CS7015 (Deep Learning) : Lecture 5

Gradient Descent (GD), Momentum Based GD, Nesterov Accelerated GD, Stochastic GD, AdaGrad, RMSProp, Adam

February 2, 2018

Mitesh M. Khapra

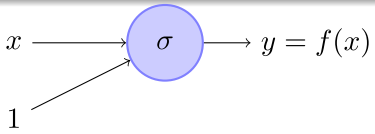
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Acknowledgements

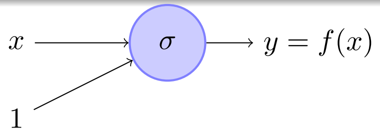
- For most of the lecture, I have borrowed ideas from the videos by Ryan Harris on "visualize backpropagation" (available on youtube)
- Some content is based on the course CS231n[a] by Andrej Karpathy and others

[a] http://cs231n.stanford.edu/2016/

Module 5.1: Learning Parameters : Infeasible (Guess Work)



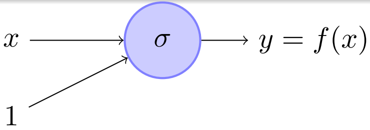
$$f(x) = \frac{1}{1+e^{-(w \cdot x+b)}}$$



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

Input for training

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



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

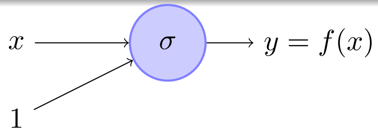
Input for training

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

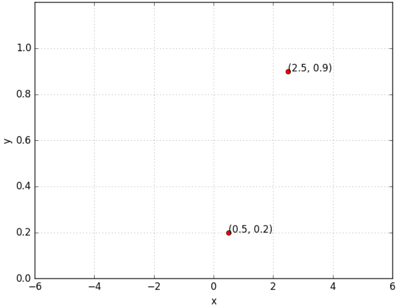
Training objective

Find w and b such that:

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

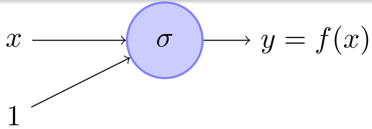


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

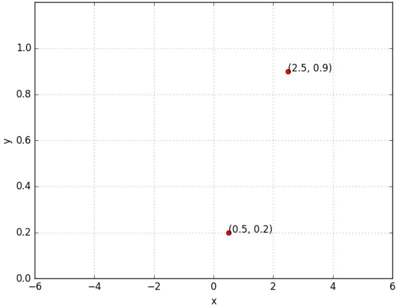


What does it mean to train the network?

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

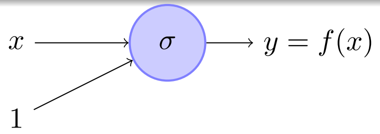


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

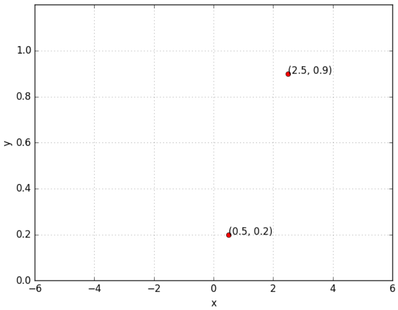


What does it mean to train the network?

- 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:

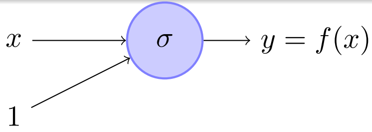


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

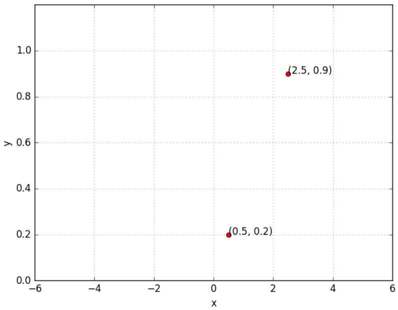


What does it mean to train the network?

- 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 $f(2.5) \rightarrow 0.9$



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

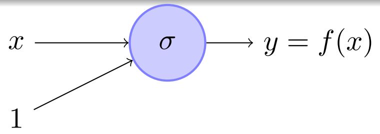


What does it mean to train the network?

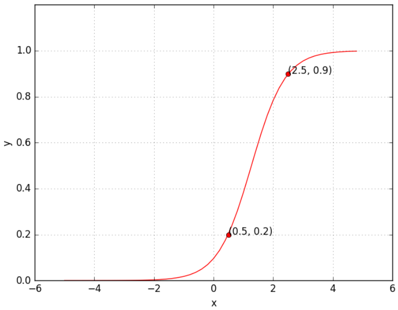
- Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$
- At the end of training we expect to find w^*, b^* such that:
- $f(0.5) \rightarrow 0.2$ and $f(2.5) \rightarrow 0.9$

In other words...

- We hope to find a sigmoid function such that $(0.5, 0.2)$ and $(2.5, 0.9)$ lie on this sigmoid



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



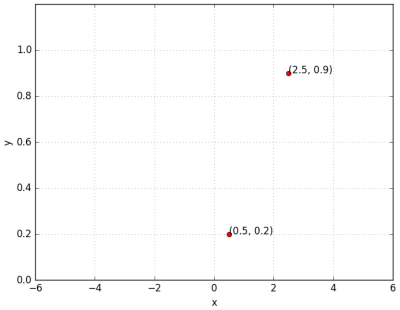
What does it mean to train the network?

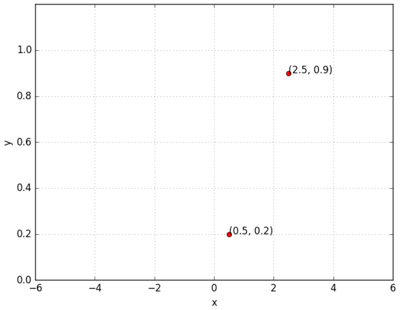
- Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$
- At the end of training we expect to find w^*, b^* such that:
- $f(0.5) \rightarrow 0.2$ and $f(2.5) \rightarrow 0.9$

In other words...

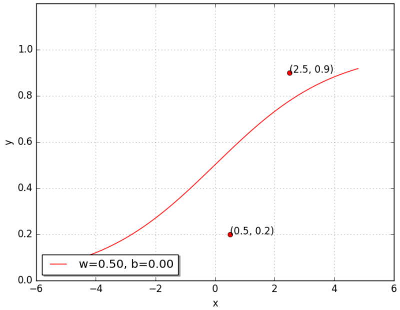
- We hope to find a sigmoid function such that $(0.5, 0.2)$ and $(2.5, 0.9)$ lie on this sigmoid

Let us see this in more detail....

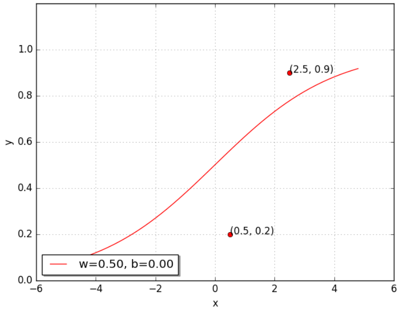

(2.5, 0.9)
(0.5, 0.2)
1.0
0.8
0.6
0.4
0.2
0.0
y
−6
−4
−2
0
2
4
6
x



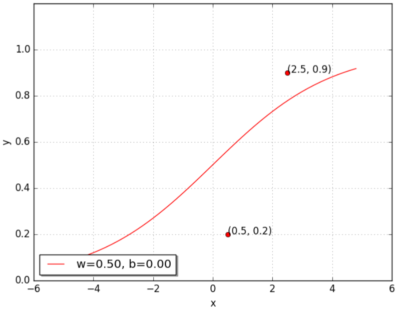
- Can we try to find such a w^*, b^* manually



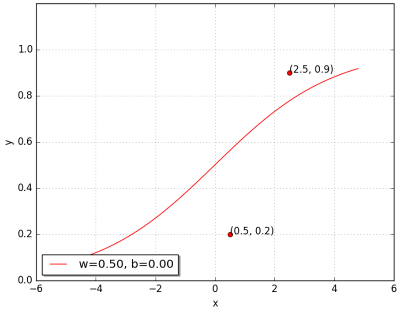
- Can we try to find such a w^*, b^* manually
- Let us try a random guess.. (say, $w = 0.5, b = 0$)



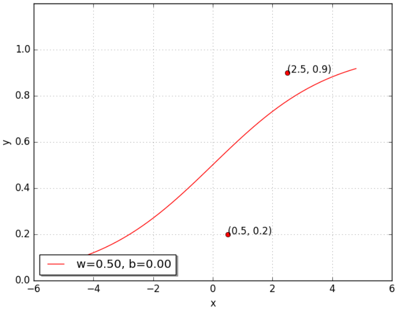
- Can we try to find such a w^*, b^* manually
- Let us try a random guess.. (say, $w = 0.5, b = 0$)
- Clearly not good, but how bad is it ?



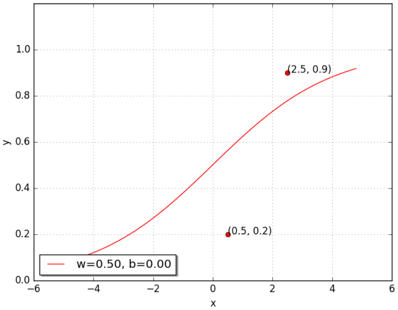
- Can we try to find such a w^*, b^* manually
- Let us try a random guess.. (say, $w = 0.5, b = 0$)
- Clearly not good, but how bad is it ?
- Let us revisit $\mathscr{L}(w, b)$ to see how bad it is ...



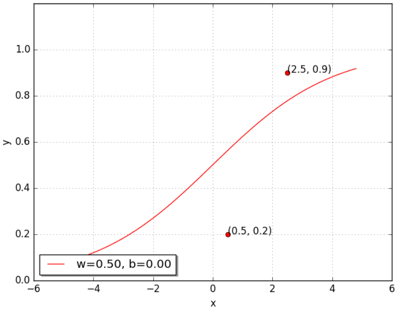
$$\mathscr{L}(w,b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$



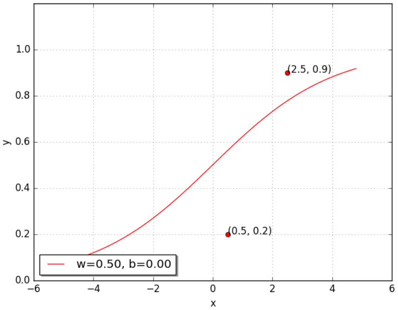
$$\mathscr{L}(w,b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$
$$= \frac{1}{2} * ((y_1 - f(x_1))^2 + (y_2 - f(x_2))^2)$$



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



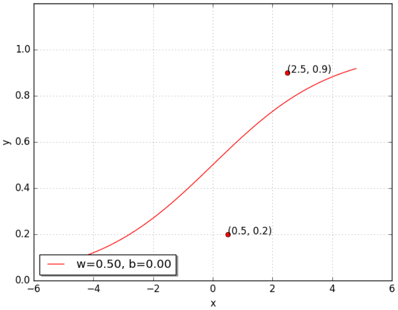
$$\begin{aligned}\mathscr{L}(w,b) &= \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2 \\ &= \frac{1}{2} * ((y_1 - f(x_1))^2 + (y_2 - f(x_2))^2) \\ &= \frac{1}{2} * ((0.9 - f(2.5))^2 + (0.2 - f(0.5))^2) \\ &= 0.073\end{aligned}$$



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

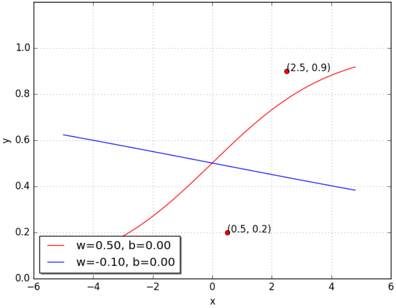
We want $\mathscr{L}(w,b)$ to be as close to 0 as possible

Let us try some other values of w, b



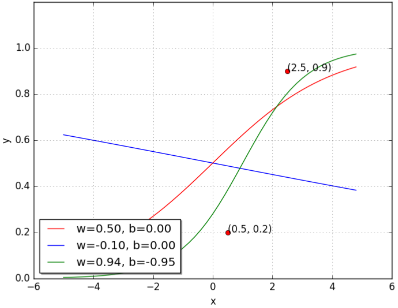
w	b	$\mathscr{L}(w,b)$
0.50	0.00	0.0730

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

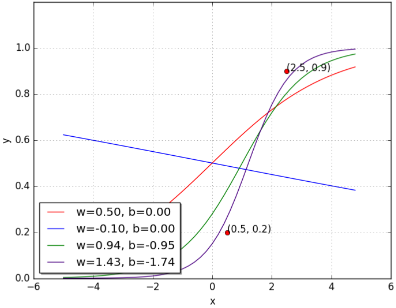
Oops!! this made things even worse...



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
0.94	-0.94	0.0214

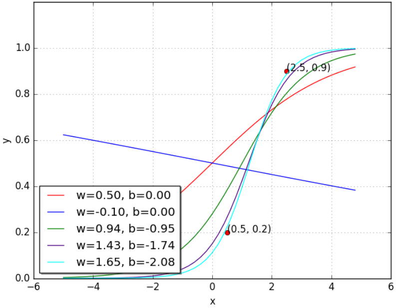
Perhaps it would help to push w and b in the other direction...



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
0.94	-0.94	0.0214
1.42	-1.73	0.0028

Let us keep going in this direction, *i.e.*, increase w and decrease b

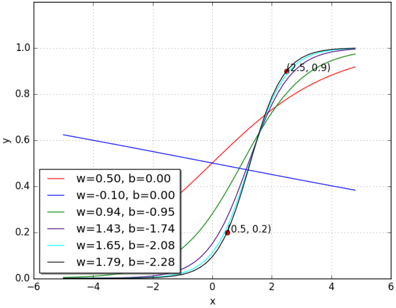


Let us try some other values of w, b

w	b	$\mathscr{L}(w,b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003

Let us keep going in this direction, *i.e.*, increase w and decrease b

Let us try some other values of w, b

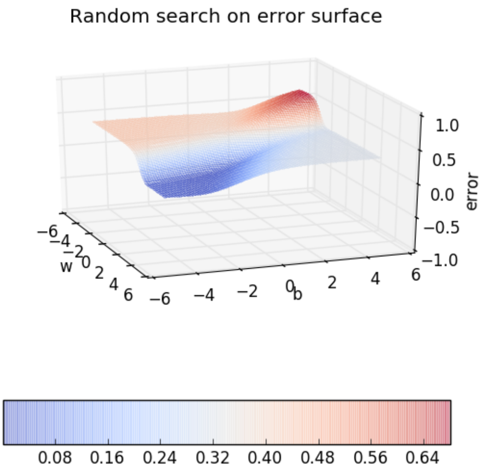


w	b	$\mathscr{L}(w,b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003
1.78	-2.27	0.0000

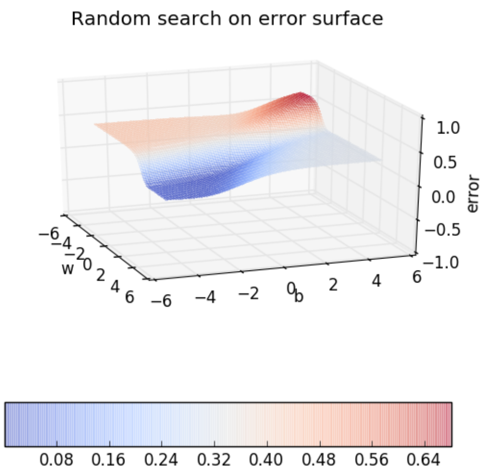
With some guess work and intuition we were able to find the right values for w and b

Let us look at something better than our "guess work" algorithm....

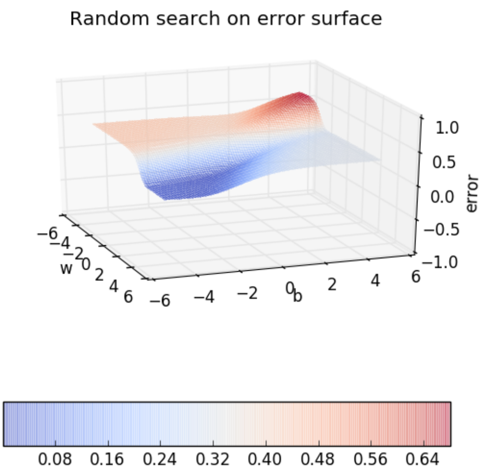
- Since we have only 2 points and 2 parameters $(w,\ b)$ we can easily plot $\mathscr{L}(w,b)$ for different values of $(w,\ b)$ and pick the one where $\mathscr{L}(w,b)$ is minimum



- 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

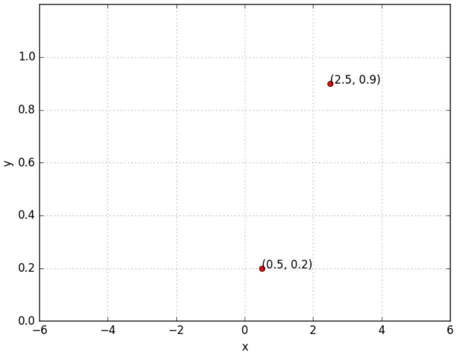


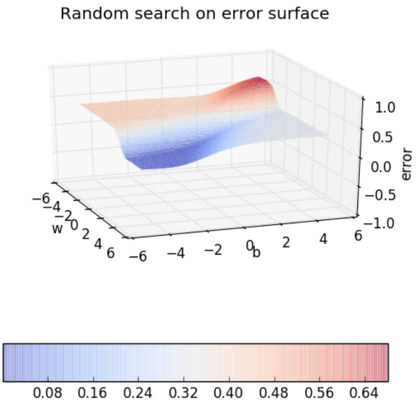
- Since we have only 2 points and 2 parameters $(w,\ b)$ we can easily plot $\mathscr{L}(w,b)$ for different values of $(w,\ b)$ and pick the one where $\mathscr{L}(w,b)$ is minimum
- But of course this becomes intractable once you have many more data points and many more parameters !!

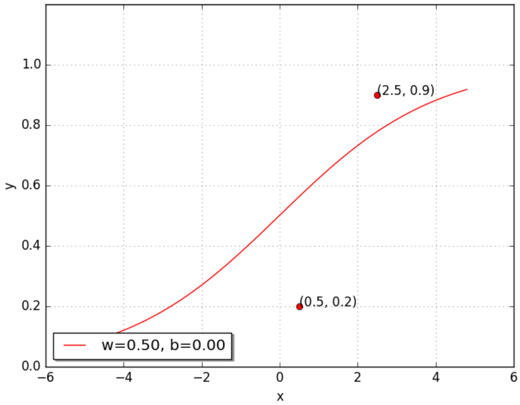


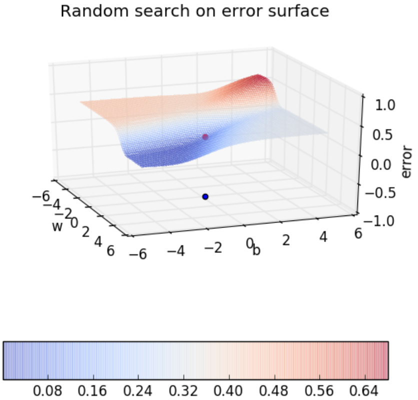
- Since we have only 2 points and 2 parameters $(w,\ b)$ we can easily plot $\mathscr{L}(w,b)$ for different values of $(w,\ b)$ and pick the one where $\mathscr{L}(w,b)$ is minimum
- But of course this becomes intractable once you have many more data points and many more parameters !!
- Further, even here we have plotted the error surface only for a small range of $(w,\ b)$ [from $(-6,6)$ and not from $(-\inf,\inf)$]

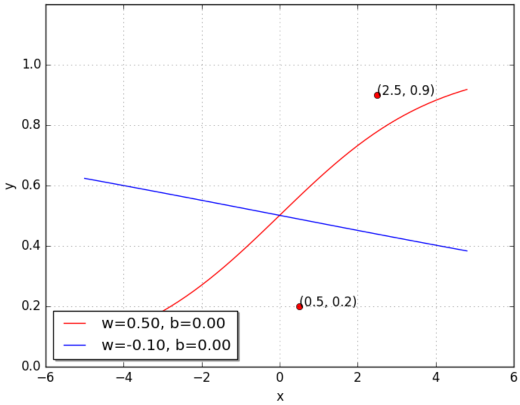
Let us look at the geometric interpretation of our "guess work" algorithm in terms of this error surface


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

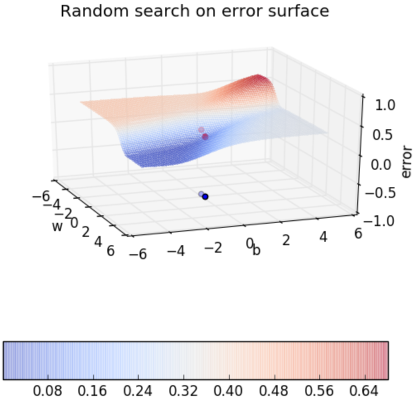

Random search on error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64


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

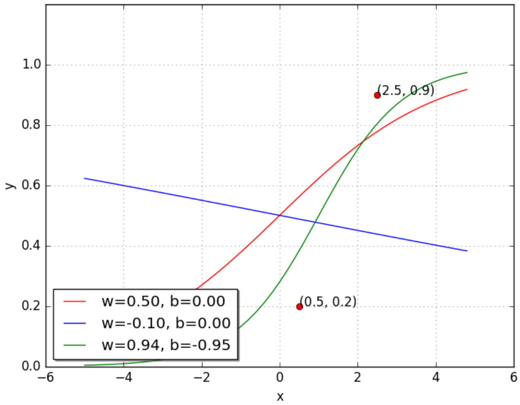

Random search on error surface
error
w
b



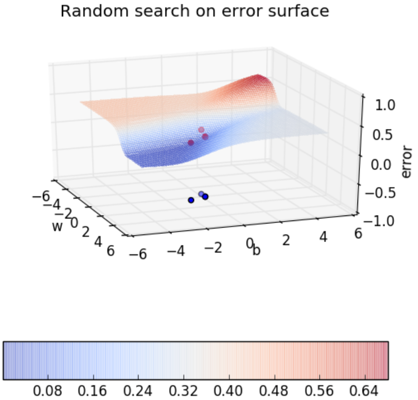
(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
y
x



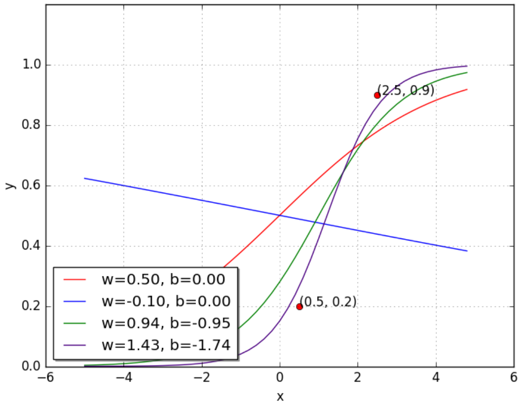
Random search on error surface
w
b
error
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



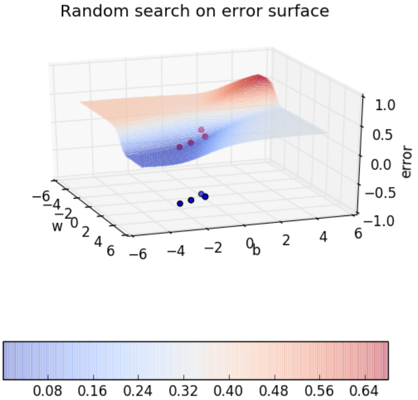
(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
x
y



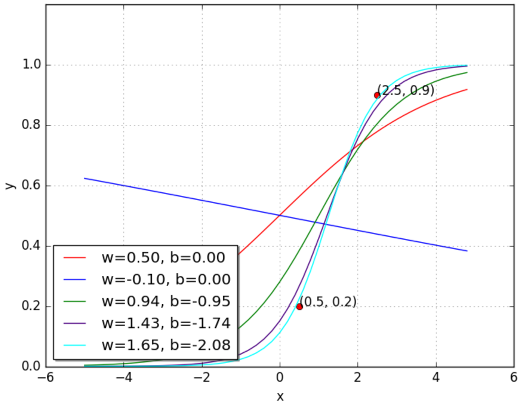
Random search on error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

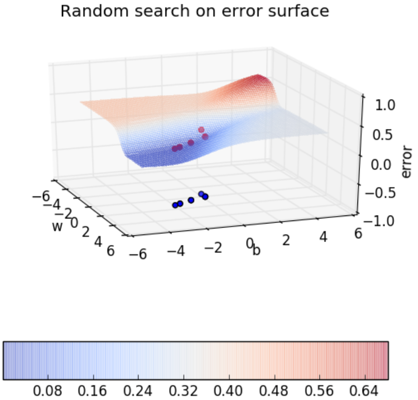


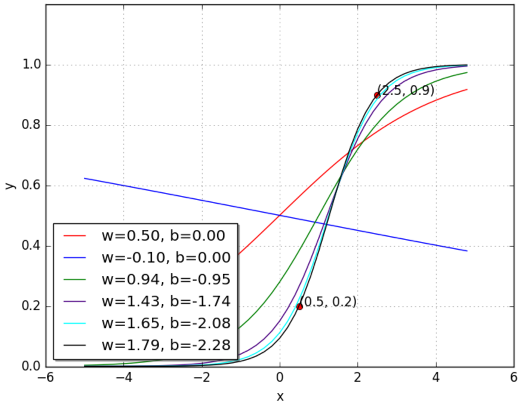
w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
w=1.43, b=-1.74
(2.5, 0.9)
(0.5, 0.2)
x
y



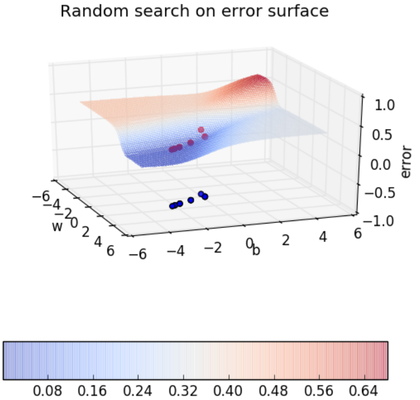
Random search on error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64


(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
w=1.43, b=-1.74
w=1.65, b=-2.08
x
y


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
y 0.6
0.4
0.2
0.0
−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
w=1.79, b=-2.28



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

Module 5.2: Learning Parameters : Gradient Descent

Now let's see if there is a more efficient and principled way of doing this

Goal

Find a better way of traversing the error surface so that we can reach the minimum value quickly without resorting to brute force search!

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

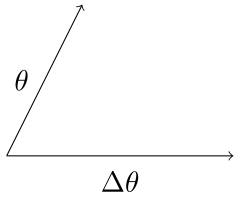
change in the values of w, b

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b

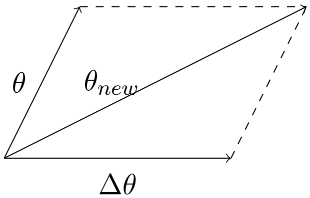


vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b

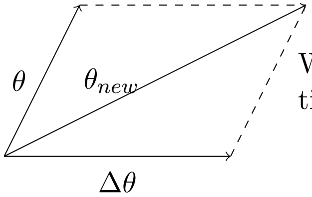


vector of parameters, say, randomly initialized

$\theta = [w, b]$

$\Delta\theta = [\Delta w, \Delta b]$

change in the values of w, b



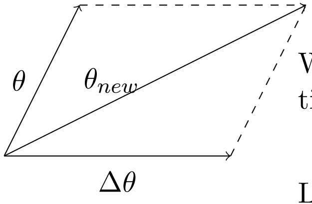
We moved in the direction of $\Delta\theta$

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



We moved in the direction of $\Delta\theta$

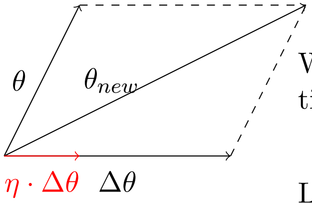
Let us be a bit conservative: move only by a small amount η

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



We moved in the direction of $\Delta\theta$

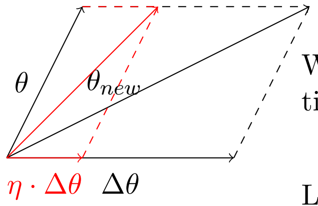
Let us be a bit conservative: move only by a small amount η

vector of parameters, say, randomly initialized

$\theta = [w, b]$

$\Delta\theta = [\Delta w, \Delta b]$

change in the values of w, b



We moved in the direction of $\Delta\theta$

Let us be a bit conservative: move only by a small amount η

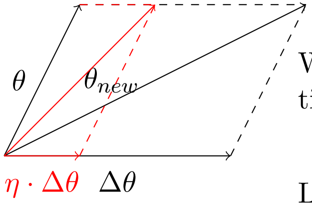
vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b

$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$



We moved in the direction of $\Delta\theta$

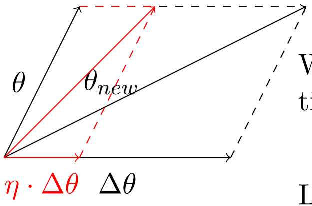
Let us be a bit conservative: move only by a small amount η

vector of parameters, say, randomly initialized

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

change in the values of w, b



We moved in the direction of $\Delta\theta$

Let us be a bit conservative: move only by a small amount η

$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$

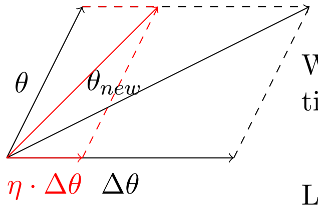
Question: What is the right $\Delta\theta$ to use?

vector of parameters, say, randomly initialized

$\theta = [w, b]$

$\Delta\theta = [\Delta w, \Delta b]$

change in the values of w, b



We moved in the direction of $\Delta\theta$

Let us be a bit conservative: move only by a small amount η

$\theta_{new} = \theta + \eta \cdot \Delta\theta$

Question: What is the right $\Delta\theta$ to use?

The answer comes from Taylor series

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

$$\mathscr{L}(\theta + \eta u) = \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathscr{L}(\theta) u + \frac{\eta^3}{3!} * ... + \frac{\eta^4}{4!} * ...$$

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

$$\mathscr{L}(\theta + \eta u) = \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathscr{L}(\theta) u + \frac{\eta^3}{3!} * ... + \frac{\eta^4}{4!} * ...$$
$$= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) \; [\eta \; is \; typically \; small, \; so \; \eta^2, \eta^3, ... \to 0]$$

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

$$\begin{aligned}\mathscr{L}(\theta + \eta u) &= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathscr{L}(\theta) u + \frac{\eta^3}{3!} * \ldots + \frac{\eta^4}{4!} * \ldots \\ &= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) \ [\eta \ \textit{is typically small, so} \ \eta^2, \eta^3, \ldots \to 0]\end{aligned}$$

Note that the move (ηu) would be favorable only if,

$$\mathscr{L}(\theta + \eta u) - \mathscr{L}(\theta) < 0 \ [\textit{i.e., if the new loss is less than the previous loss}]$$

For ease of notation, let $\Delta\theta = u$, then from Taylor series, we have,

$$\begin{aligned}
\mathscr{L}(\theta + \eta u) &= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathscr{L}(\theta) u + \frac{\eta^3}{3!} * \ldots + \frac{\eta^4}{4!} * \ldots \\
&= \mathscr{L}(\theta) + \eta * u^T \nabla \mathscr{L}(\theta) \ [\eta \ \textit{is typically small, so} \ \eta^2, \eta^3, \ldots \to 0]
\end{aligned}$$

Note that the move (ηu) would be favorable only if,

$$\mathscr{L}(\theta + \eta u) - \mathscr{L}(\theta) < 0 \ [\textit{i.e., if the new loss is less than the previous loss}]$$

This implies,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$?

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Let's see....

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Let's see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Let's see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

$$-1 \le cos(\beta) = \frac{u^T \nabla \mathscr{L}(\theta)}{||u|| * ||\nabla \mathscr{L}(\theta)||} \le 1$$

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Let's see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

$$-1 \leq cos(\beta) = \frac{u^T \nabla \mathscr{L}(\theta)}{||u|| * ||\nabla \mathscr{L}(\theta)||} \leq 1$$

Multiply throughout by $k = ||u|| * ||\nabla \mathscr{L}(\theta)||$

$$-k \leq k * cos(\beta) = u^T \nabla \mathscr{L}(\theta) \leq k$$

Okay, so we have,

$$u^T \nabla \mathscr{L}(\theta) < 0$$

But, what is the range of $u^T \nabla \mathscr{L}(\theta)$? Let's see....
Let β be the angle between u^T and $\nabla \mathscr{L}(\theta)$, then we know that,

$$-1 \leq cos(\beta) = \frac{u^T \nabla \mathscr{L}(\theta)}{||u|| * ||\nabla \mathscr{L}(\theta)||} \leq 1$$

Multiply throughout by $k = ||u|| * ||\nabla \mathscr{L}(\theta)||$

$$-k \leq k * cos(\beta) = u^T \nabla \mathscr{L}(\theta) \leq k$$

Thus, $\mathscr{L}(\theta + \eta u) - \mathscr{L}(\theta) = u^T \nabla \mathscr{L}(\theta) = k * cos(\beta)$ will be most negative when $cos(\beta) = -1$ $i.e.$, when β is 180°

Gradient Descent Rule

- The direction u that we intend to move in should be at 180° w.r.t. the gradient

Gradient Descent Rule

- The direction u that we intend to move in should be at $180°$ w.r.t. the gradient
- In other words, move in a direction opposite to the gradient

Gradient Descent Rule

- The direction u that we intend to move in should be at $180°$ w.r.t. the gradient
- In other words, move in a direction opposite to the gradient

Parameter Update Equations

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$b_{t+1} = b_t - \eta \nabla b_t$$

$$where, \nabla w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}_{at\ w=w_t, b=b_t}, \nabla b_t = \frac{\partial \mathscr{L}(w,b)}{\partial b}_{at\ w=w_t, b=b_t}$$

Gradient Descent Rule

- The direction u that we intend to move in should be at 180° w.r.t. the gradient
- In other words, move in a direction opposite to the gradient

Parameter Update Equations

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$b_{t+1} = b_t - \eta \nabla b_t$$

$$where, \nabla w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}_{at\ w = w_t, b = b_t}, \nabla b_t = \frac{\partial \mathscr{L}(w,b)}{\partial b}_{at\ w = w_t, b = b_t}$$

So we now have a more principled way of moving in the w-b plane than our "guess work" algorithm

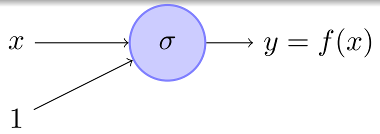
- Let's create an algorithm from this rule ...

- Let's create an algorithm from this rule ...

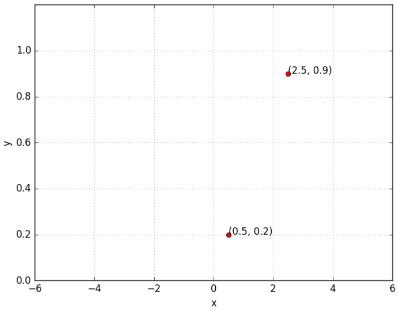
Algorithm 1: gradient_descent()

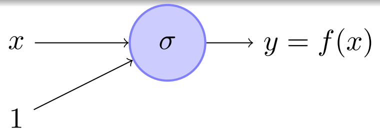
$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
while $t < max\_iterations$ **do**
    $w_{t+1} \leftarrow w_t - \eta \nabla w_t$;
    $b_{t+1} \leftarrow b_t - \eta \nabla b_t$;
end

- To see this algorithm in practice let us first derive ∇w and ∇b for our toy neural network

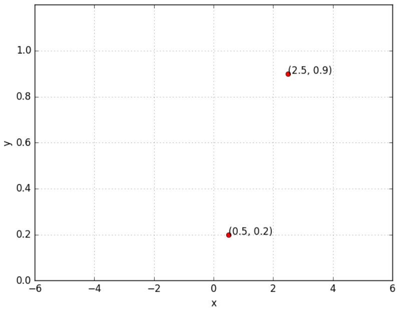


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

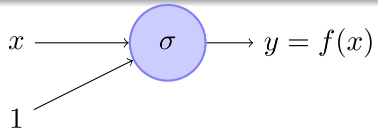




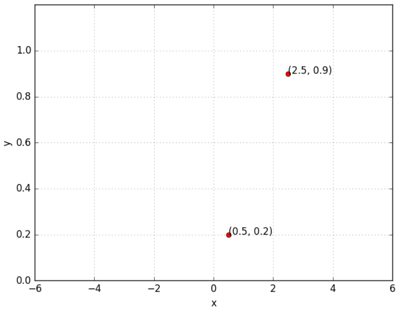
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



Let's assume there is only 1 point to fit (x, y)

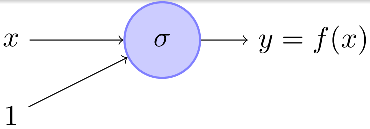


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

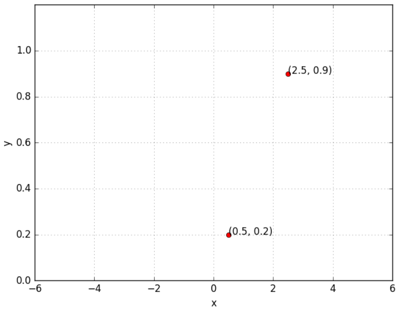


Let's assume there is only 1 point to fit (x, y)

$$\mathscr{L}(w, b) = \frac{1}{2} * (f(x) - y)^2$$



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



Let's assume there is only 1 point to fit (x, y)

$$\mathscr{L}(w, b) = \frac{1}{2} * (f(x) - y)^2$$

$$\nabla w = \frac{\partial \mathscr{L}(w, b)}{\partial w} = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$

$$
\begin{aligned}
\nabla w &= \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2] \\
&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)] \\
&= (f(x) - y) * \frac{\partial}{\partial w}(f(x))
\end{aligned}
$$

$$
\begin{aligned}
\nabla w &= \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2] \\
&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)] \\
&= (f(x) - y) * \frac{\partial}{\partial w}(f(x)) \\
&= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)
\end{aligned}
$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$
\begin{aligned}
\nabla w &= \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2] \\
&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)] \\
&= (f(x) - y) * \frac{\partial}{\partial w}(f(x)) \\
&= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)
\end{aligned}
$$

$$
\begin{aligned}
&\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right) \\
&= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w}(e^{-(wx+b)}))
\end{aligned}
$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2}\frac{\partial}{\partial w}(e^{-(wx+b)}))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)})\frac{\partial}{\partial w}(-(wx + b)))$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$

$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$

$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$

$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

$$= \frac{-1}{(1 + e^{-(wx+b)})^2}\frac{\partial}{\partial w}(e^{-(wx+b)}))$$

$$= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)})\frac{\partial}{\partial w}(-(wx + b)))$$

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

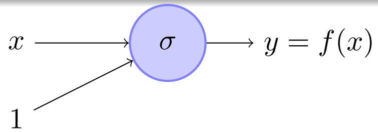
$$= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x)$$

$$\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$$
$$= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$$
$$= (f(x) - y) * \frac{\partial}{\partial w}(f(x))$$
$$= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$

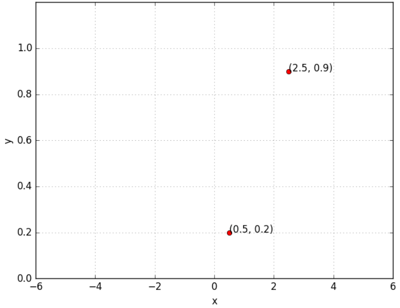
$$\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right)$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2}\frac{\partial}{\partial w}(e^{-(wx+b)}))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)})\frac{\partial}{\partial w}(-(wx + b)))$$
$$= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x)$$
$$= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x)$$
$$= f(x) * (1 - f(x)) * x$$

$$
\begin{aligned}
\nabla w &= \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2] \\
&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)] \\
&= (f(x) - y) * \frac{\partial}{\partial w}(f(x)) \\
&= (f(x) - y) * \frac{\partial}{\partial w}\Big(\frac{1}{1 + e^{-(wx+b)}}\Big) \\
&= {\color{red}(f(x) - y) * f(x) * (1 - f(x)) * x}
\end{aligned}
$$

$$
\begin{aligned}
&\frac{\partial}{\partial w}\Big(\frac{1}{1 + e^{-(wx+b)}}\Big) \\
&= \frac{-1}{(1 + e^{-(wx+b)})^2}\frac{\partial}{\partial w}(e^{-(wx+b)})) \\
&= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)})\frac{\partial}{\partial w}(-(wx + b))) \\
&= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x) \\
&= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x) \\
&= f(x) * (1 - f(x)) * x
\end{aligned}
$$



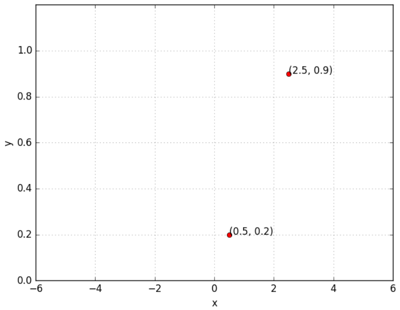
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



So if there is only 1 point (x, y), we have,

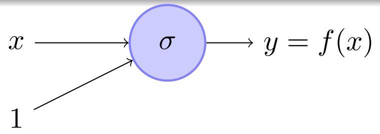
$x \longrightarrow \sigma \longrightarrow y = f(x)$, with input 1 also feeding σ

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

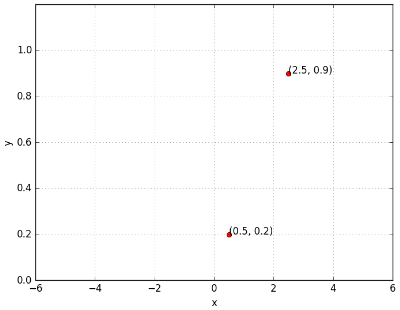


So if there is only 1 point (x, y), we have,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$



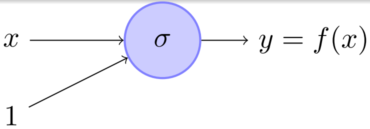
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



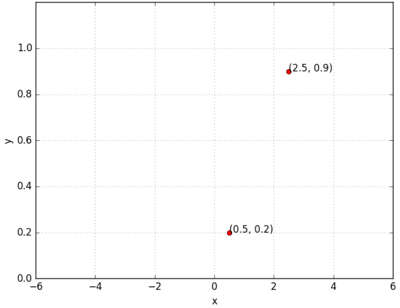
So if there is only 1 point (x, y), we have,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

For two points,



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

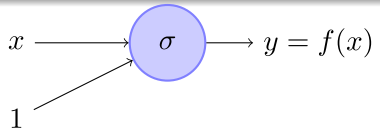


So if there is only 1 point (x, y), we have,

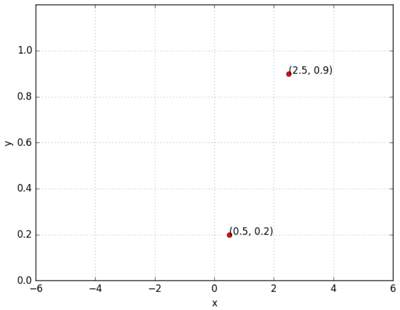
$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

For two points,

$$\nabla w = \sum_{i=1}^{2} (f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) * x_i$$



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



So if there is only 1 point (x, y), we have,

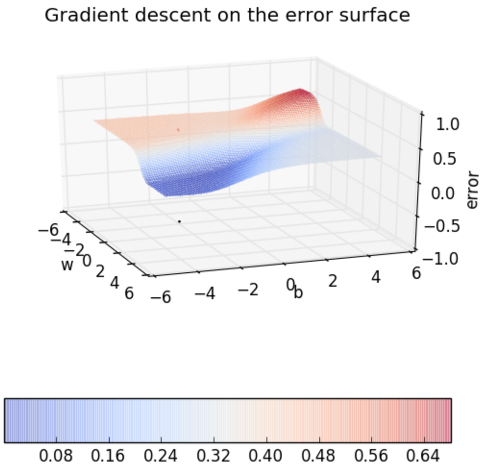
$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

For two points,

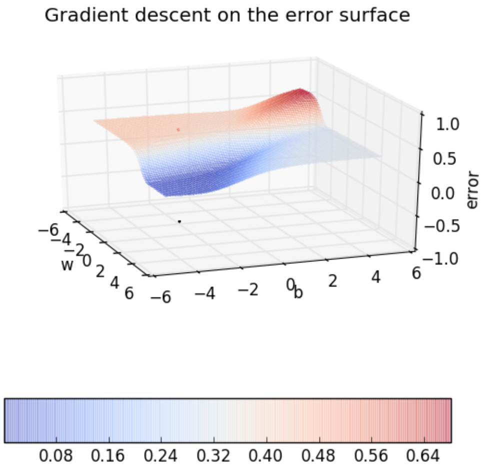
$$\nabla w = \sum_{i=1}^{2} (f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) * x_i$$

$$\nabla b = \sum_{i=1}^{2} (f(x_i) - y_i) * f(x_i) * (1 - f(x_i))$$

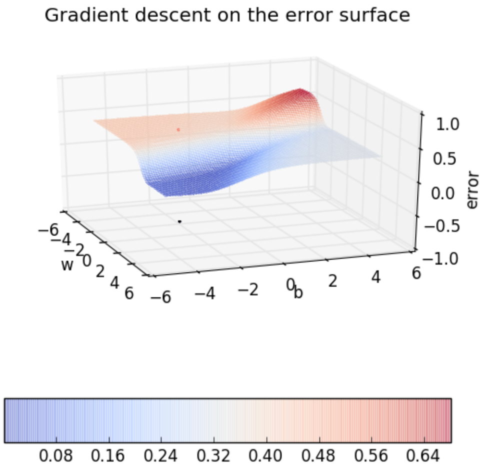
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



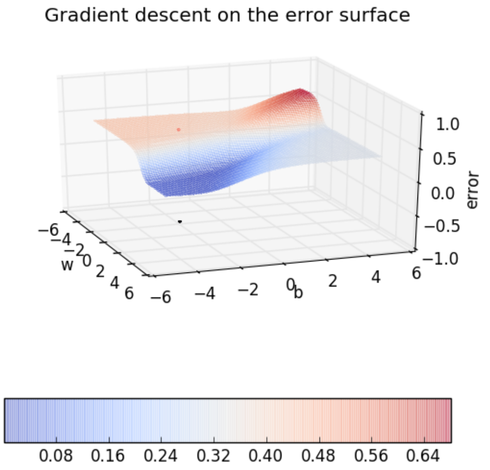
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



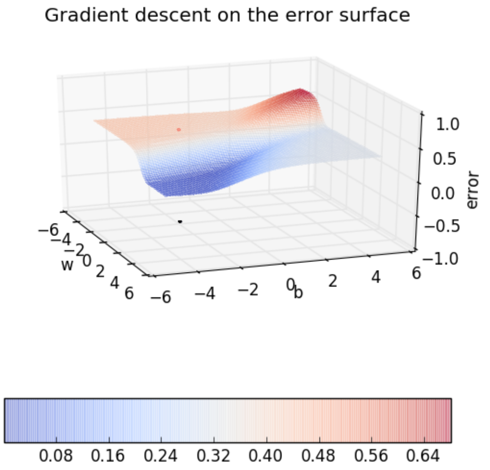
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



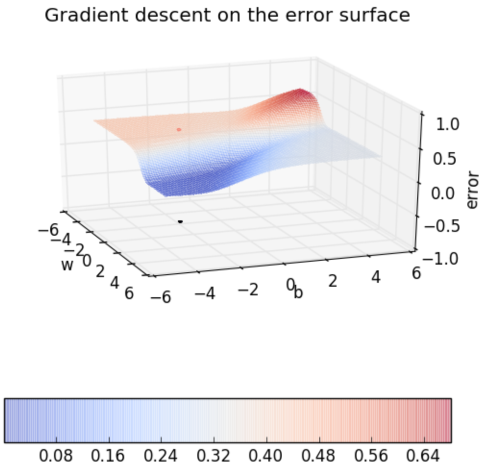
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



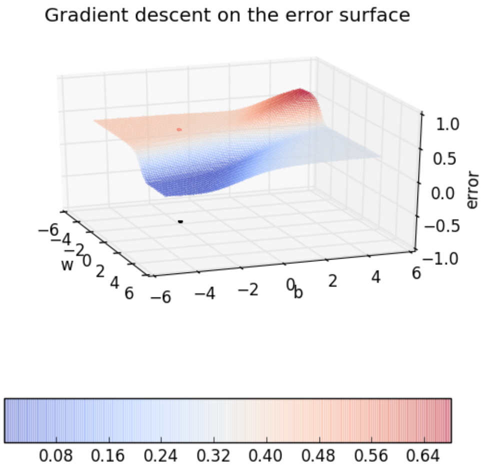
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



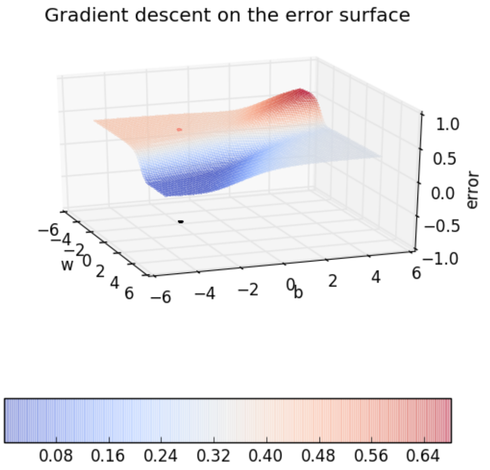
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



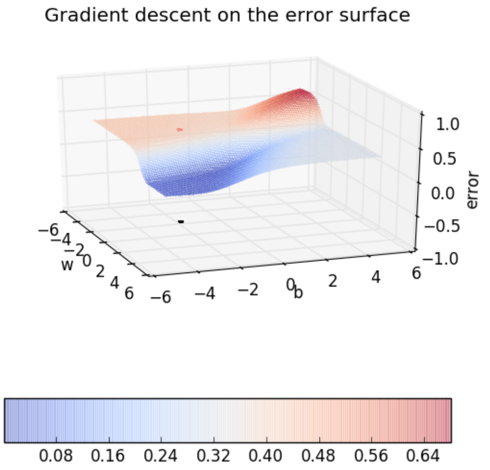
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



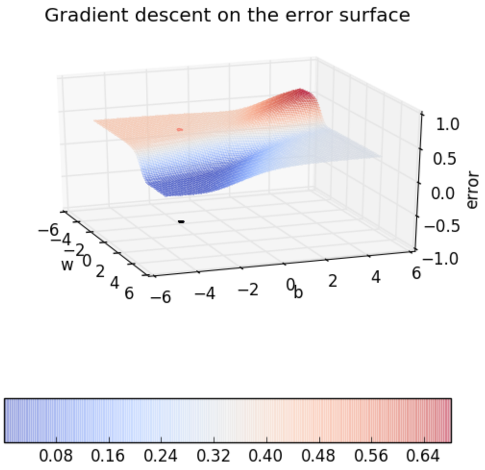
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

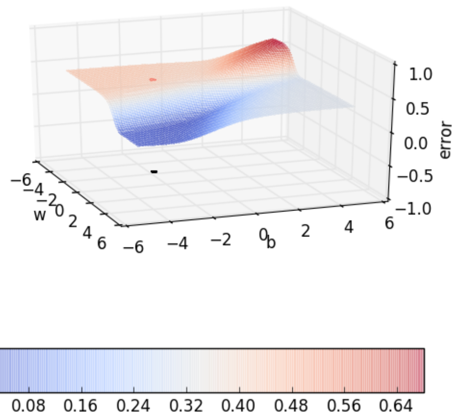


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

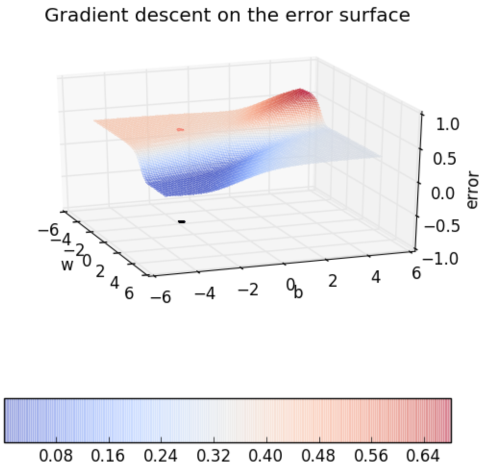


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

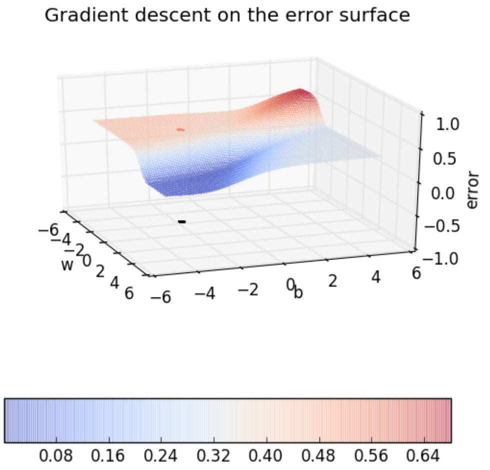
Gradient descent on the error surface



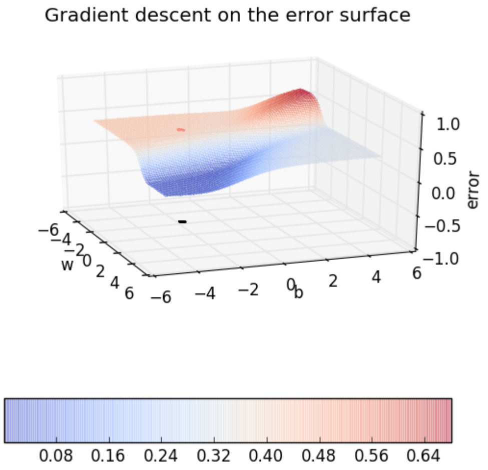
```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



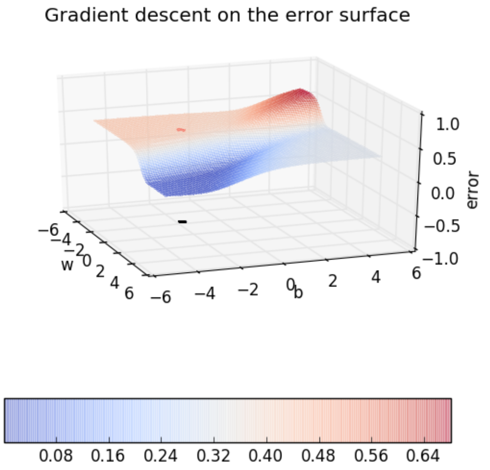
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



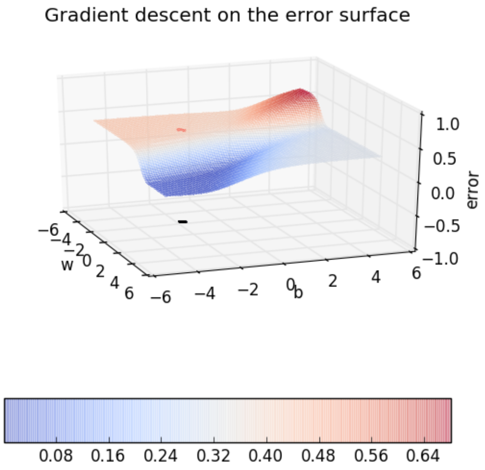
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

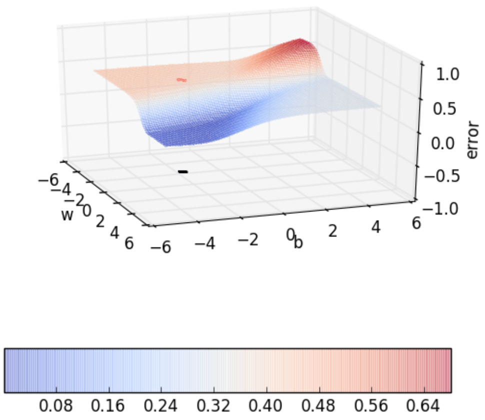


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

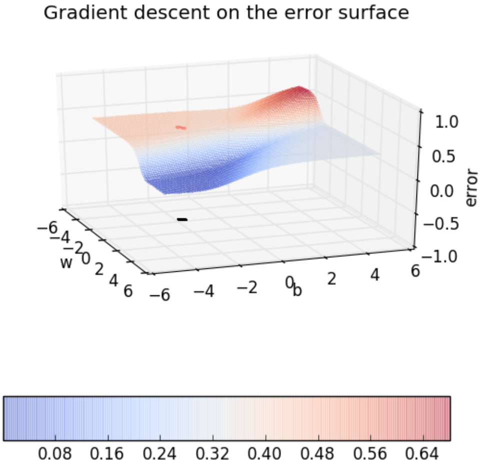


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

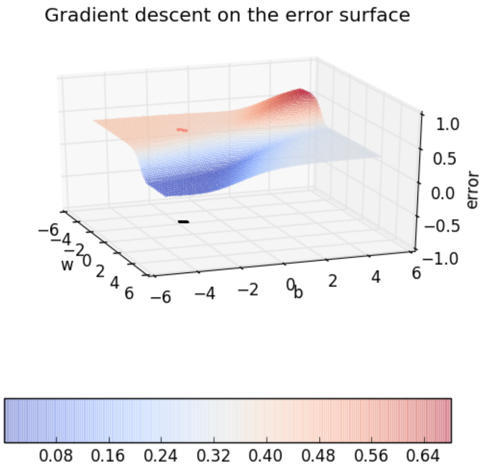
Gradient descent on the error surface



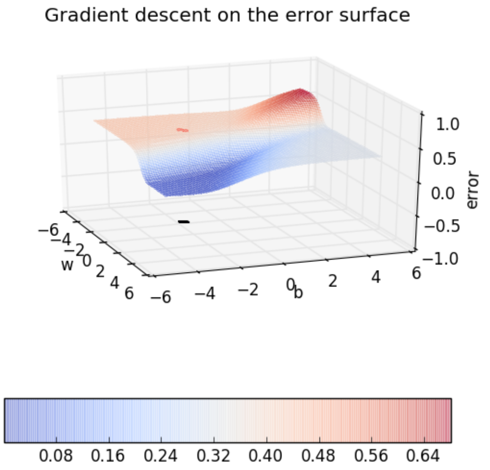
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



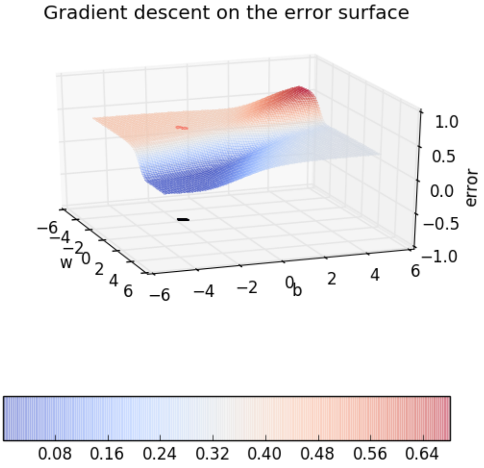
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



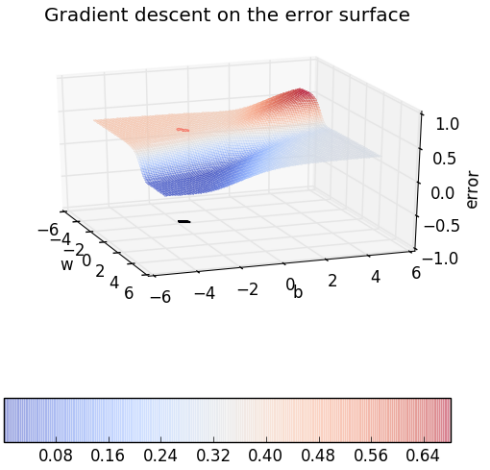
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

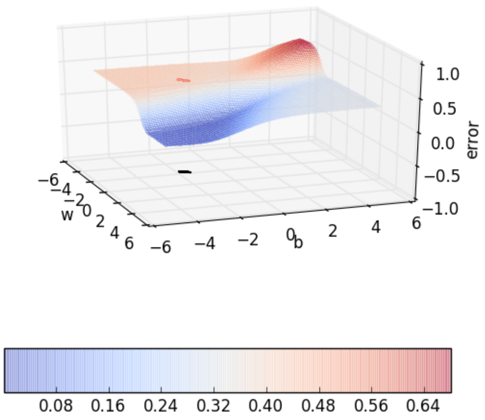


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



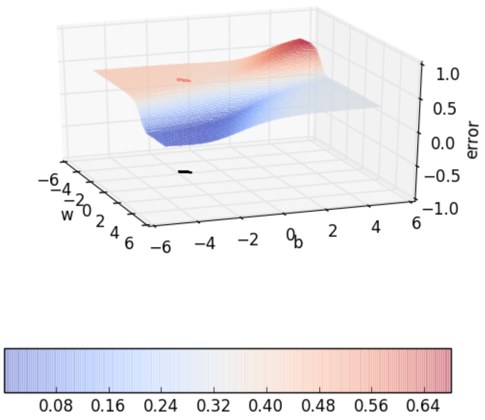
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

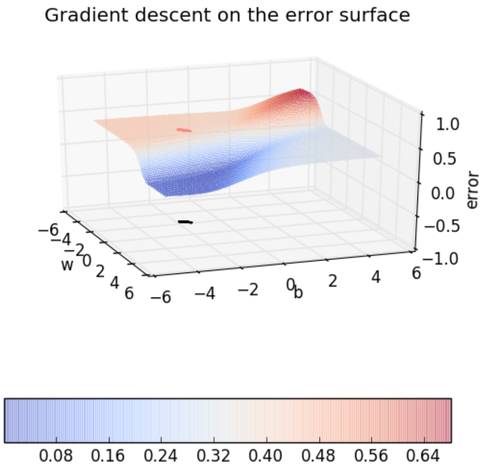


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

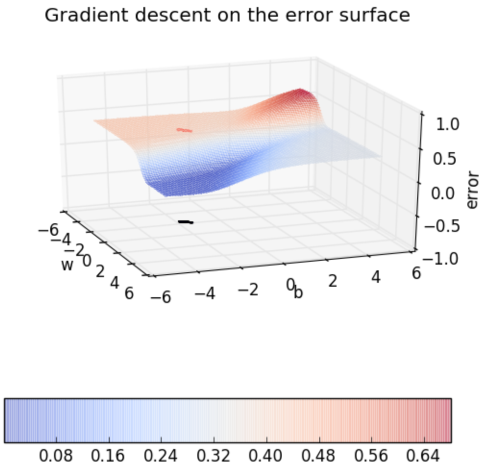
Gradient descent on the error surface



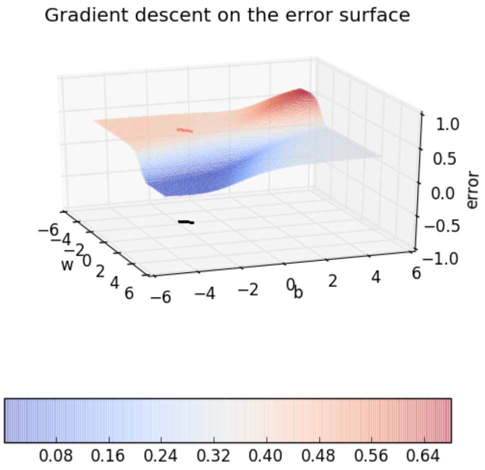
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

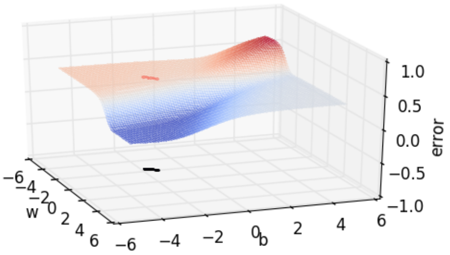


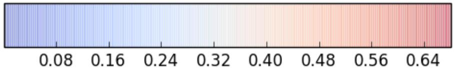
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



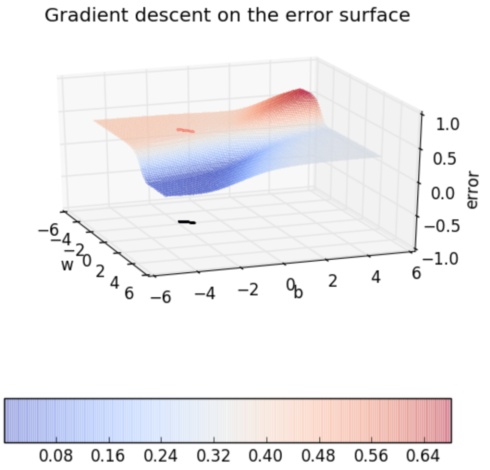
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

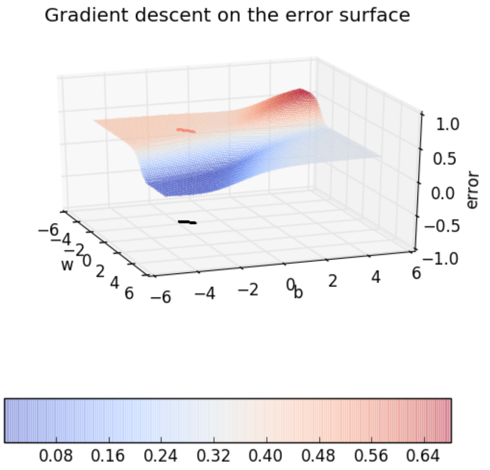




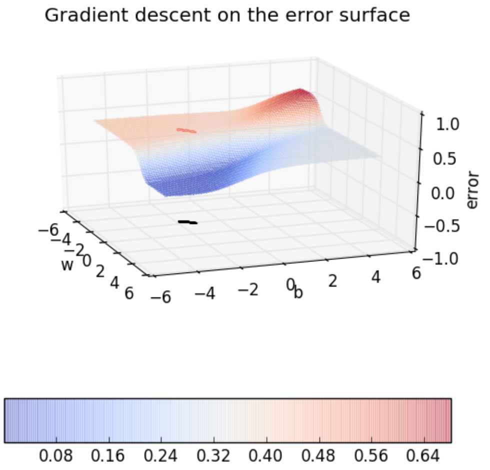
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



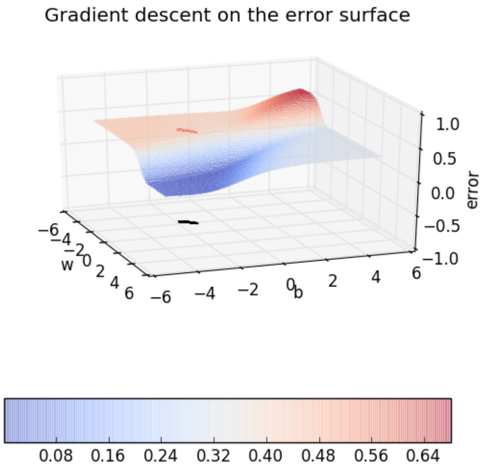
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

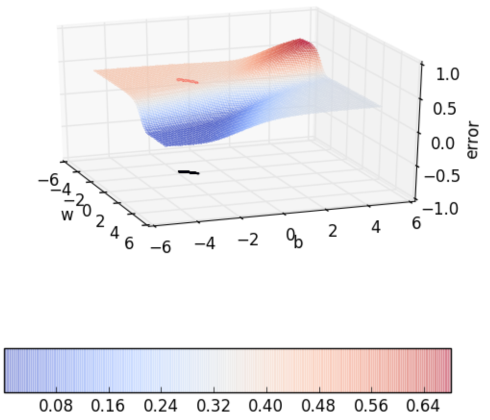


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

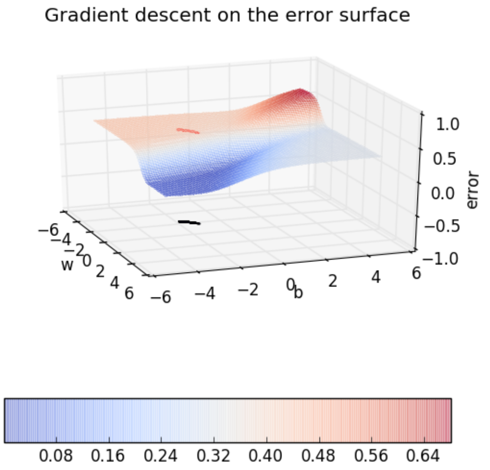


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

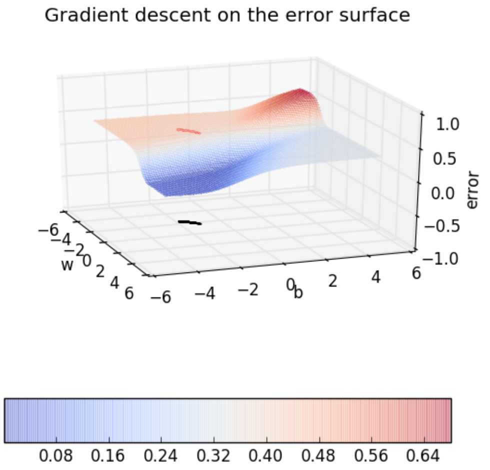
Gradient descent on the error surface



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

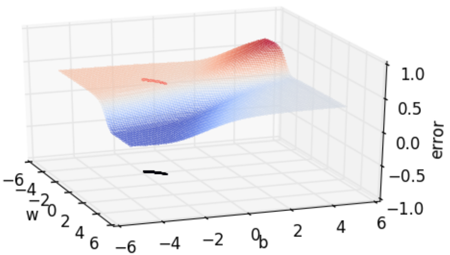


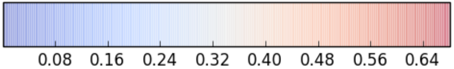
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



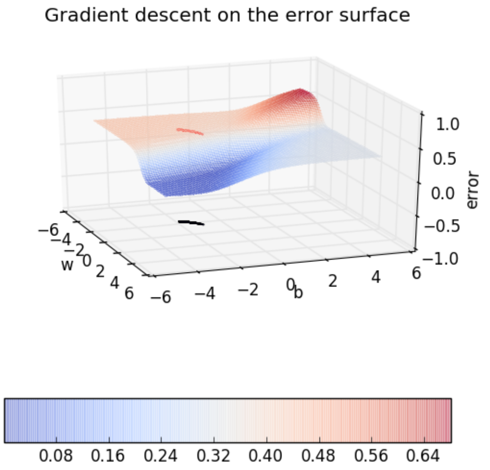
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

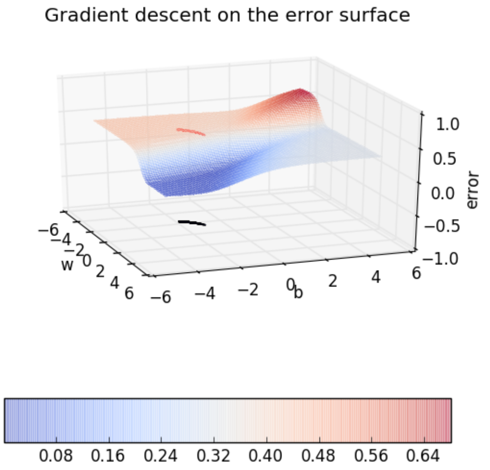




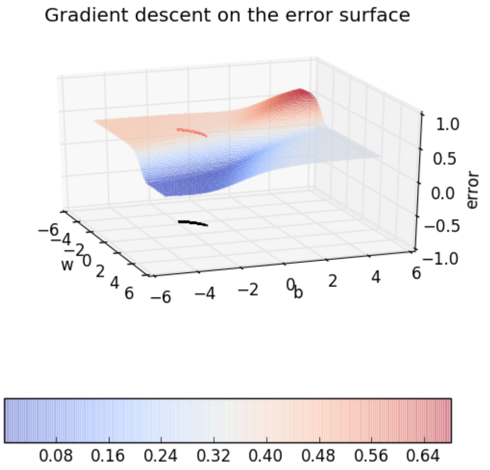
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



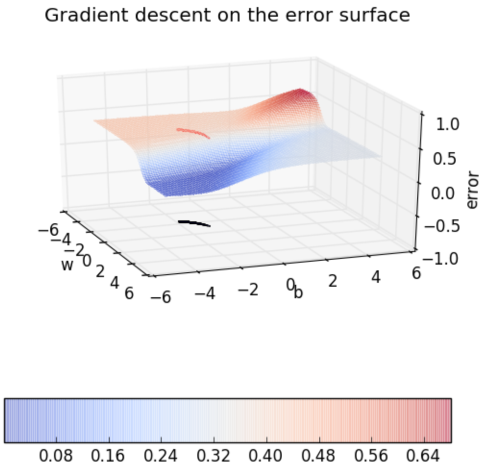
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



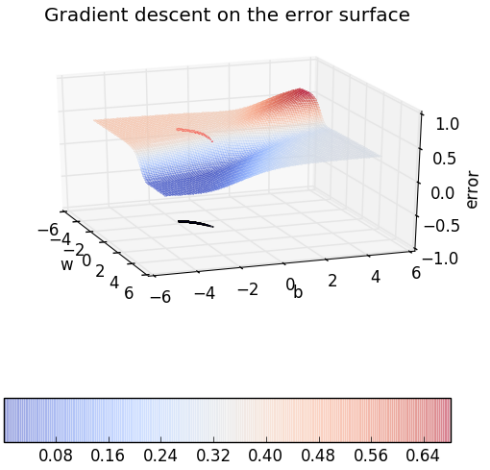
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

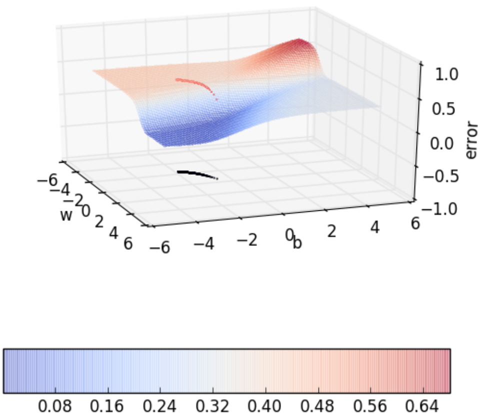


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



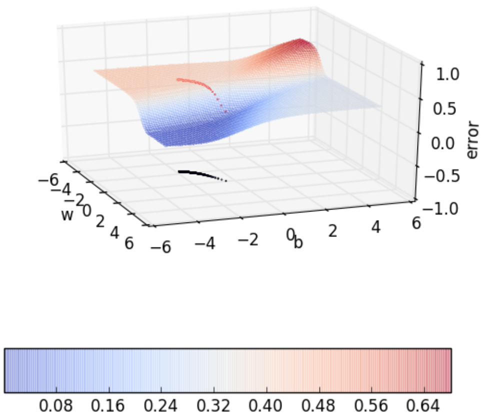
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

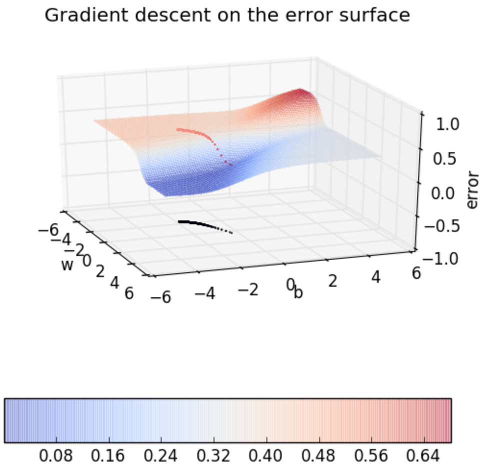


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

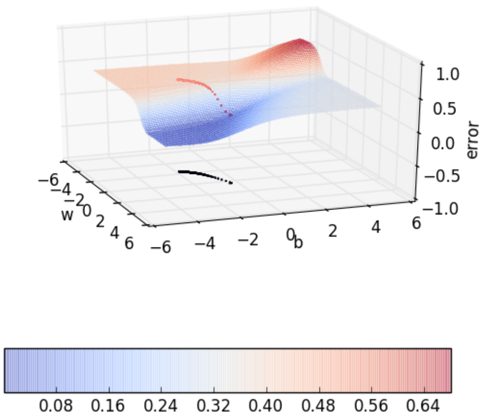


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

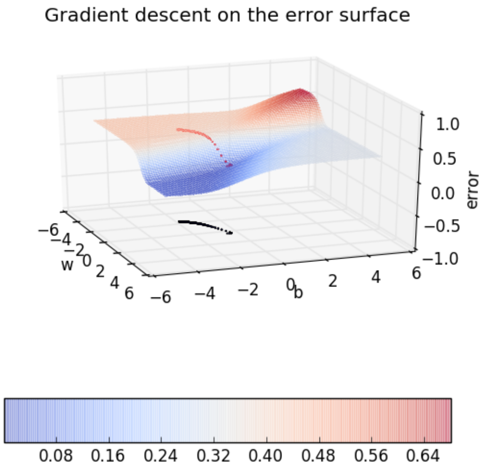


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

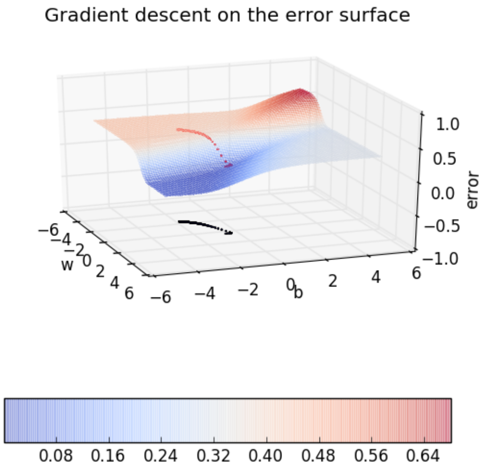
Gradient descent on the error surface



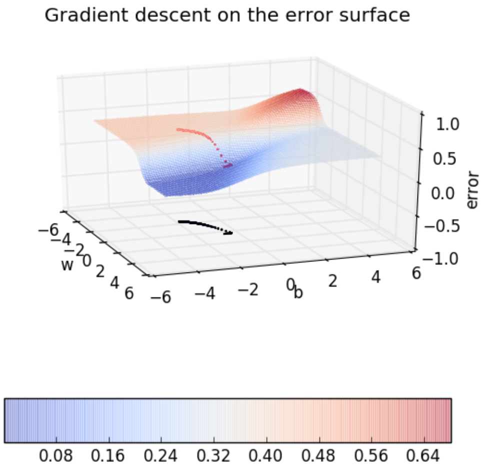
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



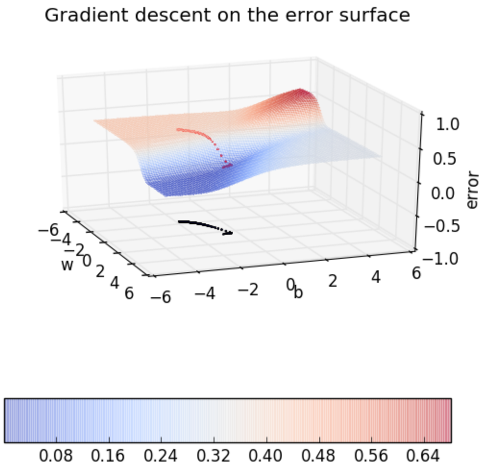
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

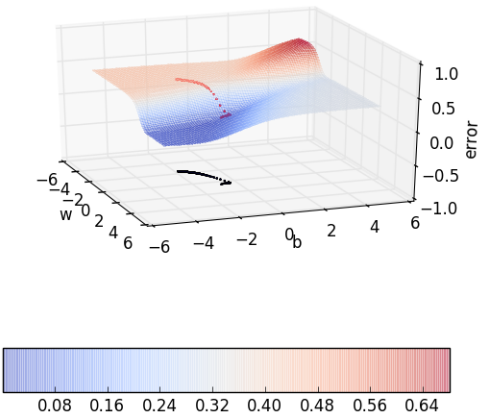


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

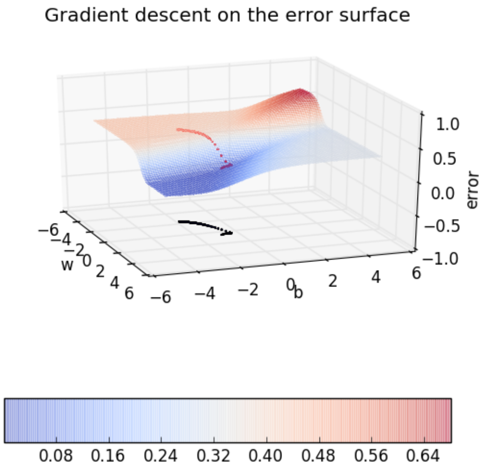


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

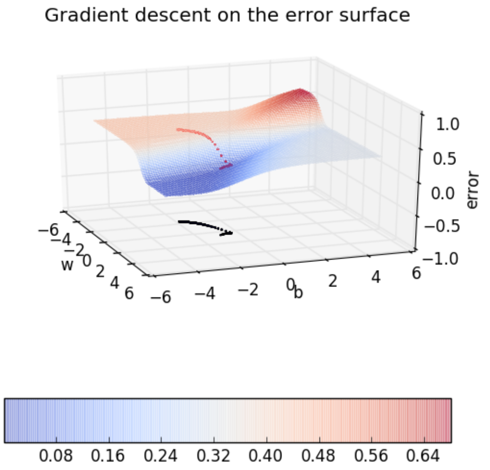
Gradient descent on the error surface



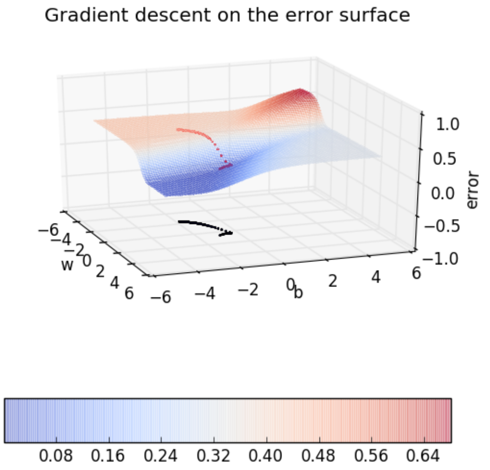
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



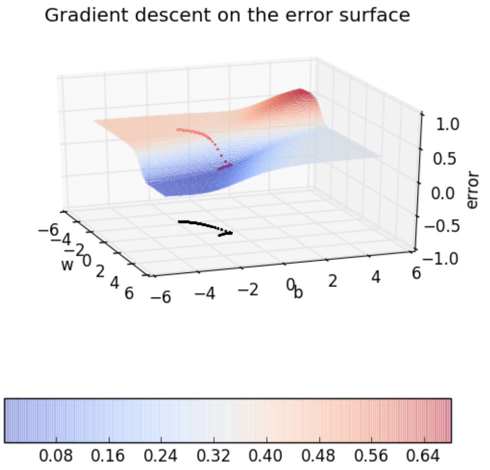
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



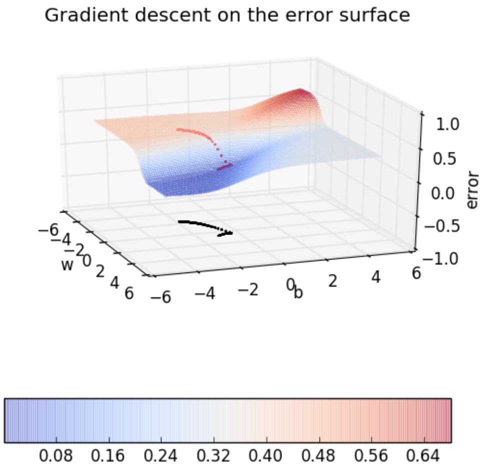
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



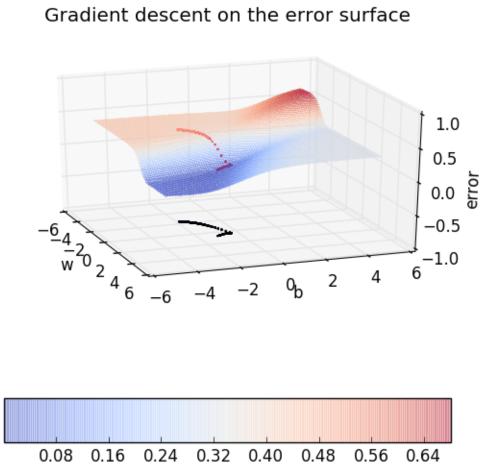
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



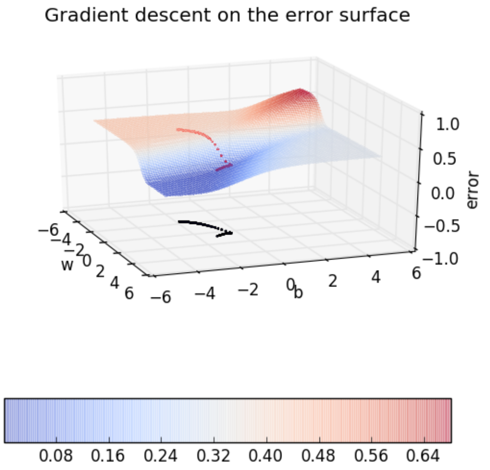
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



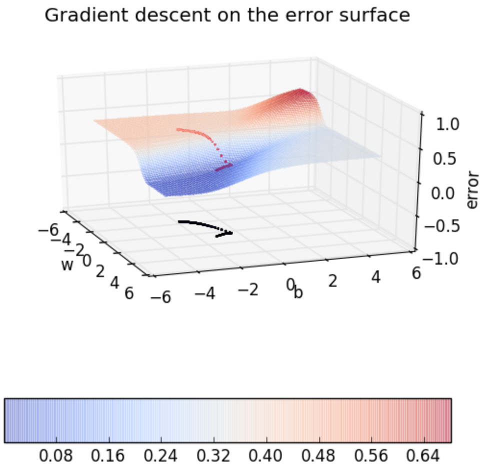
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



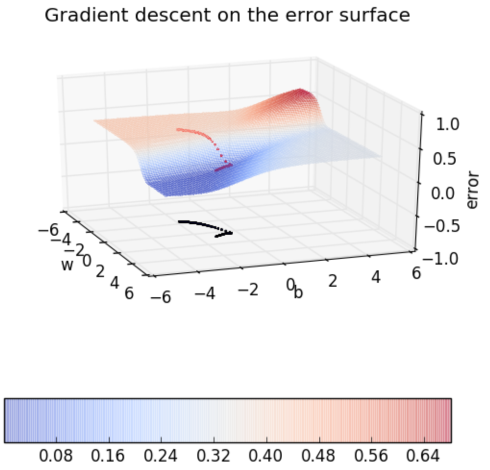
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



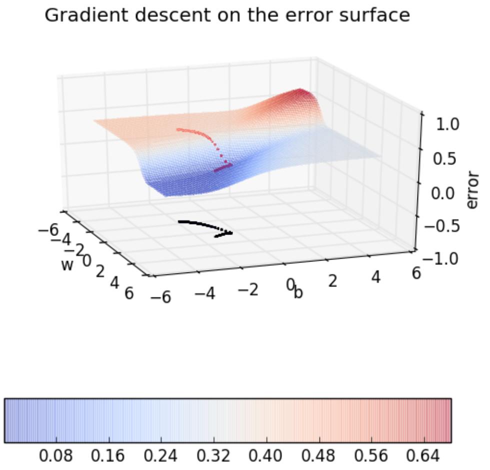
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



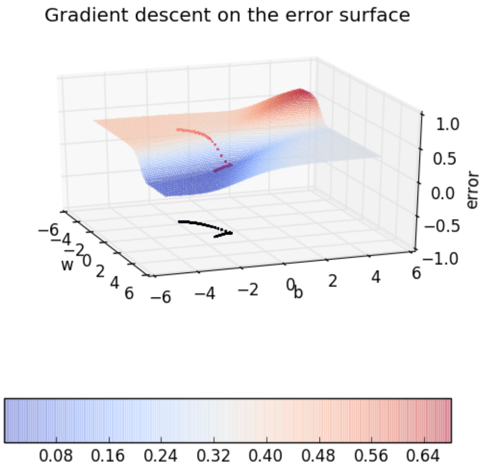
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



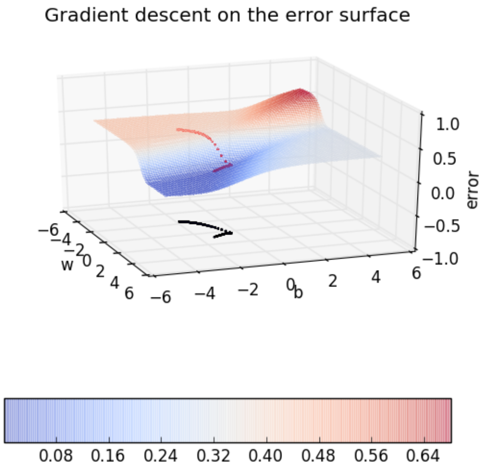
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



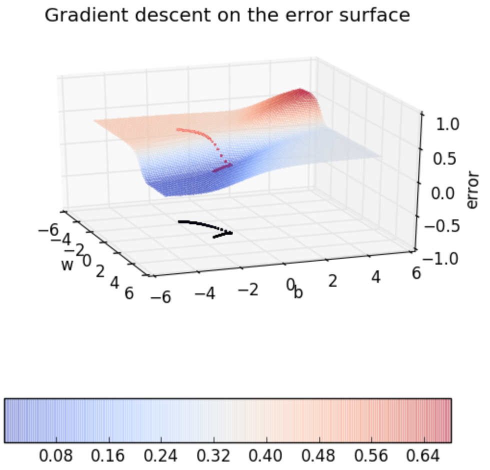
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

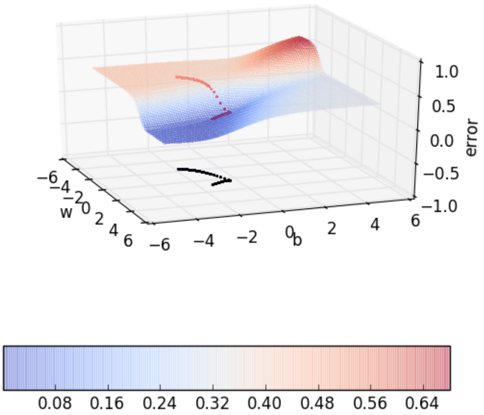


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

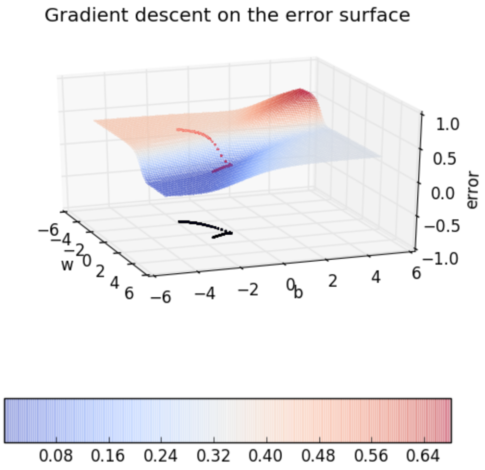


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

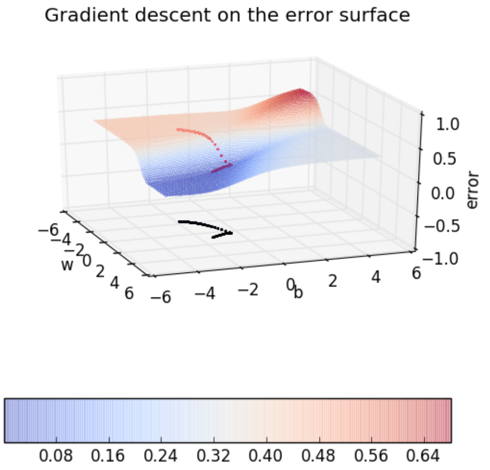
Gradient descent on the error surface



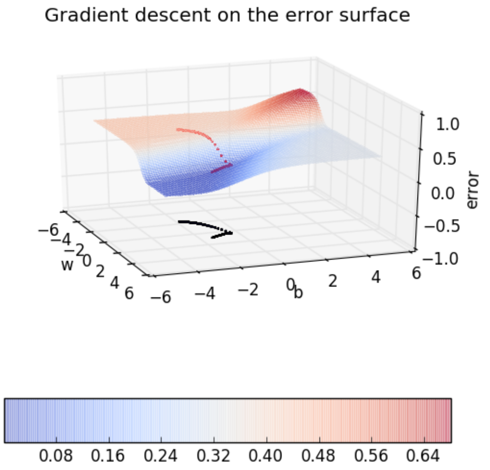
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

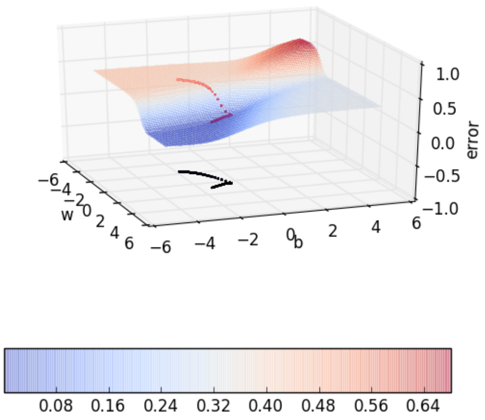


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

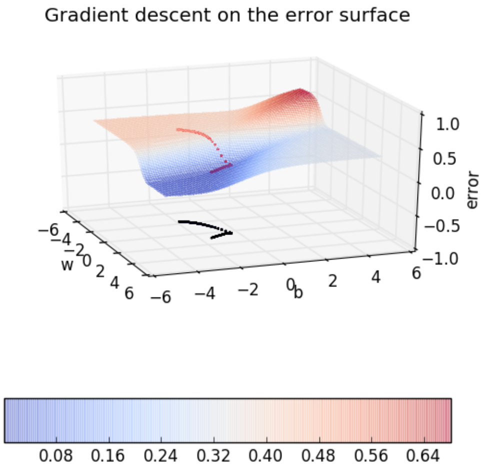


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

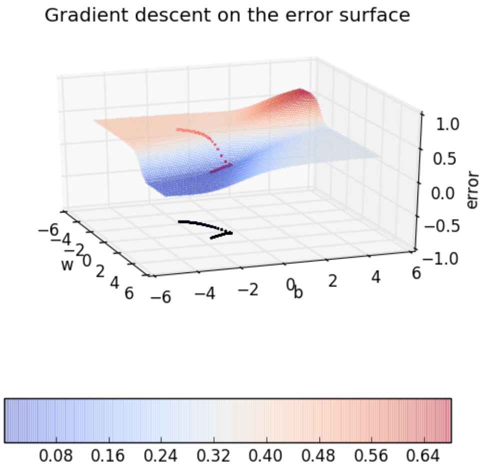
Gradient descent on the error surface



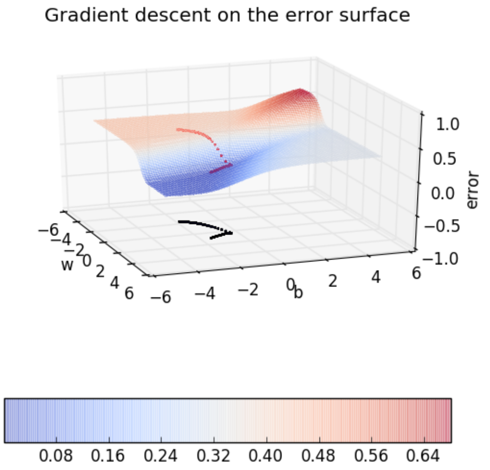
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



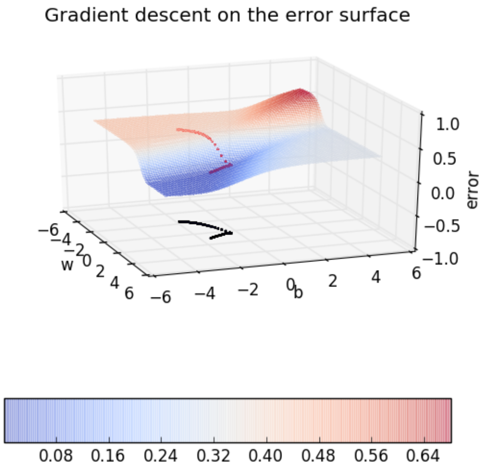
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

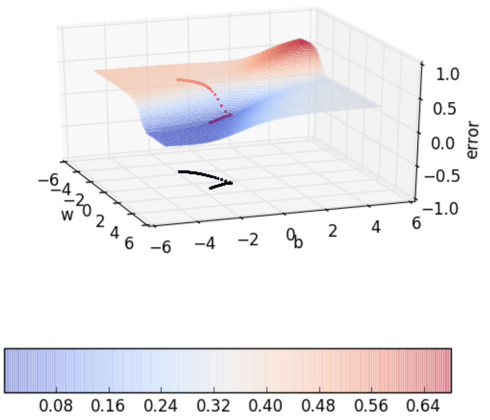


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

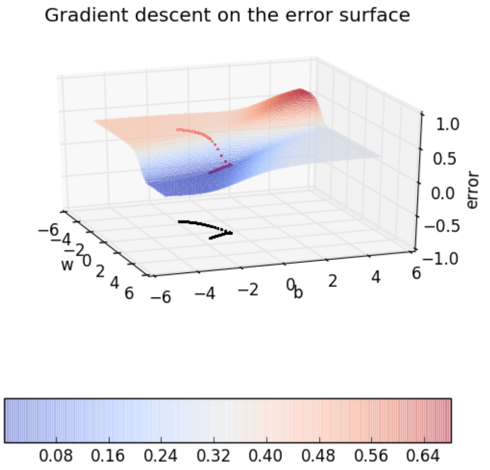


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

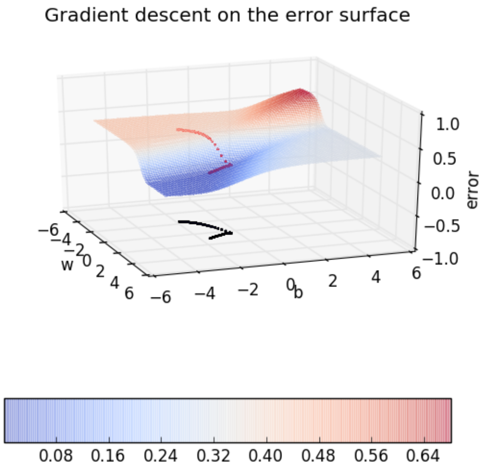
Gradient descent on the error surface



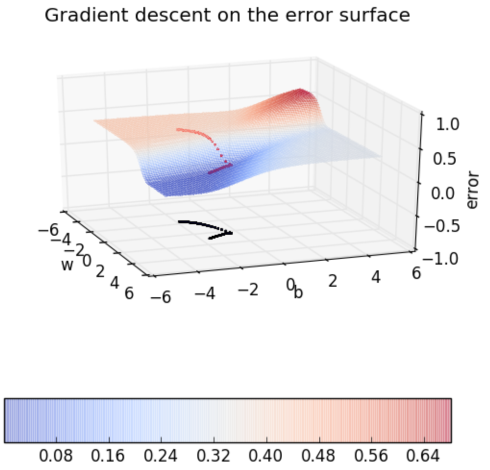
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



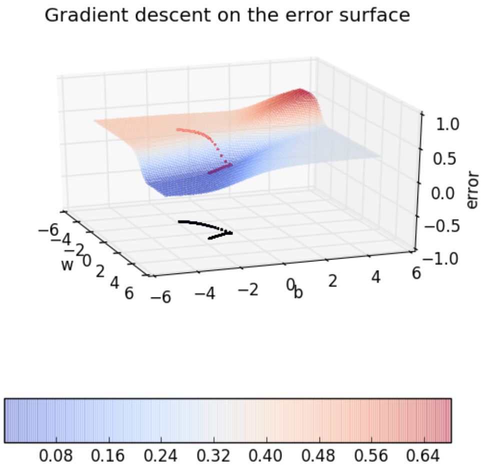
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

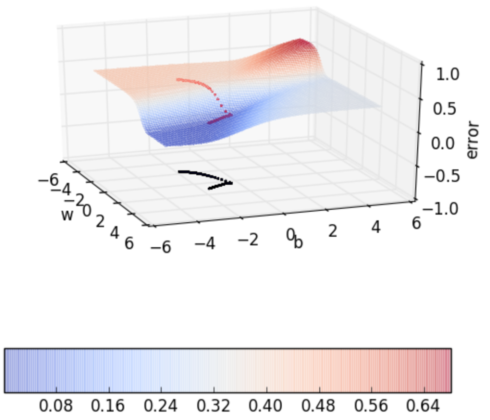


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

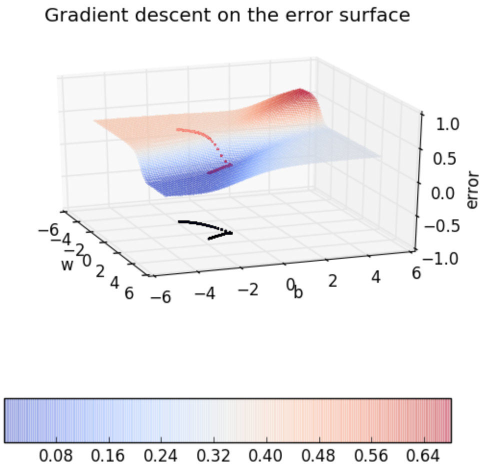


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

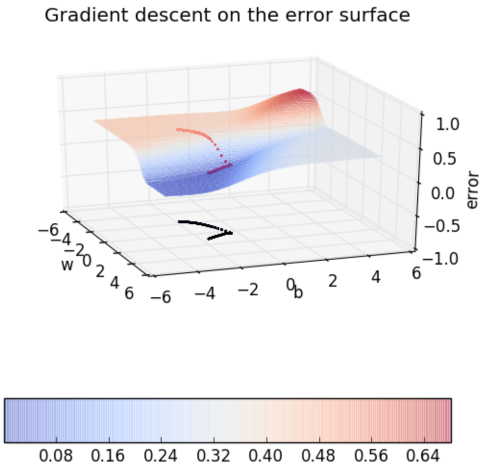
Gradient descent on the error surface



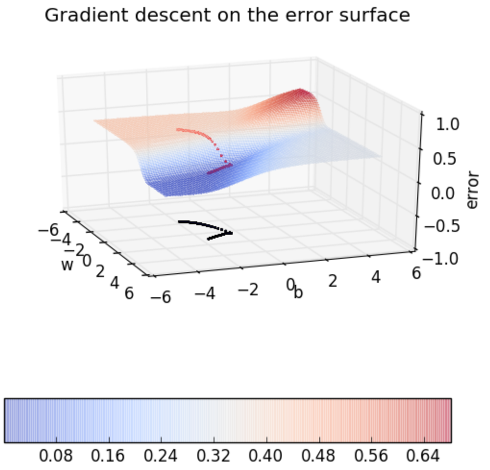
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



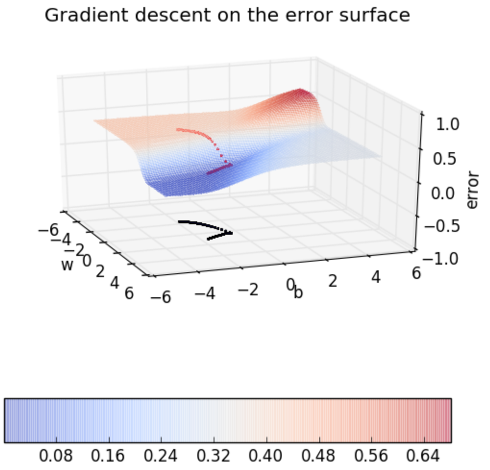
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



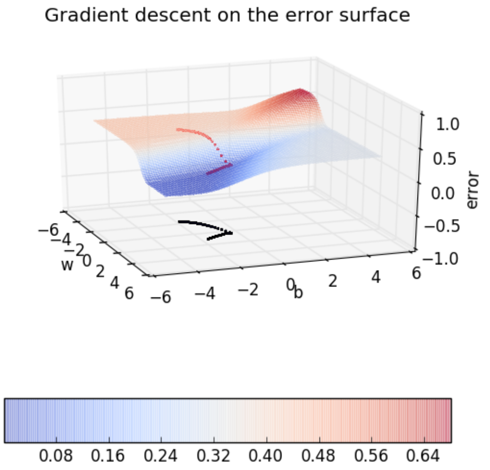
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



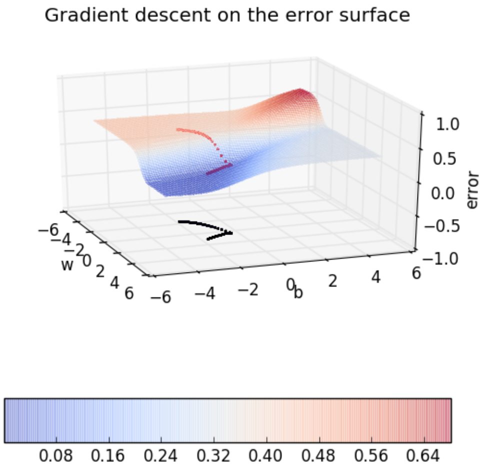
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

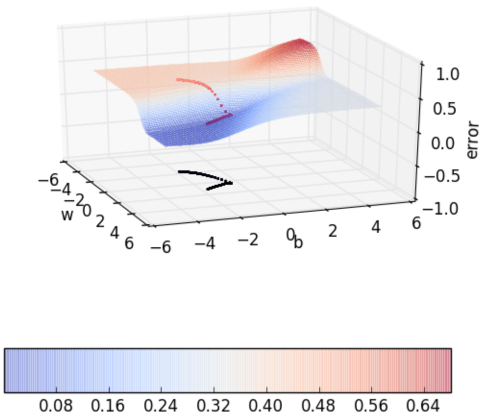


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

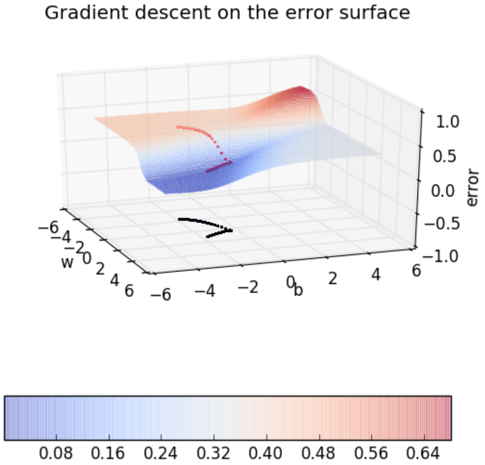


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface

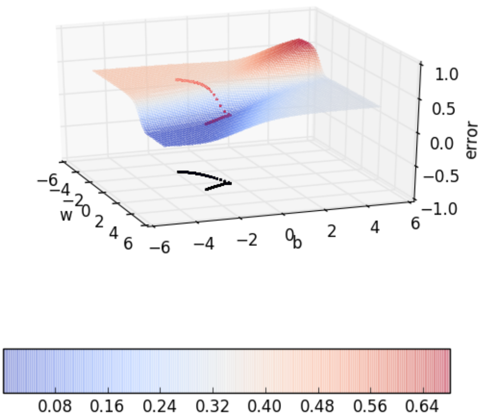


```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

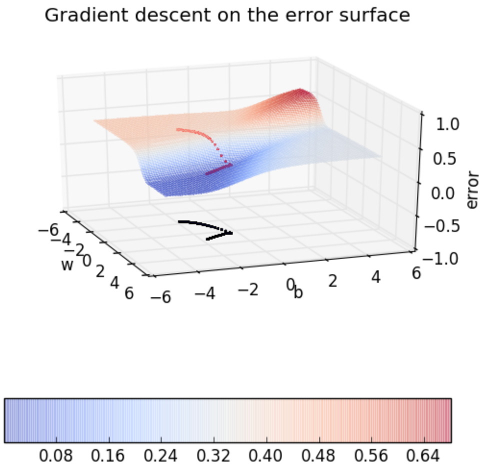


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

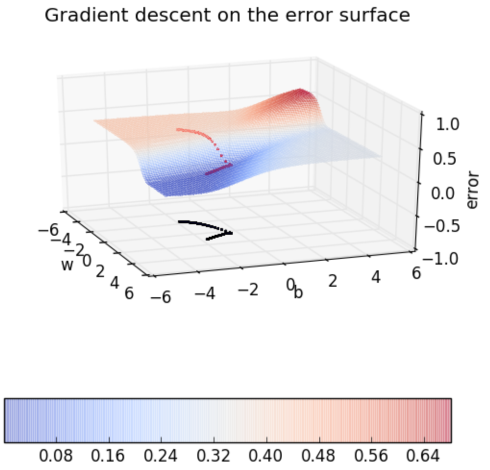
Gradient descent on the error surface



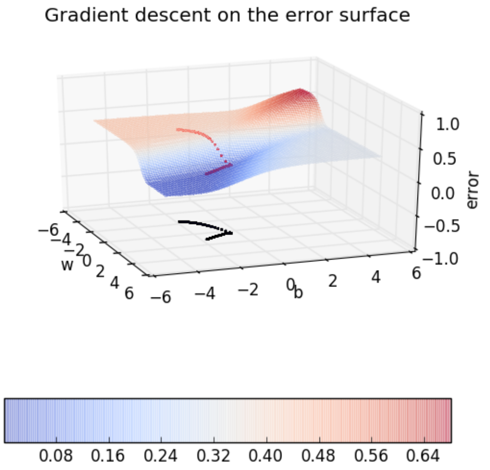
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



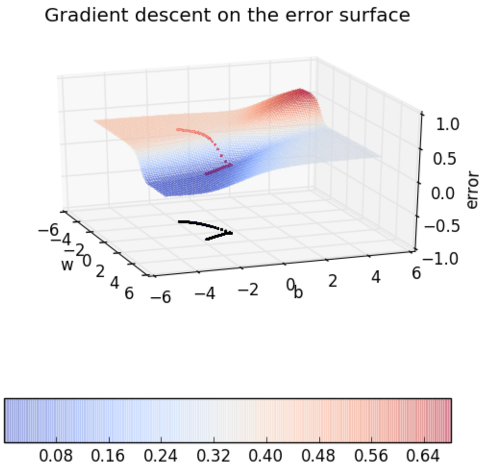
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



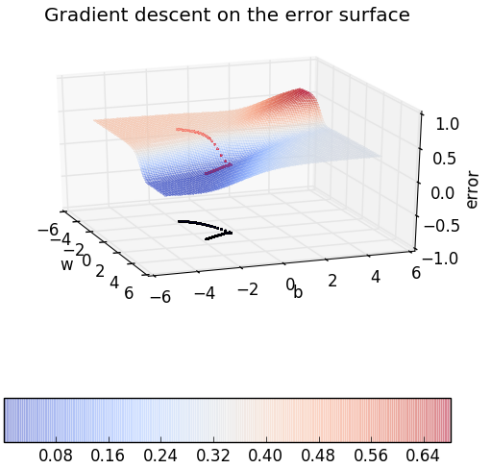
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



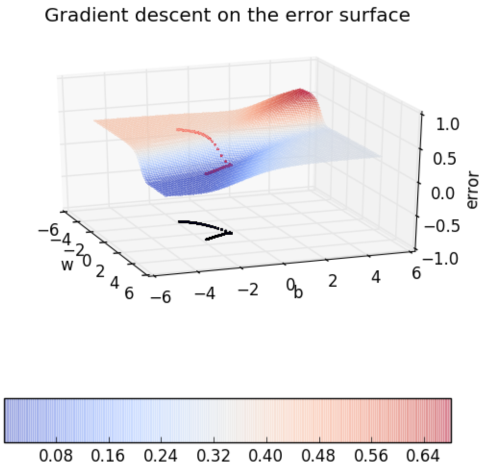
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



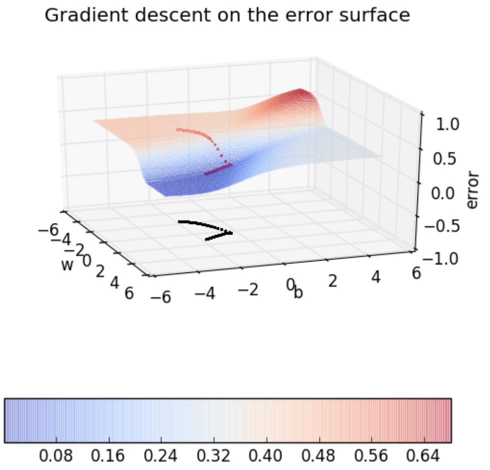
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

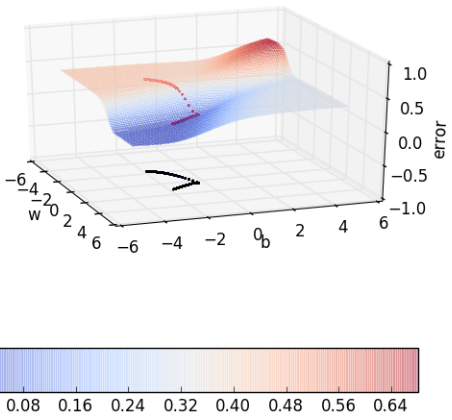


```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

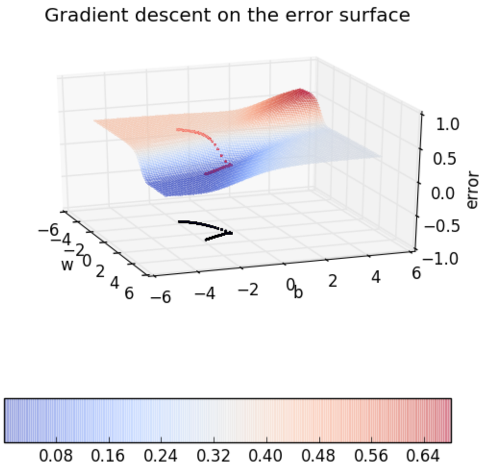


```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

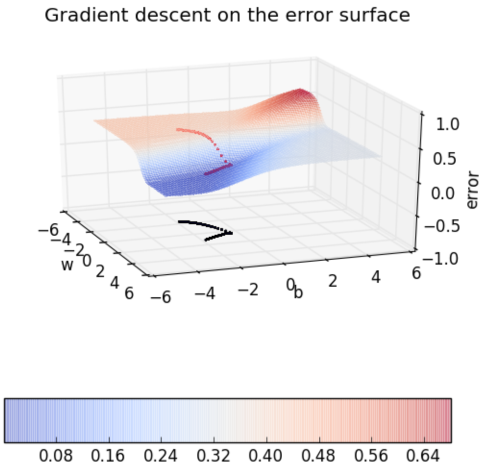
Gradient descent on the error surface



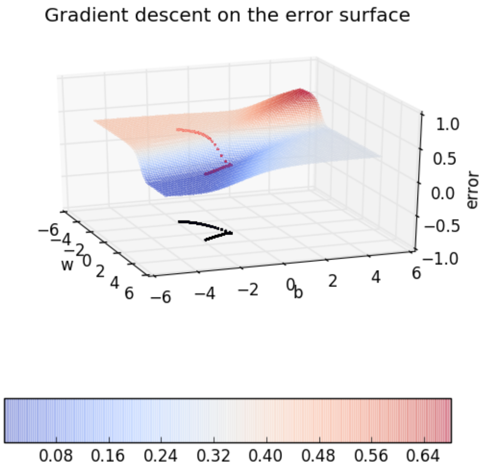
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



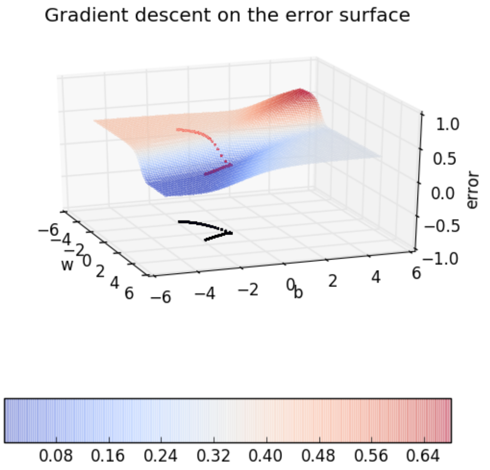
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

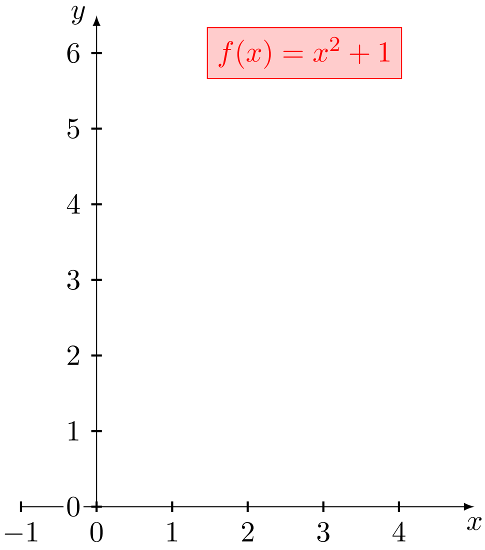


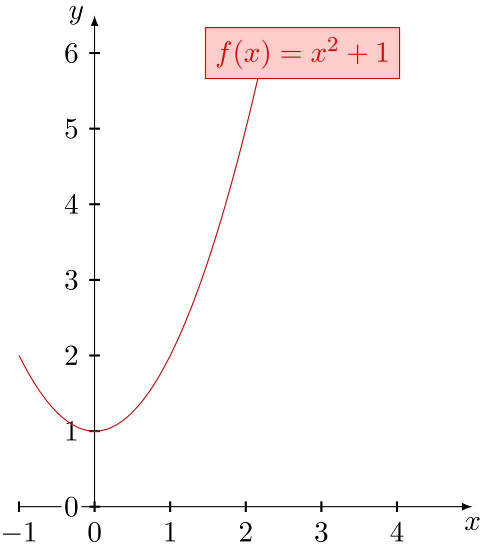
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



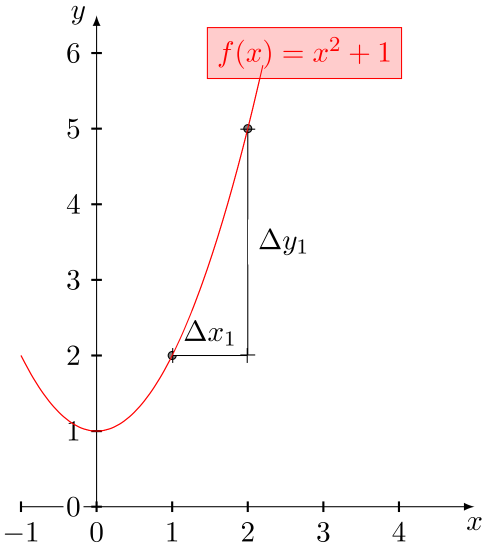
```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



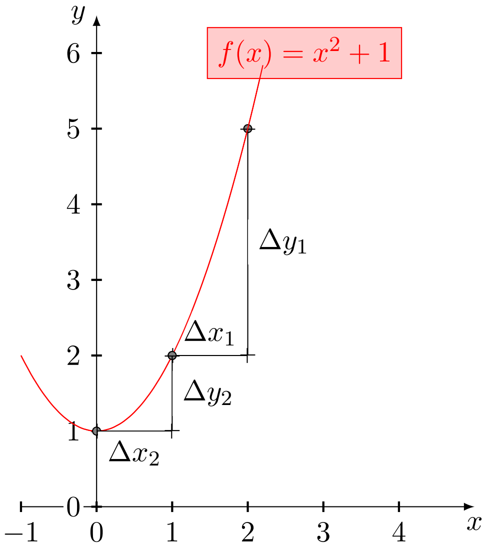

$f(x) = x^2 + 1$
y
6
5
4
3
2
1
0
−1
0
1
2
3
4
x



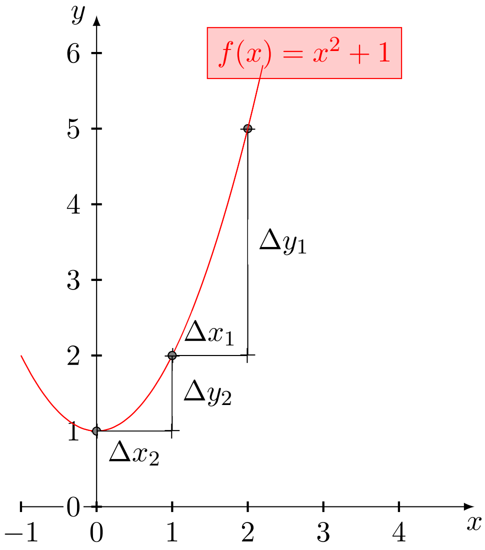
- When the curve is steep the gradient $(\frac{\Delta y_1}{\Delta x_1})$ is large



- When the curve is steep the gradient $(\frac{\Delta y_1}{\Delta x_1})$ is large
- When the curve is gentle the gradient $(\frac{\Delta y_2}{\Delta x_2})$ is small

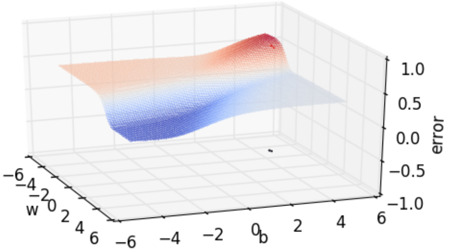


- When the curve is steep the gradient $(\frac{\Delta y_1}{\Delta x_1})$ is large
- When the curve is gentle the gradient $(\frac{\Delta y_2}{\Delta x_2})$ is small
- Recall that our weight updates are proportional to the gradient $w = w - \eta \nabla w$

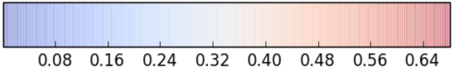


- When the curve is steep the gradient $(\frac{\Delta y_1}{\Delta x_1})$ is large
- When the curve is gentle the gradient $(\frac{\Delta y_2}{\Delta x_2})$ is small
- Recall that our weight updates are proportional to the gradient $w = w - \eta \nabla w$
- Hence in the areas where the curve is gentle the updates are small whereas in the areas where the curve is steep the updates are large

- *Let's see what happens when we start from a different point*

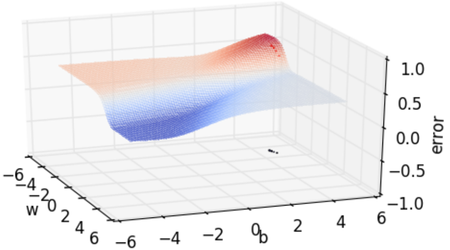


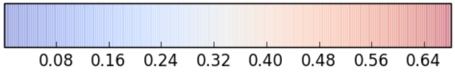
1.0
0.5
0.0
−0.5
−1.0
error
−6
−4
−2
0
2
4
6
w
−6
−4
−2
0
2
4
6
b



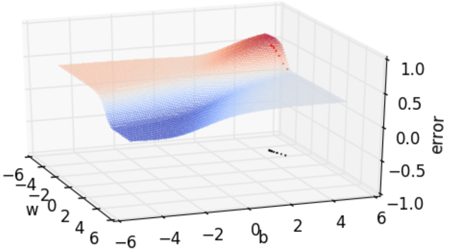
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

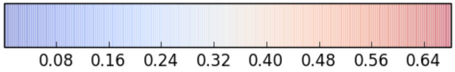
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



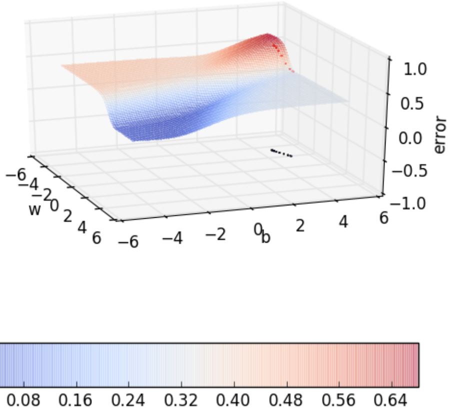


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

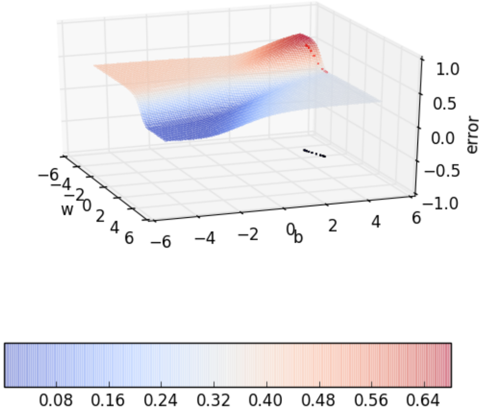




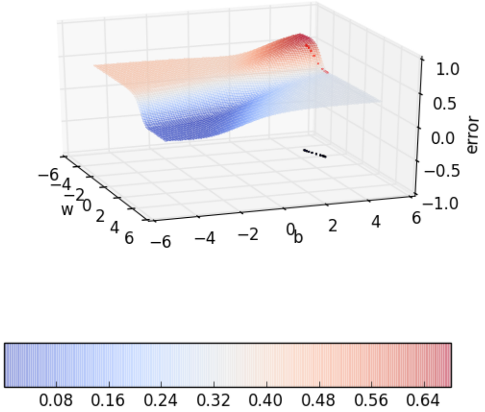
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



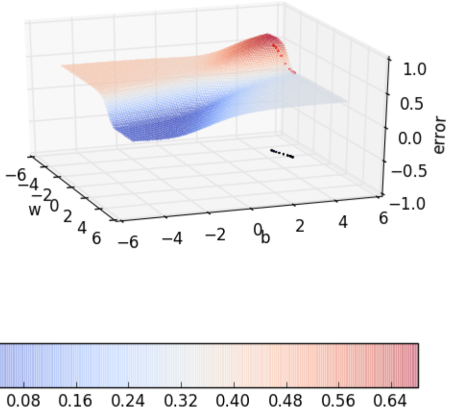
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



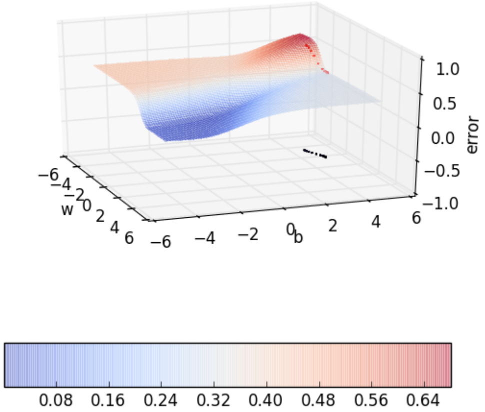
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



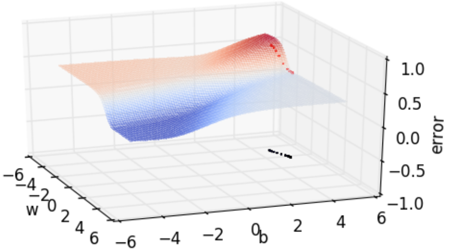
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

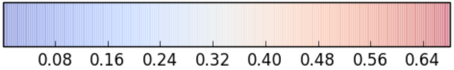


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

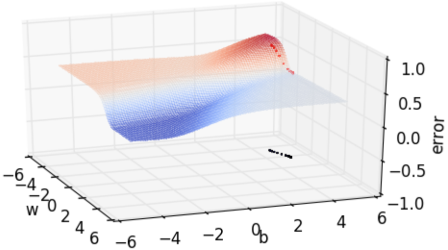


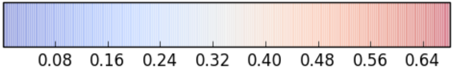
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



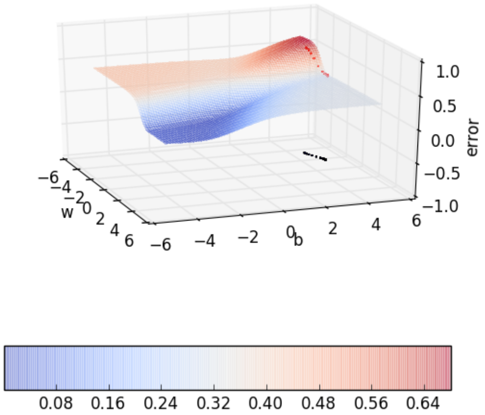


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

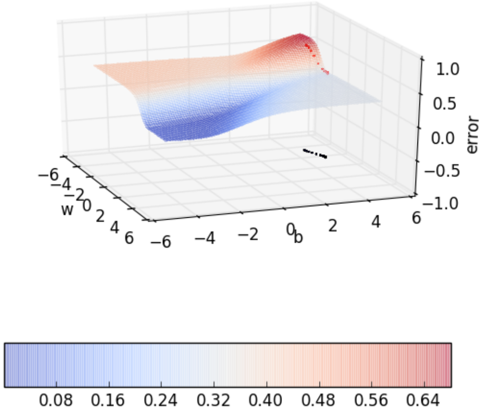




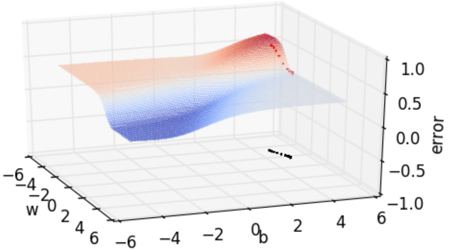
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

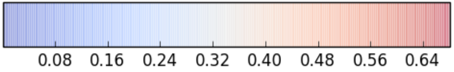


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

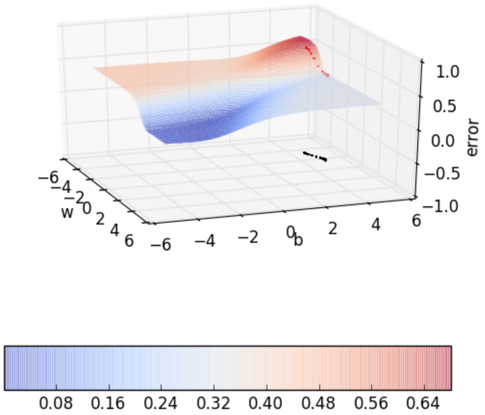


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

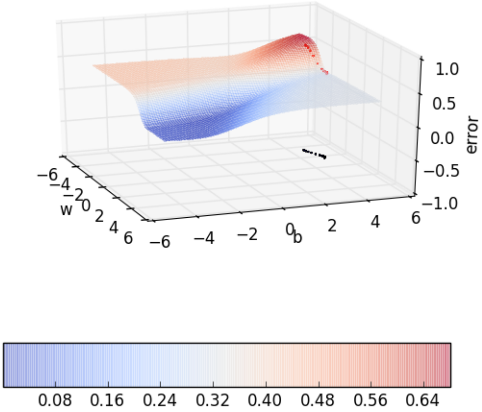




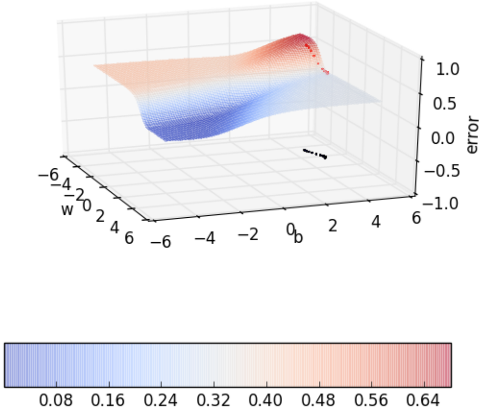
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



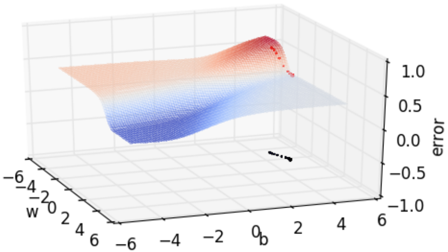
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

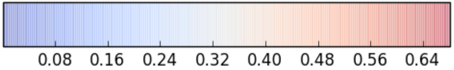


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

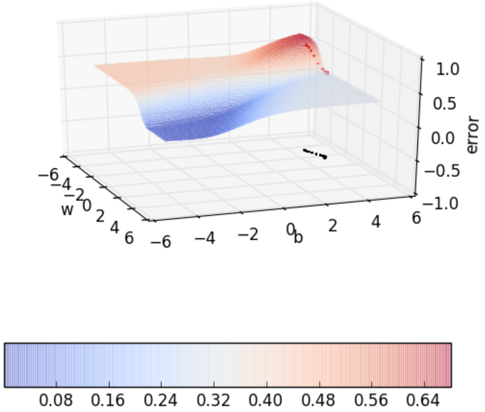


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

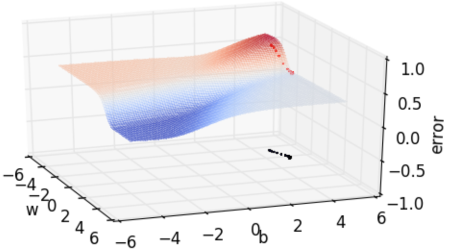


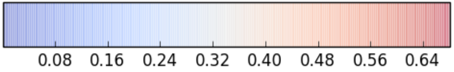


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

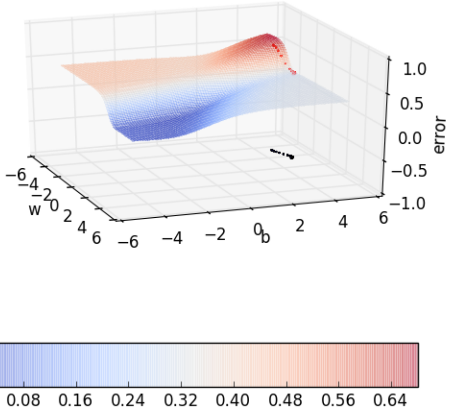


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

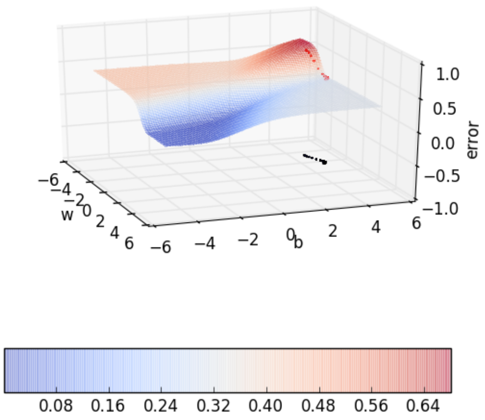




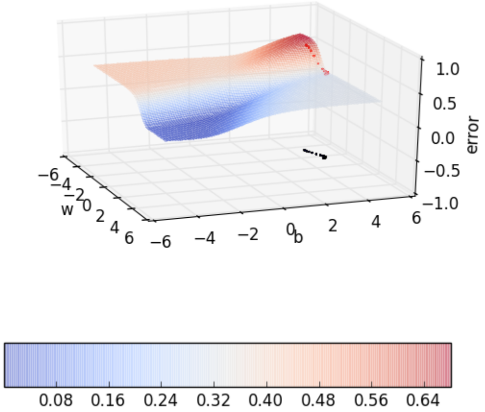
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



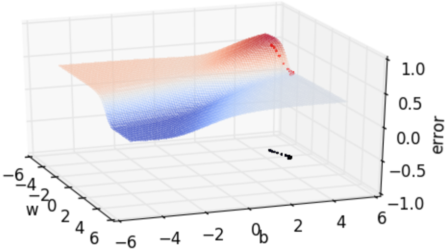
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

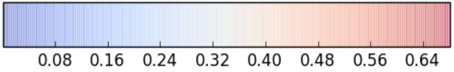


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

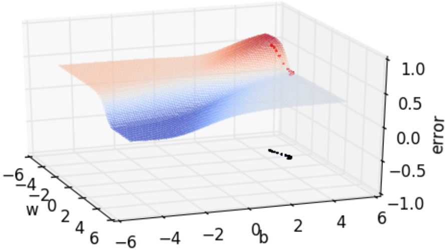


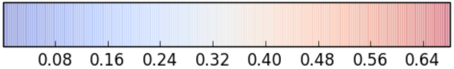
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



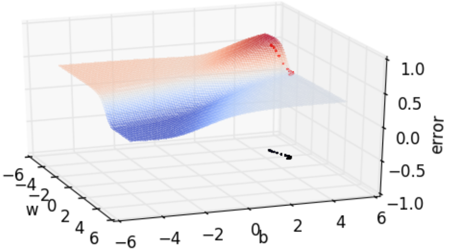


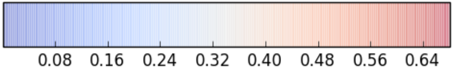
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



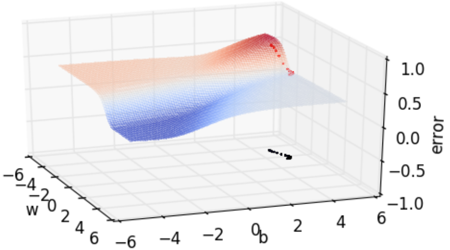


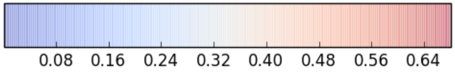
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



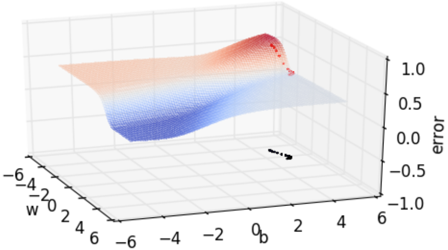


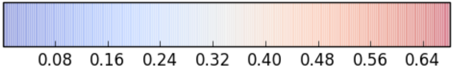
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



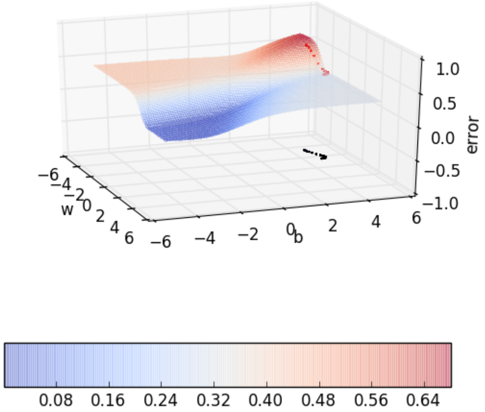


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

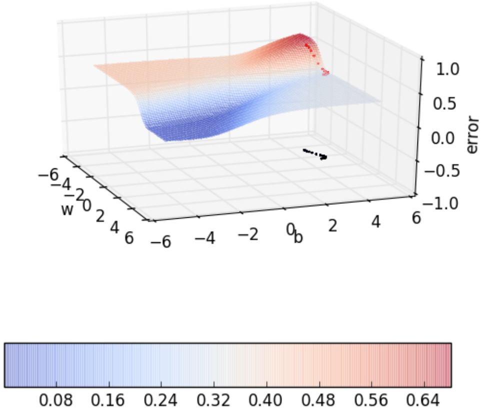




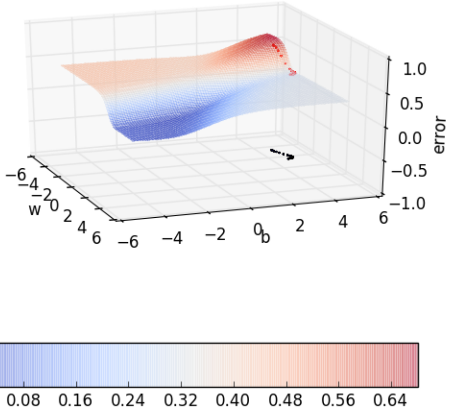
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



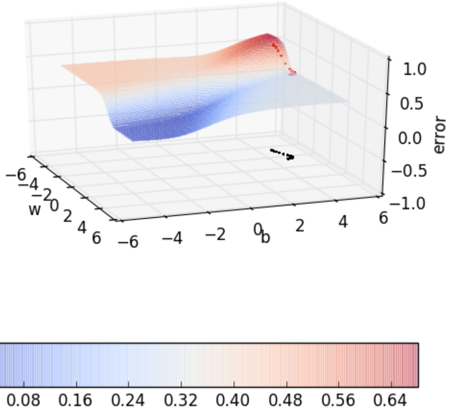
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



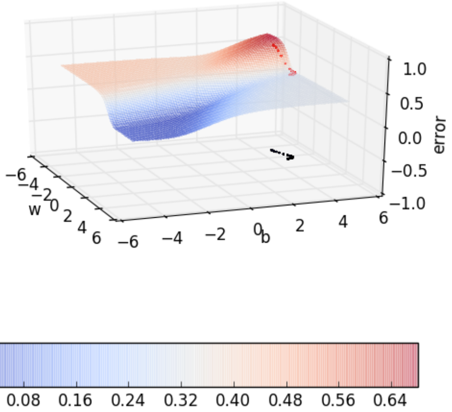
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



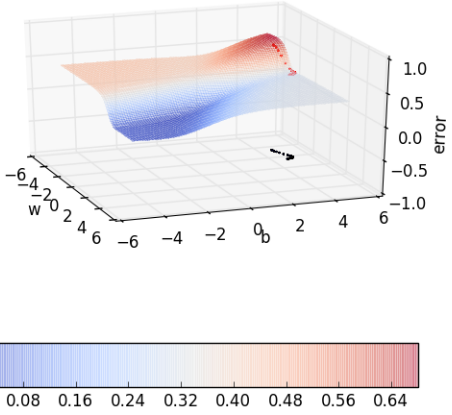
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



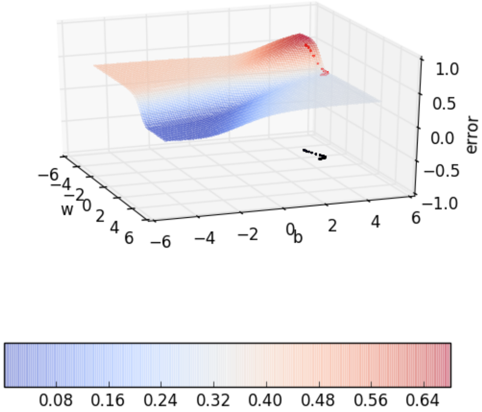
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



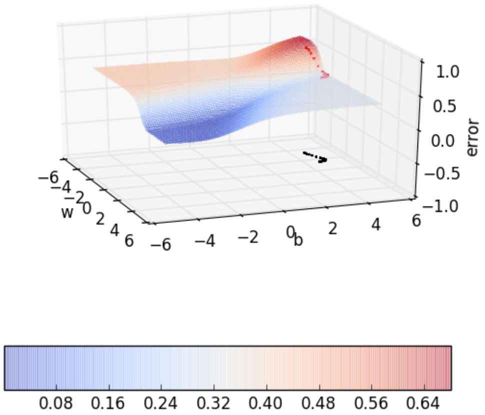
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



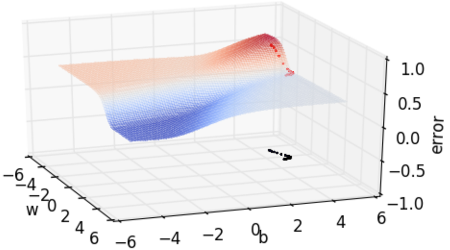
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

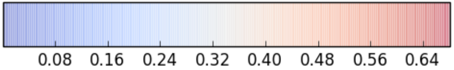


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

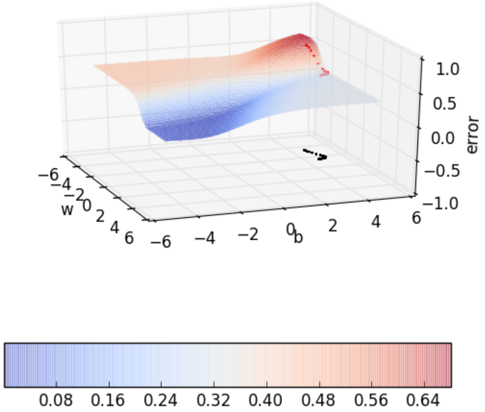


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

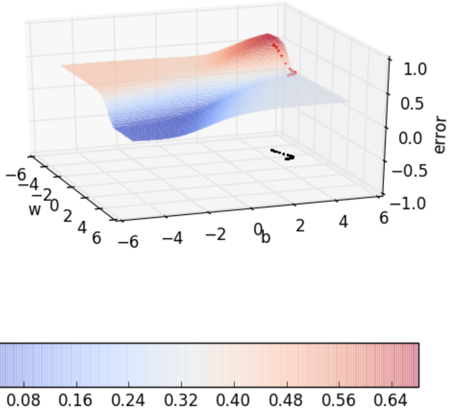




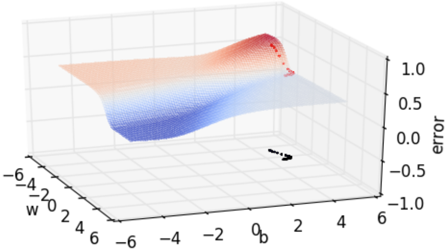
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

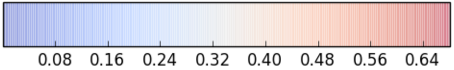


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

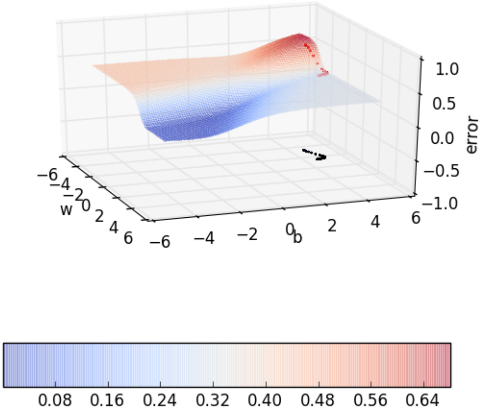


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

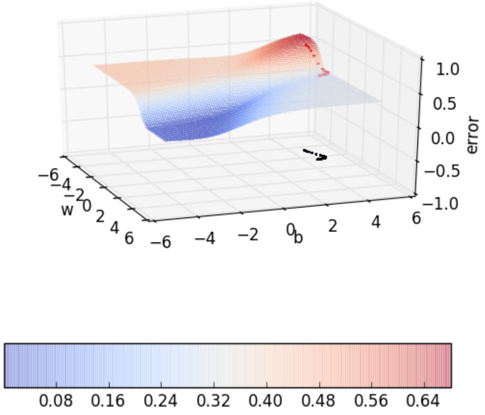




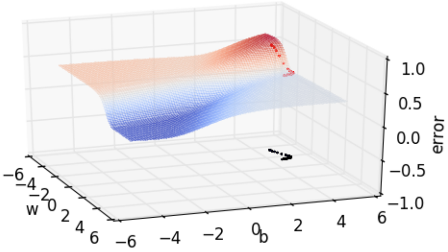
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

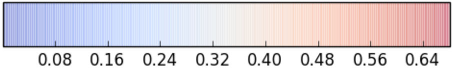


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

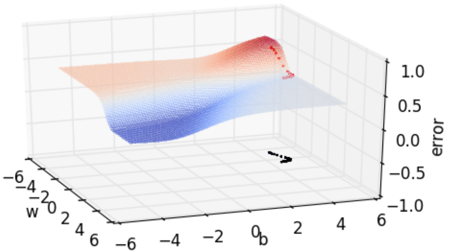


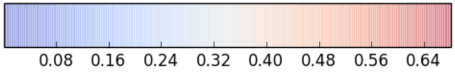
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



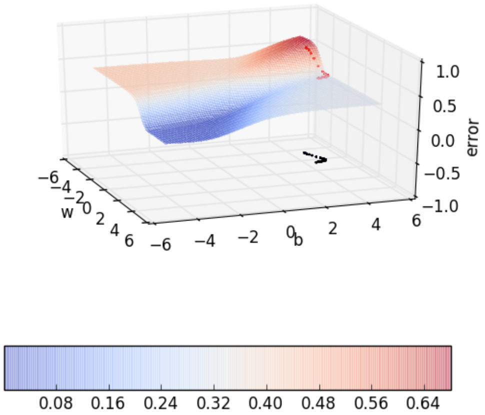


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

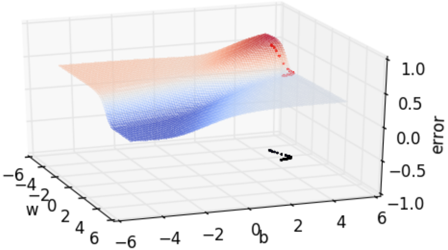


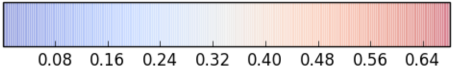


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

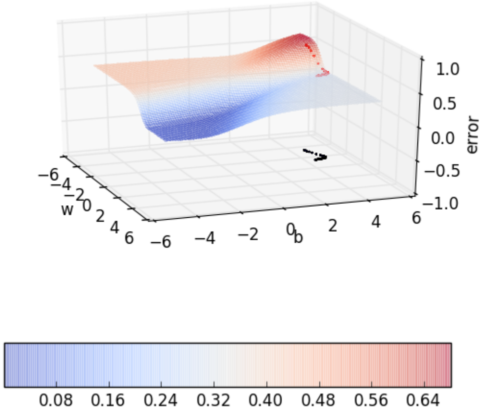


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

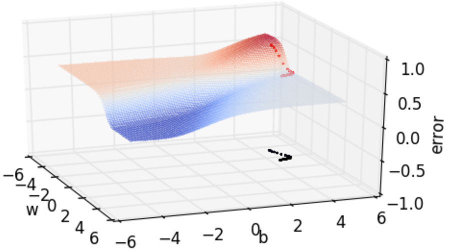


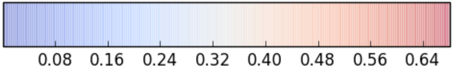


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

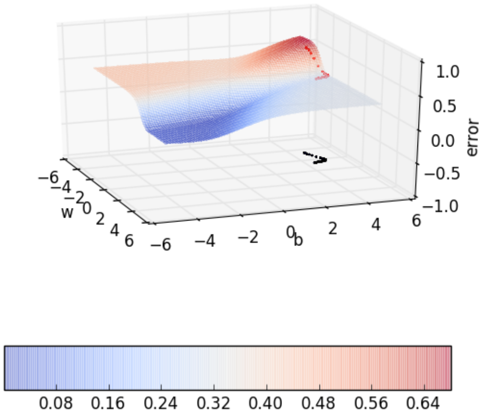


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

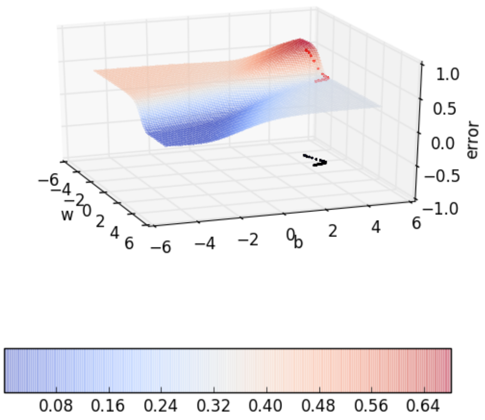




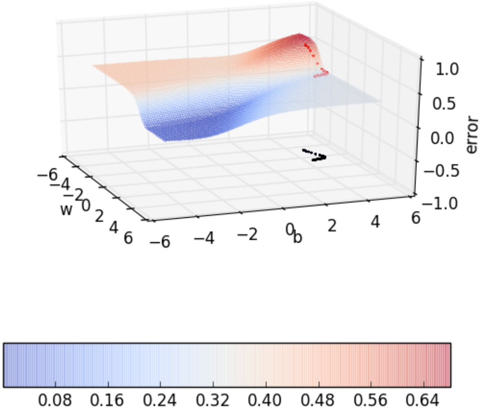
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



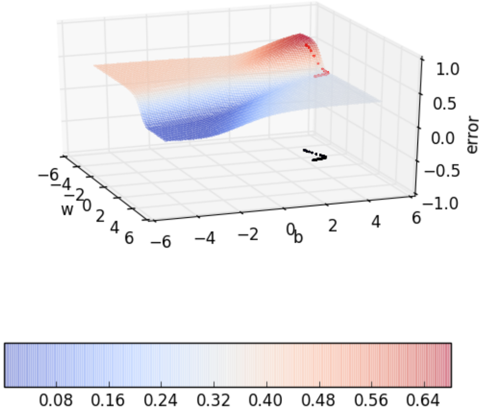
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



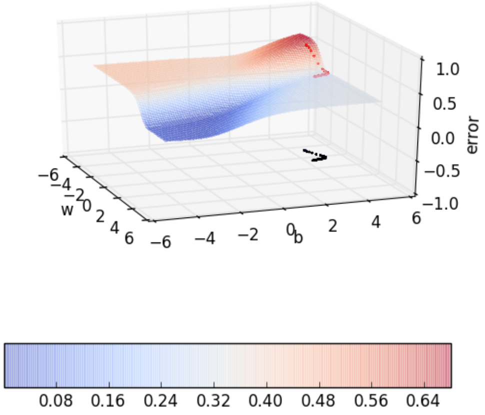
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



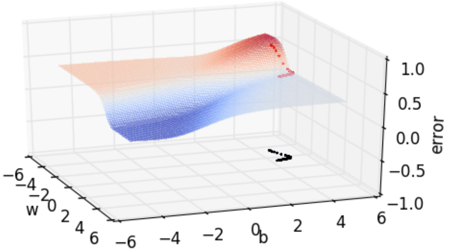
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

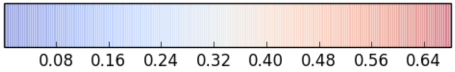


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

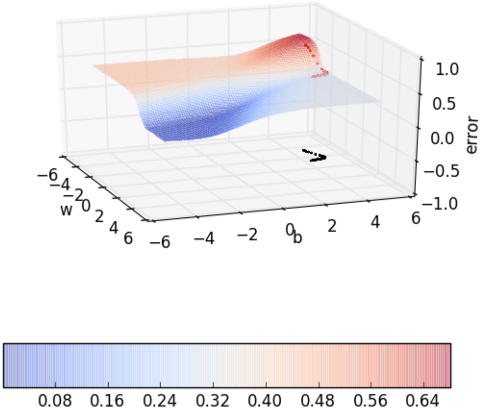


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

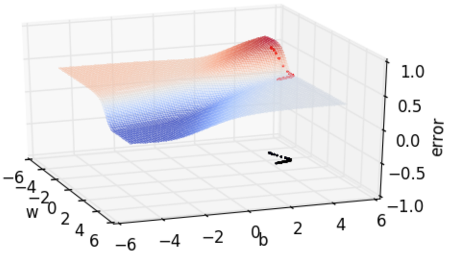


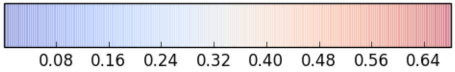


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

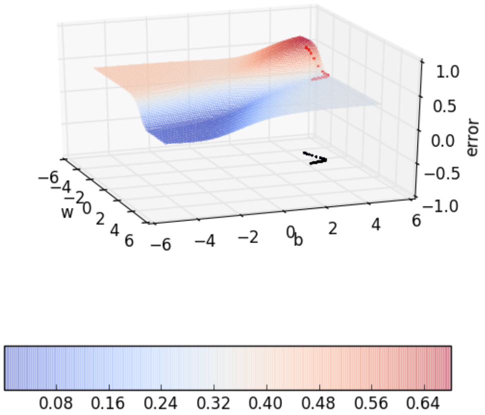


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

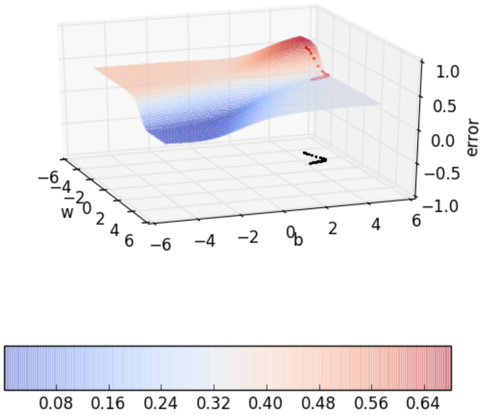




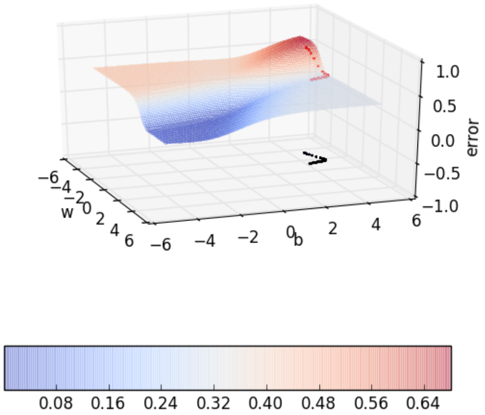
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



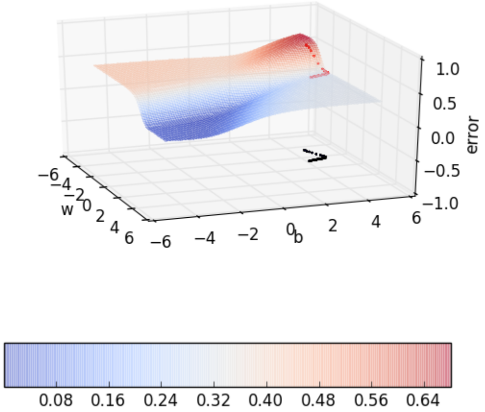
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



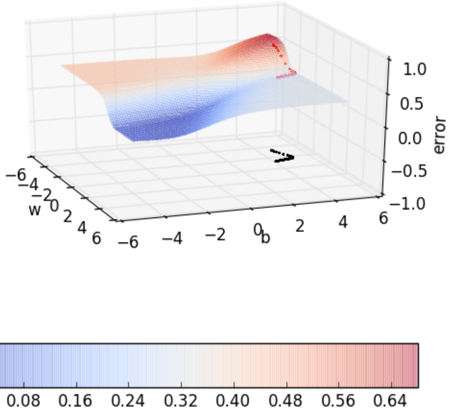
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



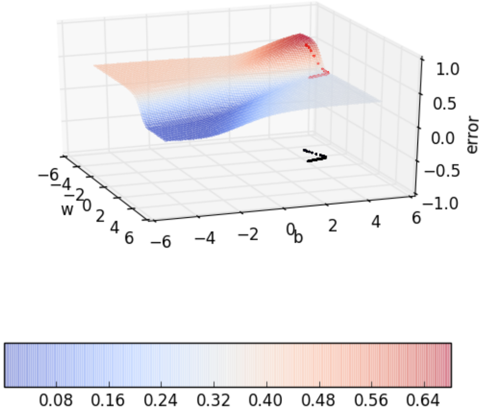
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



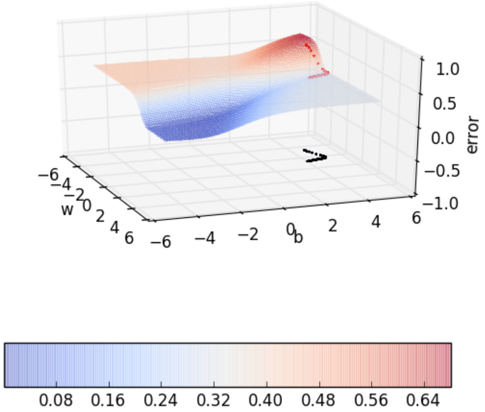
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



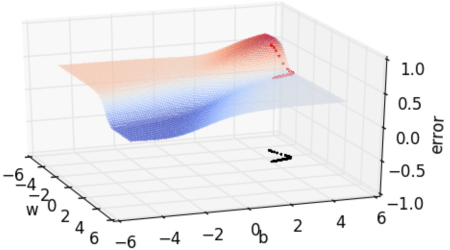
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

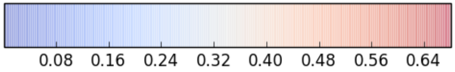


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

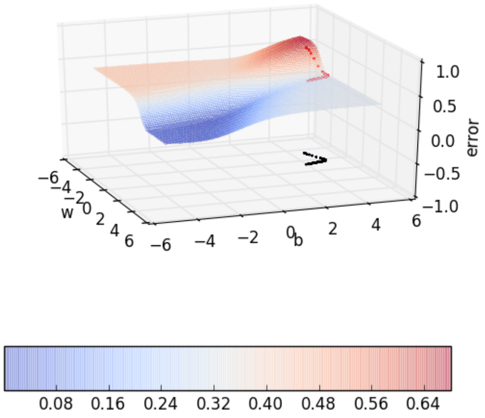


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

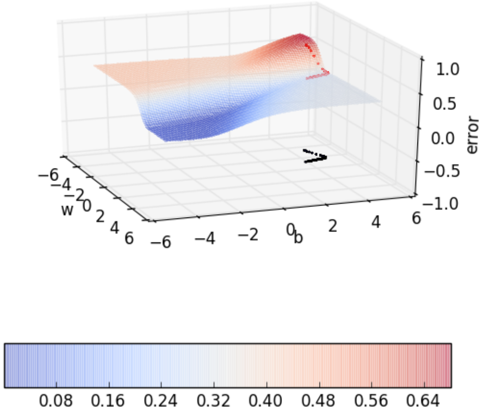




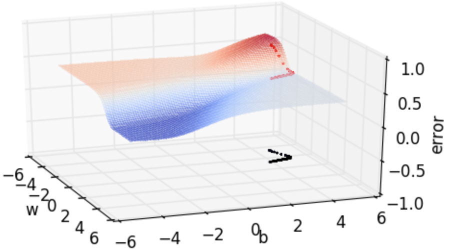
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

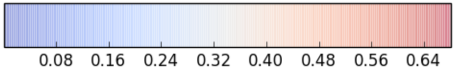


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

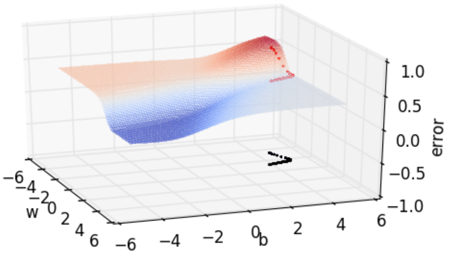


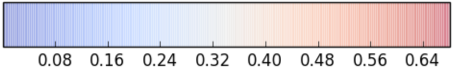
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



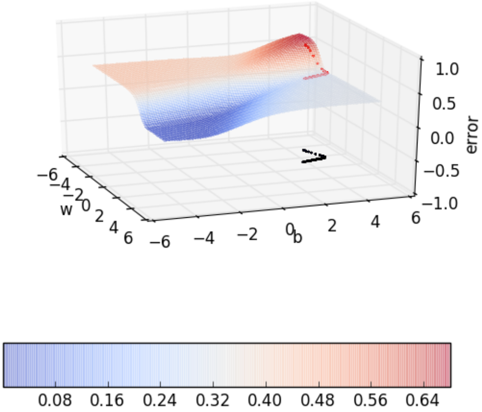


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

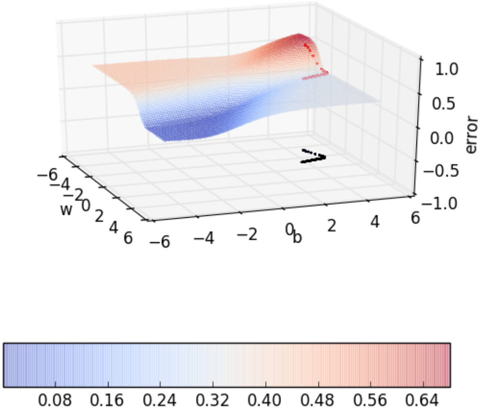




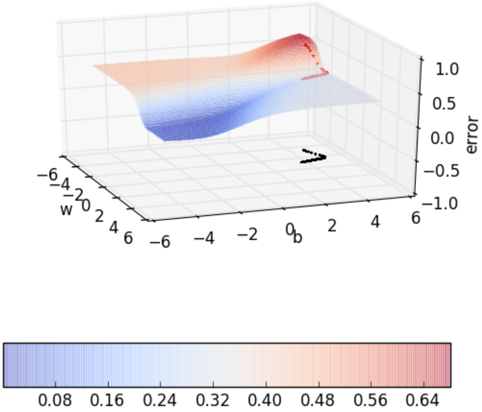
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



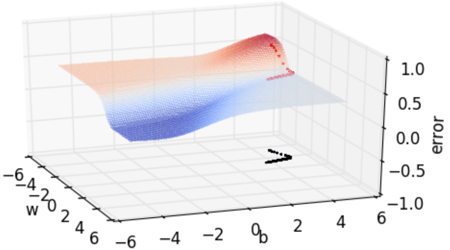
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

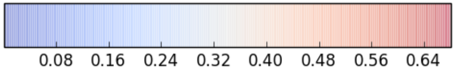


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

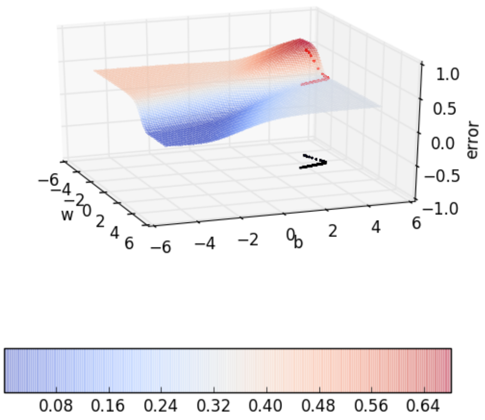


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

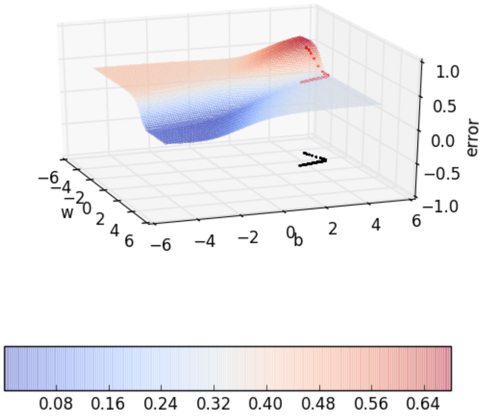




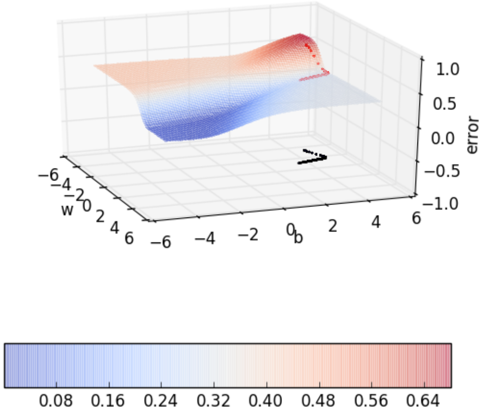
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



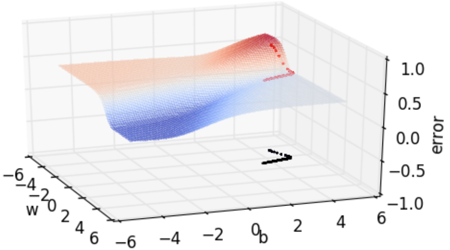
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

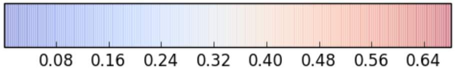


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

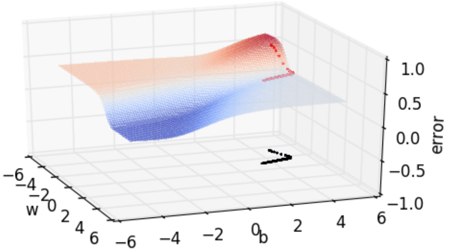


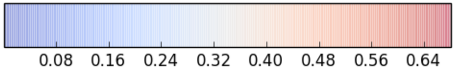
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



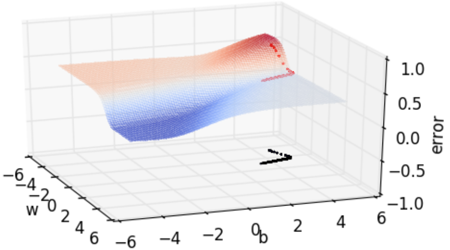


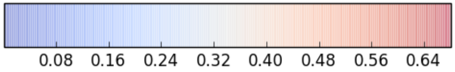
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



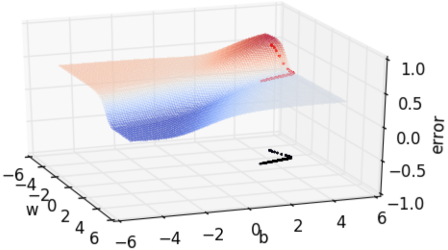


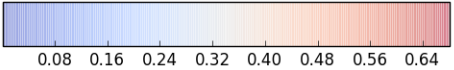
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



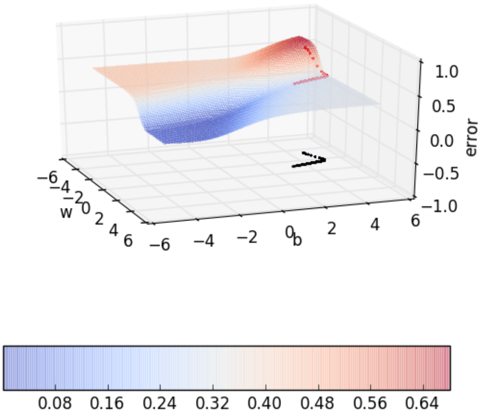


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

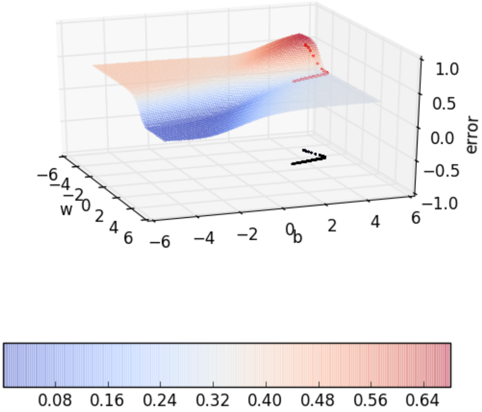




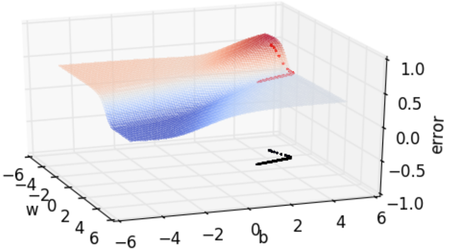
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

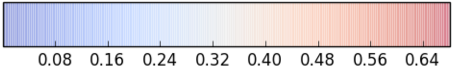


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

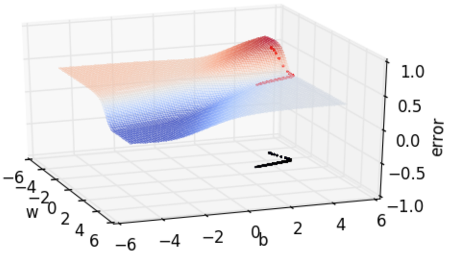


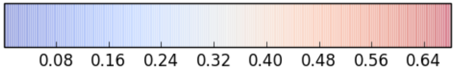
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



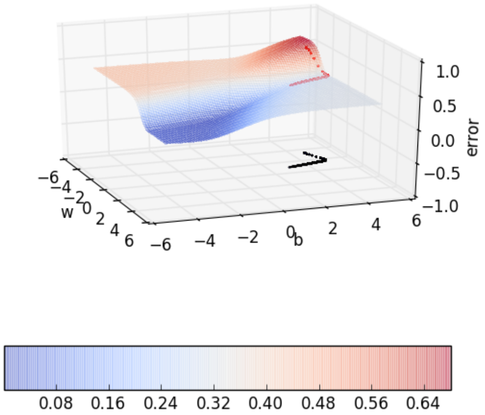


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

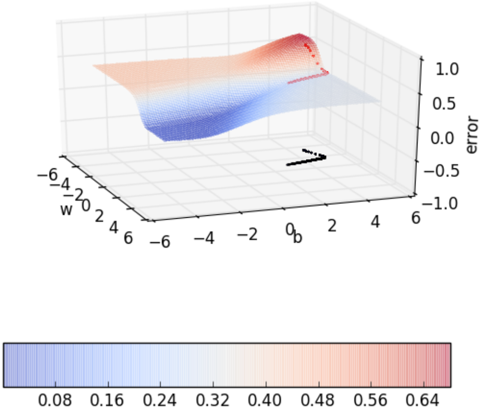




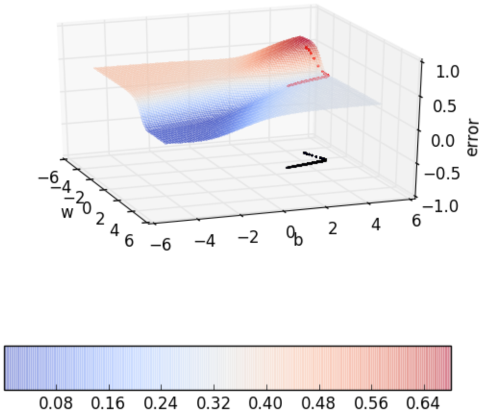
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



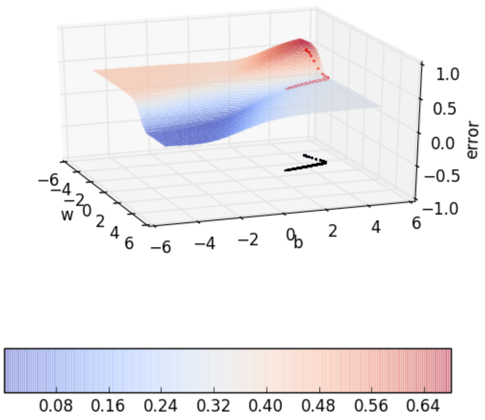
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



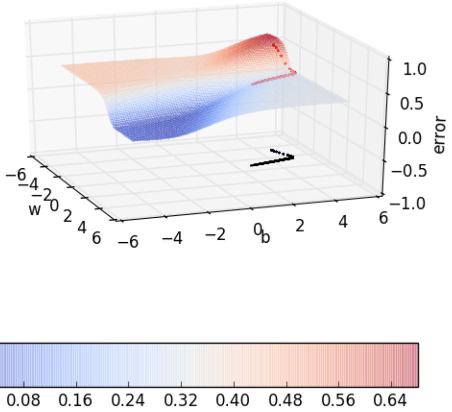
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



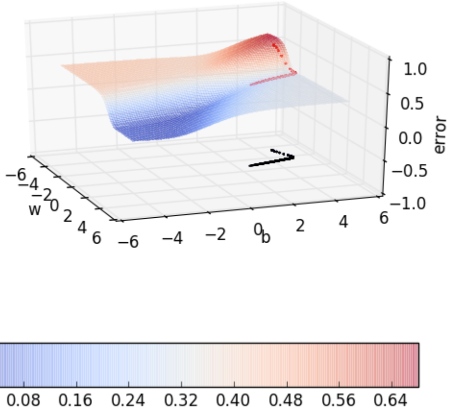
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



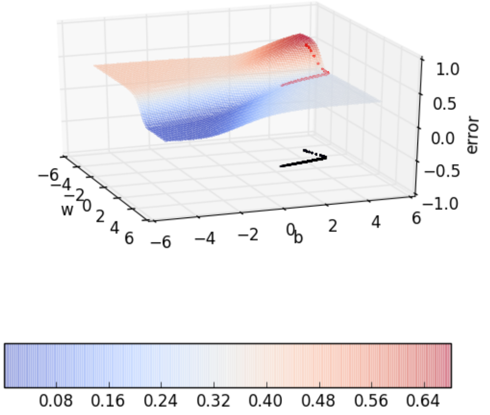
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



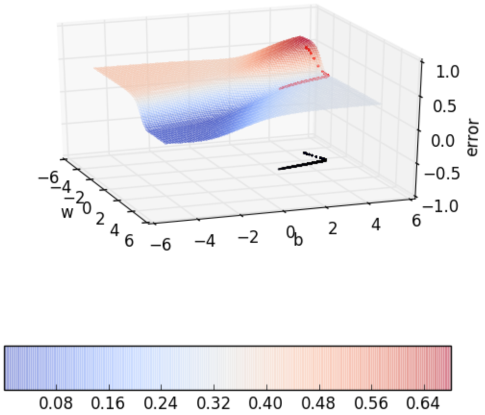
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



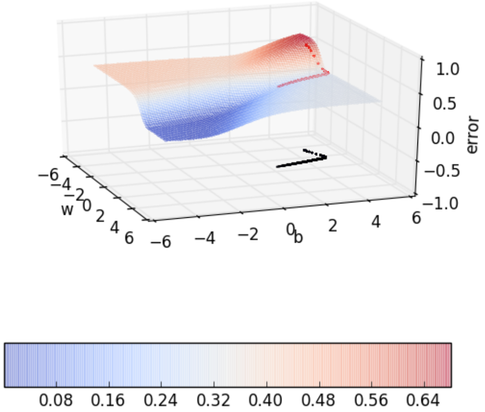
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



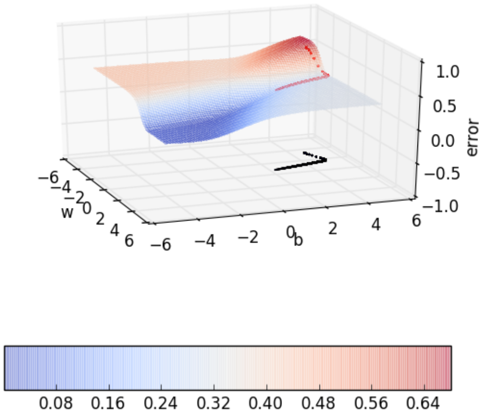
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



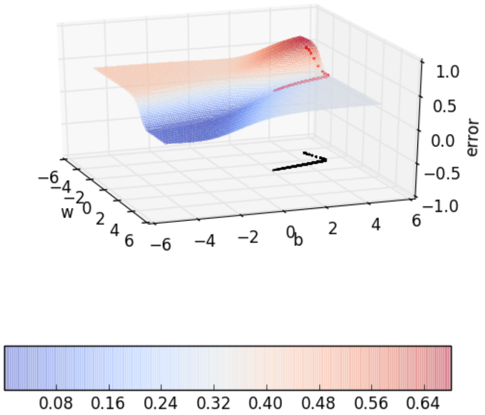
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



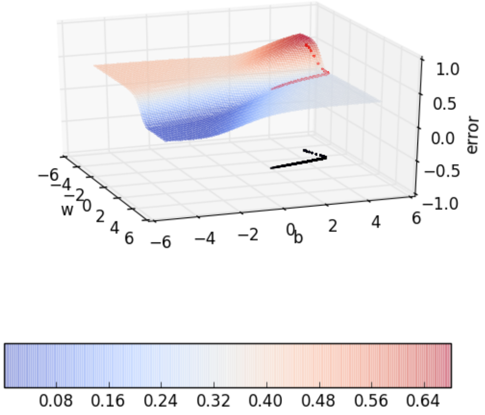
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



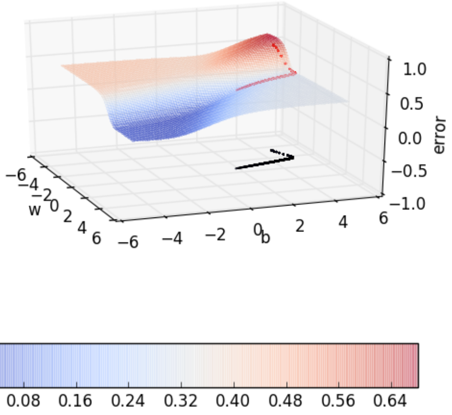
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



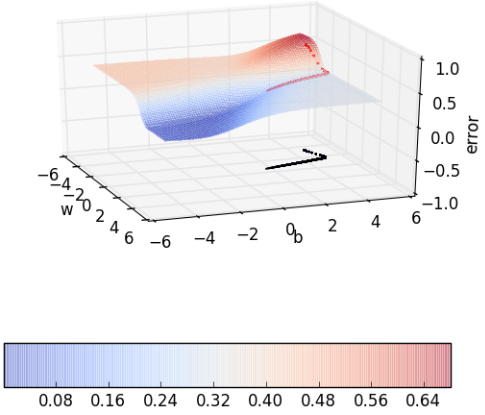
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



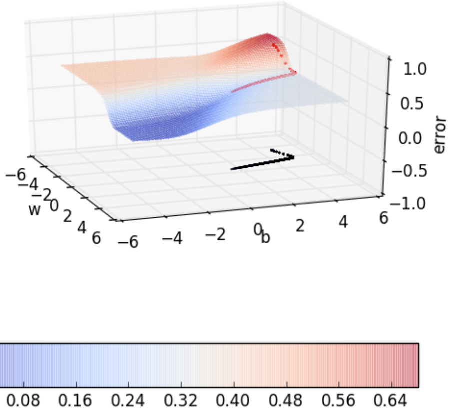
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



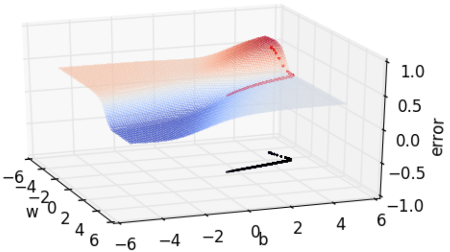
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

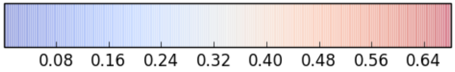


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

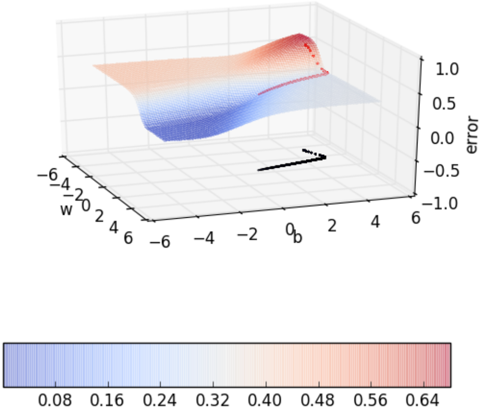


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

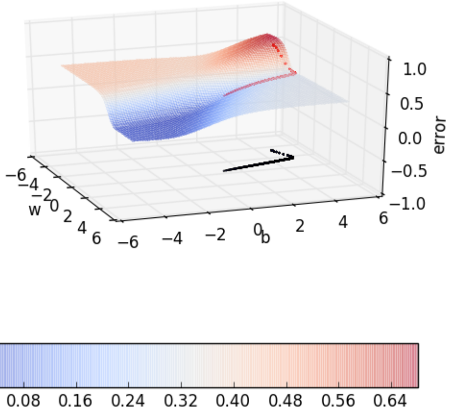




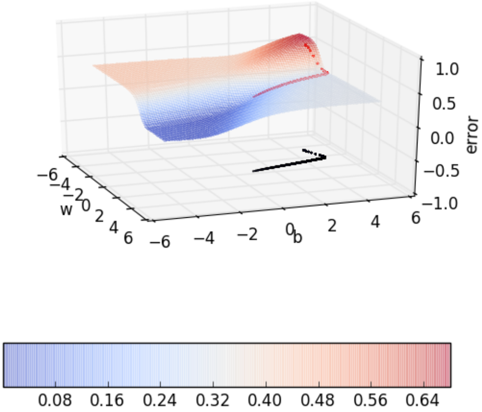
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



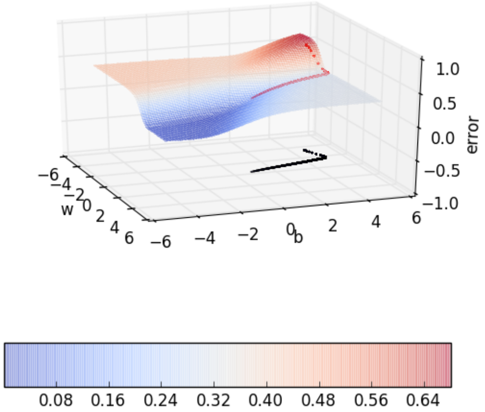
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



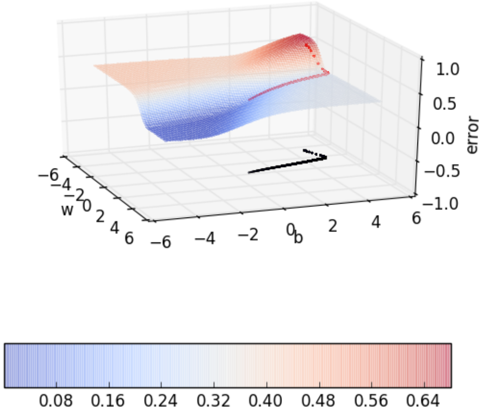
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



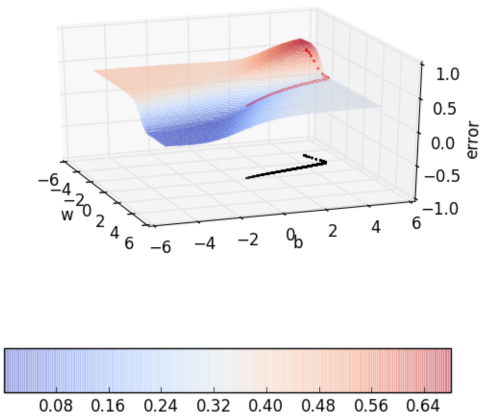
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



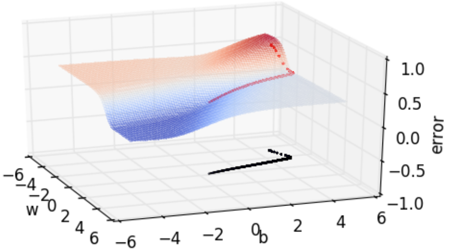
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

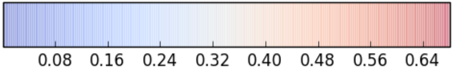


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

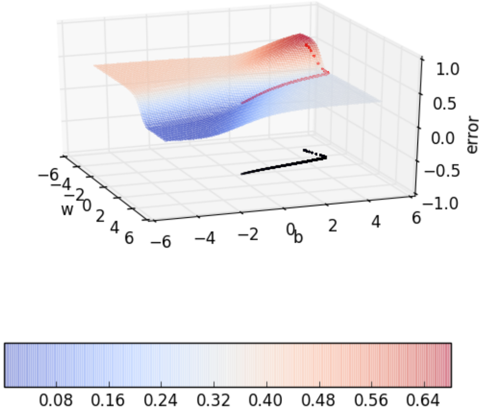


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

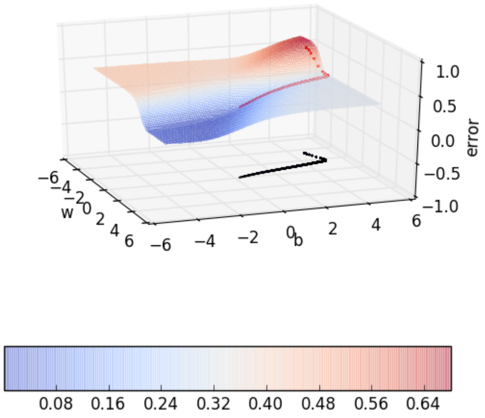




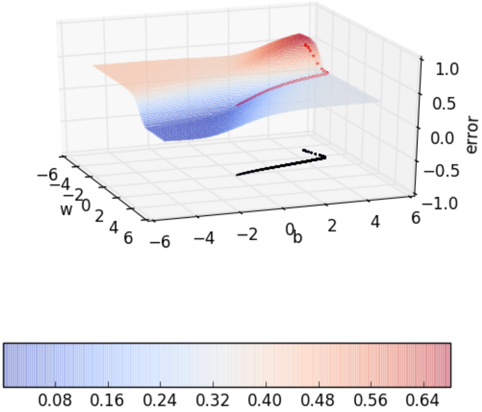
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



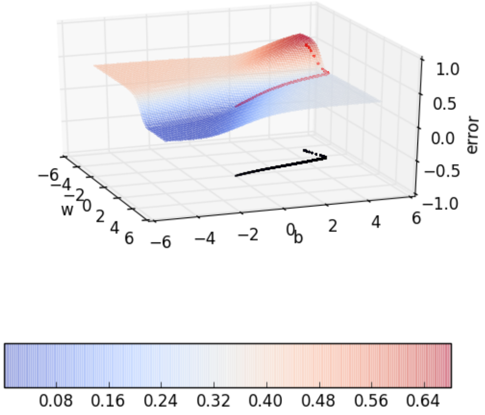
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



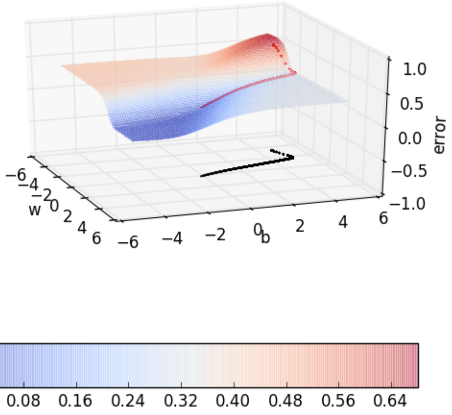
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



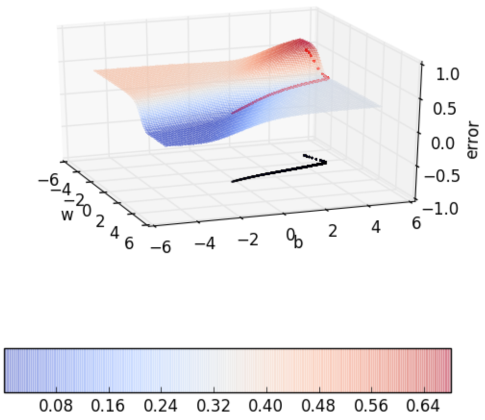
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



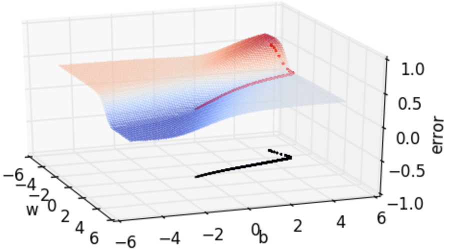
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

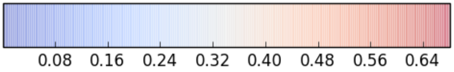


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

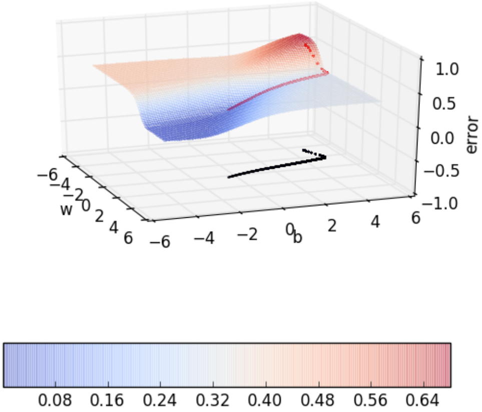


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

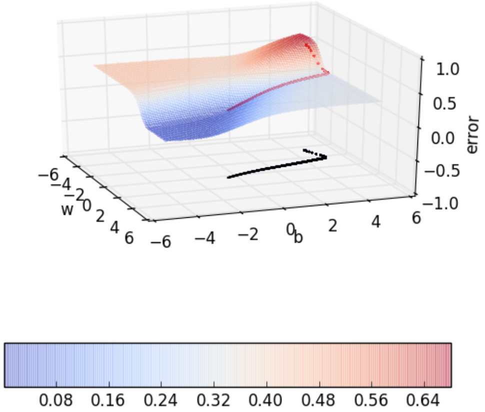




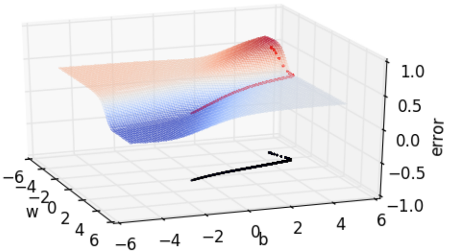
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

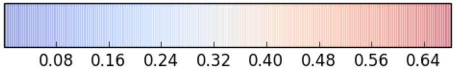


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

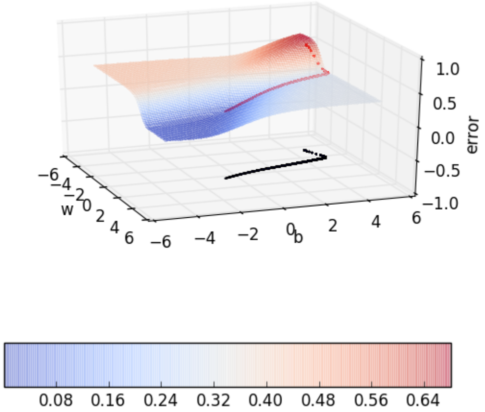


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

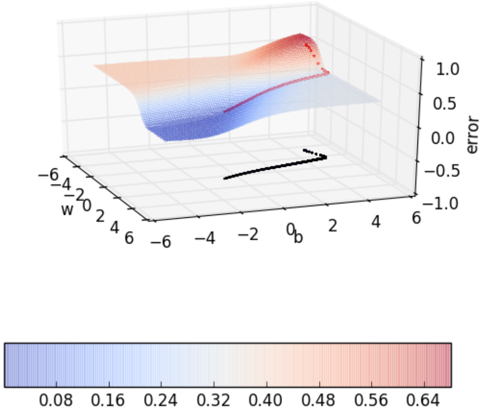




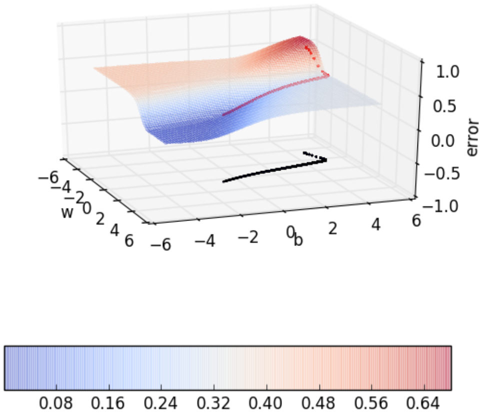
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



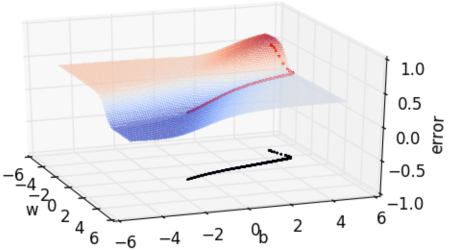
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

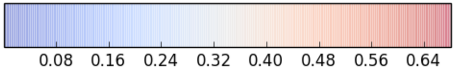


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

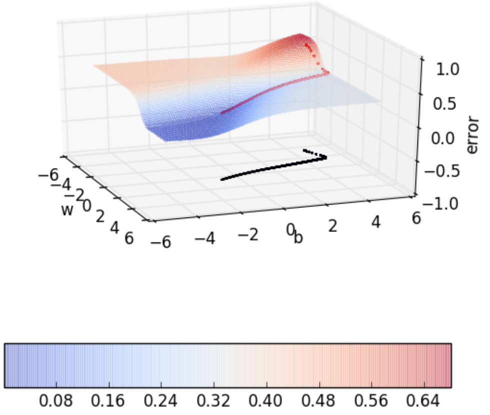


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

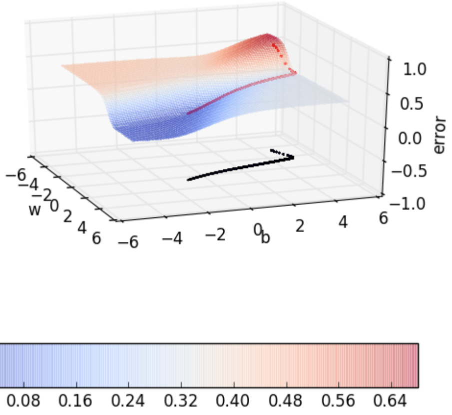




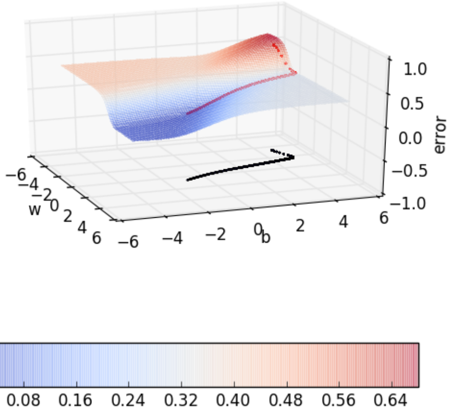
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



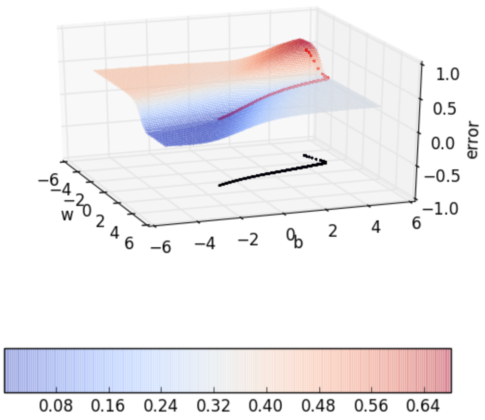
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



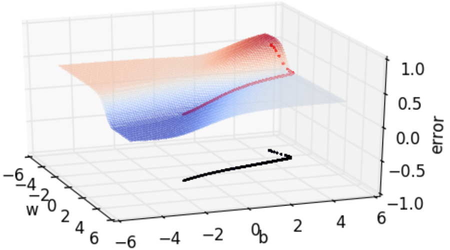
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

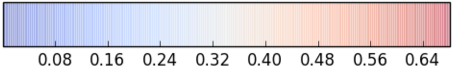


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

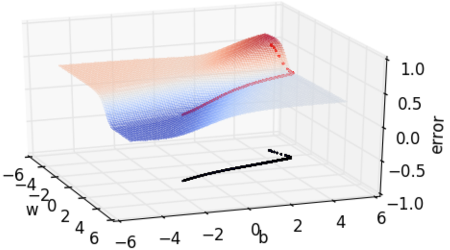


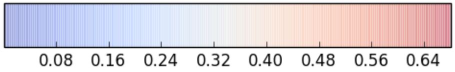
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



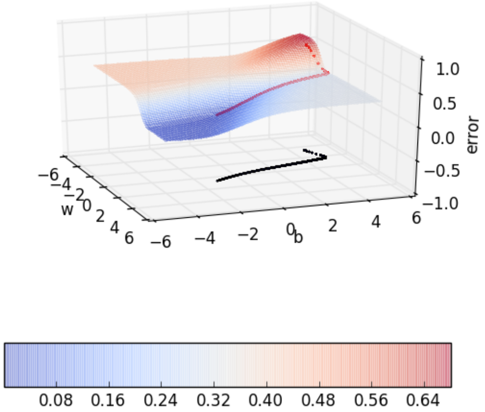


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

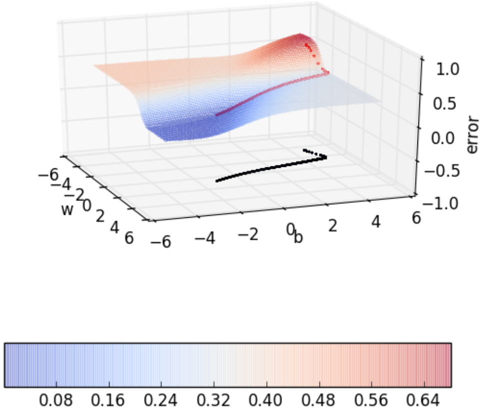




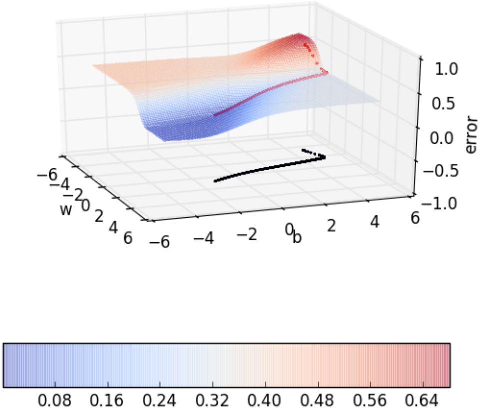
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



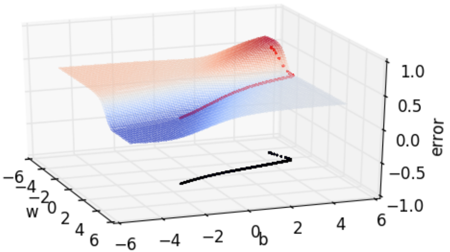
- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down

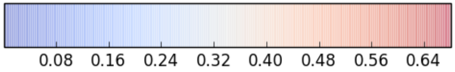


- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down



- Irrespective of where we start from once we hit a surface which has a gentle slope, the progress slows down





Module 5.3 : Contour Maps

- *Visualizing things in 3d can sometimes become a bit cumbersome*

- *Visualizing things in 3d can sometimes become a bit cumbersome*
- *Can we do a 2d visualization of this traversal along the error surface*

- *Visualizing things in 3d can sometimes become a bit cumbersome*
- *Can we do a 2d visualization of this traversal along the error surface*
- *Yes, let's take a look at something known as contours*

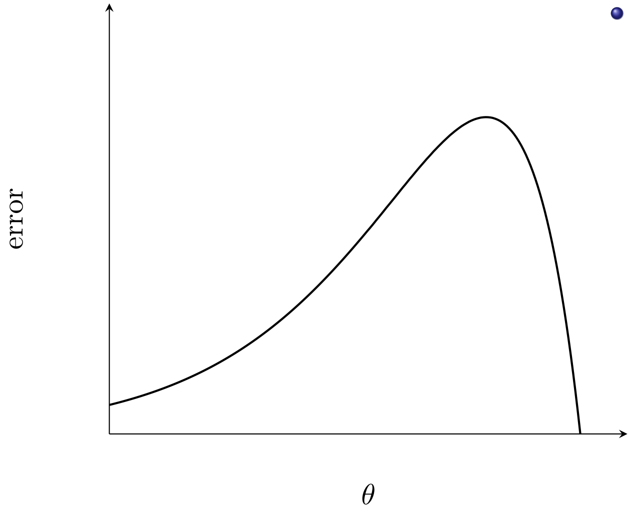
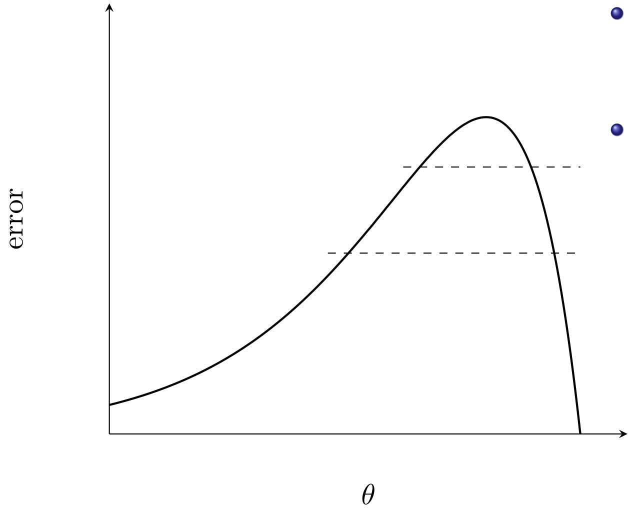


Figure : Front view of a 3d error surface

- Suppose I take horizontal slices of this error surface at regular intervals along the vertical axis



- Suppose I take horizontal slices of this error surface at regular intervals along the vertical axis
- How would this look from the top-view ?

Figure : Front view of a 3d error surface

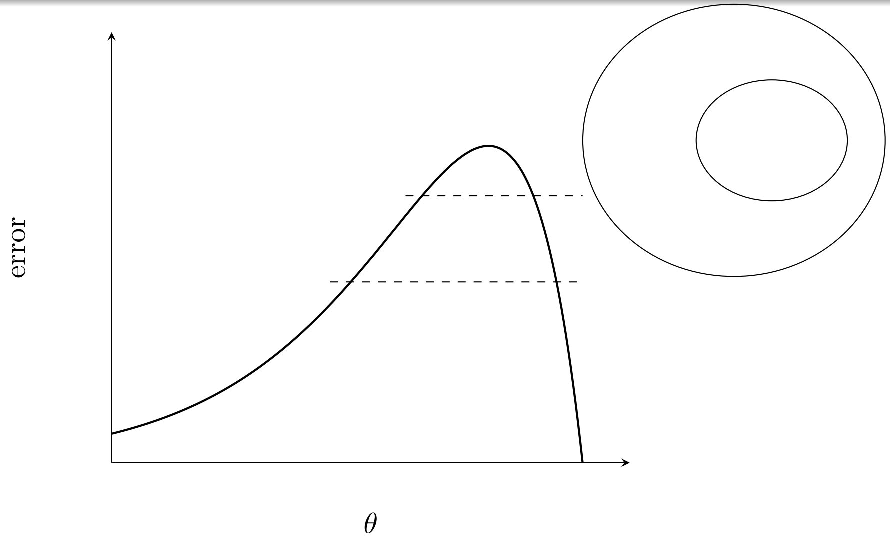
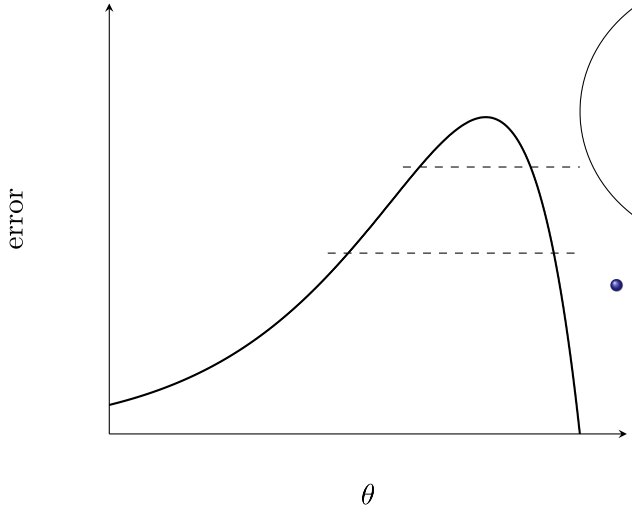
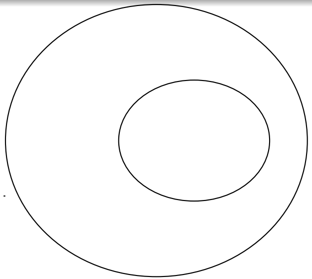


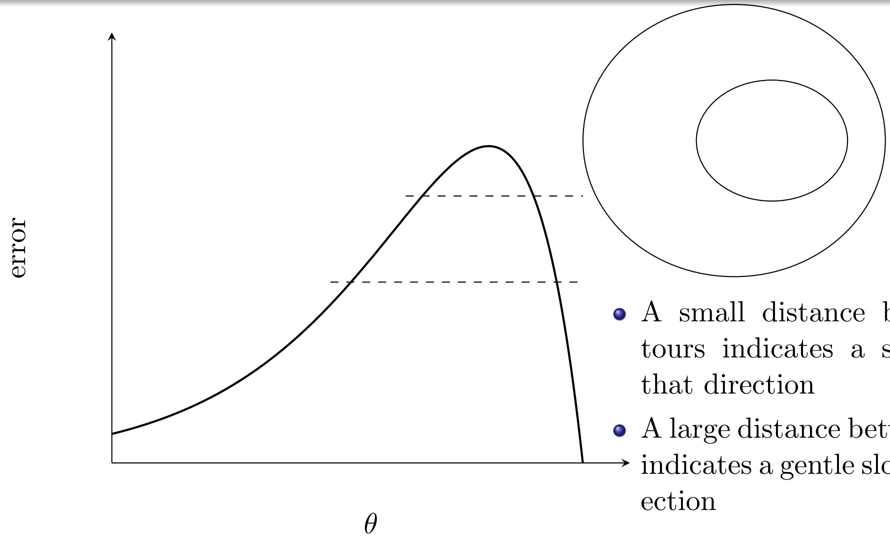
Figure : Front view of a 3d error surface





- A small distance between the contours indicates a steep slope along that direction

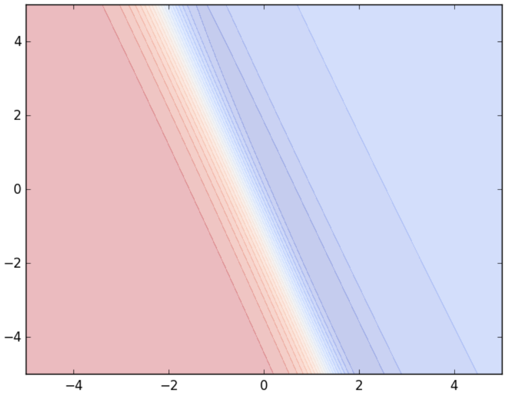
Figure : Front view of a 3d error surface



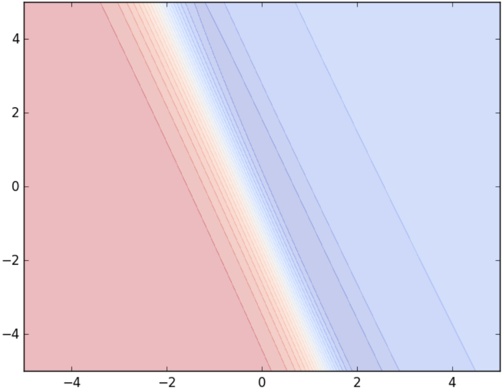
- A small distance between the contours indicates a steep slope along that direction
- A large distance between the contours indicates a gentle slope along that direction

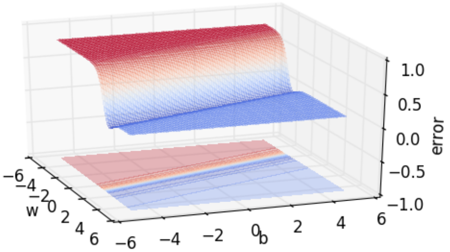
Figure : Front view of a 3d error surface

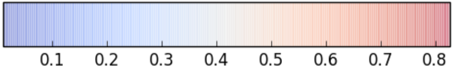
- *Just to ensure that we understand this properly let us do a few exercises ...*



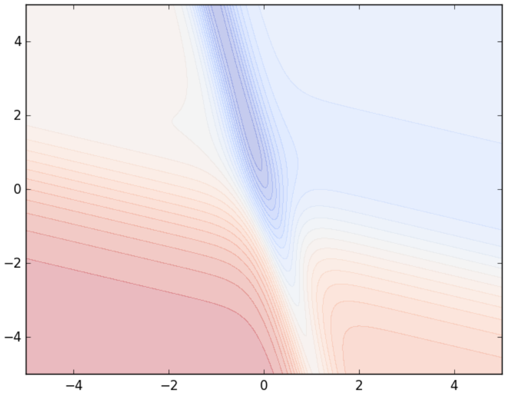
Guess the 3d surface



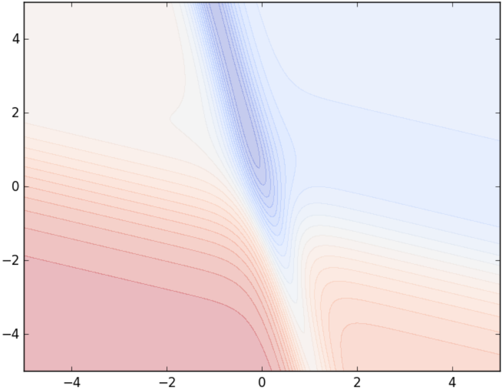


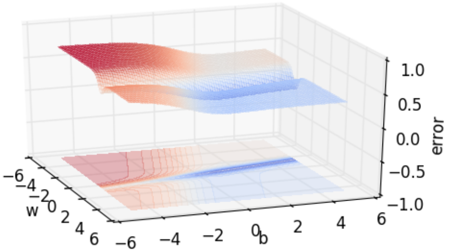


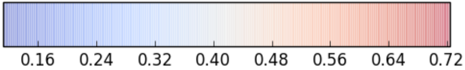
Guess the 3d surface



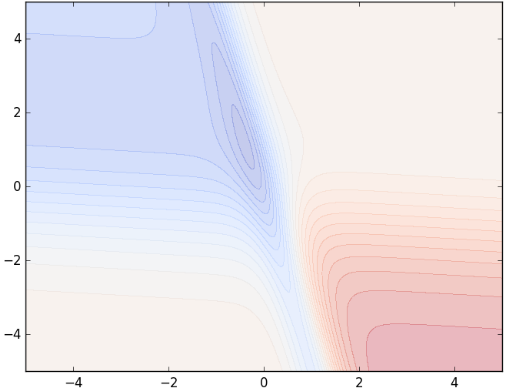
Guess the 3d surface



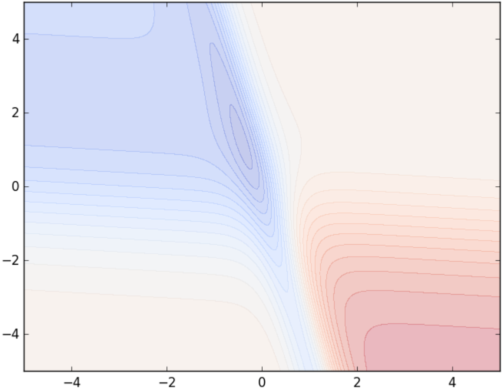


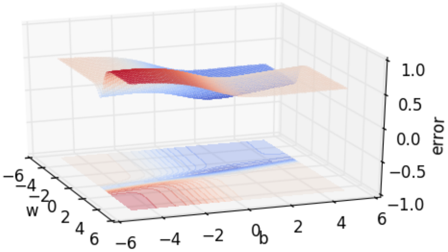


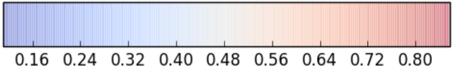
Guess the 3d surface



Guess the 3d surface

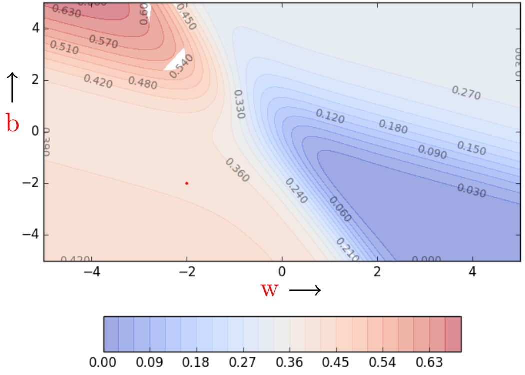


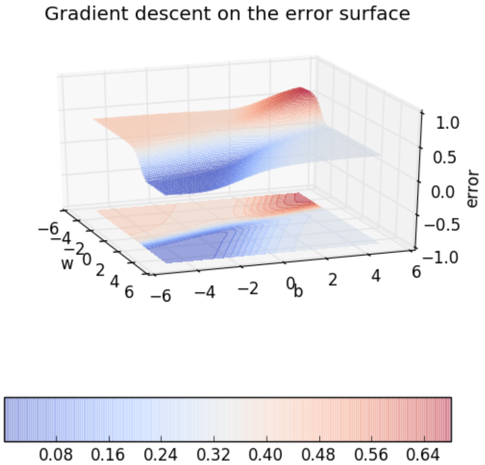


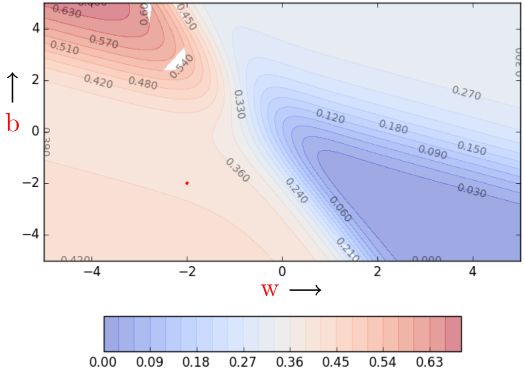


Guess the 3d surface

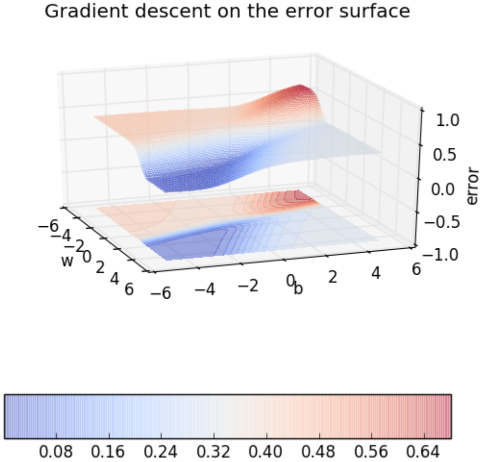
- *Now that we know what are contour maps and how to read them let us go back to our toy example and visualize gradient descent from the point of view of contours...*


0.630
0.570
0.510
0.450
0.540
0.420
0.480
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

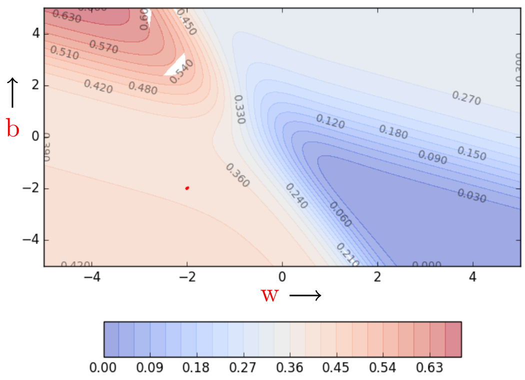

Gradient descent on the error surface
1.0
0.5
0.0
−0.5
−1.0
error
w
b
−6
−4
−2
0
2
4
6
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

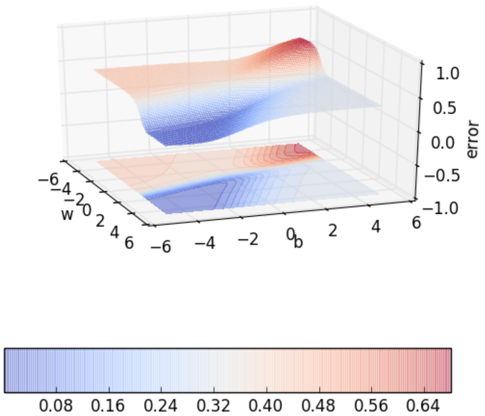


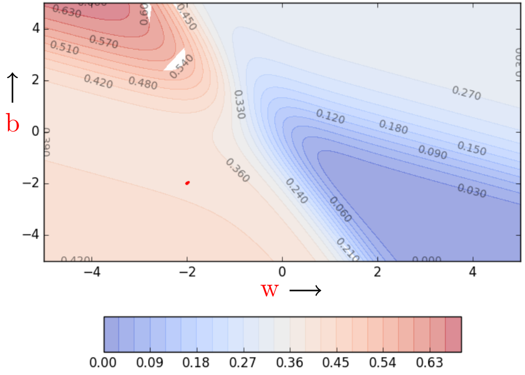
b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



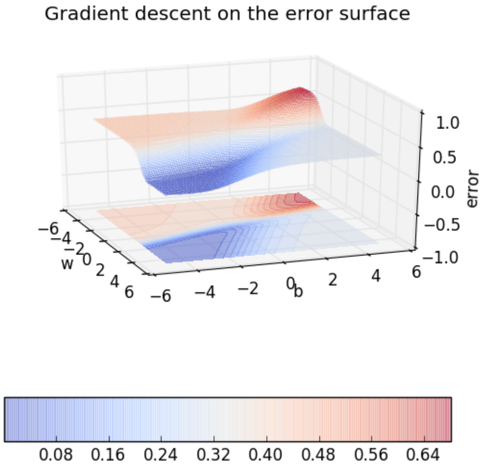
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

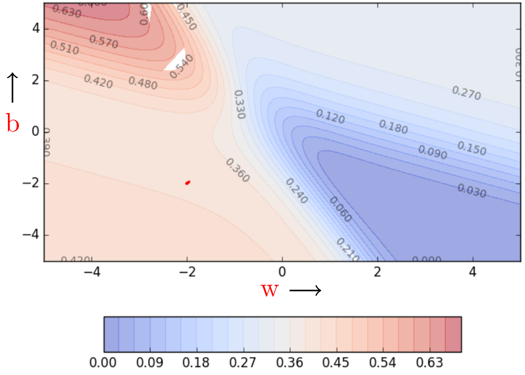

Gradient descent on the error surface
error
w
b
1.0
0.5
0.0
−0.5
−1.0
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

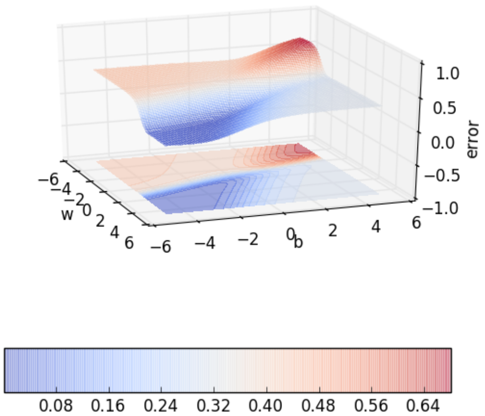


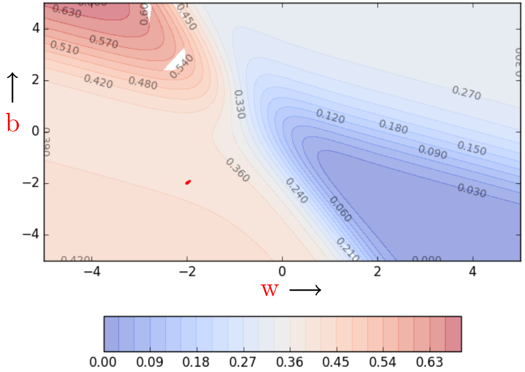
b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



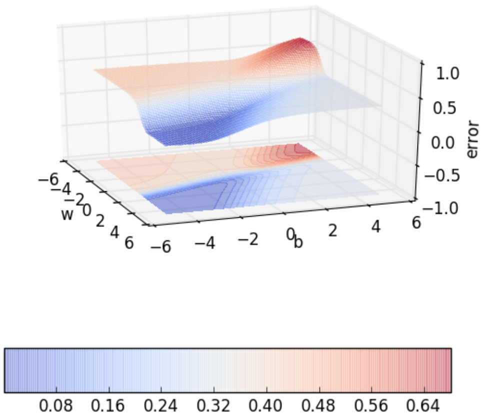
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

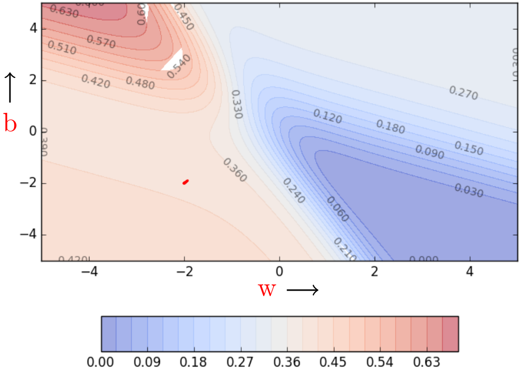

b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

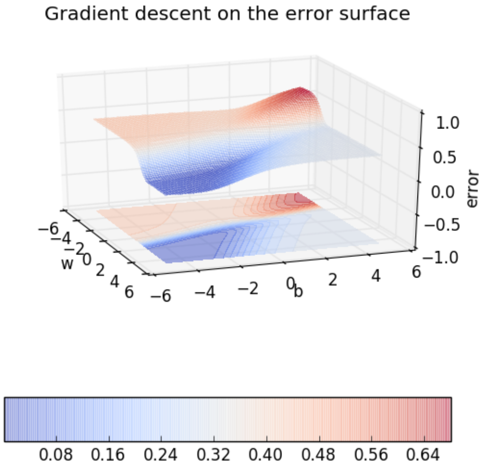

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

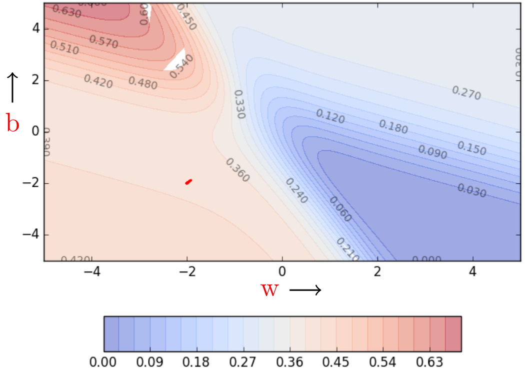


Gradient descent on the error surface

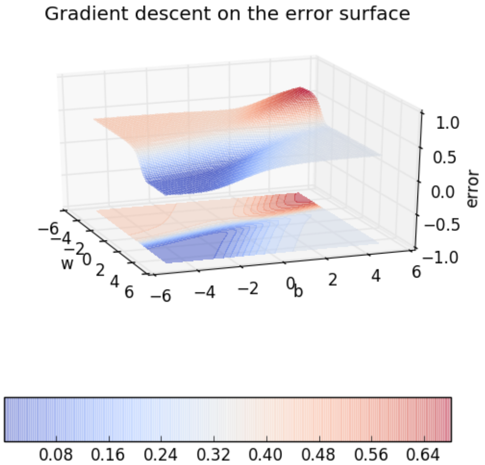



b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.150
0.090
0.030
0.360
0.240
0.060
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

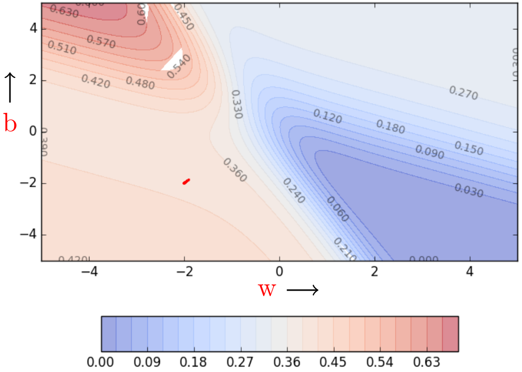

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



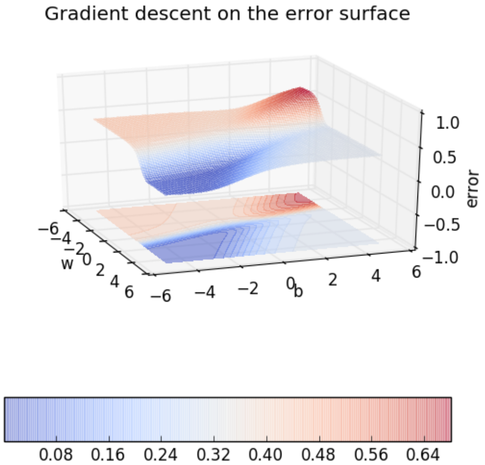
b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



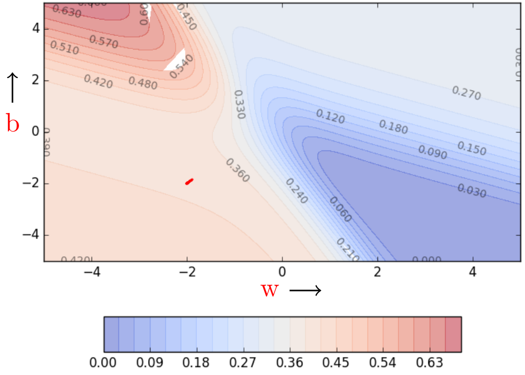
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

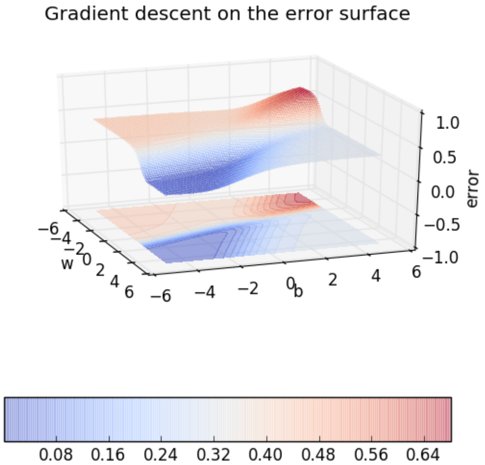


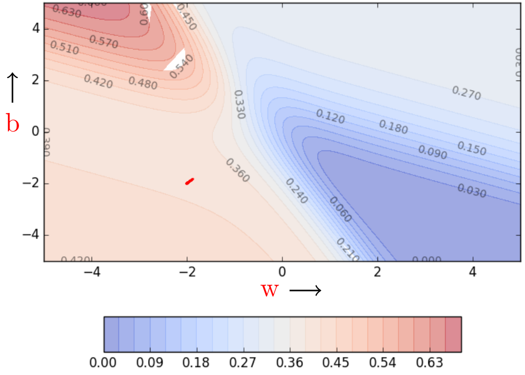
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



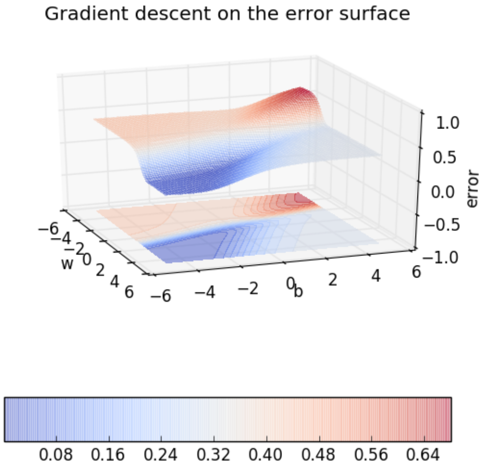
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

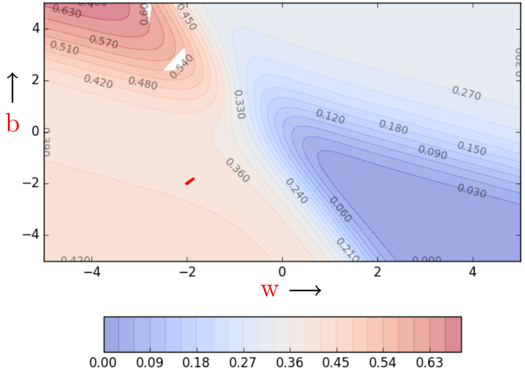

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

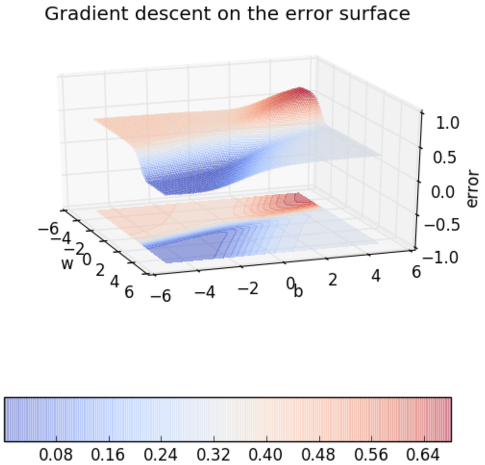


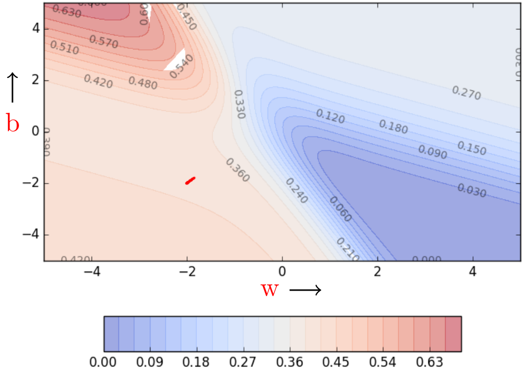
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

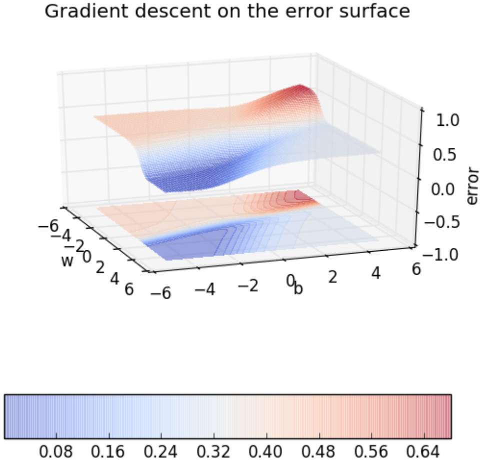


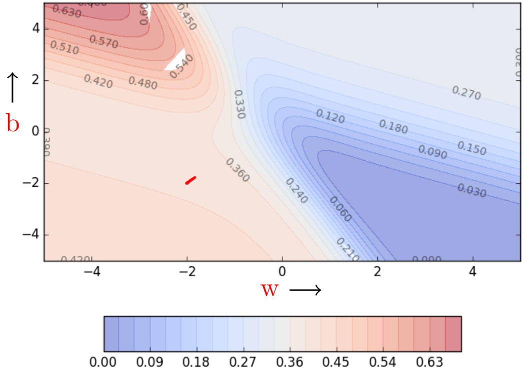
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

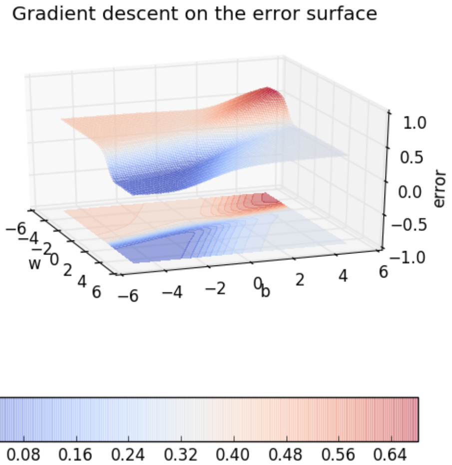

b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

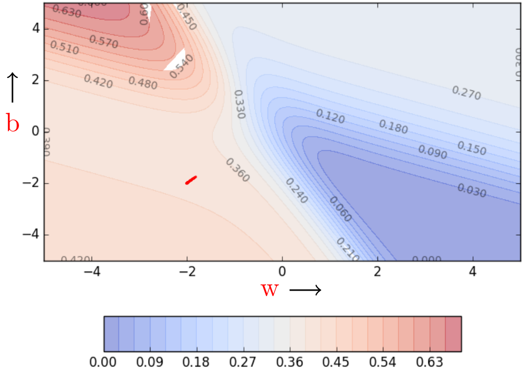

Gradient descent on the error surface
error
w
b
1.0
0.5
0.0
−0.5
−1.0
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

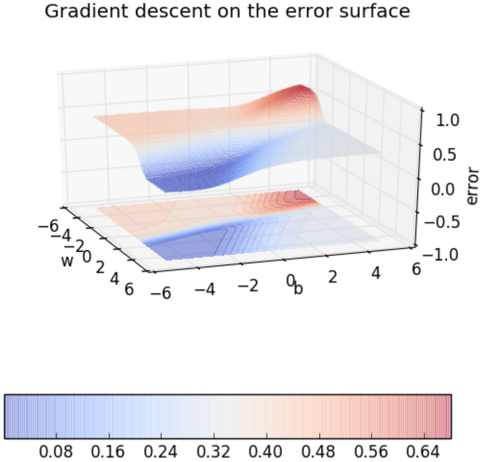

b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

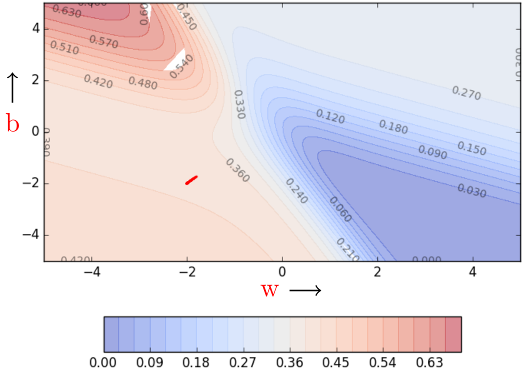

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.360
0.270
0.120
0.180
0.090
0.150
0.030
0.240
0.060
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

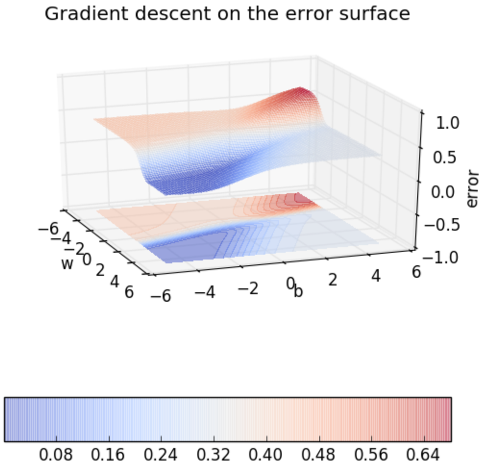

Gradient descent on the error surface
w
b
error
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

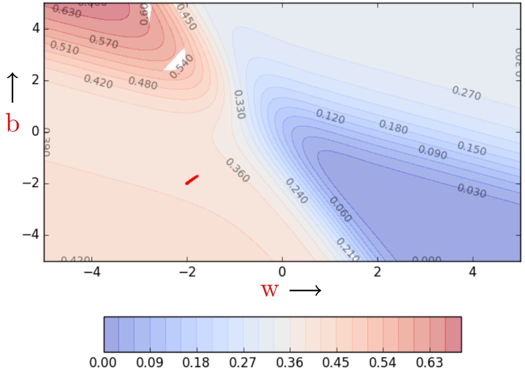

Gradient descent on the error surface
w
b
error
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



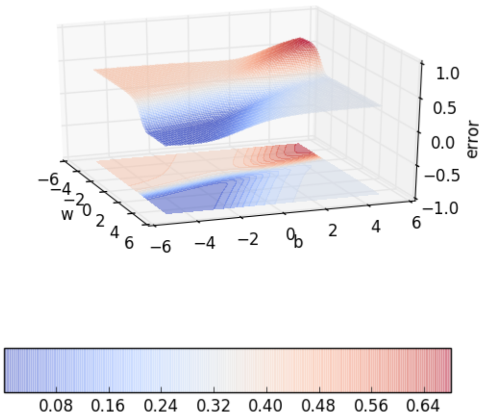
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.270
0.120
0.180
0.150
0.090
0.030
0.360
0.240
0.060
0.210
0.000
b
w
4
2
0
−2
−4
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

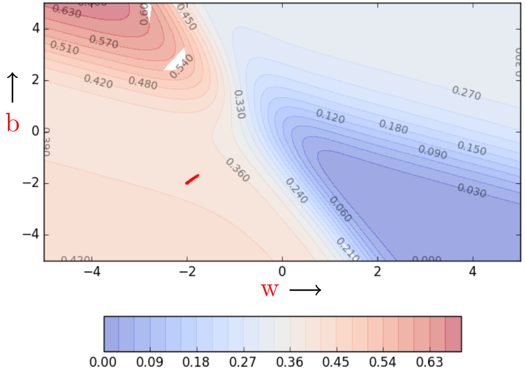


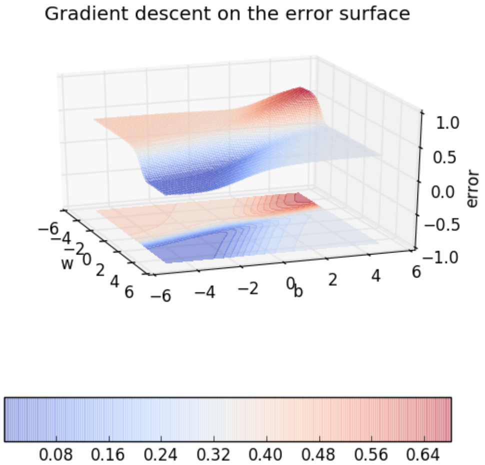
Gradient descent on the error surface
error
1.0
0.5
0.0
−0.5
−1.0
w
b
−6
−4
−2
0
2
4
6
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

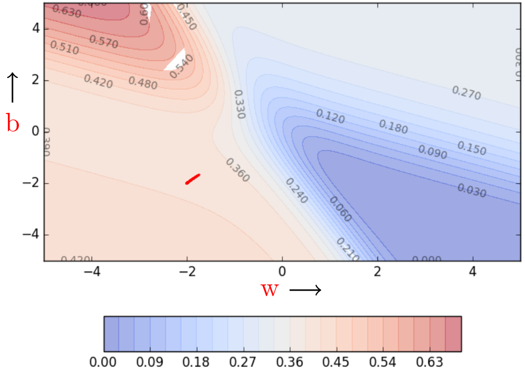


Gradient descent on the error surface

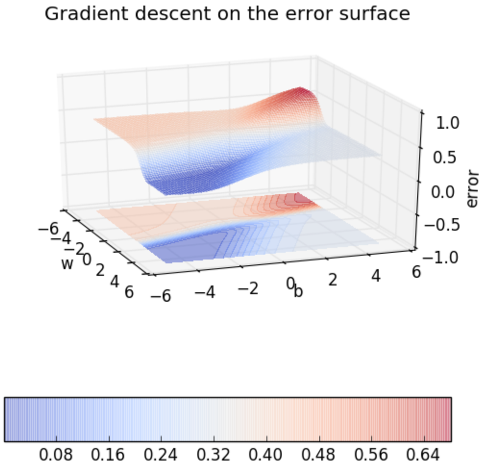



b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

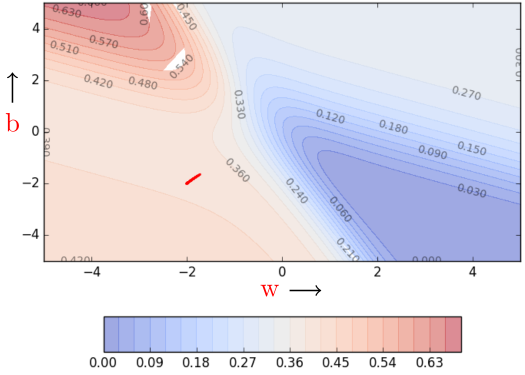

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

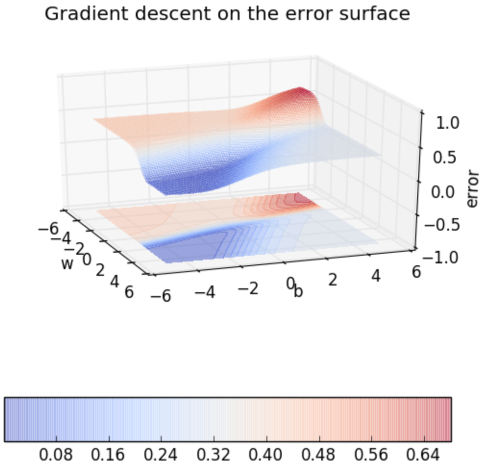


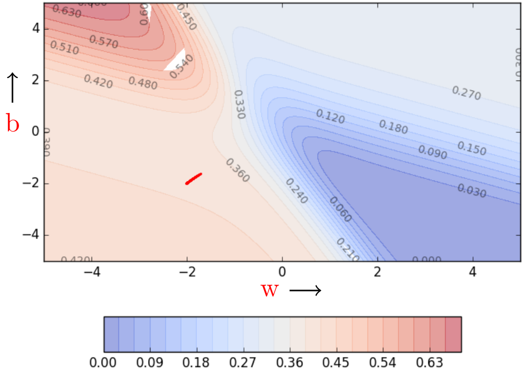
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



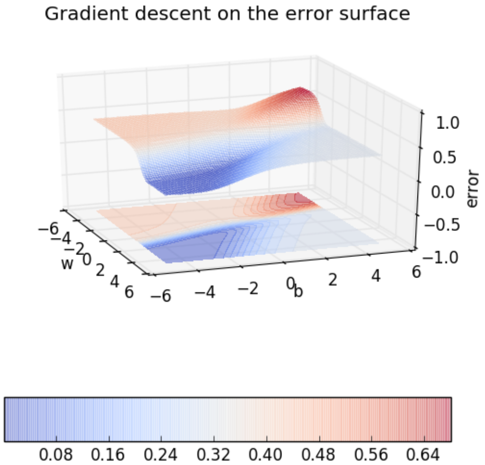
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

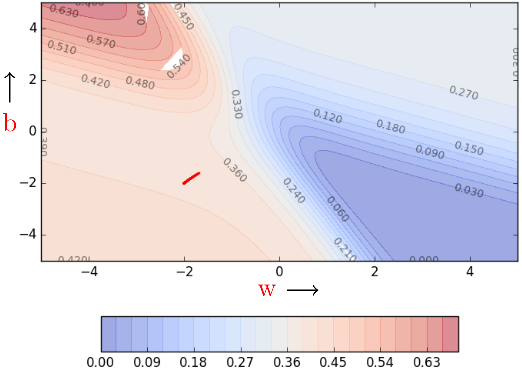

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



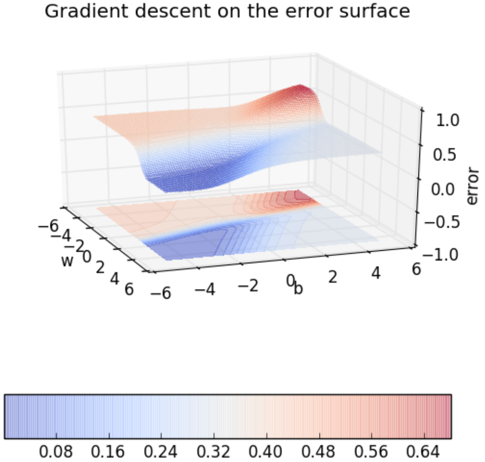
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.150
0.090
0.030
0.360
0.240
0.060
0.210
−4
−2
0
2
4
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



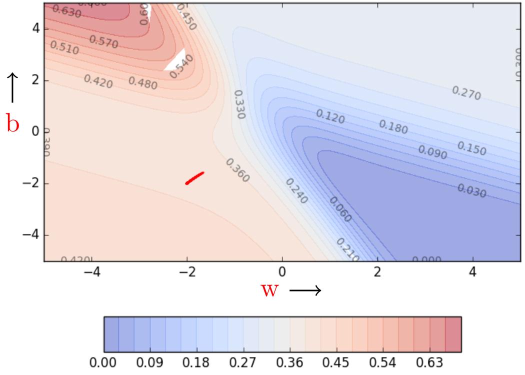
Gradient descent on the error surface
error
w
b
1.0
0.5
0.0
−0.5
−1.0
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



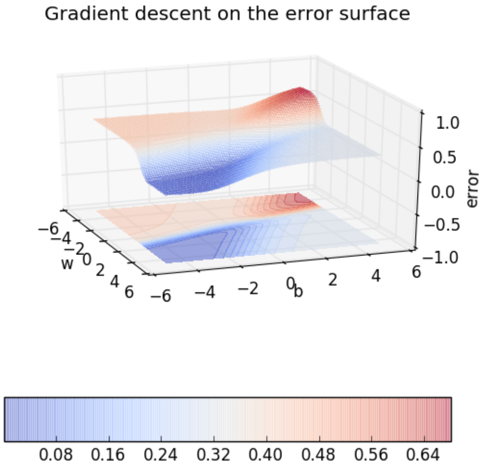
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.000
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



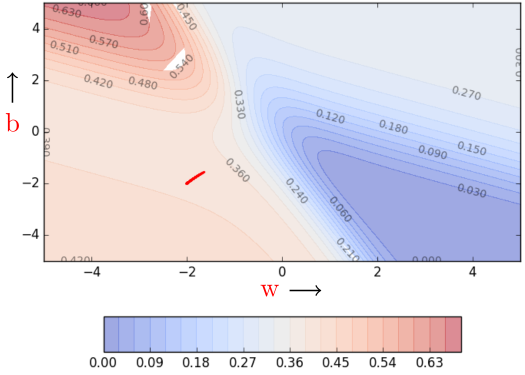
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

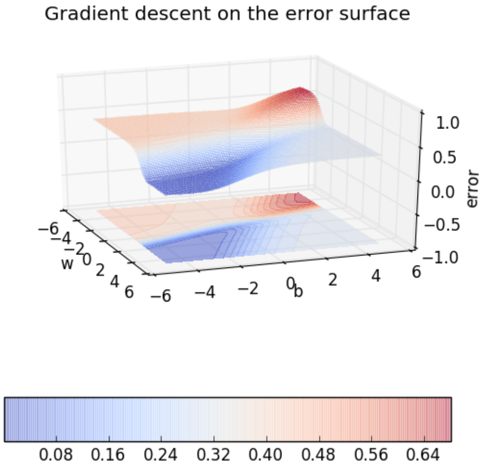


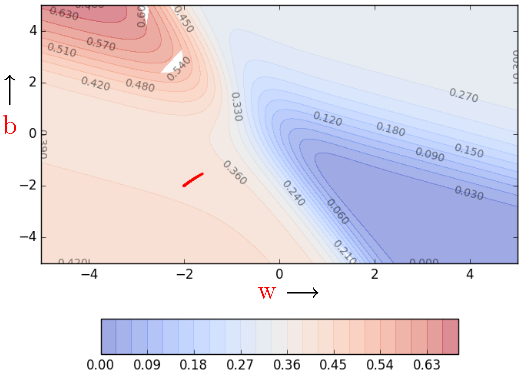
b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.270
0.120
0.180
0.150
0.090
0.030
0.360
0.240
0.060
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



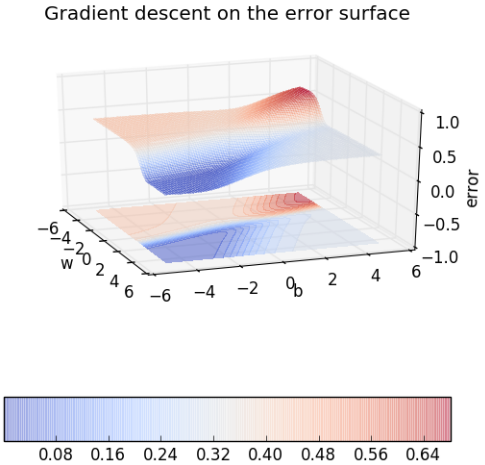
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

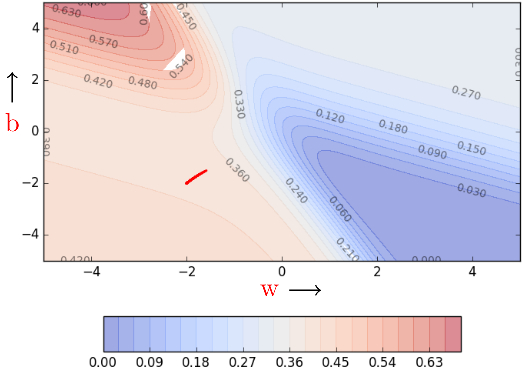

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



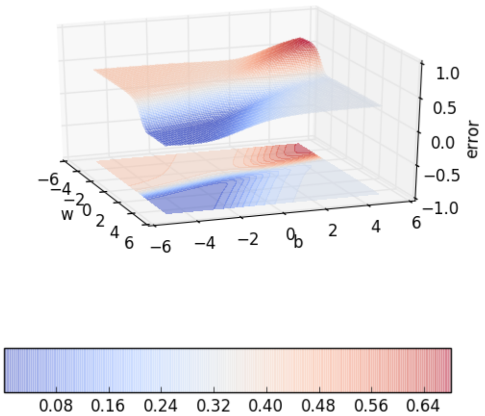
b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

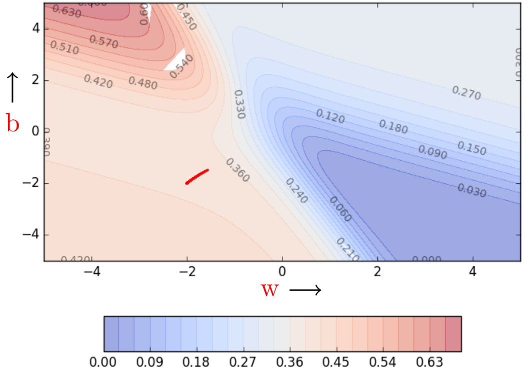


Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

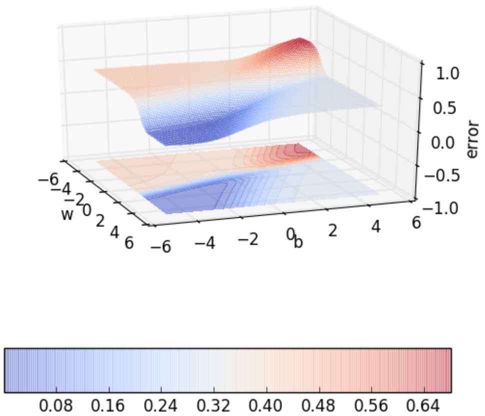


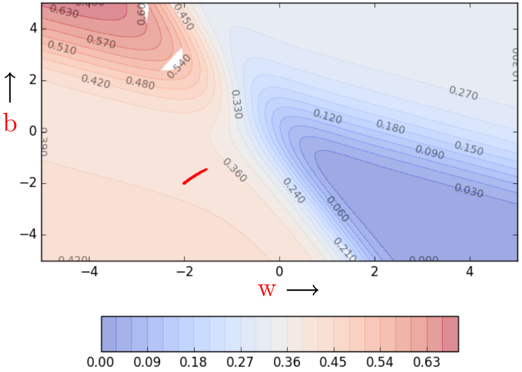
Gradient descent on the error surface



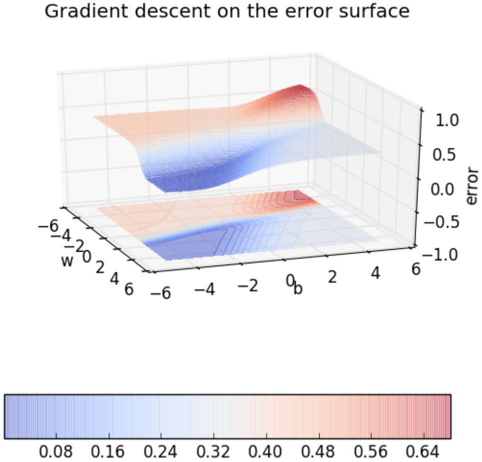


Gradient descent on the error surface

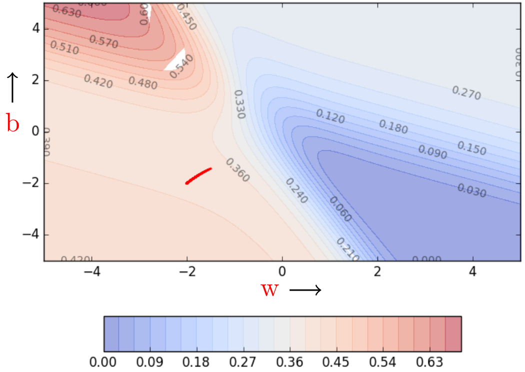




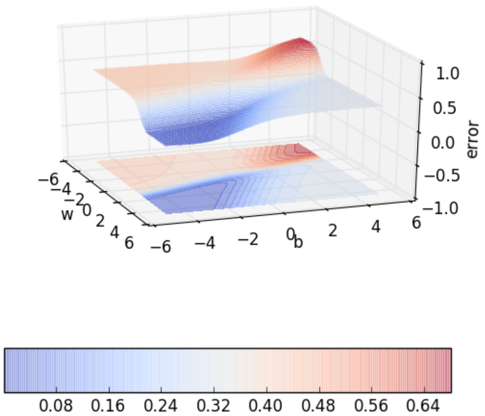
b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

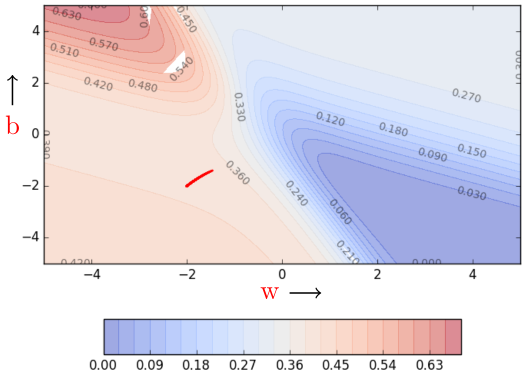


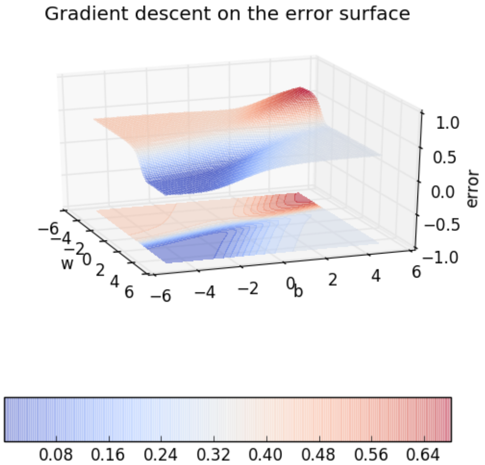
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

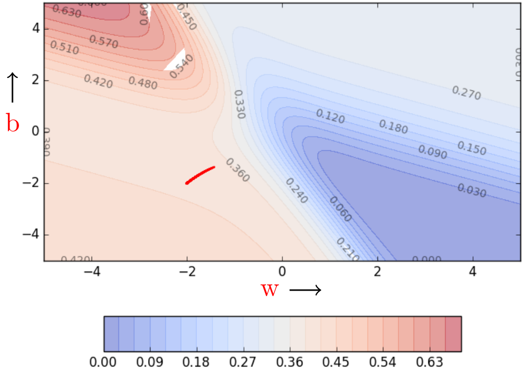


Gradient descent on the error surface

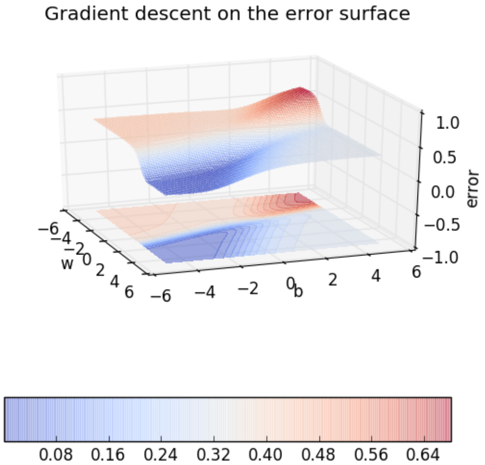



b
w
−4
−2
0
2
4
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.360
0.240
0.060
0.210
0.120
0.180
0.090
0.150
0.030
0.270
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

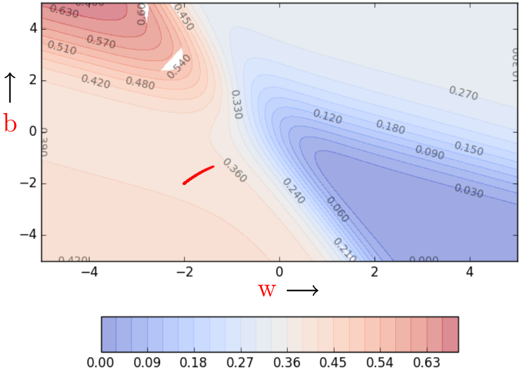

Gradient descent on the error surface
error
w
b
1.0
0.5
0.0
−0.5
−1.0
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



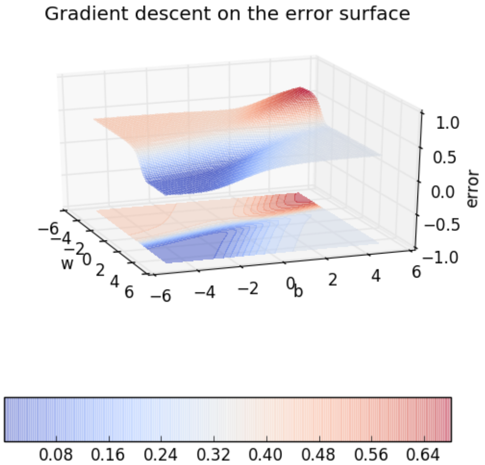
b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



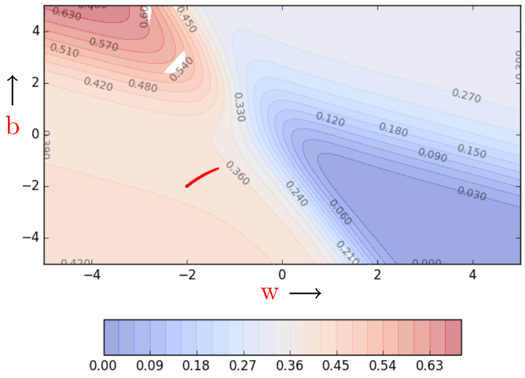
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



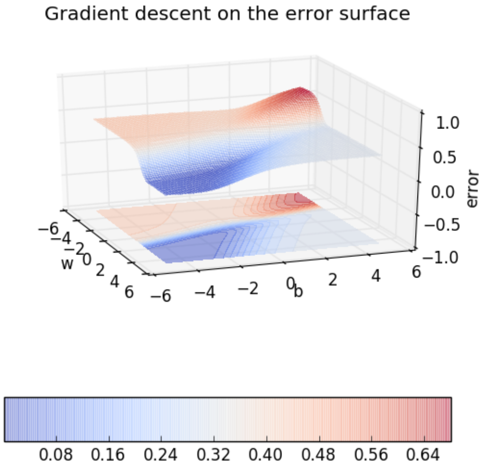
b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.270
0.120
0.180
0.150
0.090
0.030
0.360
0.240
0.060
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



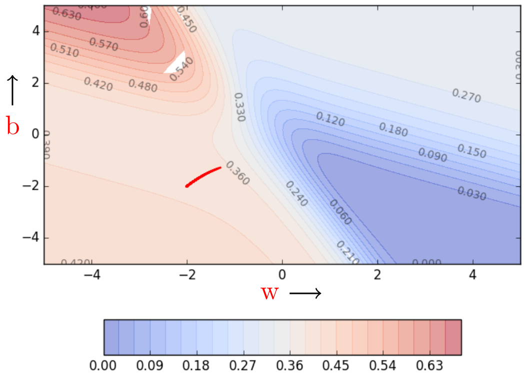
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



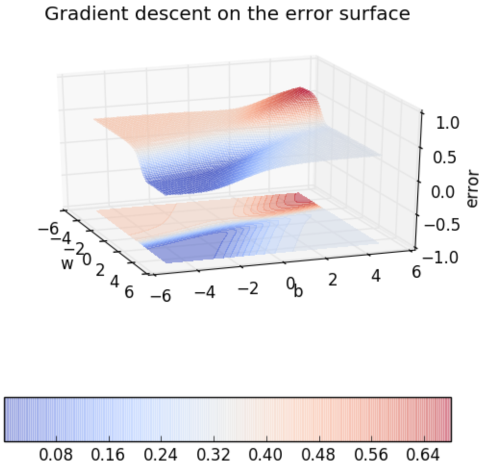
0.630
0.570
0.510
0.450
0.540
0.420
0.480
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



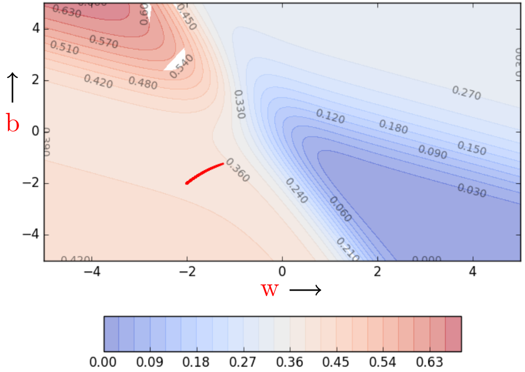
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



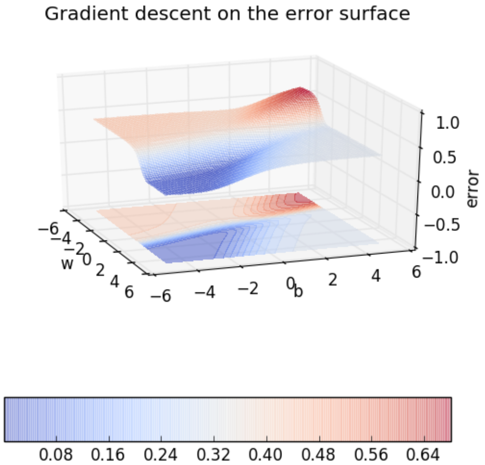
0.630
0.570
0.510
0.450
0.540
0.420
0.480
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



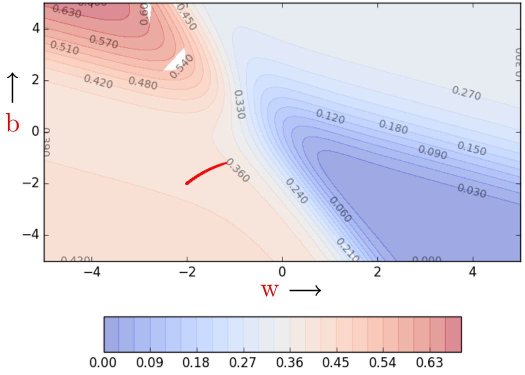
Gradient descent on the error surface
error
w
b
1.0 0.5 0.0 −0.5 −1.0
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



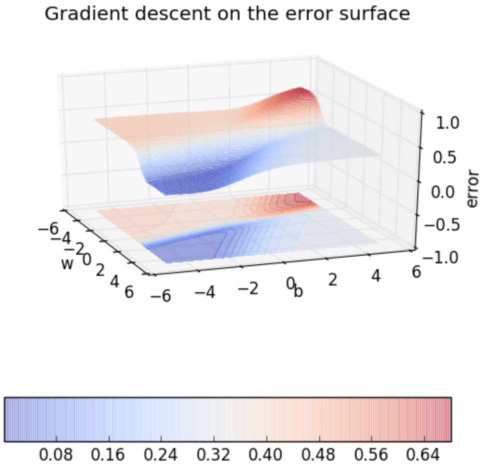
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



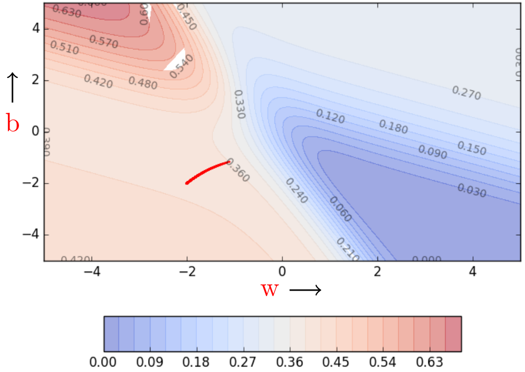
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



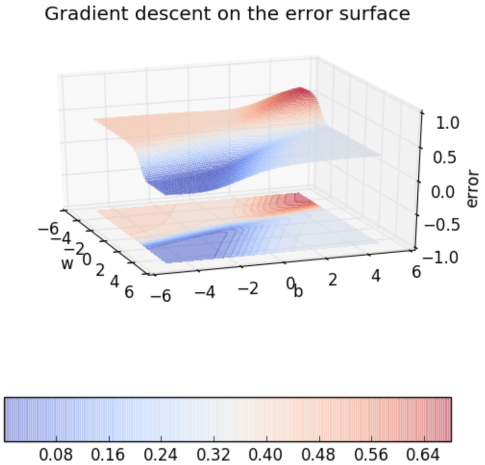
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



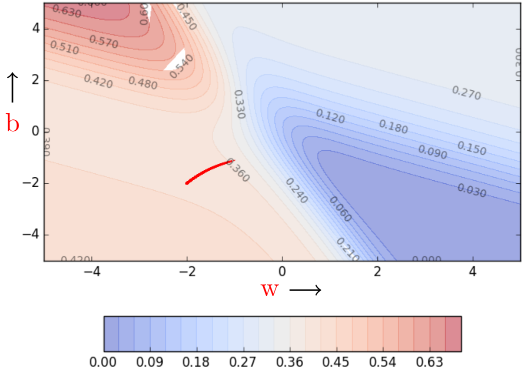
Gradient descent on the error surface
error
w
b
1.0
0.5
0.0
−0.5
−1.0
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

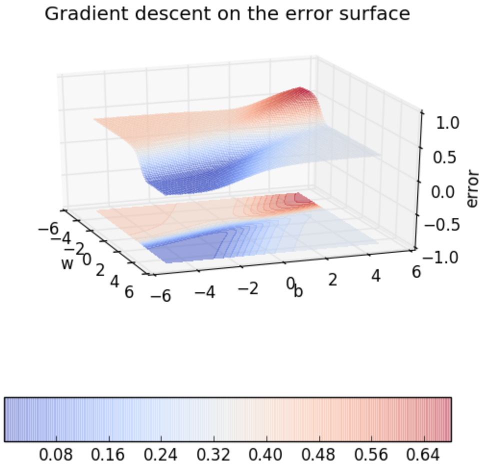


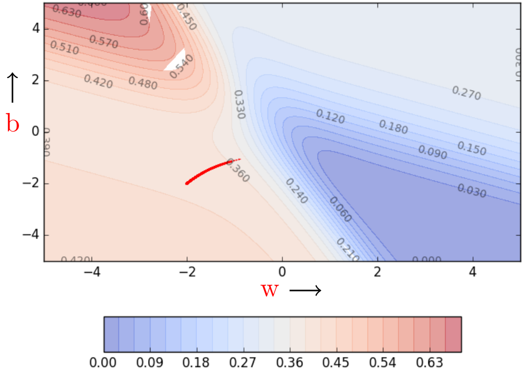
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

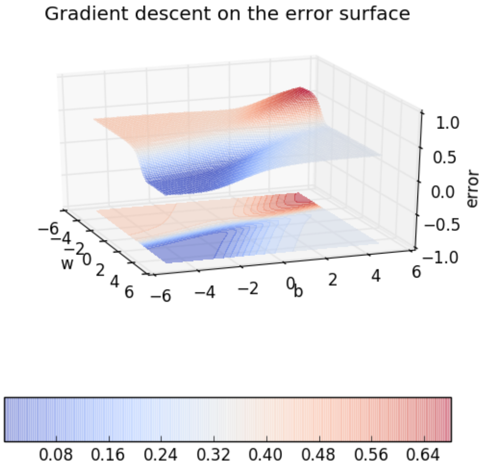


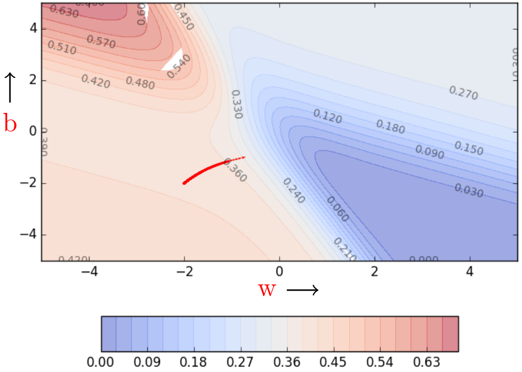
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

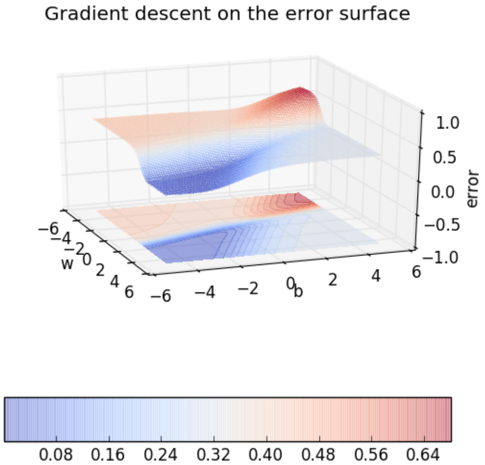

b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.360
0.270
0.120
0.180
0.090
0.150
0.030
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

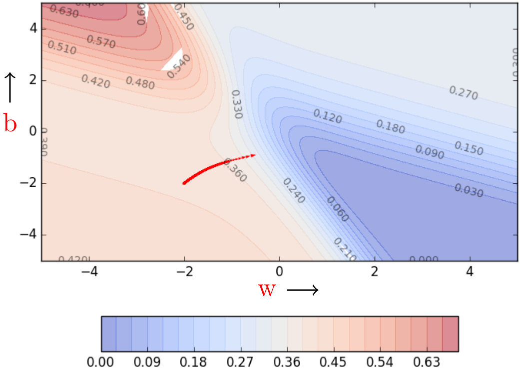

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

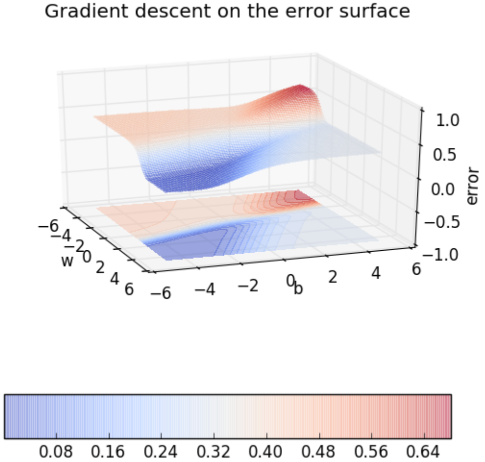

b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.360
0.240
0.060
0.210
0.120
0.180
0.270
0.090
0.150
0.030
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

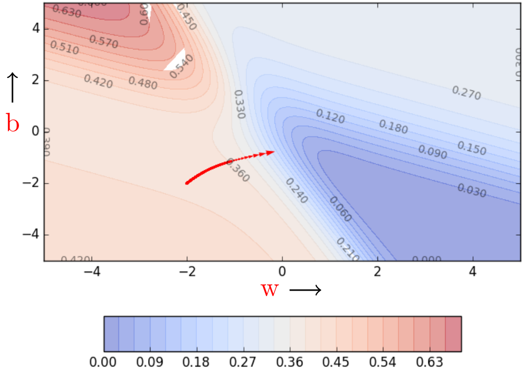

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

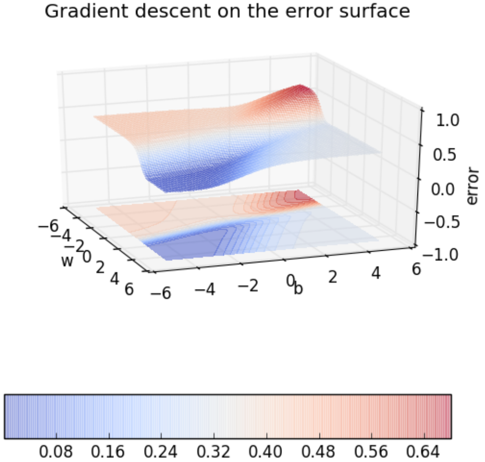

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

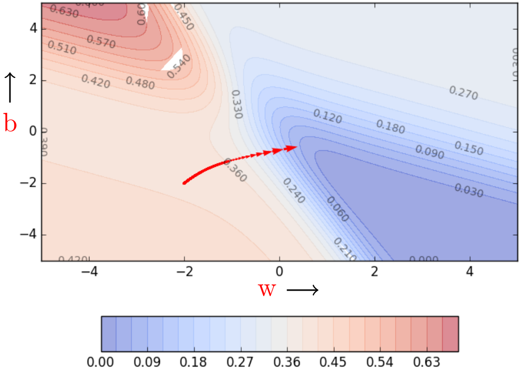

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



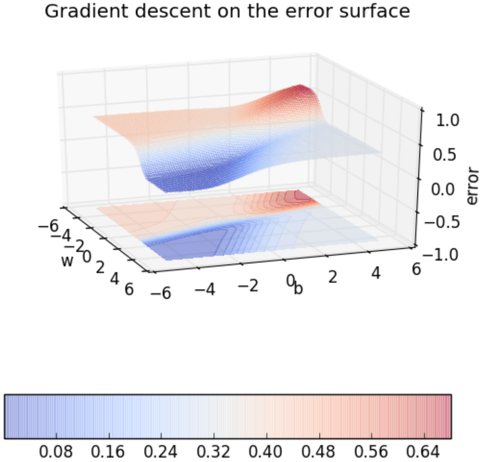
0.630
0.570
0.510
0.450
0.540
0.420
0.480
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



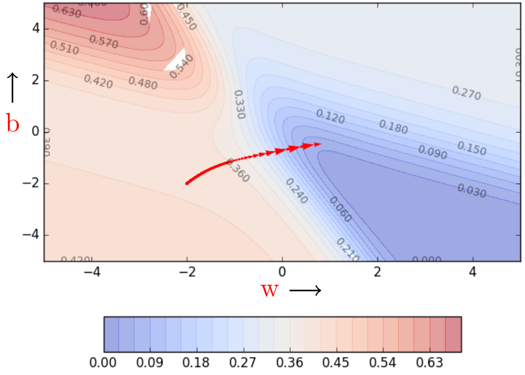
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

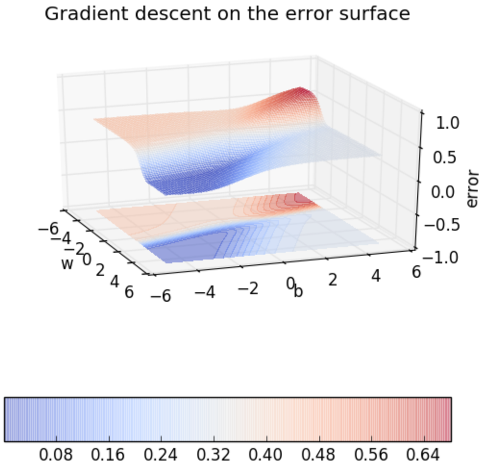


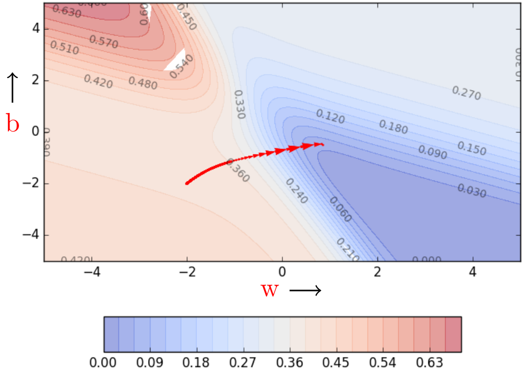
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



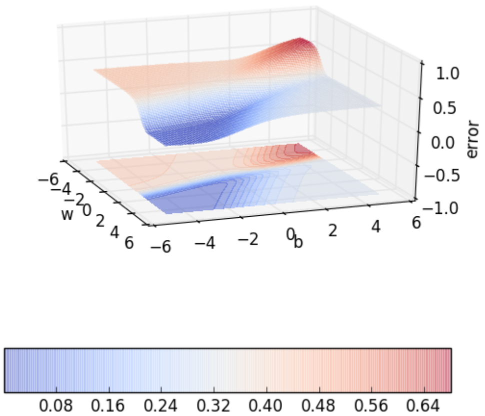
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

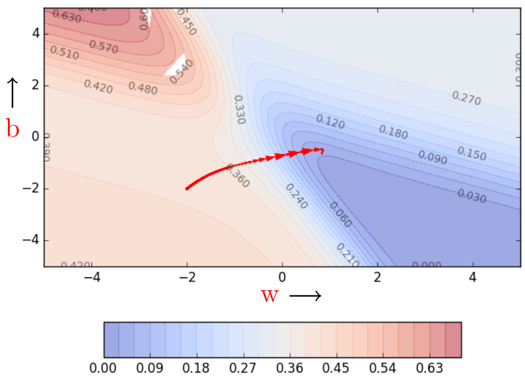

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



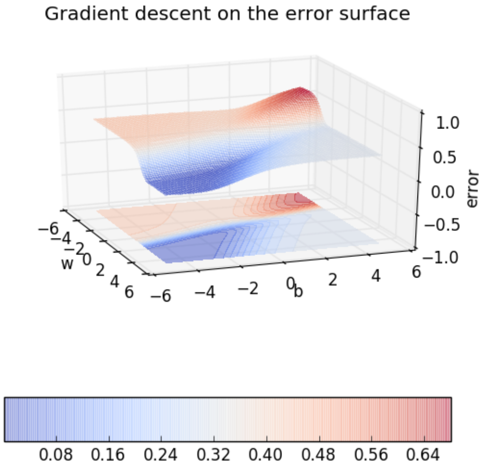
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.000
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



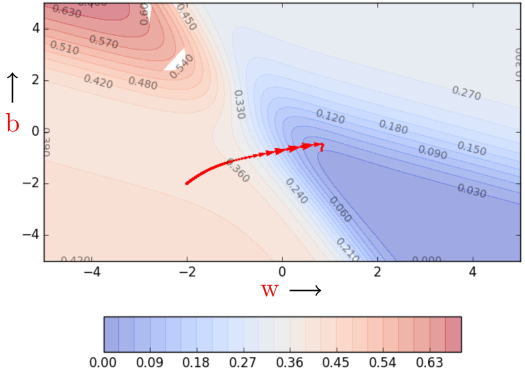
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

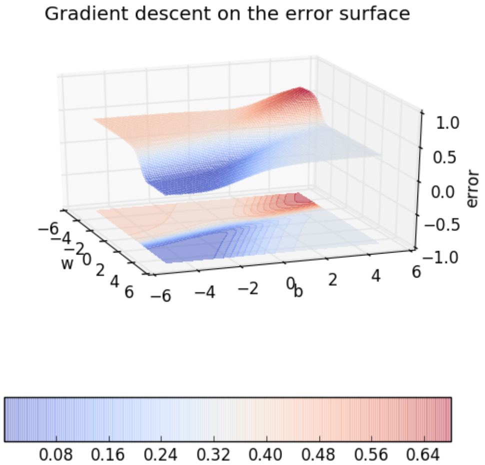


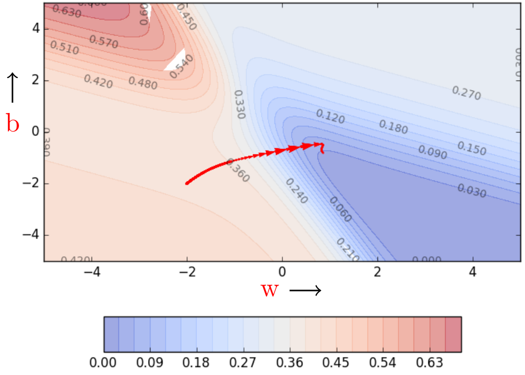
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.000
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

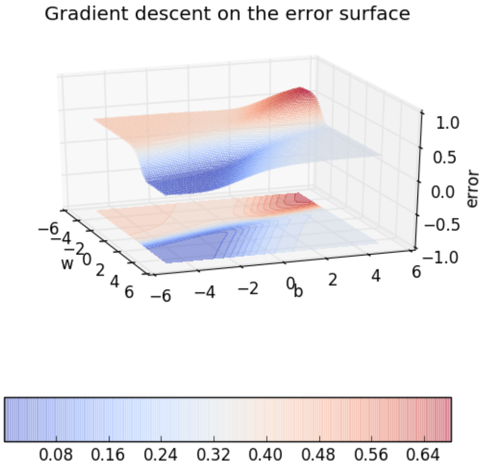


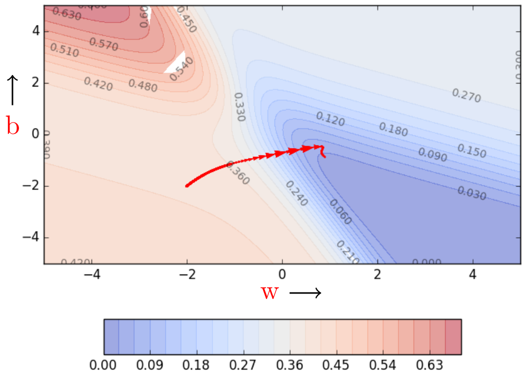
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

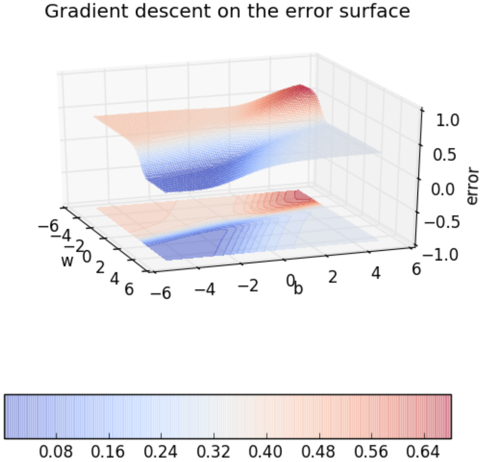

b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

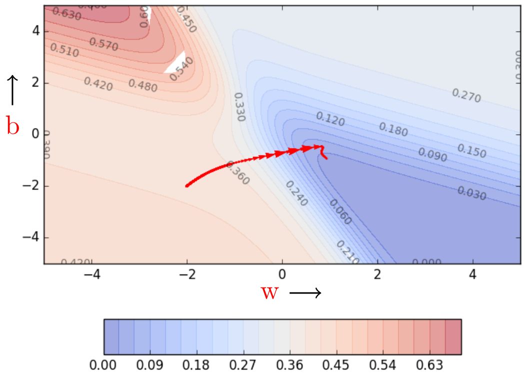

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

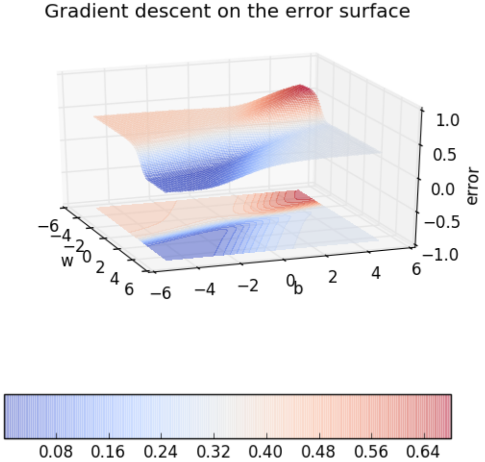

b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

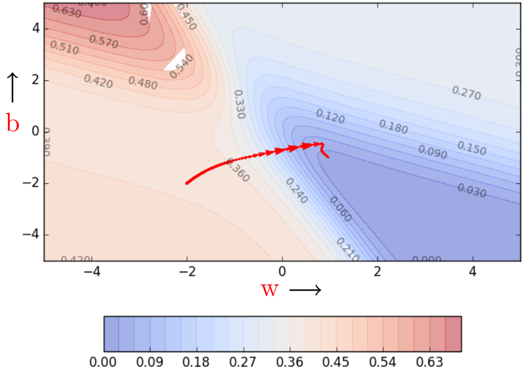

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

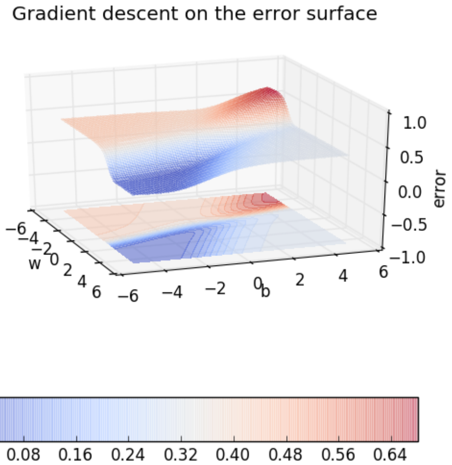

b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

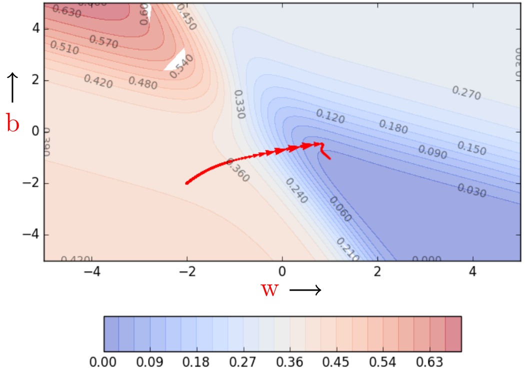

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

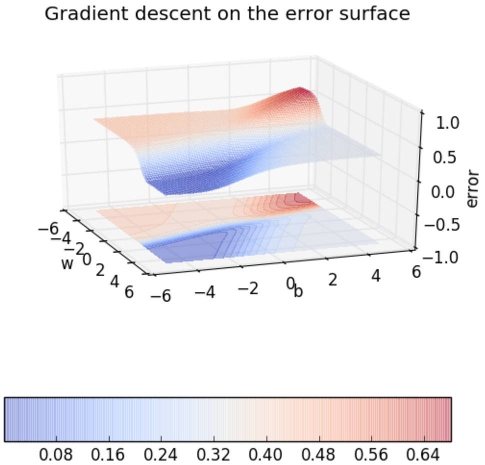

b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

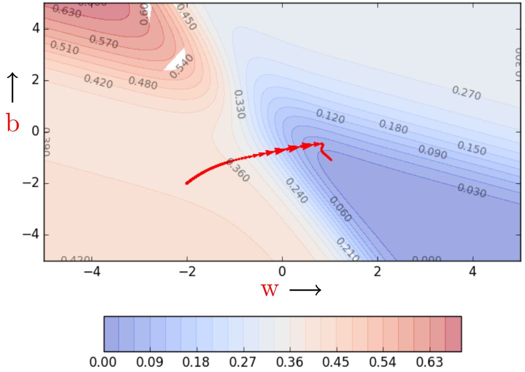

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

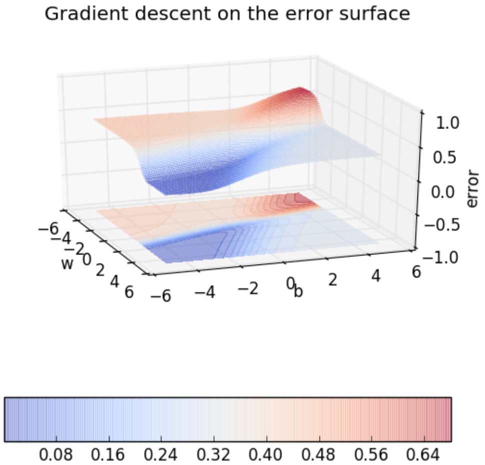

b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.360
0.240
0.060
0.210
0.120
0.180
0.090
0.150
0.030
0.270
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

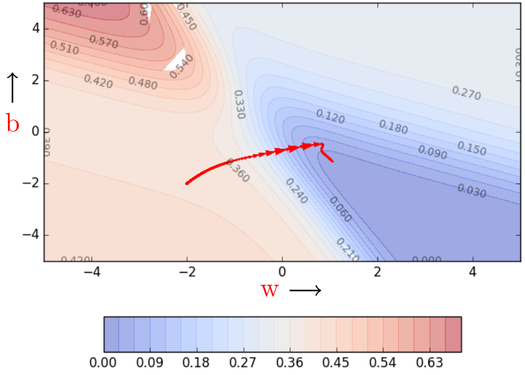

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

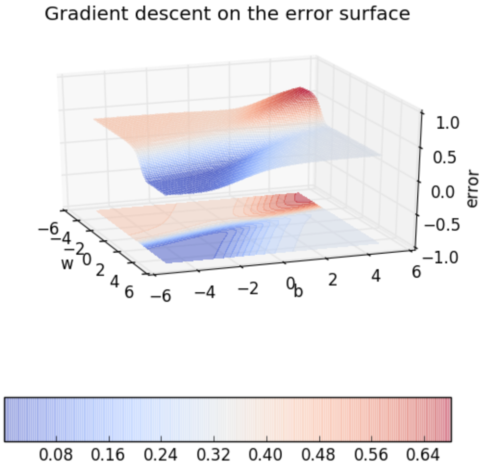

0.630
0.570
0.510
0.450
0.540
0.420
0.480
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

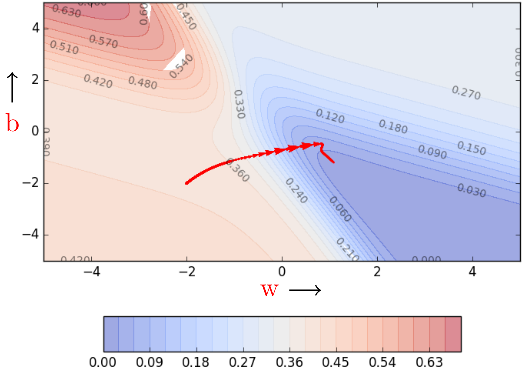

Gradient descent on the error surface
error
1.0
0.5
0.0
−0.5
−1.0
w
b
−6
−4
−2
0
2
4
6
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.000
−4
−2
0
2
4
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

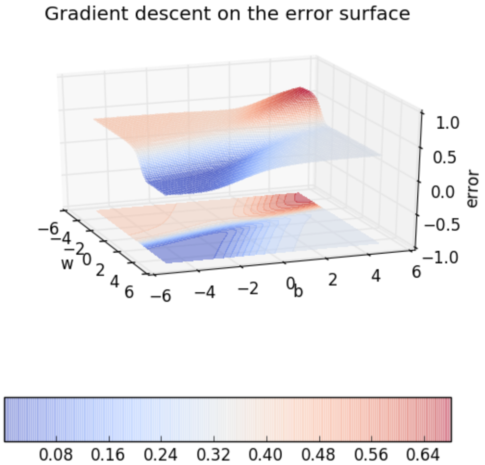

Gradient descent on the error surface
error
1.0
0.5
0.0
−0.5
−1.0
w
b
−6
−4
−2
0
2
4
6
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.450
0.540
0.420
0.480
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

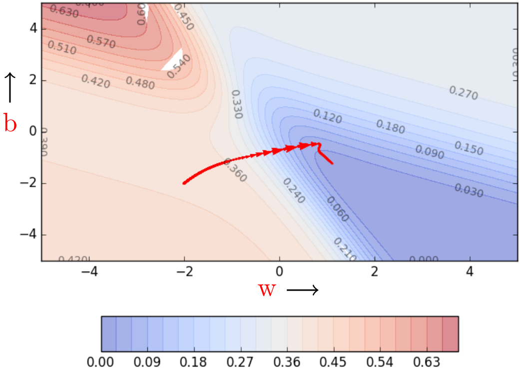

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

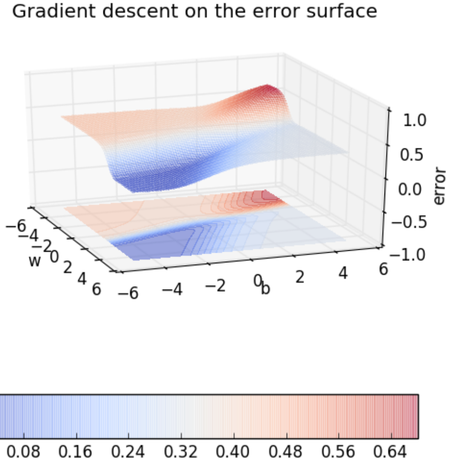


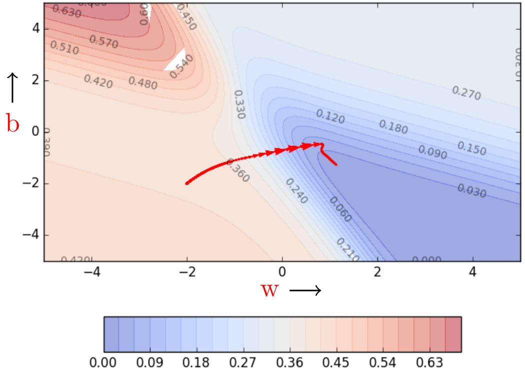
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



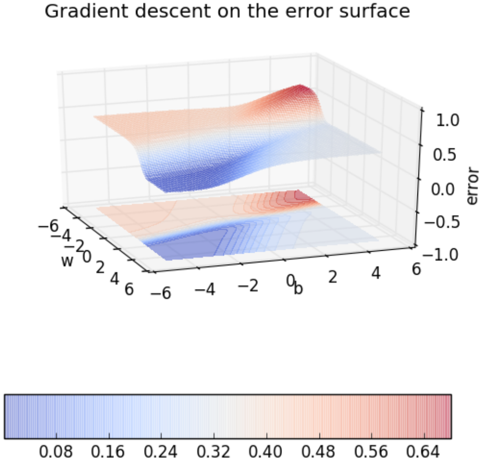
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

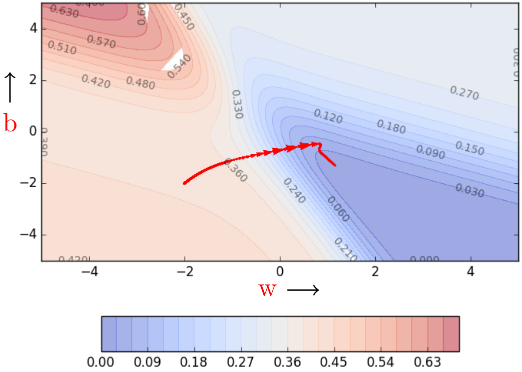

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

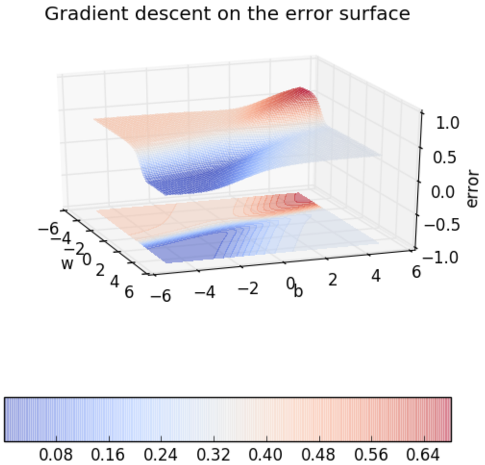


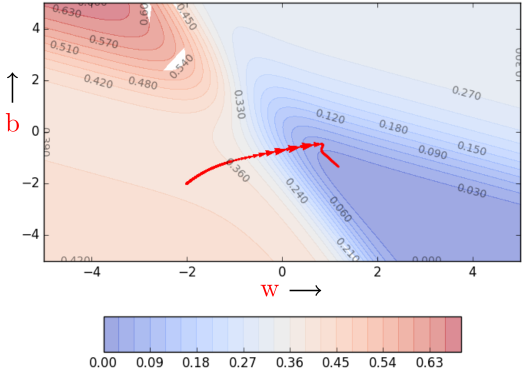
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.000
0.420
4
2
0
−2
−4
−4
−2
0
2
4
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



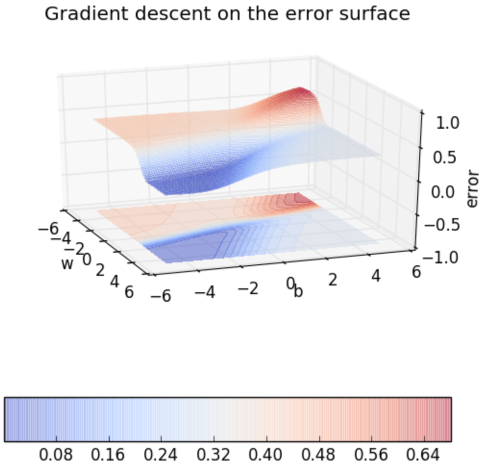
Gradient descent on the error surface
1.0
0.5
0.0
−0.5
−1.0
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

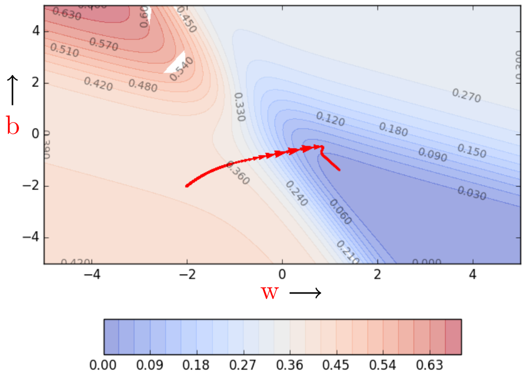

Gradient descent on the error surface
w
b
error
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

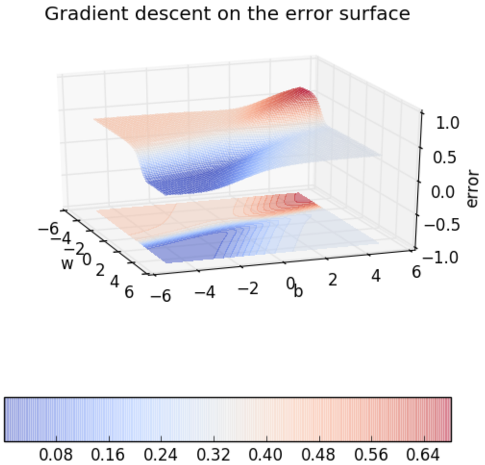


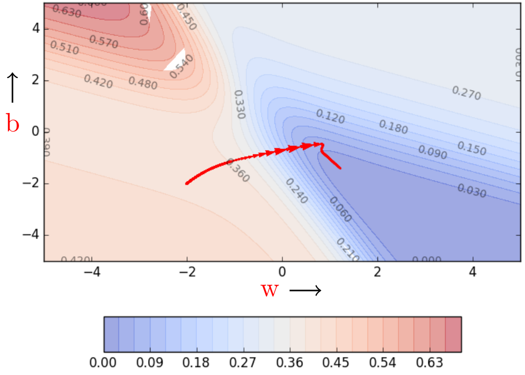
b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.120
0.180
0.270
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

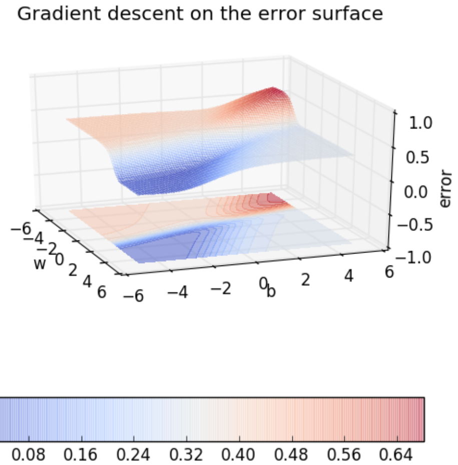


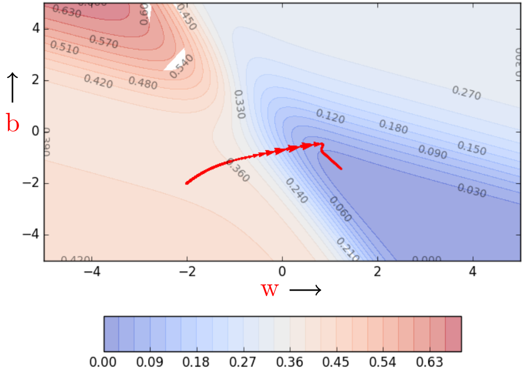
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.360
0.270
0.120
0.180
0.090
0.150
0.030
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

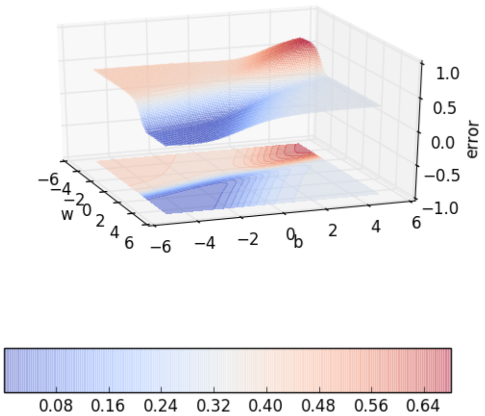

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

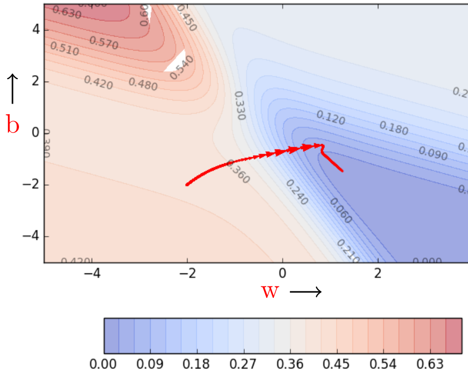

b
w
0.630
0.570
0.510
0.450
0.540
0.420
0.480
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.000
0.420
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

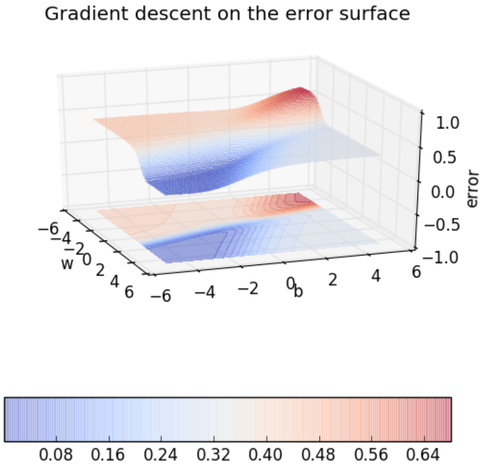

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

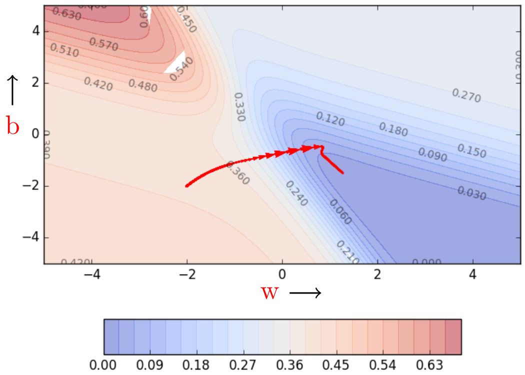


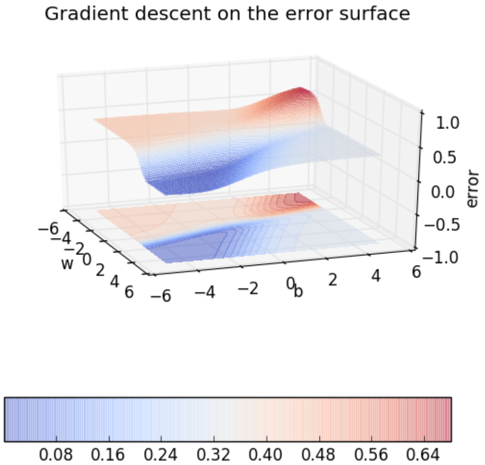
Gradient descent on the error surface

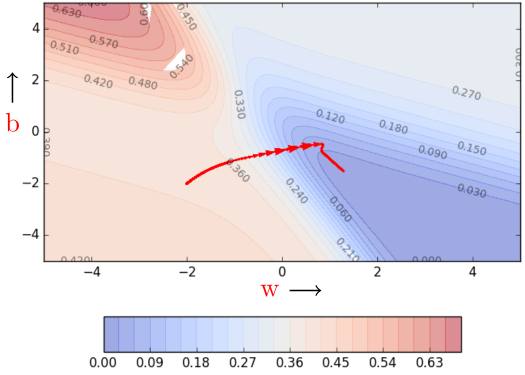


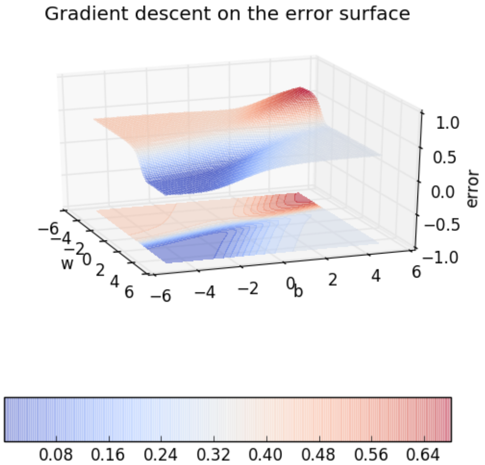

b
w
0.630
0.570
0.510
0.450
0.540
0.420
0.480
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.000
4
2
0
−2
−4
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

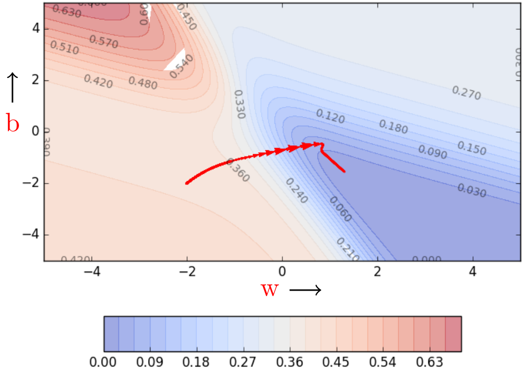

Gradient descent on the error surface
error
1.0
0.5
0.0
−0.5
−1.0
w
b
−6
−4
−2
0
2
4
6
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

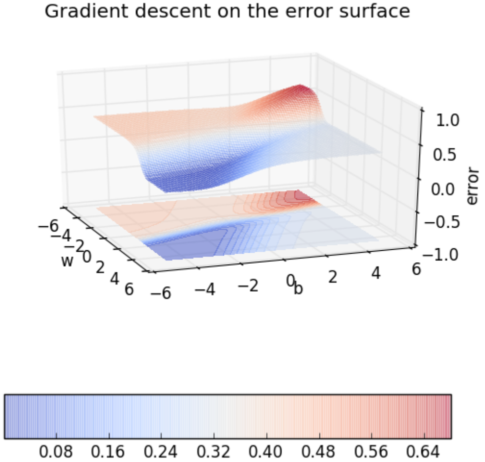

b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

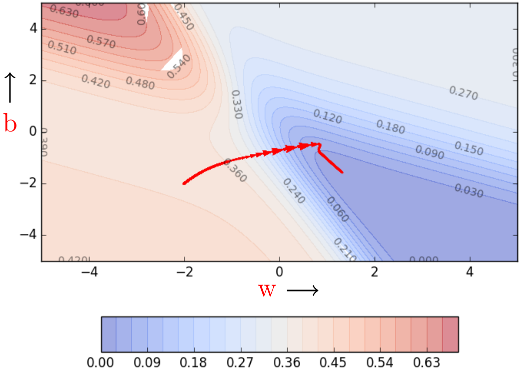

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

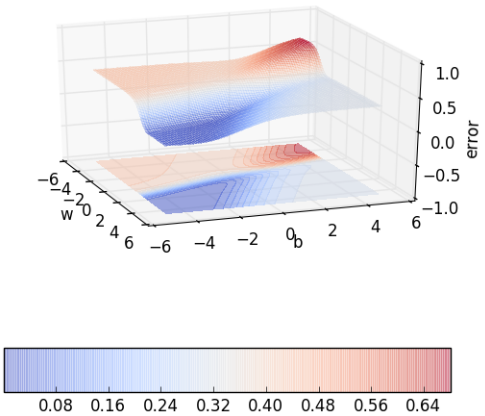

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

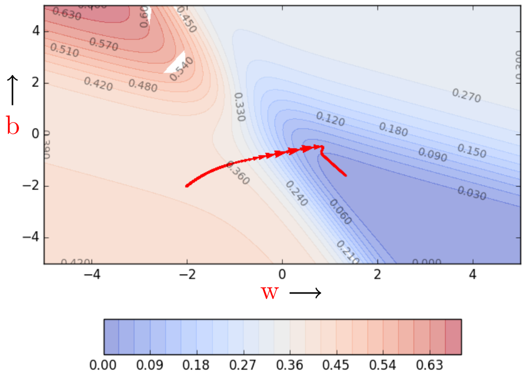

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

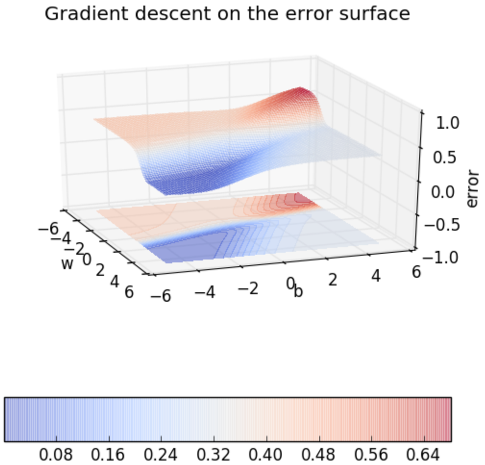


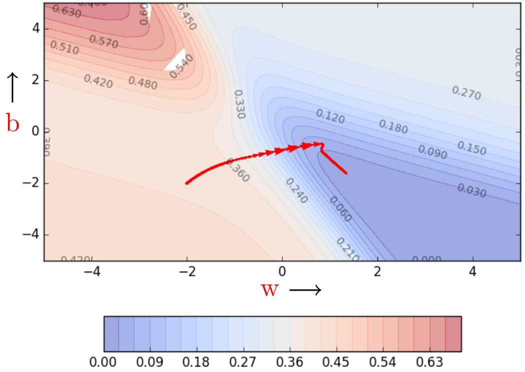
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

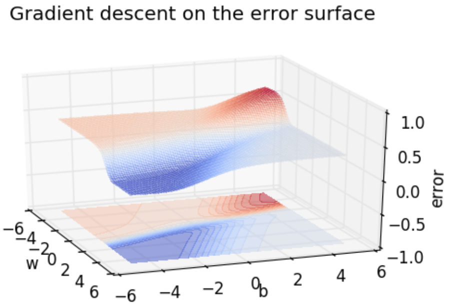


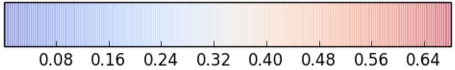
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

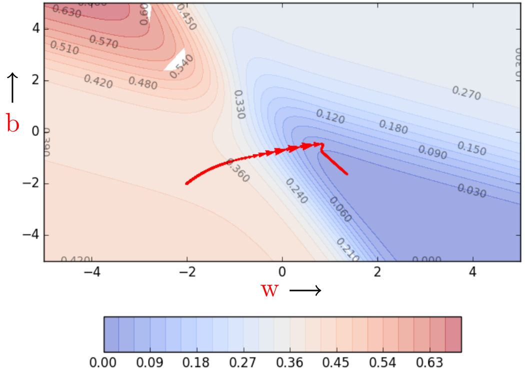

b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63


Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

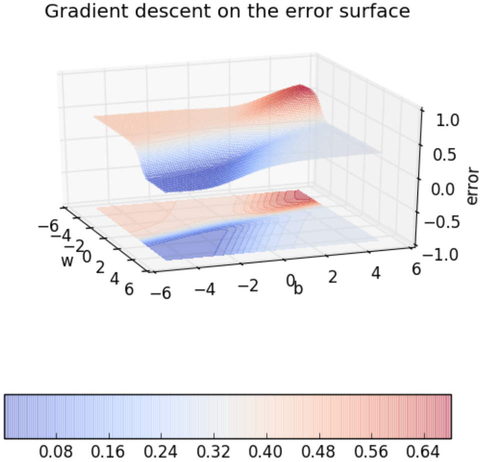

b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.000
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63


Gradient descent on the error surface
error
w
b

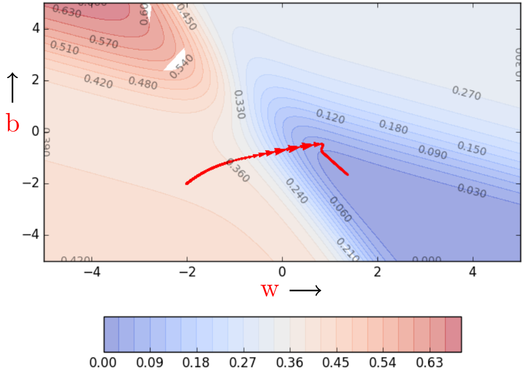

0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



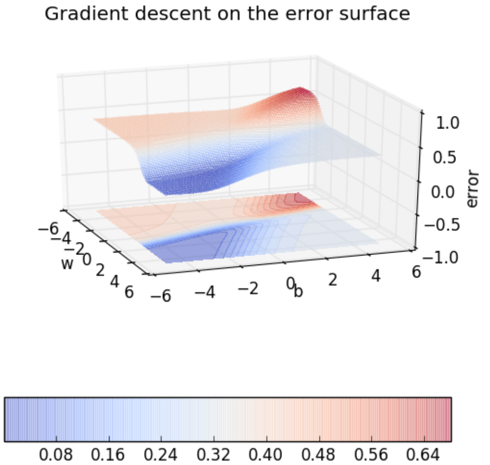
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



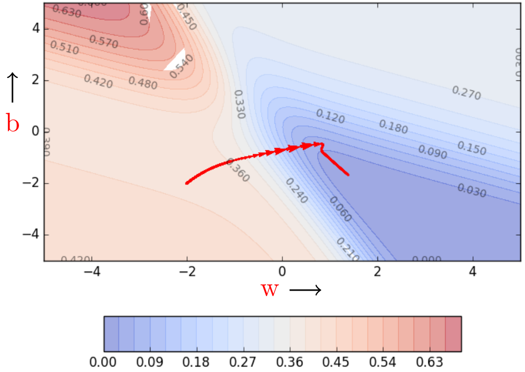
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64



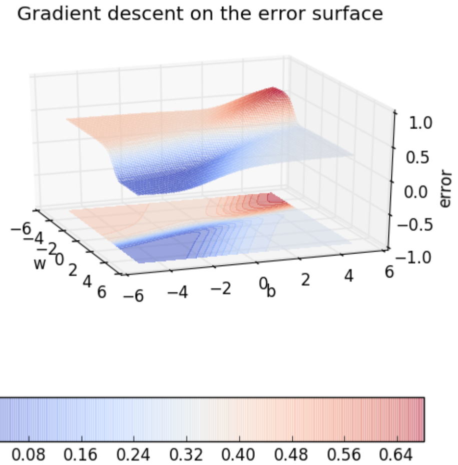
b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



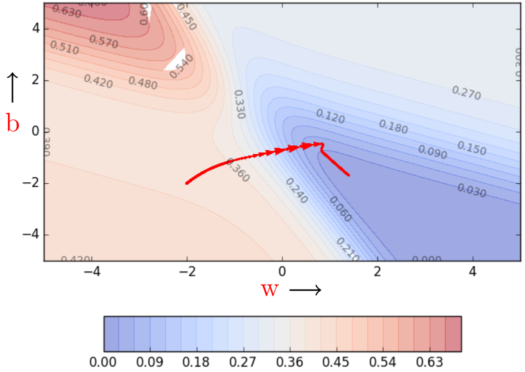
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

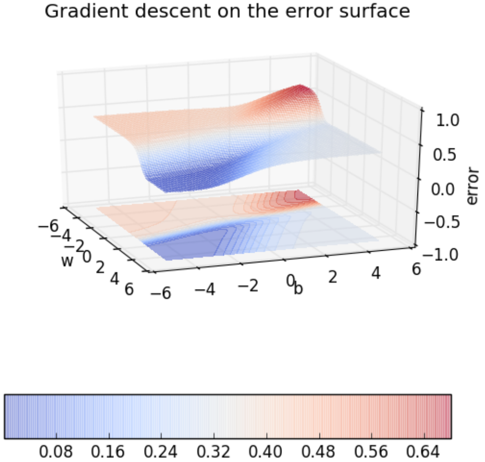


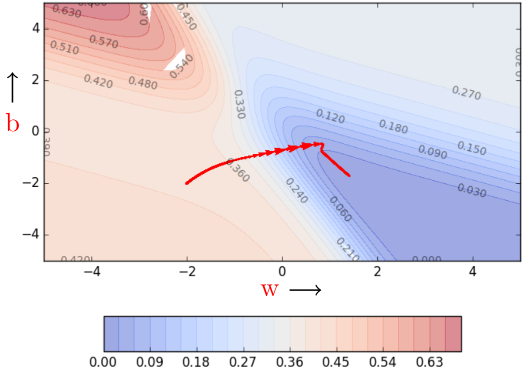
b
w
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63



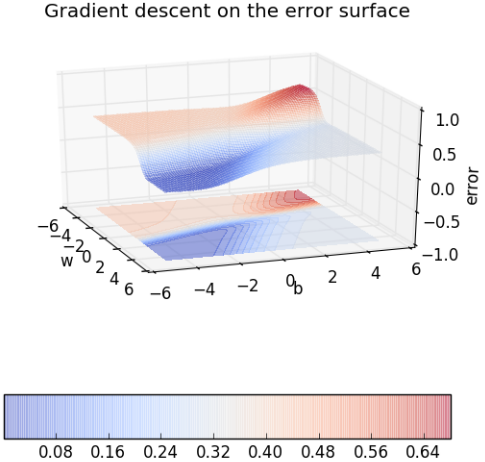
Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

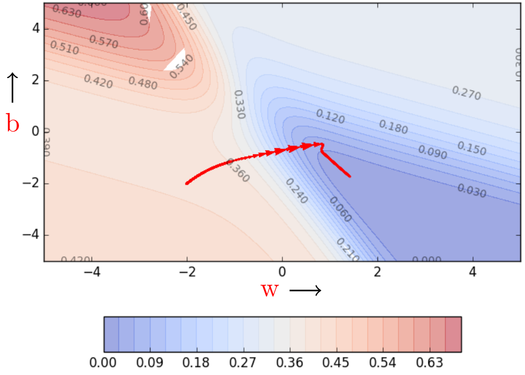

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

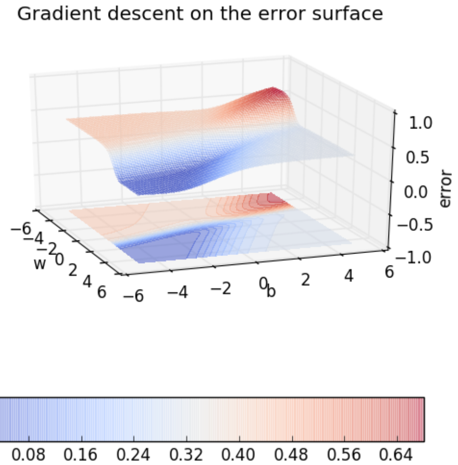


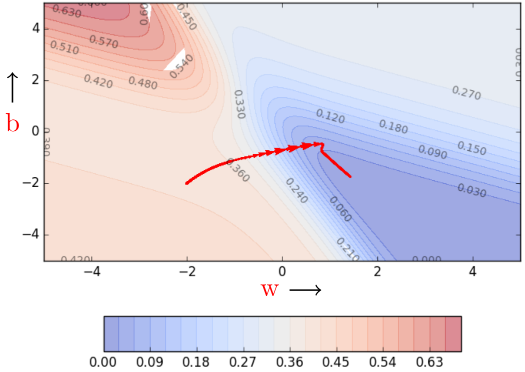
b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

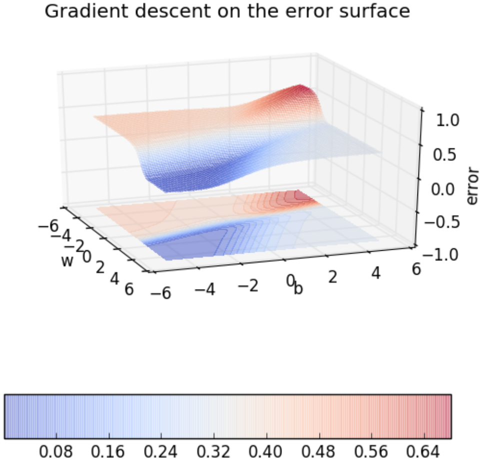


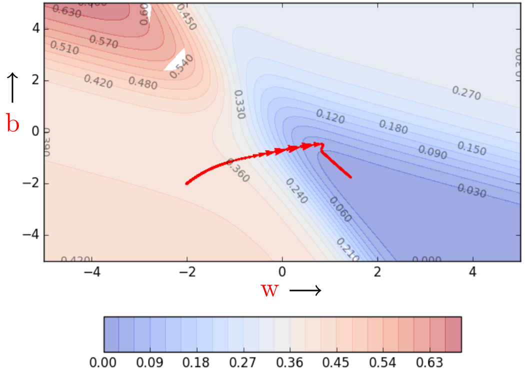
Gradient descent on the error surface
error
w
b
1.0
0.5
0.0
−0.5
−1.0
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.000
0.00
0.09
0.18
0.27
0.36
0.45
0.54
0.63

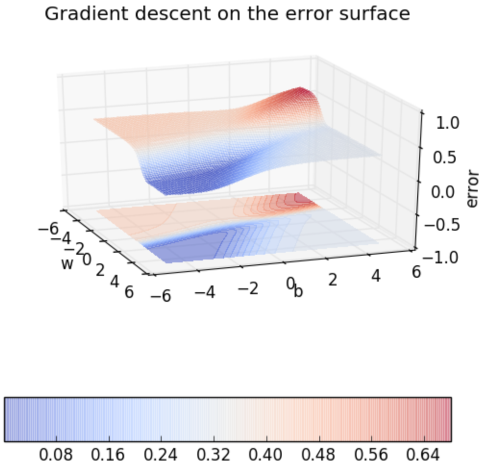

Gradient descent on the error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.270
0.120
0.180
0.090
0.150
0.360
0.240
0.060
0.030
0.210
0.000
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63

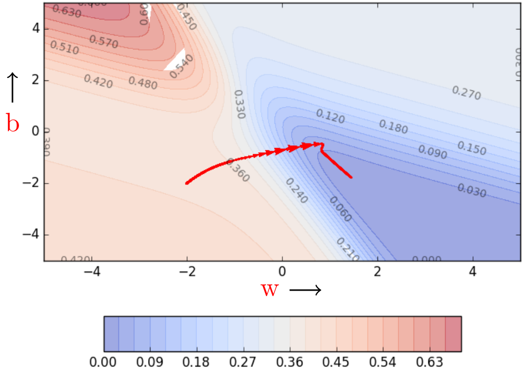

Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



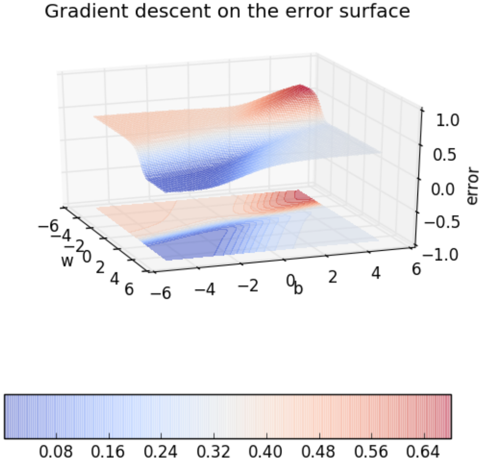
b
w
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



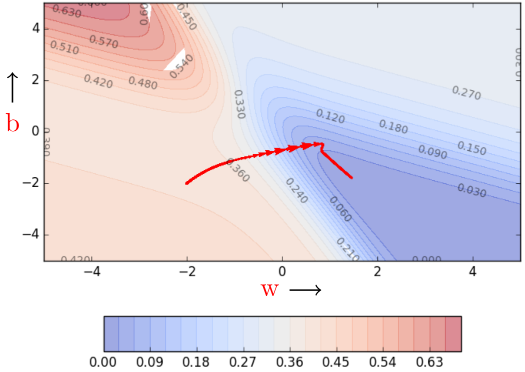
Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

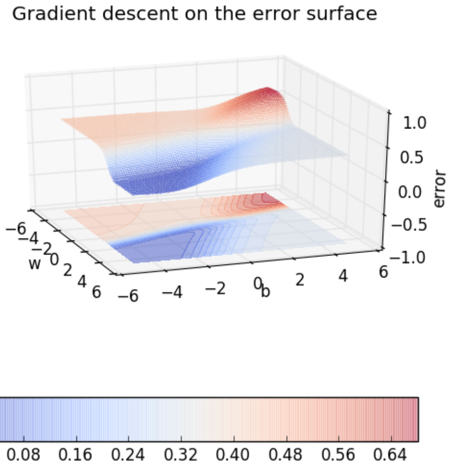


b
w
0.630
0.570
0.510
0.420
0.480
0.540
0.450
0.330
0.120
0.180
0.270
0.090
0.150
0.030
0.360
0.240
0.060
0.210
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63



Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64


b
w
0.630
0.570
0.510
0.420
0.480
0.450
0.540
0.330
0.360
0.270
0.120
0.180
0.090
0.150
0.030
0.240
0.060
0.210
0.000
0.420
0.00 0.09 0.18 0.27 0.36 0.45 0.54 0.63


Gradient descent on the error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64

Module 5.4 : Momentum based Gradient Descent

Some observations about gradient descent

- It takes a lot of time to navigate regions having a gentle slope

Some observations about gradient descent

- It takes a lot of time to navigate regions having a gentle slope
- This is because the gradient in these regions is very small

Some observations about gradient descent

- It takes a lot of time to navigate regions having a gentle slope
- This is because the gradient in these regions is very small
- Can we do something better ?

Some observations about gradient descent

- It takes a lot of time to navigate regions having a gentle slope
- This is because the gradient in these regions is very small
- Can we do something better ?
- Yes, let's take a look at 'Momentum based gradient descent'

Intuition

- If I am repeatedly being asked to move in the same direction then I should probably gain some confidence and start taking bigger steps in that direction

Intuition

- If I am repeatedly being asked to move in the same direction then I should probably gain some confidence and start taking bigger steps in that direction
- Just as a ball gains momentum while rolling down a slope

Intuition

- If I am repeatedly being asked to move in the same direction then I should probably gain some confidence and start taking bigger steps in that direction
- Just as a ball gains momentum while rolling down a slope

Update rule for momentum based gradient descent

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

Intuition

- If I am repeatedly being asked to move in the same direction then I should probably gain some confidence and start taking bigger steps in that direction
- Just as a ball gains momentum while rolling down a slope

Update rule for momentum based gradient descent

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

- In addition to the current update, also look at the history of updates.

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$update_0 = 0$$

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$update_0 = 0$$
$$update_1 = \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1$$

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$update_0 = 0$$
$$update_1 = \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1$$
$$update_2 = \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2$$

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$update_0 = 0$$
$$update_1 = \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1$$
$$update_2 = \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2$$
$$update_3 = \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3$$

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$
\begin{aligned}
update_0 &= 0 \\
update_1 &= \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1 \\
update_2 &= \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2 \\
update_3 &= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\
&= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3
\end{aligned}
$$

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$\begin{aligned}
update_0 &= 0 \\
update_1 &= \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1 \\
update_2 &= \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2 \\
update_3 &= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\
&= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3 \\
update_4 &= \gamma \cdot update_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \cdot \eta \nabla w_2 + \gamma \cdot \eta \nabla w_3 + \eta \nabla w_4
\end{aligned}$$

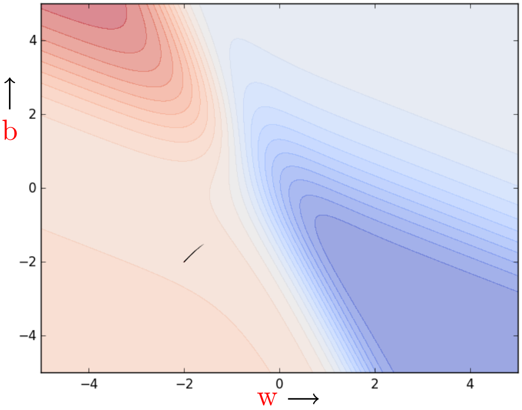
$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$\begin{aligned}
update_0 &= 0 \\
update_1 &= \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1 \\
update_2 &= \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2 \\
update_3 &= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\
&= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3 \\
update_4 &= \gamma \cdot update_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \cdot \eta \nabla w_2 + \gamma \cdot \eta \nabla w_3 + \eta \nabla w_4 \\
&\vdots
\end{aligned}$$

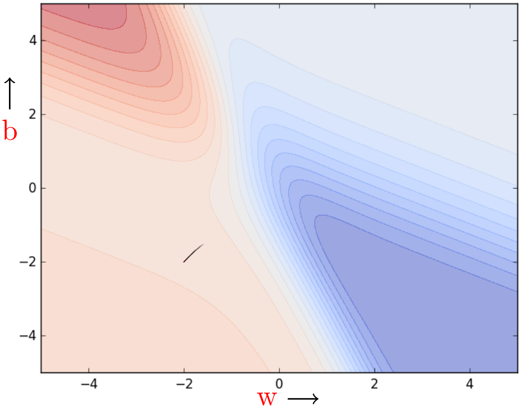
$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$\begin{aligned}
update_0 &= 0 \\
update_1 &= \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1 \\
update_2 &= \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2 \\
update_3 &= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\
&= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3 \\
update_4 &= \gamma \cdot update_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \cdot \eta \nabla w_2 + \gamma \cdot \eta \nabla w_3 + \eta \nabla w_4 \\
&\vdots \\
update_t &= \gamma \cdot update_{t-1} + \eta \nabla w_t = \gamma^{t-1} \cdot \eta \nabla w_1 + \gamma^{t-2} \cdot \eta \nabla w_1 + \ldots + \eta \nabla w_t
\end{aligned}$$

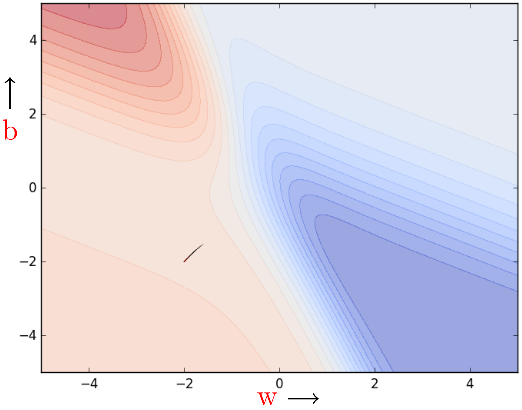
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



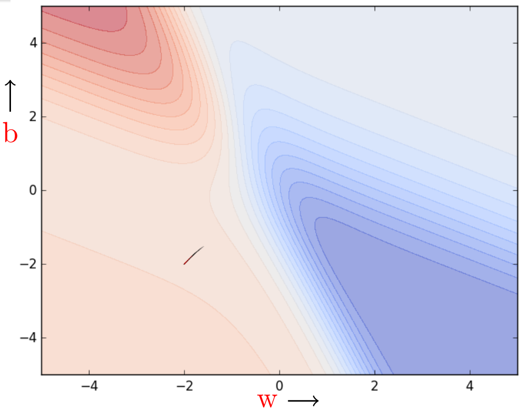
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



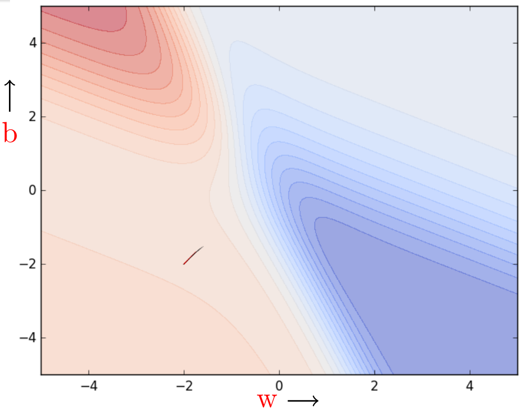
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



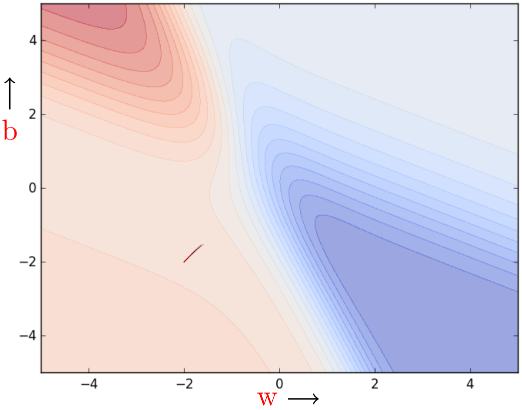
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



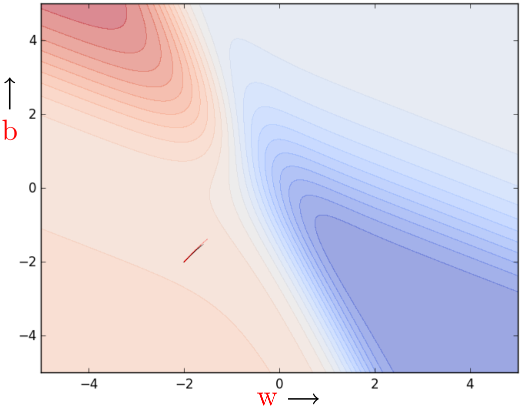
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



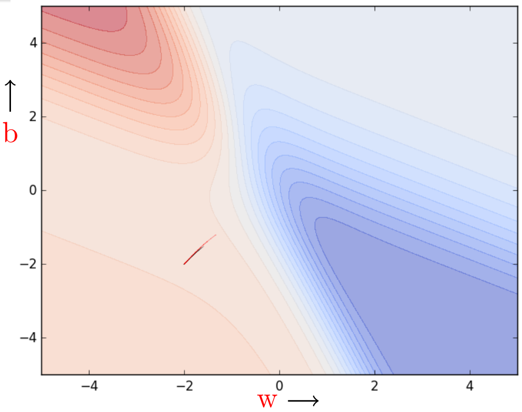
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



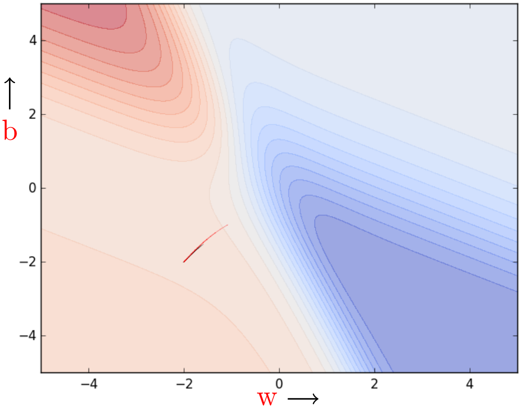
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



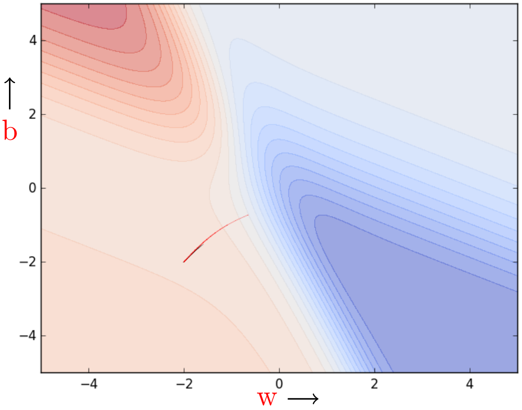
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



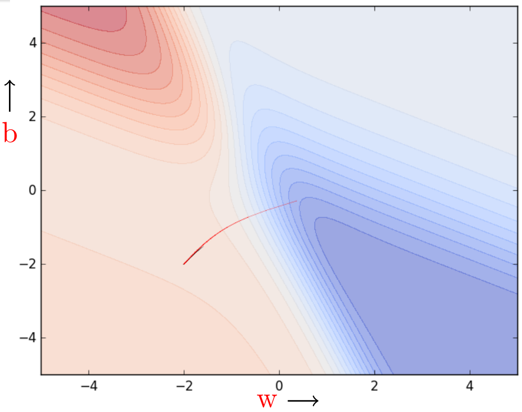
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



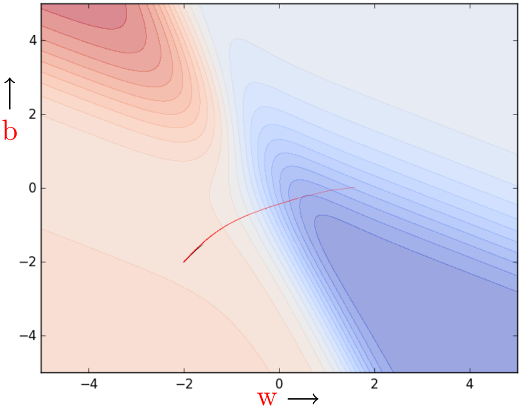
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



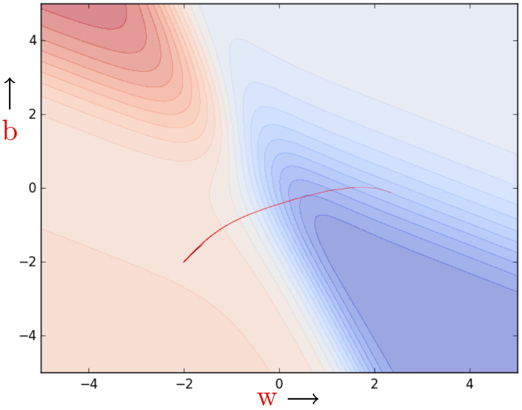
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



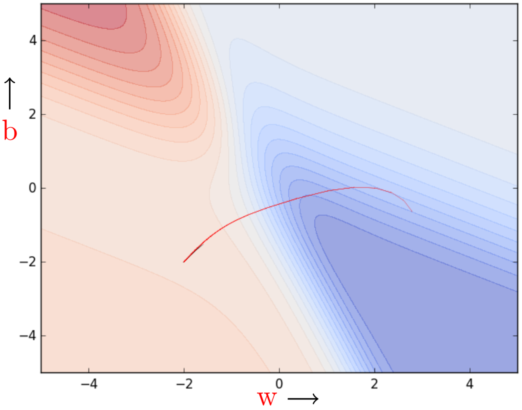
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



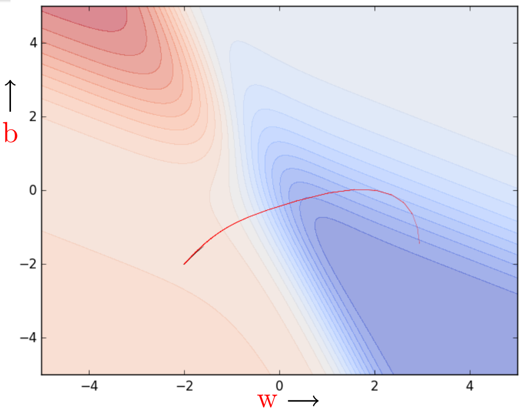
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



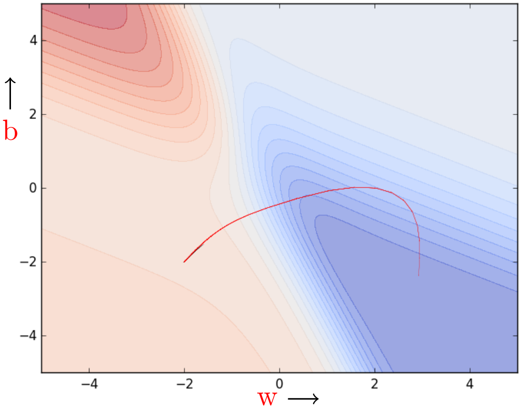
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



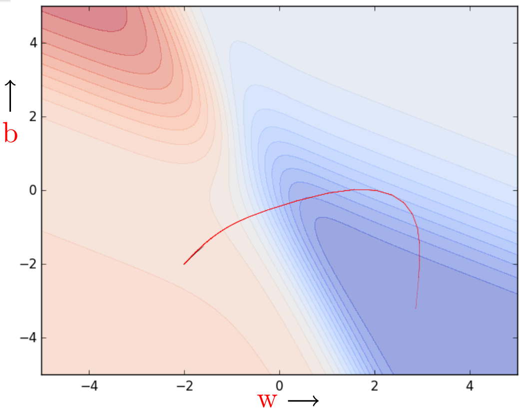
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



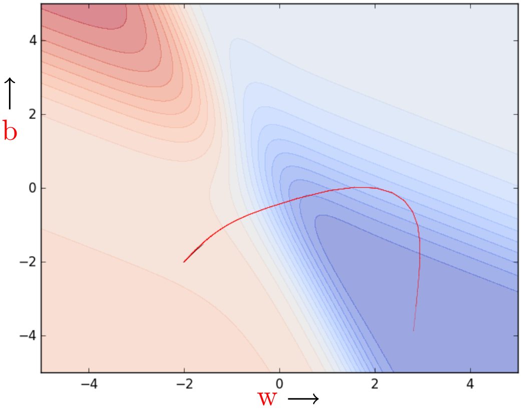
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



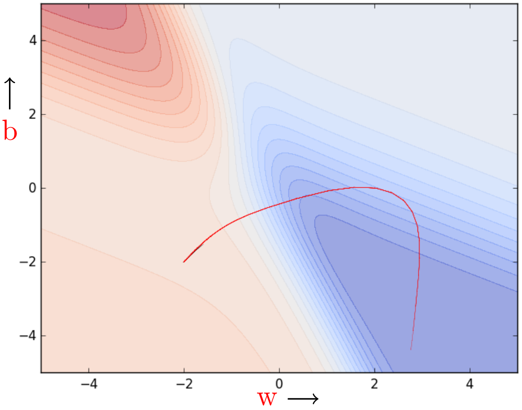
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



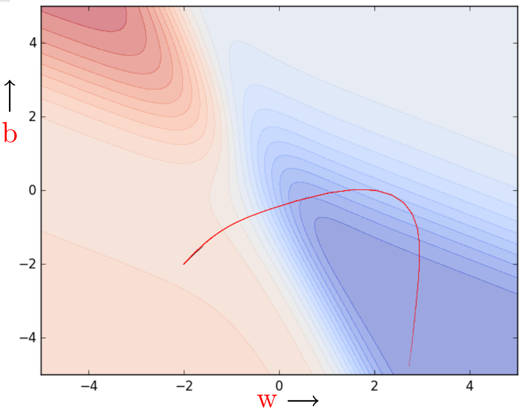
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



Some observations and questions

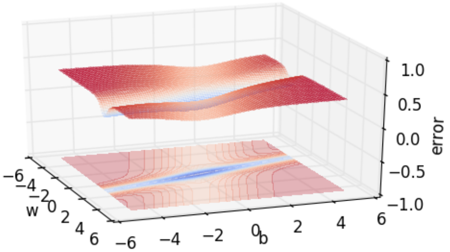
- Even in the regions having gentle slopes, momentum based gradient descent is able to take large steps because the momentum carries it along

Some observations and questions

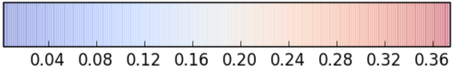
- Even in the regions having gentle slopes, momentum based gradient descent is able to take large steps because the momentum carries it along
- Is moving fast always good? Would there be a situation where momentum would cause us to run pass our goal?

Some observations and questions

- Even in the regions having gentle slopes, momentum based gradient descent is able to take large steps because the momentum carries it along
- Is moving fast always good? Would there be a situation where momentum would cause us to run pass our goal?
- Let us change our input data so that we end up with a different error surface and then see what happens ...

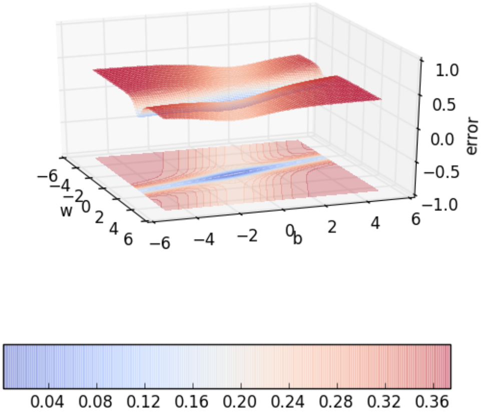


1.0
0.5
0.0
−0.5
−1.0
error
w
b
−6
−4
−2
0
2
4
6

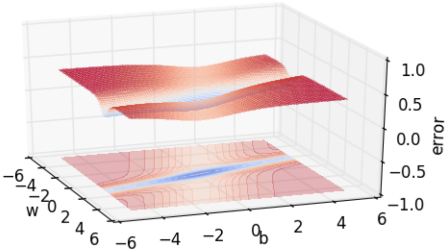


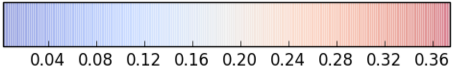
0.04 0.08 0.12 0.16 0.20 0.24 0.28 0.32 0.36

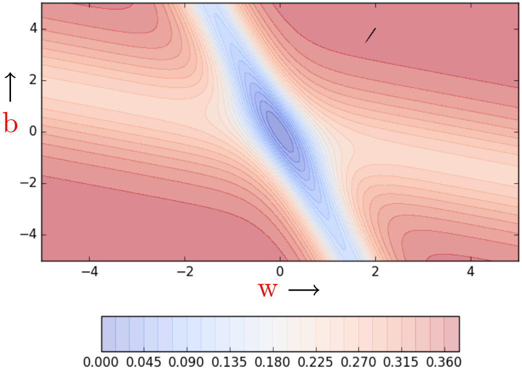
- In this case, the error is high on either side of the minima valley



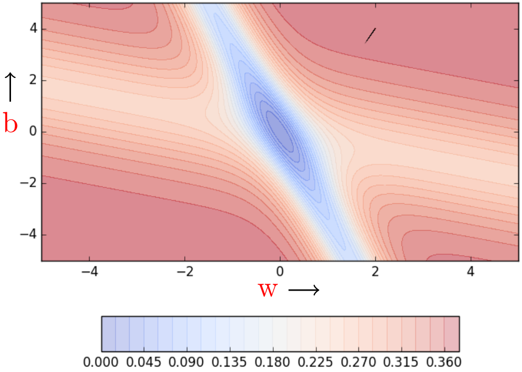
- In this case, the error is high on either side of the minima valley
- Could momentum be detrimental in such cases... let's see....

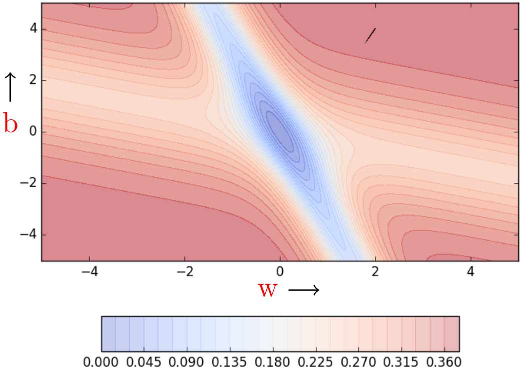




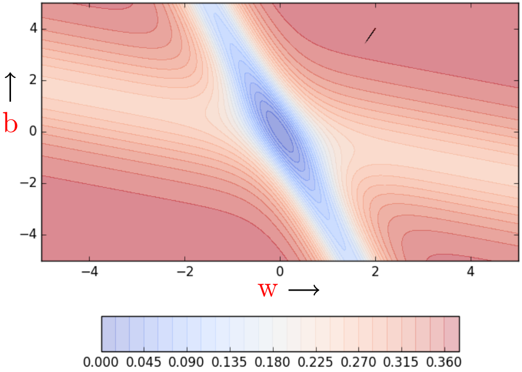


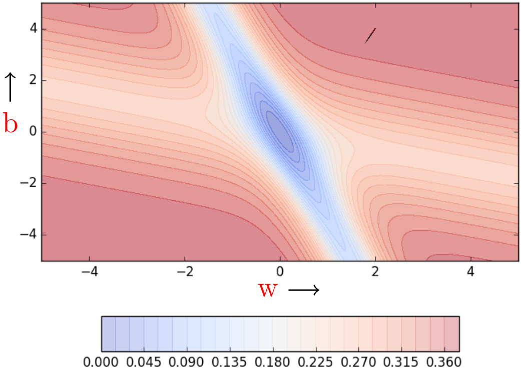
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360

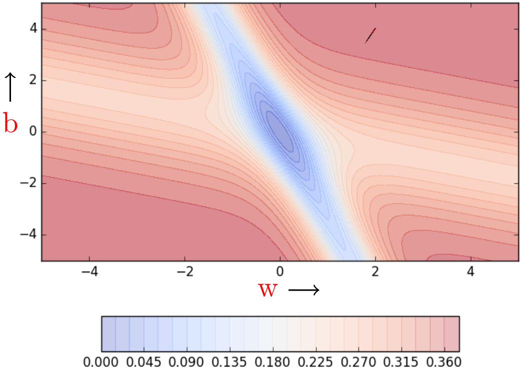


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

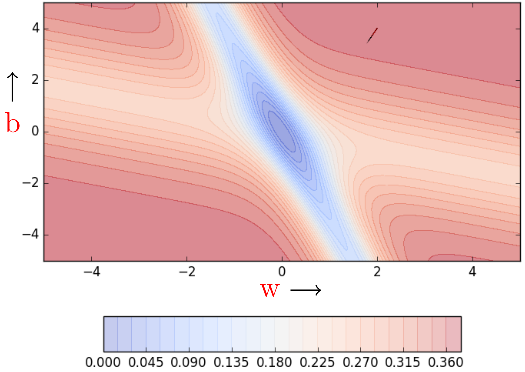

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360

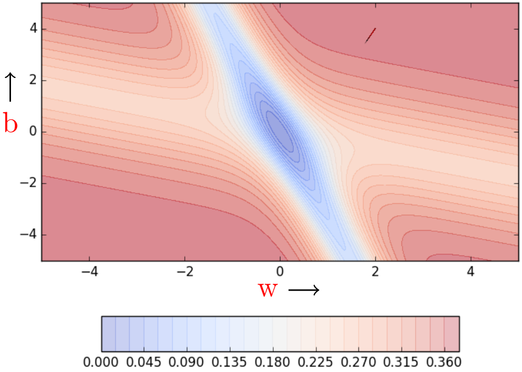


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

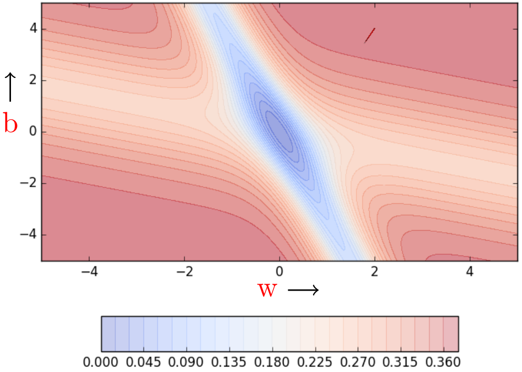


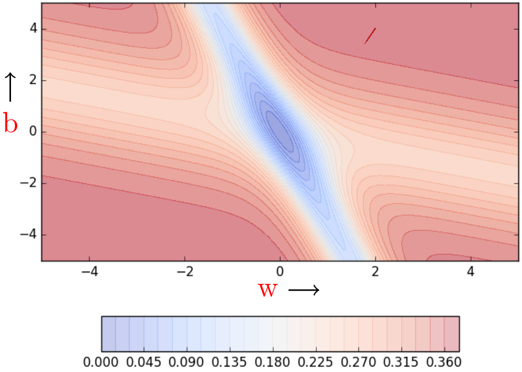
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

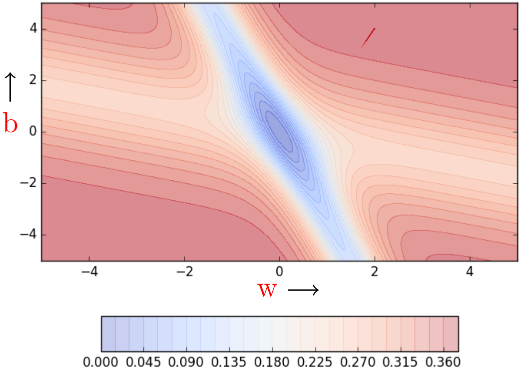

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

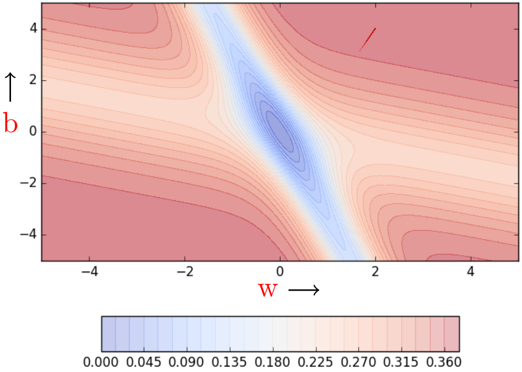


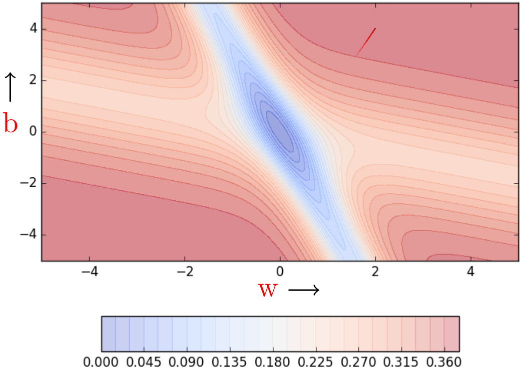
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360

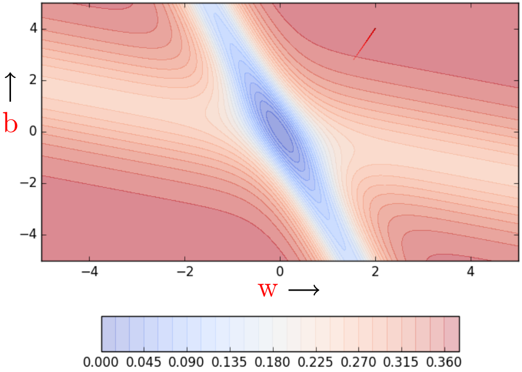

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

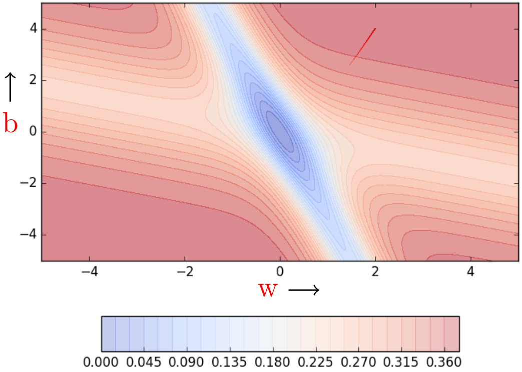

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

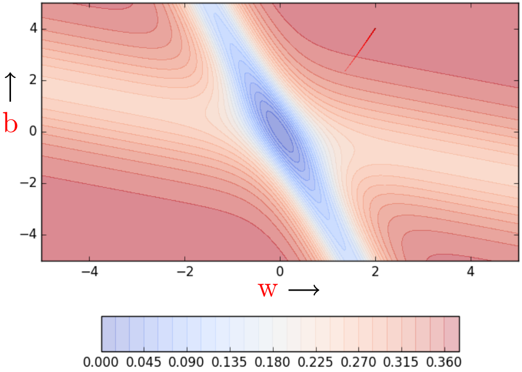

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360

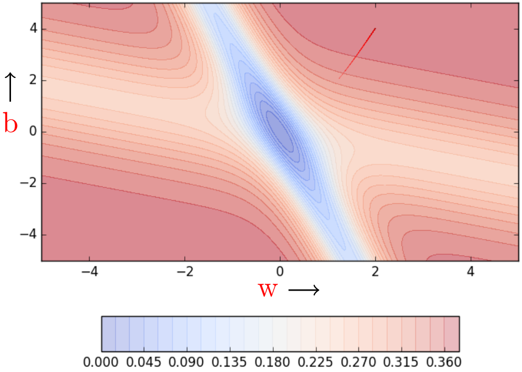


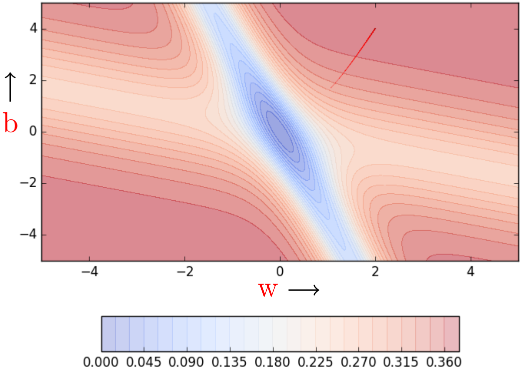
4
2
0
−2
−4
−4
−2
0
2
4
b
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360

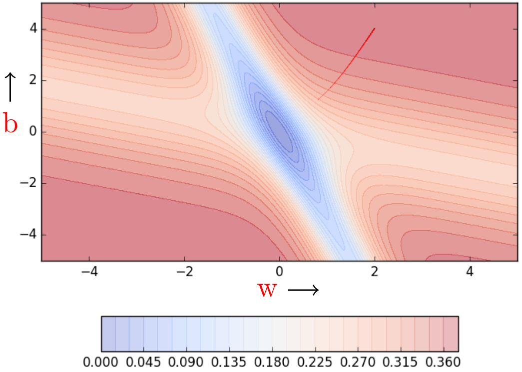

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

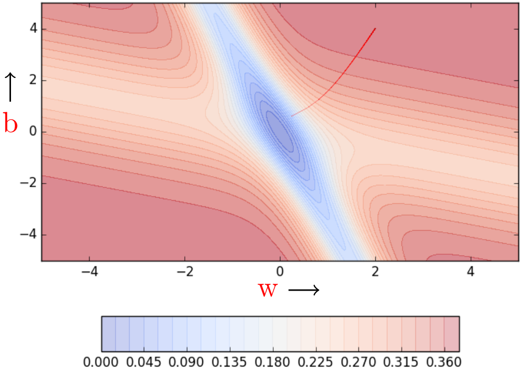

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

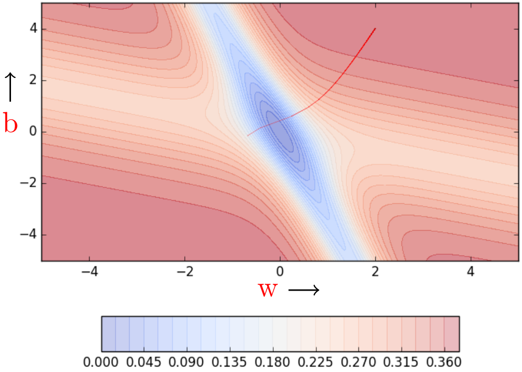

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

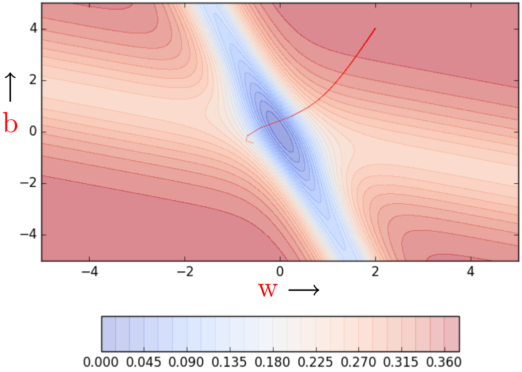

b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

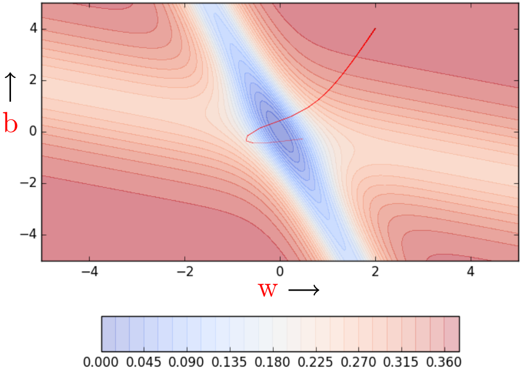

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

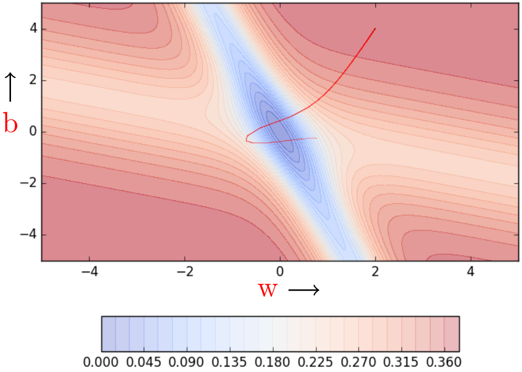

b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

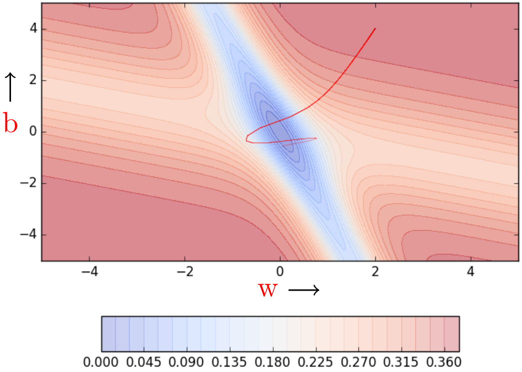

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360



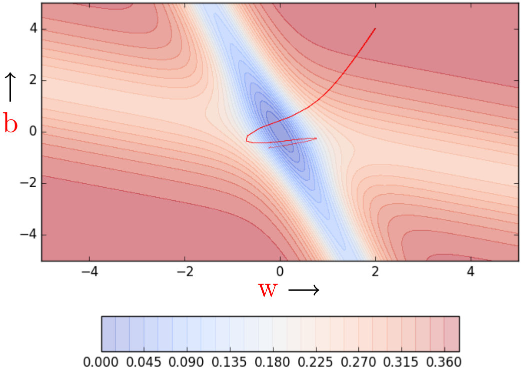
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360



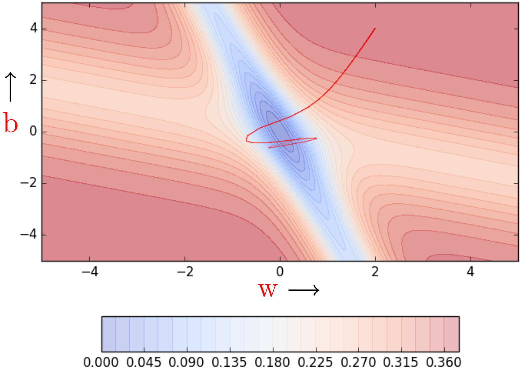
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

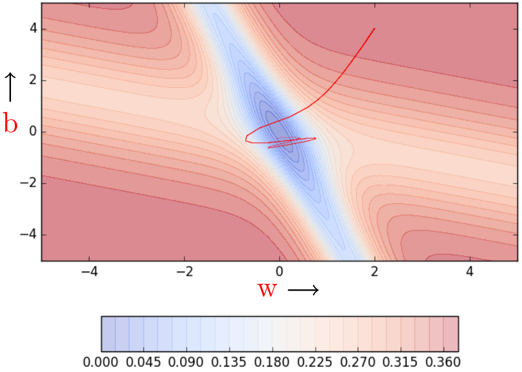


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

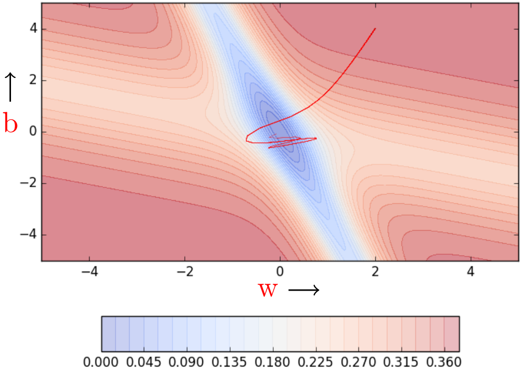


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360

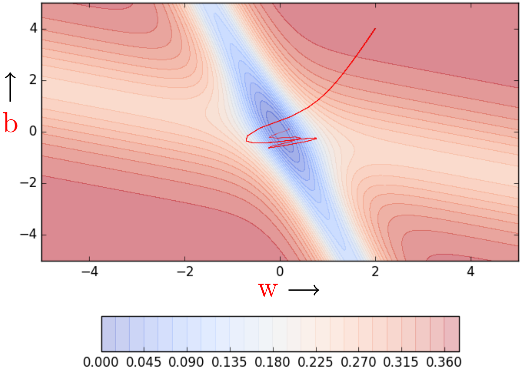

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360



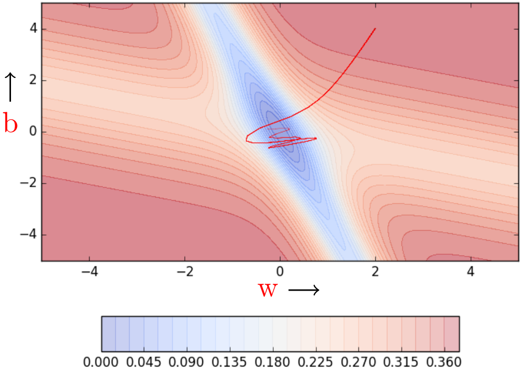
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

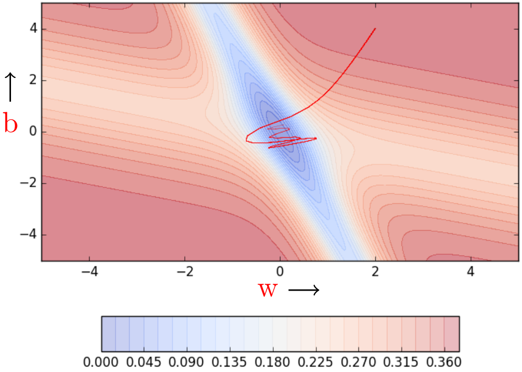


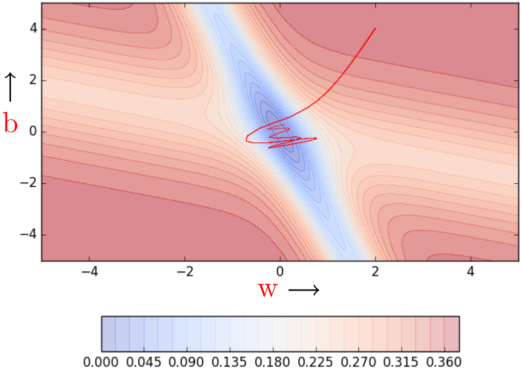
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360



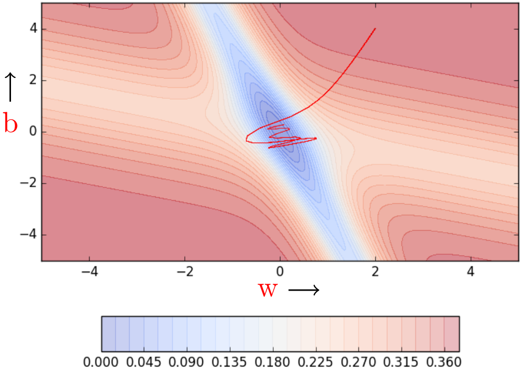
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000
0.045
0.090
0.135
0.180
0.225
0.270
0.315
0.360

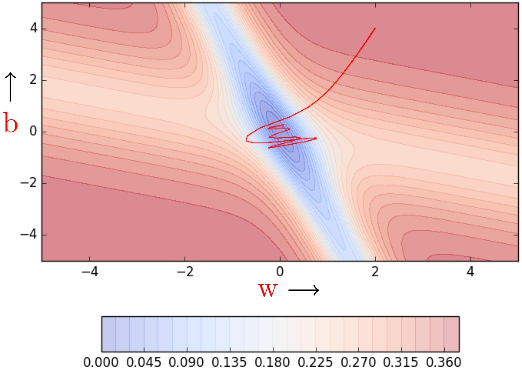

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

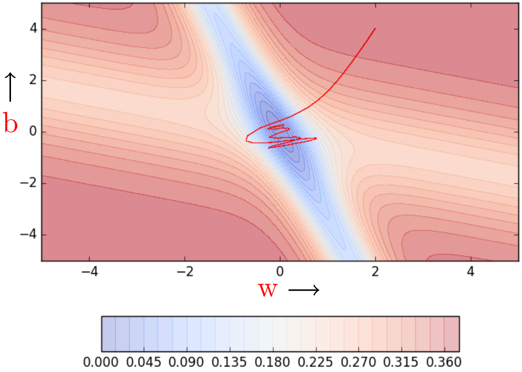


b
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

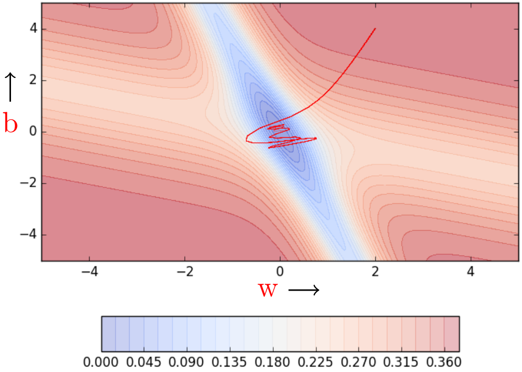


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

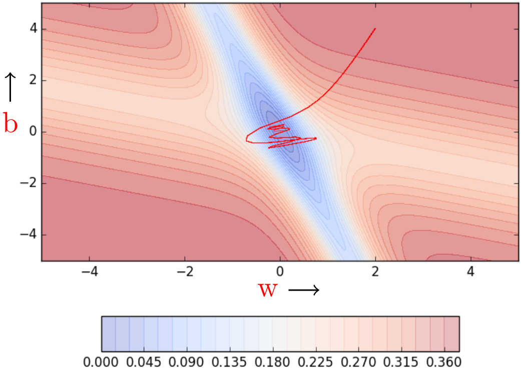

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.000 0.045 0.090 0.135 0.180 0.225 0.270 0.315 0.360

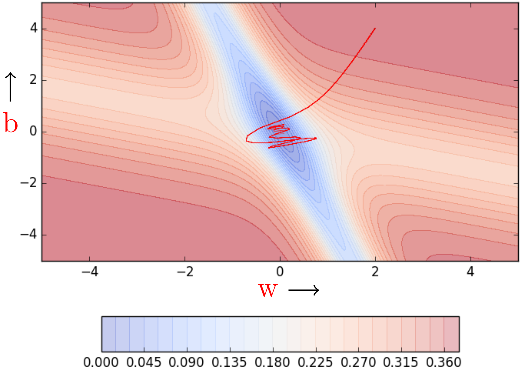
- Momentum based gradient descent oscillates in and out of the minima valley as the momentum carries it out of the valley



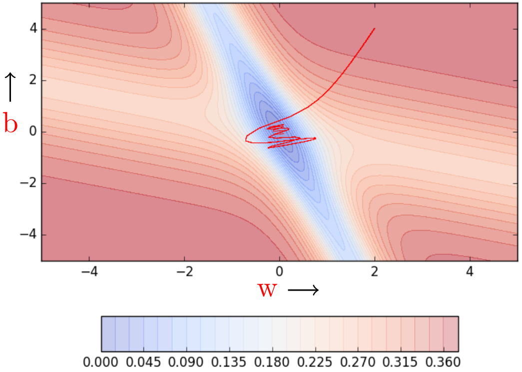
- Momentum based gradient descent oscillates in and out of the minima valley as the momentum carries it out of the valley
- Takes a lot of u-turns before finally converging



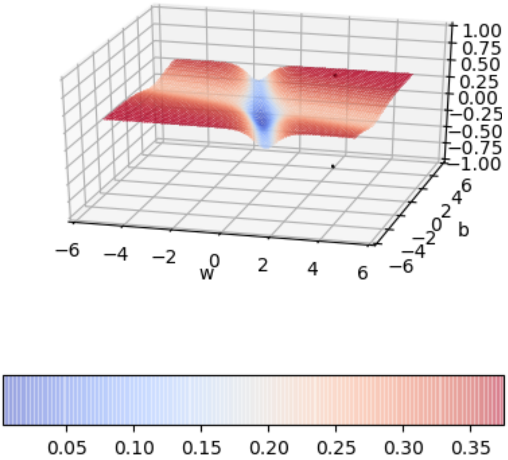
- Momentum based gradient descent oscillates in and out of the minima valley as the momentum carries it out of the valley
- Takes a lot of u-turns before finally converging
- Despite these u-turns it still converges faster than vanilla gradient descent

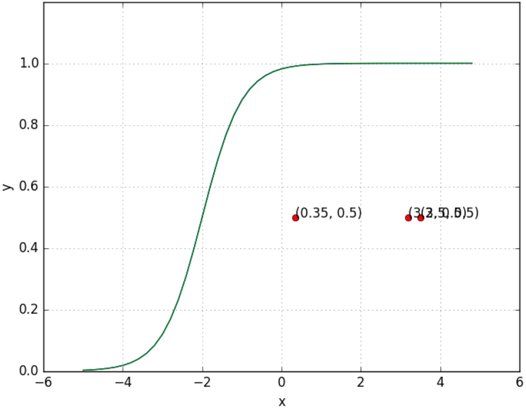


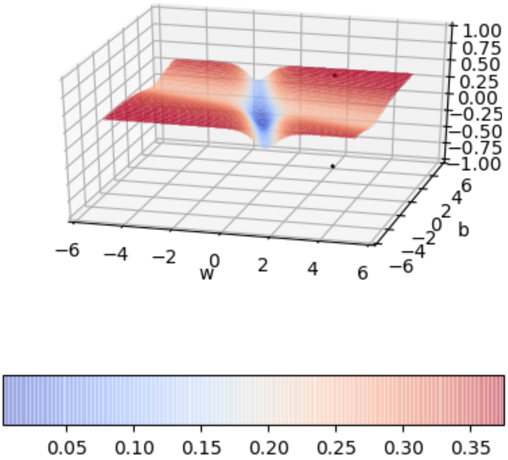
- Momentum based gradient descent oscillates in and out of the minima valley as the momentum carries it out of the valley
- Takes a lot of u-turns before finally converging
- Despite these u-turns it still converges faster than vanilla gradient descent
- After 100 iterations momentum based method has reached an error of 0.00001 whereas vanilla gradient descent is still stuck at an error of 0.36



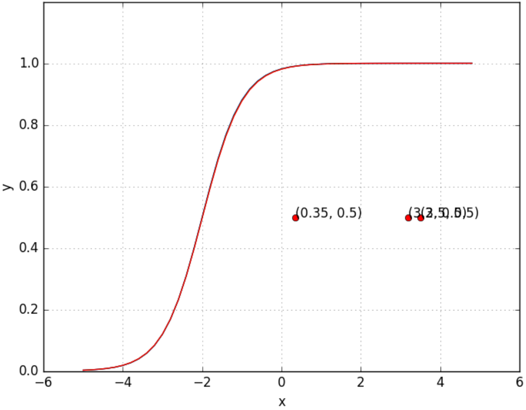
Let's look at a 3d visualization and a different geometric perspective of the same thing...


w
b
−6 −4 −2 0 2 4 6
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
0.05 0.10 0.15 0.20 0.25 0.30 0.35

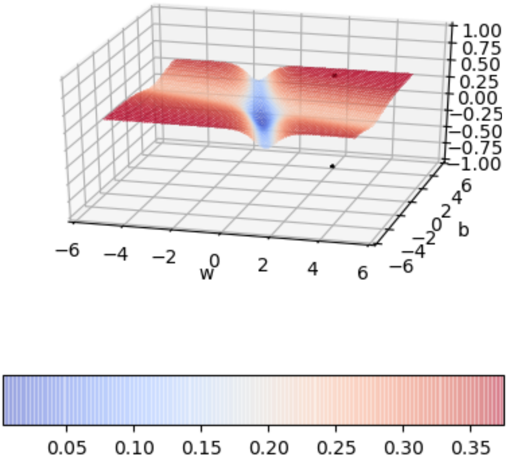

y
x
(0.35, 0.5)
0.0 0.2 0.4 0.6 0.8 1.0
−6 −4 −2 0 2 4 6

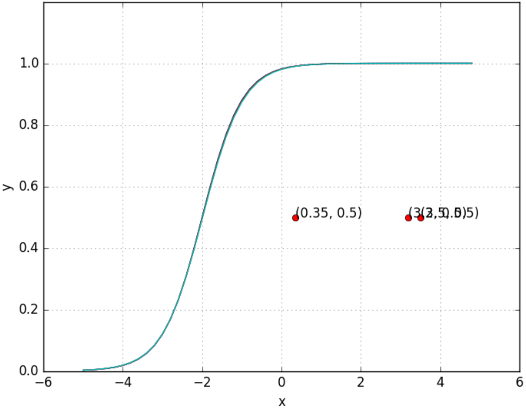


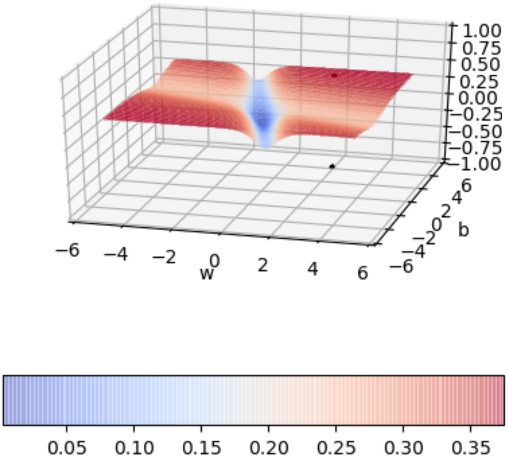
w
b
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
0.05 0.10 0.15 0.20 0.25 0.30 0.35



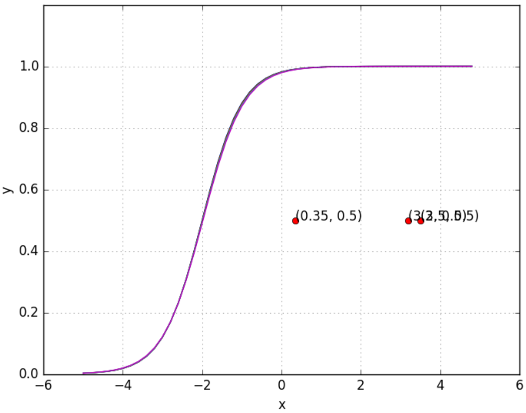
(0.35, 0.5)
y
x
0.0 0.2 0.4 0.6 0.8 1.0
−6 −4 −2 0 2 4 6


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
−6
−4
−2
0
2
4
6
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

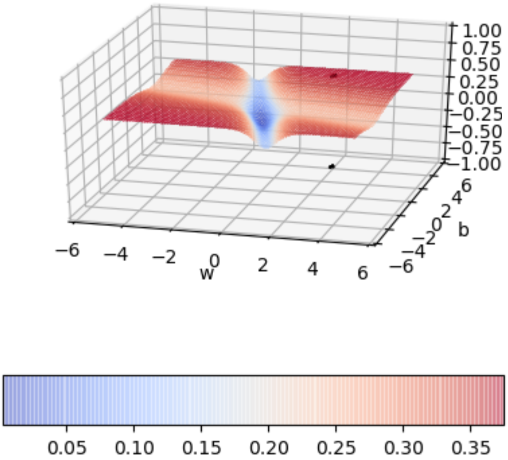

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x



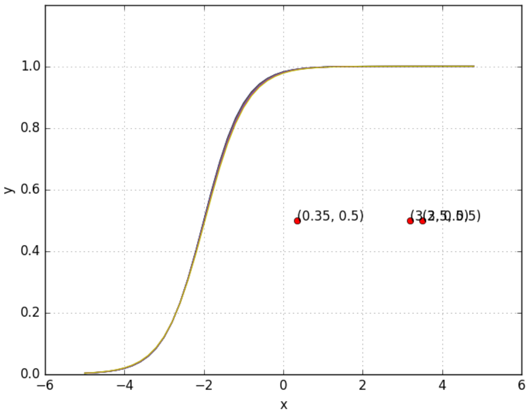
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
−6 −4 −2 0 2 4 6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35



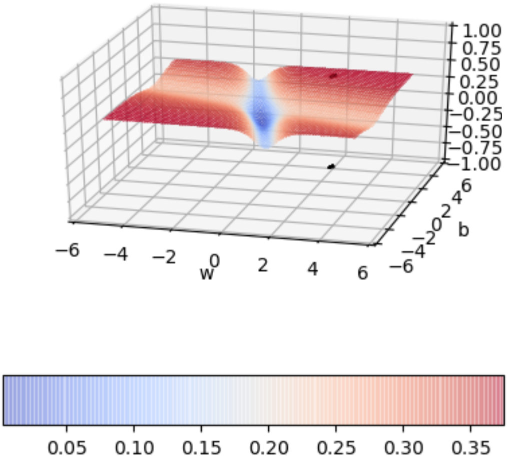
1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6 −4 −2 0 2 4 6
x



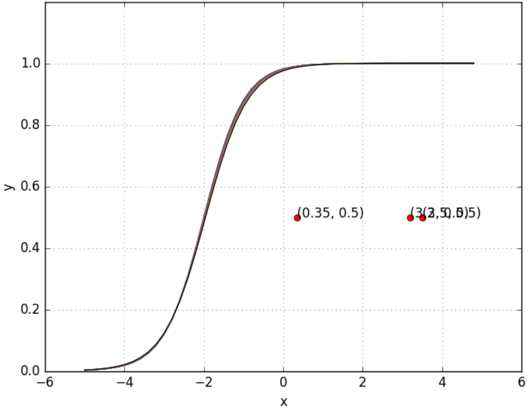
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
b
−6
−4
−2
0
2
4
6
w
0.05
0.10
0.15
0.20
0.25
0.30
0.35



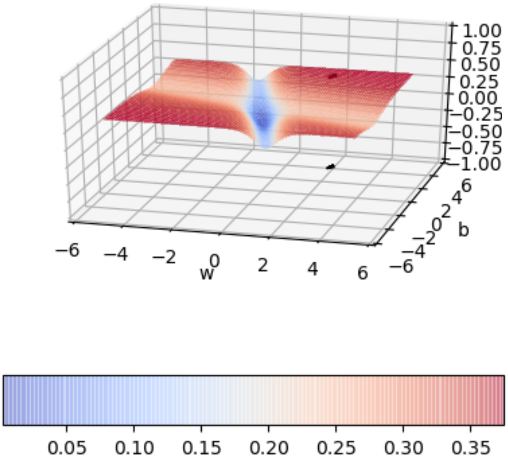
(0.35, 0.5)
y
x
0.0
0.2
0.4
0.6
0.8
1.0
−6
−4
−2
0
2
4
6

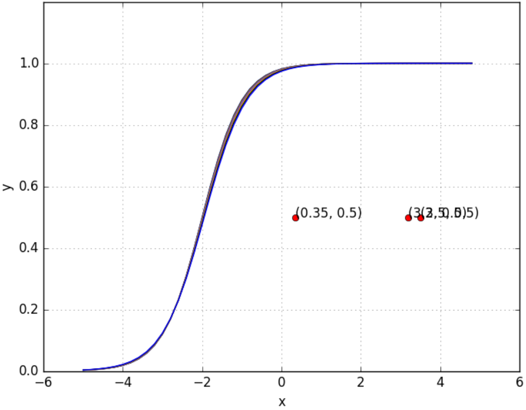


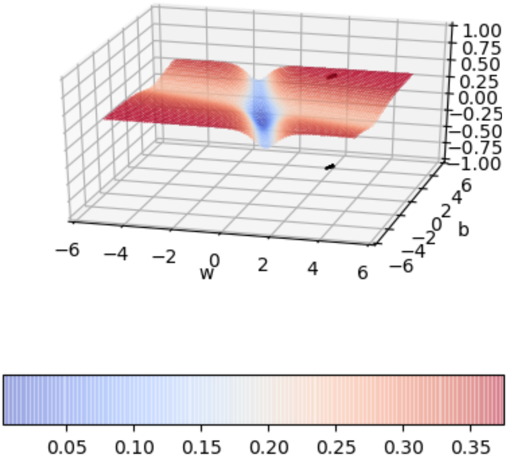
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
6 4 2 0 −2 −4 −6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35

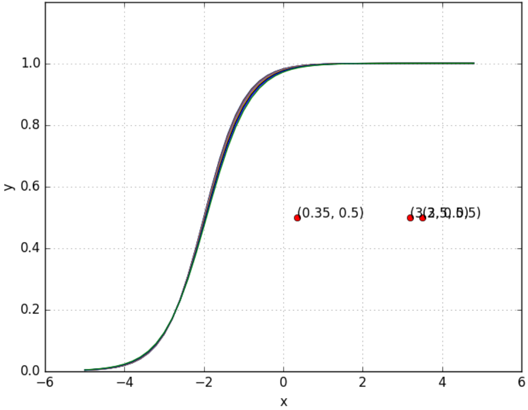


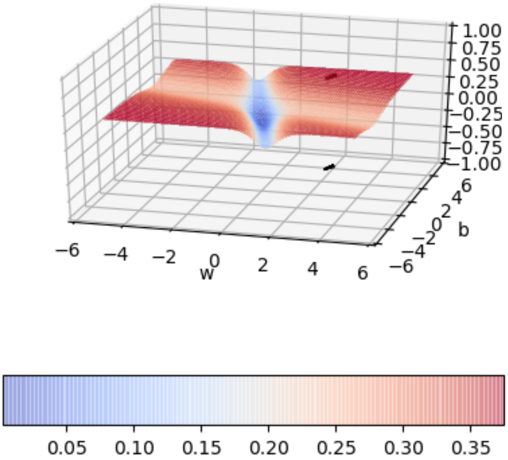
1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6 −4 −2 0 2 4 6
x

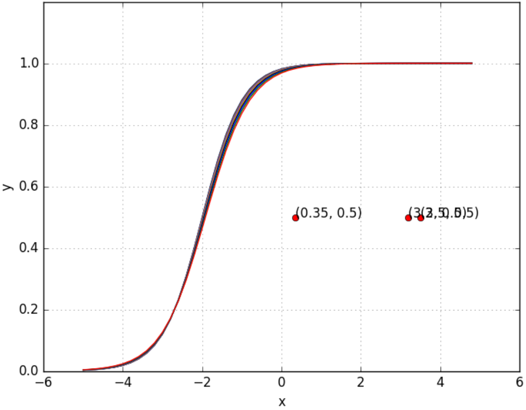

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

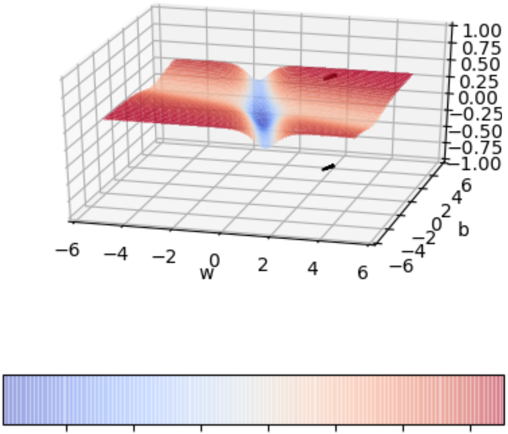

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x

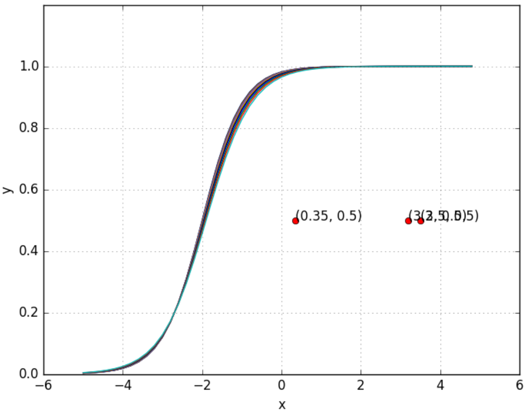

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

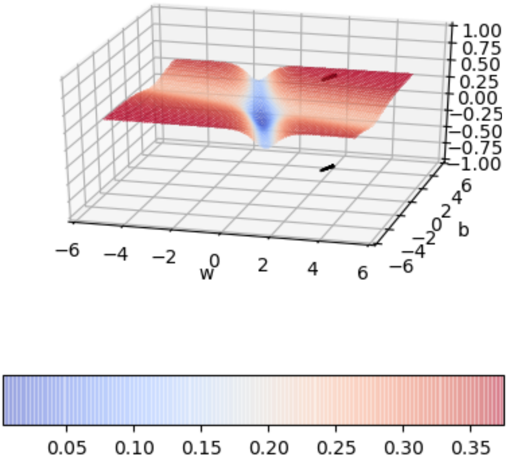

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x

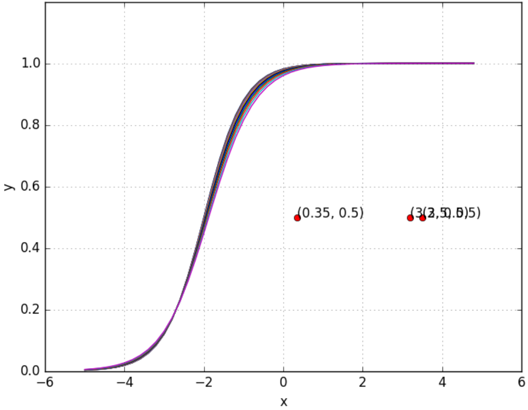

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
−6 −4 −2 0 2 4 6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35

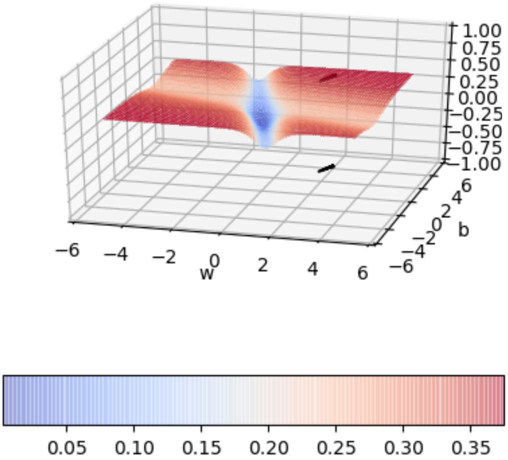

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6 −4 −2 0 2 4 6
x

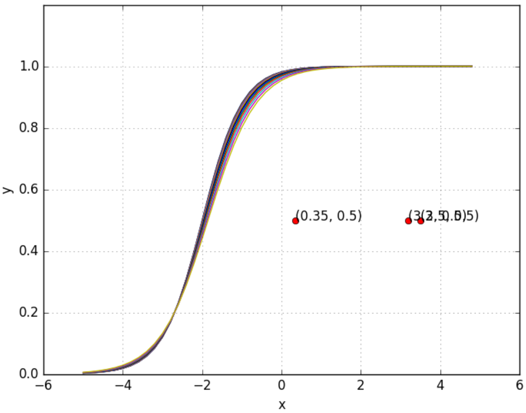

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

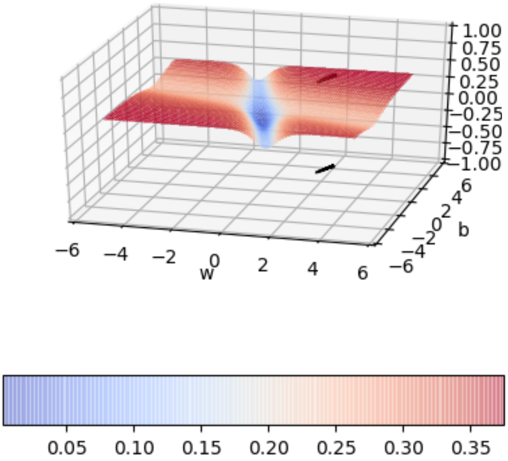

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x

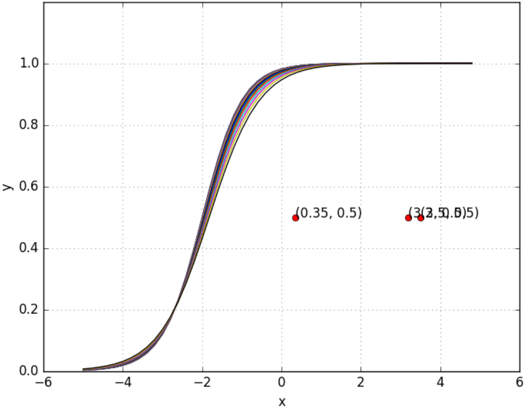

w
b
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.05 0.10 0.15 0.20 0.25 0.30 0.35

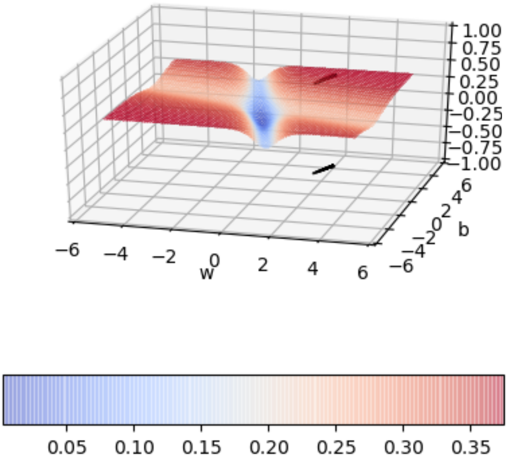

(0.35, 0.5)
x
y


w
b
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
0.05 0.10 0.15 0.20 0.25 0.30 0.35

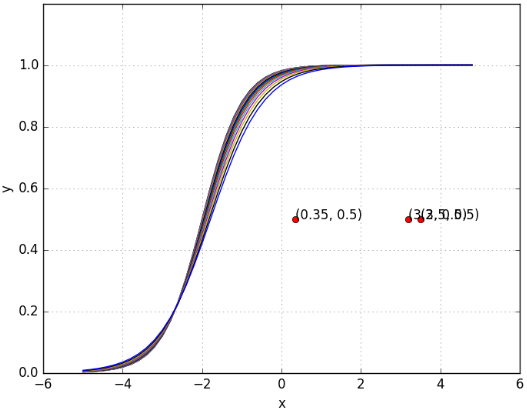

y
x
0.0 0.2 0.4 0.6 0.8 1.0
−6 −4 −2 0 2 4 6
(0.35, 0.5)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w
b
−6
−4
−2
0
2
4
6
0.05
0.10
0.15
0.20
0.25
0.30
0.35

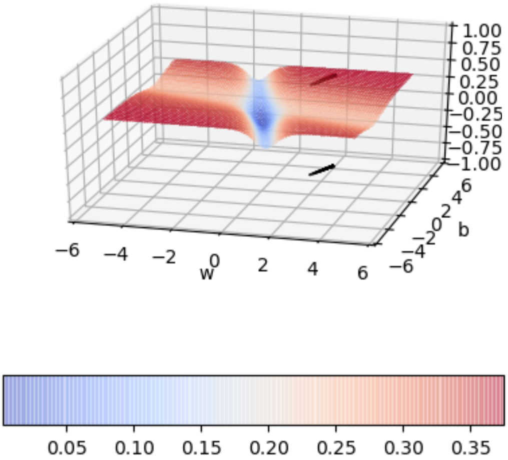

y
x
(0.35, 0.5)
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6

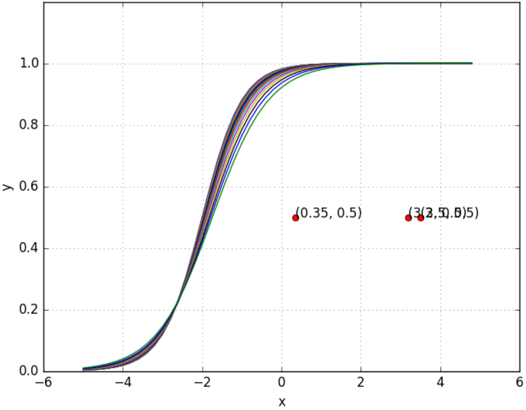


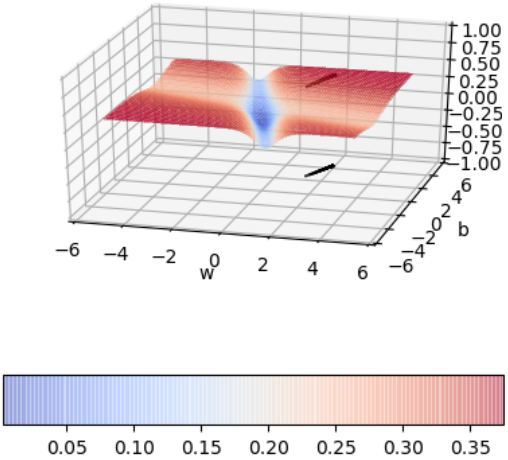
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

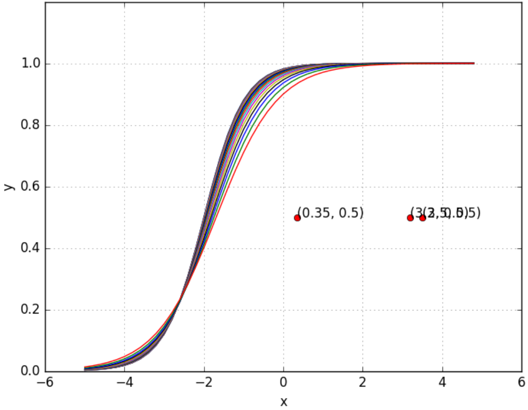


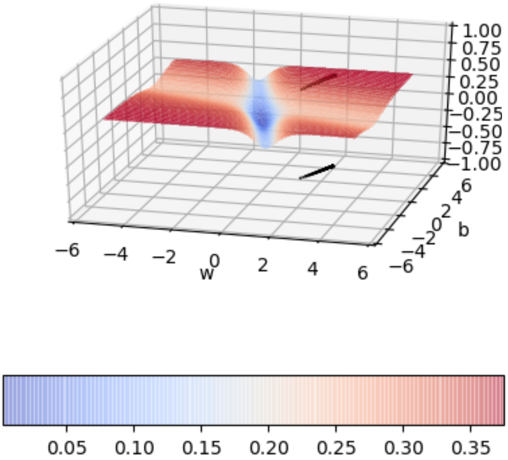
1.0
0.8
0.6
0.4
0.2
0.0
y
−6
−4
−2
0
2
4
6
x
(0.35, 0.5)

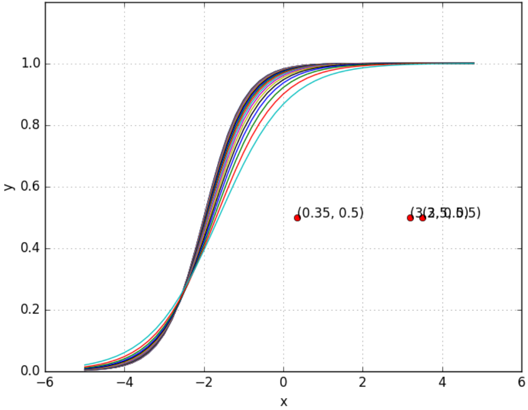

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

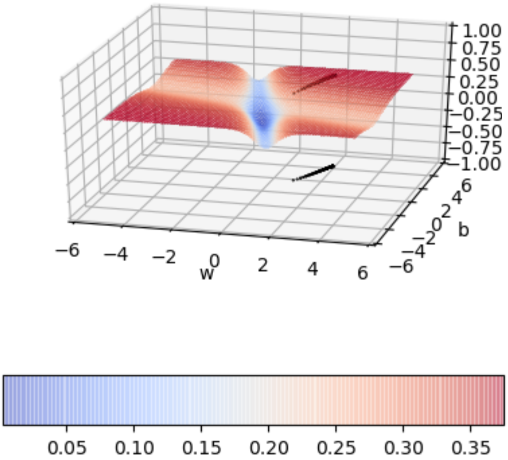

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x

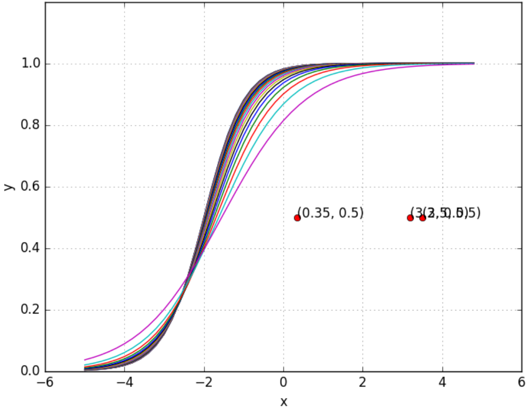

w
b
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
−6 −4 −2 0 2 4 6
0.05 0.10 0.15 0.20 0.25 0.30 0.35

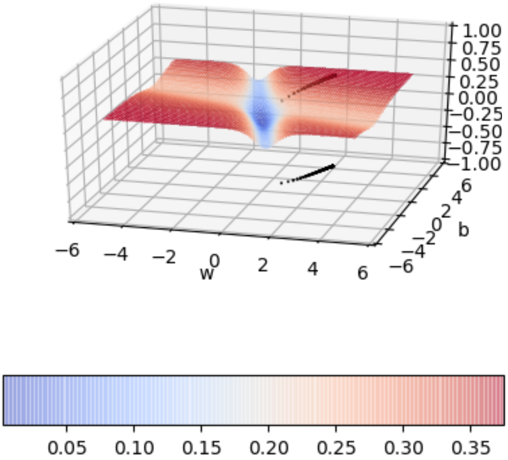

(0.35, 0.5)
x
y
0.0 0.2 0.4 0.6 0.8 1.0
−6 −4 −2 0 2 4 6

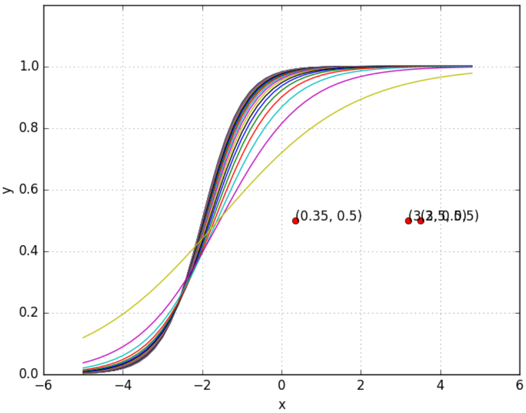

w
b
−6 −4 −2 0 2 4 6
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
0.05 0.10 0.15 0.20 0.25 0.30 0.35

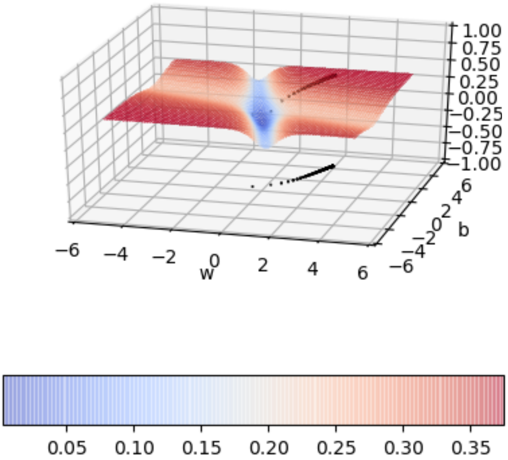

(0.35, 0.5)
y
x
0.0 0.2 0.4 0.6 0.8 1.0
−6 −4 −2 0 2 4 6

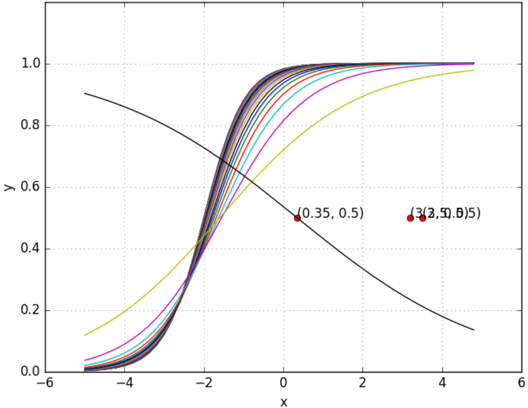

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

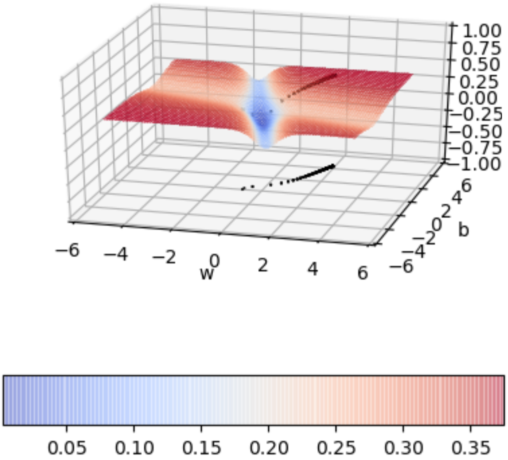

1.0
0.8
0.6
0.4
0.2
0.0
y
−6
−4
−2
0
2
4
6
x
(0.35, 0.5)

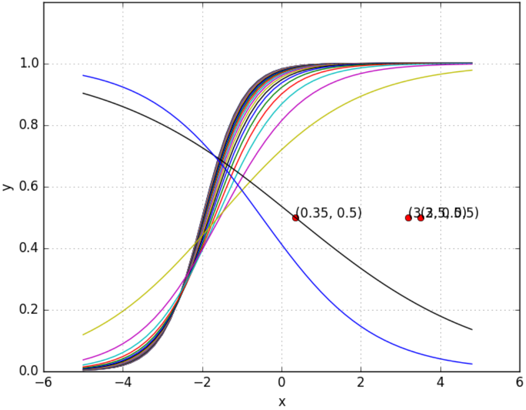

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w
b
−6 −4 −2 0 2 4 6
0.05 0.10 0.15 0.20 0.25 0.30 0.35

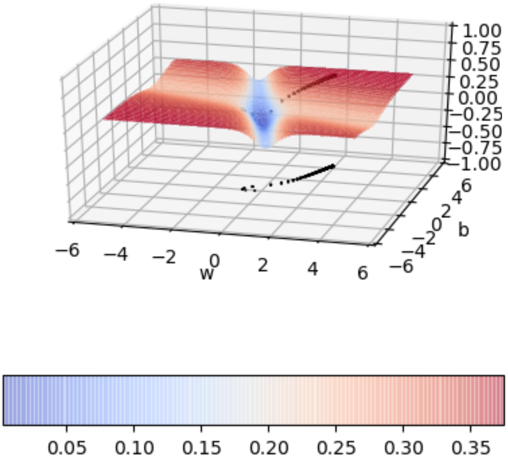

y
x
0.0 0.2 0.4 0.6 0.8 1.0
−6 −4 −2 0 2 4 6
(0.35, 0.5)

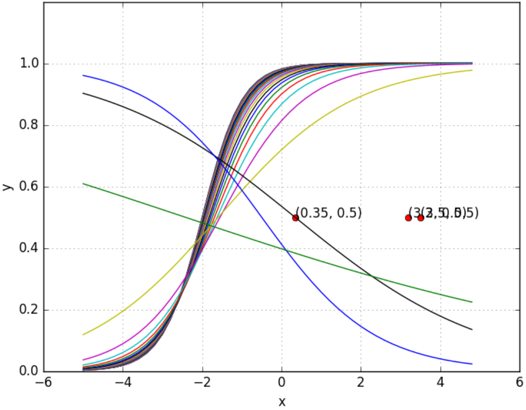

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
6 4 2 0 −2 −4 −6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35

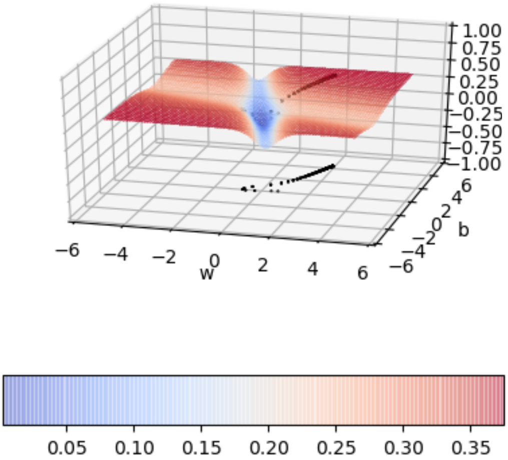

1.0
0.8
0.6
0.4
0.2
0.0
y
−6 −4 −2 0 2 4 6
x
(0.35, 0.5)

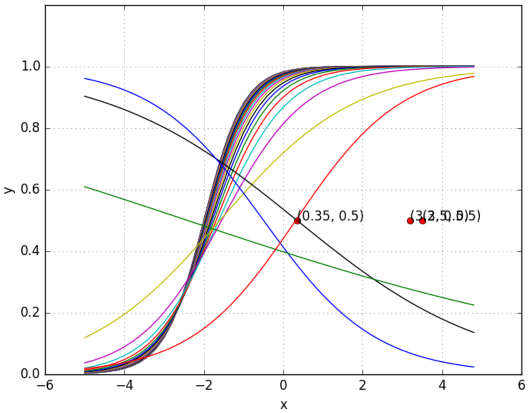

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
−6 −4 −2 0 2 4 6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35

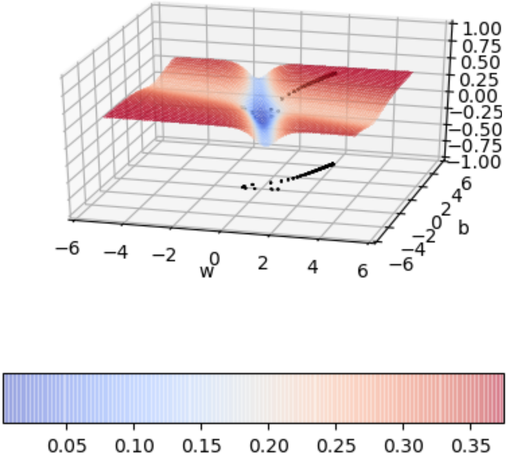

(0.35, 0.5)
y
0.0 0.2 0.4 0.6 0.8 1.0
x
−6 −4 −2 0 2 4 6

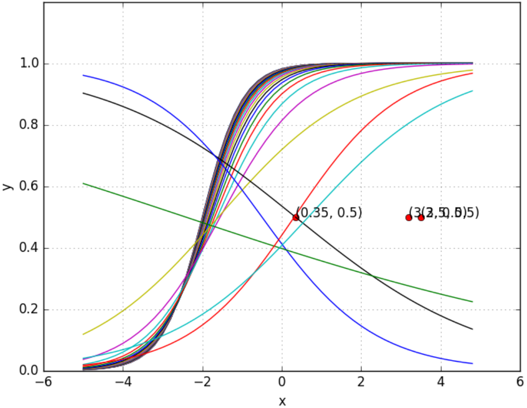

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

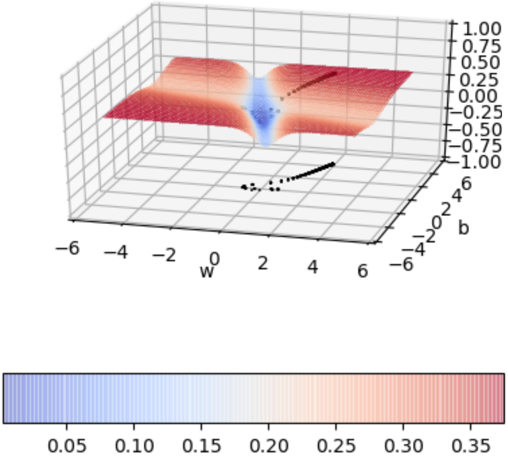

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x

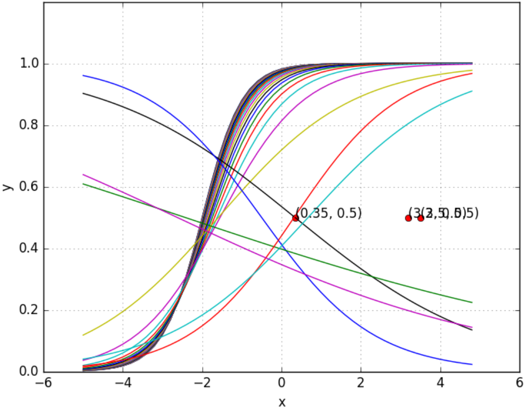

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w
b
−6
−4
−2
0
2
4
6
0.05
0.10
0.15
0.20
0.25
0.30
0.35

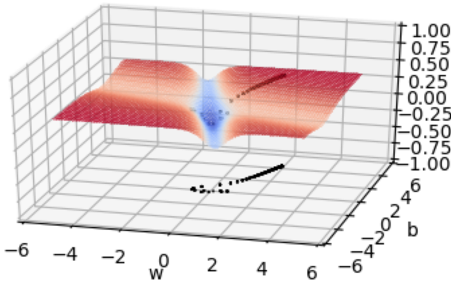

(0.35, 0.5)
y
x
0.0
0.2
0.4
0.6
0.8
1.0
−6
−4
−2
0
2
4
6


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
−6 −4 −2 0 2 4 6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35

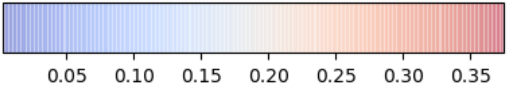

(0.35, 0.5)
1.0
0.8
0.6
0.4
0.2
0.0
y
−6 −4 −2 0 2 4 6
x


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

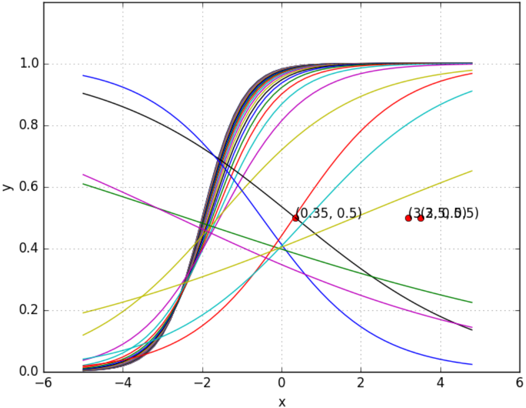

1.0
0.8
0.6
0.4
0.2
0.0
y
−6
−4
−2
0
2
4
6
x
(0.35, 0.5)



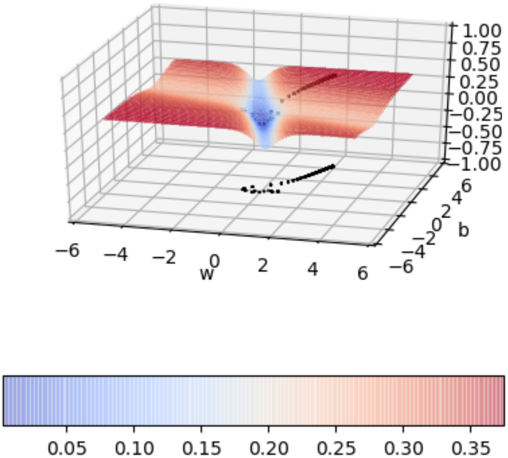
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b

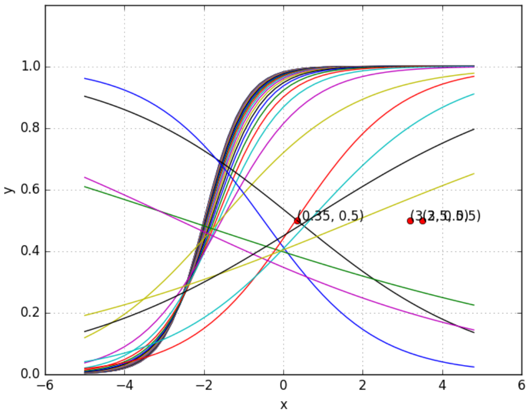


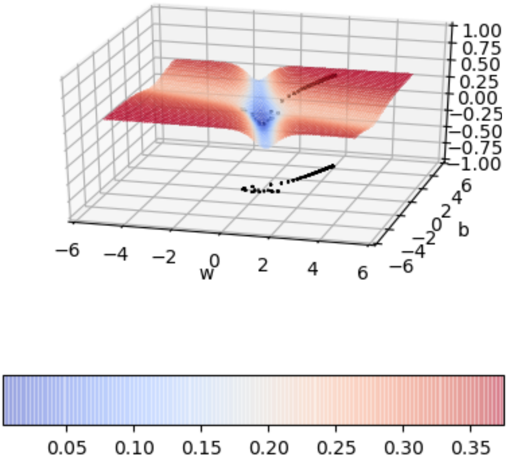
0.05
0.10
0.15
0.20
0.25
0.30
0.35

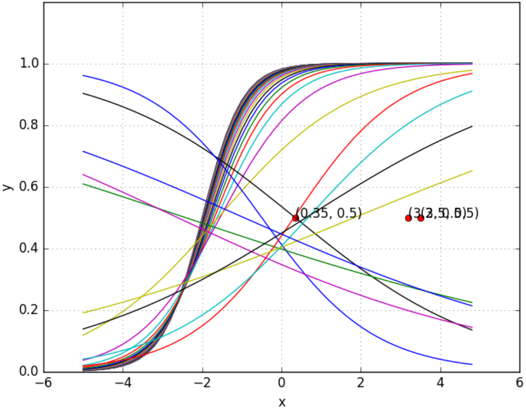


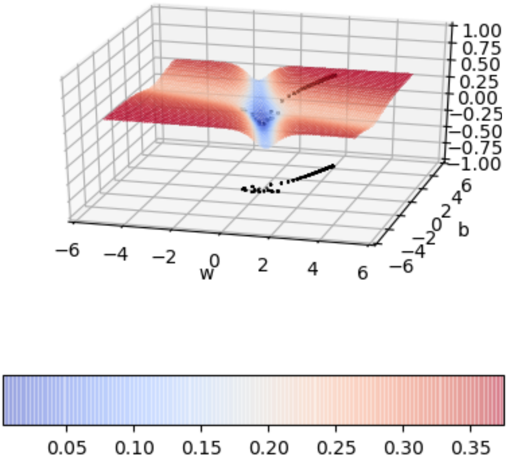
1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x

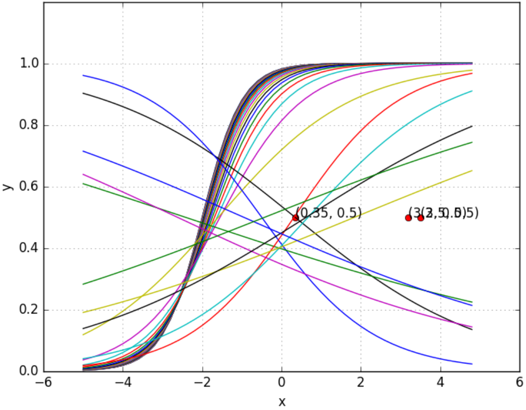

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
6 4 2 0 −2 −4 −6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35

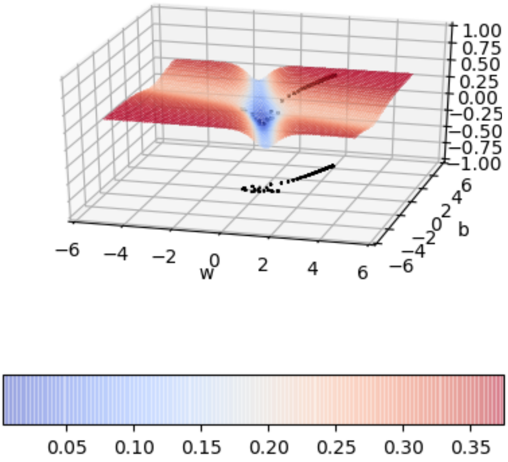

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6 −4 −2 0 2 4 6
x

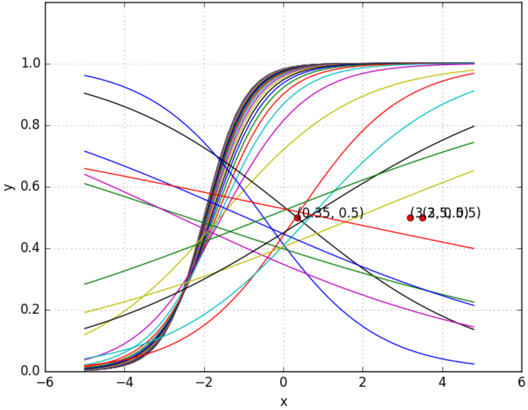

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

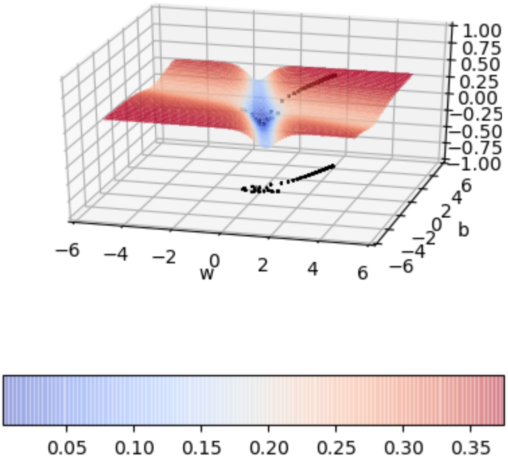

(0.35, 0.5)
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6

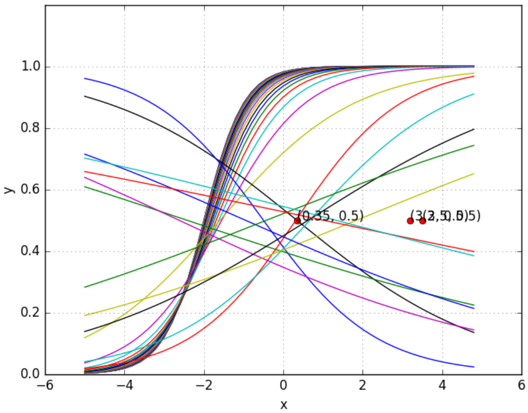

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

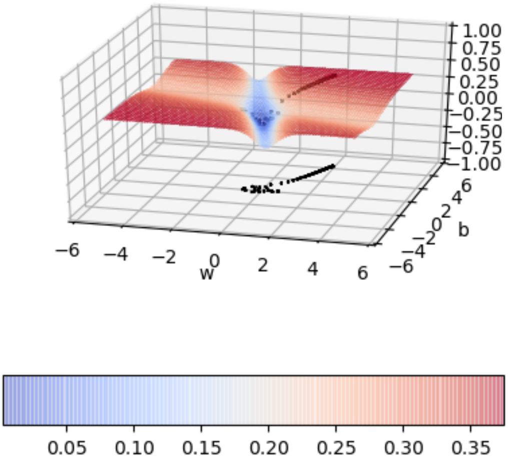

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

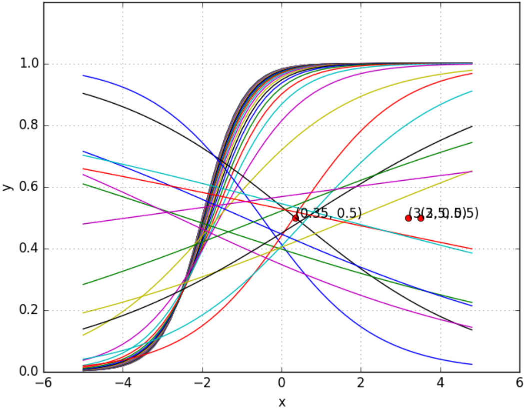

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x


w
b
−6 −4 −2 0 2 4 6
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
0.05 0.10 0.15 0.20 0.25 0.30 0.35

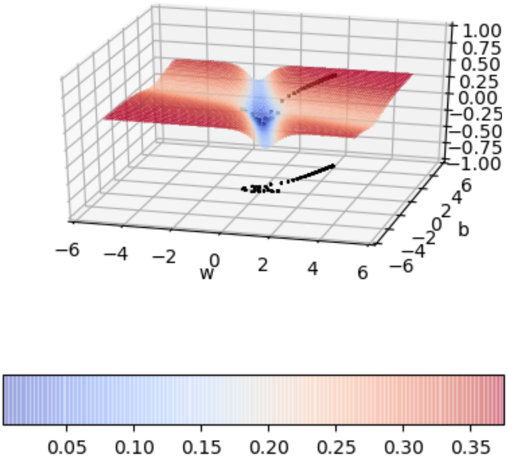

x
y
(0.35, 0.5)
−6 −4 −2 0 2 4 6
0.0 0.2 0.4 0.6 0.8 1.0



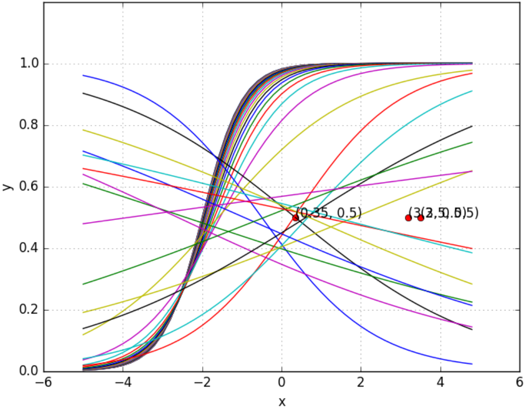
w
b



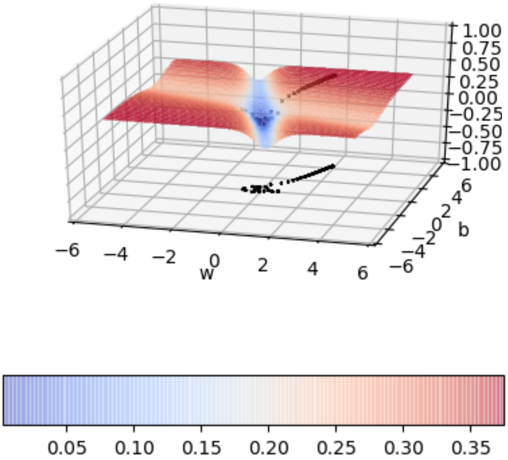
x
y
(0.35, 0.5)

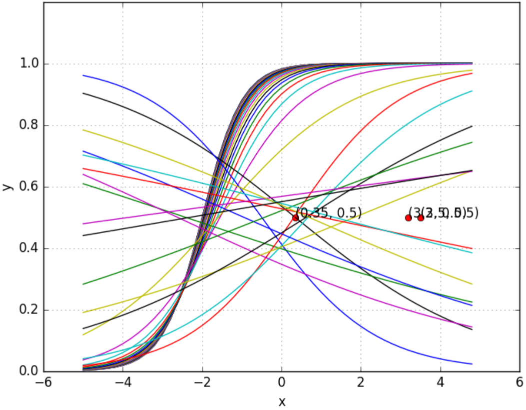


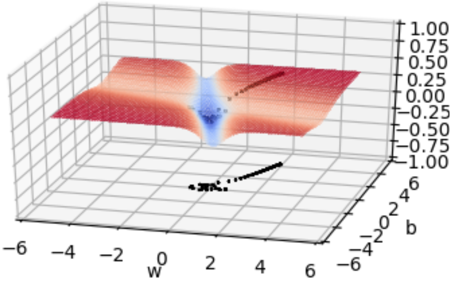
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
−6 −4 −2 0 2 4 6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35



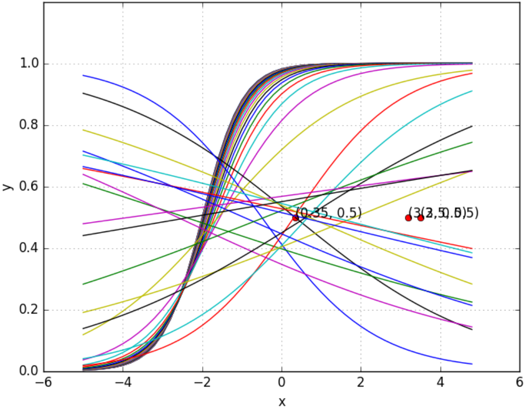
(0.35, 0.5)
y
0.0 0.2 0.4 0.6 0.8 1.0
x
−6 −4 −2 0 2 4 6


w
b
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.05 0.10 0.15 0.20 0.25 0.30 0.35

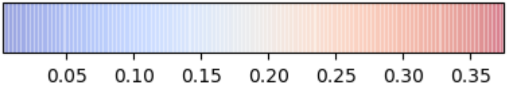

x
y
(0.35, 0.5)



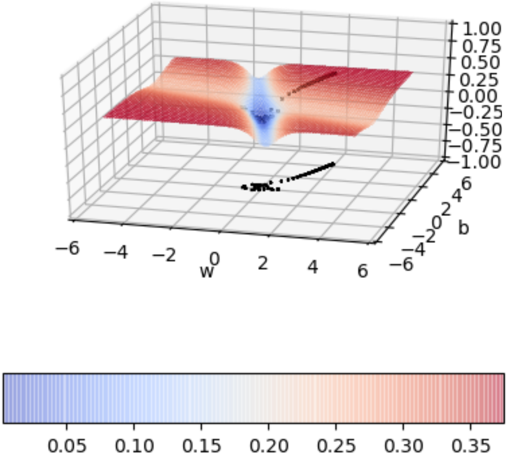
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b

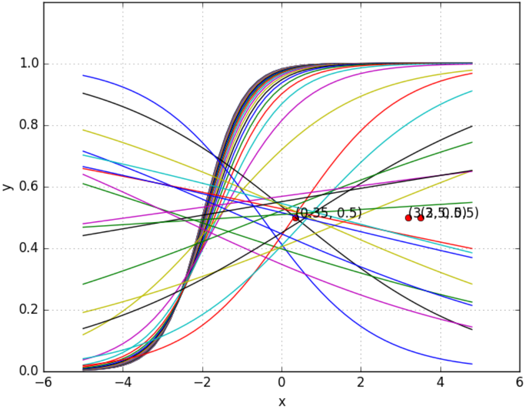


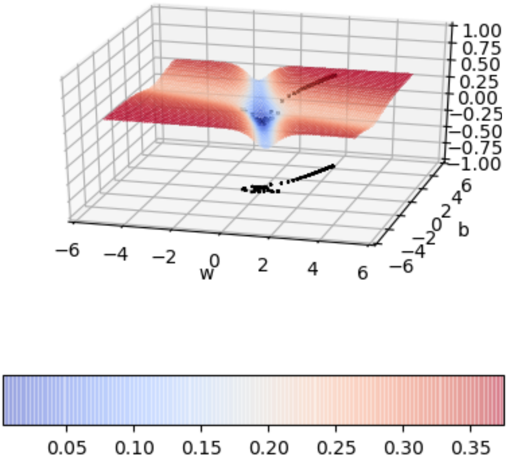
1.0
0.8
0.6
0.4
0.2
0.0
y
−6
−4
−2
0
2
4
6
x
(0.35, 0.5)

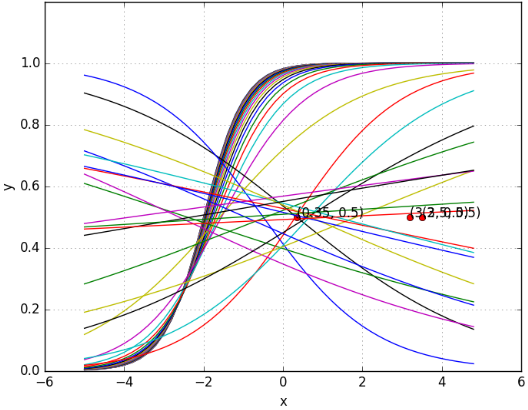


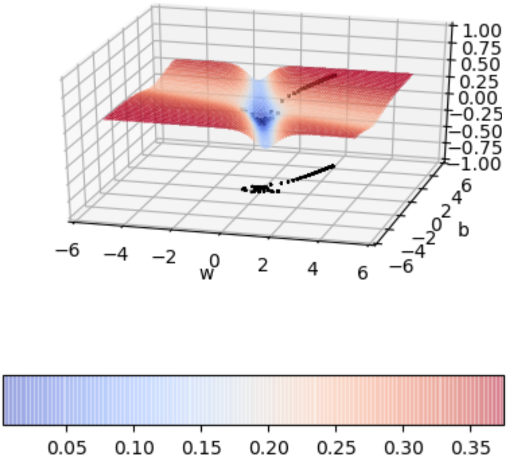
0.05
0.10
0.15
0.20
0.25
0.30
0.35


w
b
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
−6 −4 −2 0 2 4 6
0.05 0.10 0.15 0.20 0.25 0.30 0.35

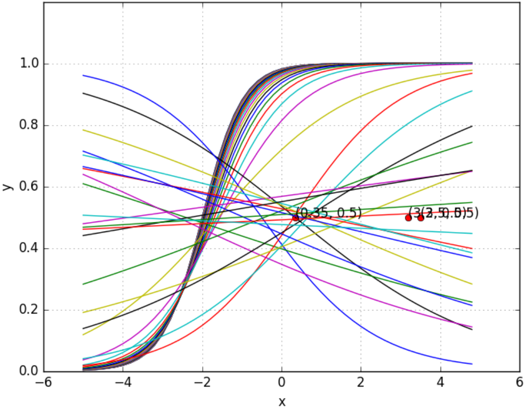

y
x
0.0 0.2 0.4 0.6 0.8 1.0
−6 −4 −2 0 2 4 6
(0.35, 0.5)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35

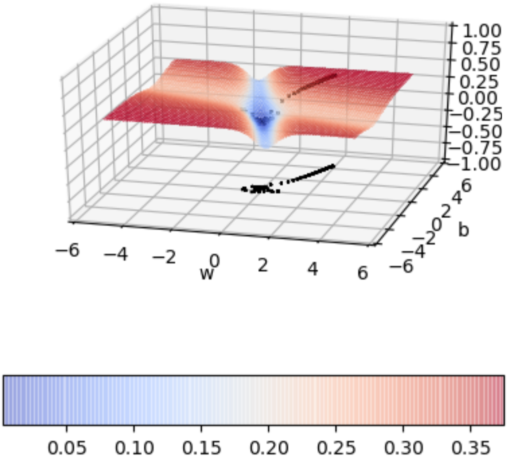

1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x



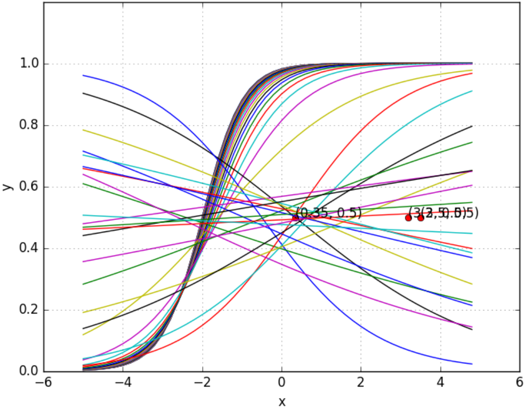
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6 −4 −2 0 2 4 6
w
−6 −4 −2 0 2 4 6
b
0.05 0.10 0.15 0.20 0.25 0.30 0.35



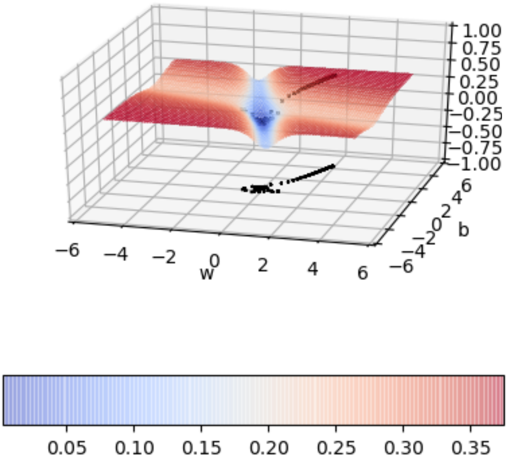
1.0
0.8
0.6
0.4
0.2
0.0
y
−6 −4 −2 0 2 4 6
x
(0.35, 0.5)

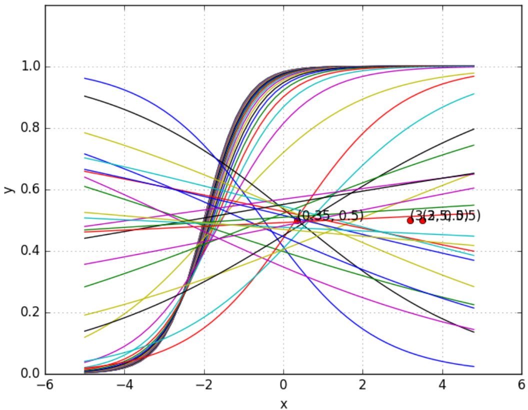


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
w
2
4
6
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35



1.0
0.8
0.6
0.4
0.2
0.0
y
(0.35, 0.5)
−6
−4
−2
0
2
4
6
x


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6
w
b
0.05
0.10
0.15
0.20
0.25
0.30
0.35


1.0
0.8
0.6
0.4
0.2
0.0
y
−6
−4
−2
0
2
4
6
x
(0.35, 0.5)

Module 5.5 : Nesterov Accelerated Gradient Descent

Question

- Can we do something to reduce these oscillations ?

Question

- Can we do something to reduce these oscillations ?
- Yes, let's look at Nesterov accelerated gradient

Intuition

- Look before you leap

Intuition

- Look before you leap
- Recall that $update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$

Intuition

- Look before you leap
- Recall that $update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$
- So we know that we are going to move by at least by $\gamma \cdot update_{t-1}$ and then a bit more by $\eta \nabla w_t$

Intuition

- Look before you leap
- Recall that $update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$
- So we know that we are going to move by at least by $\gamma \cdot update_{t-1}$ and then a bit more by $\eta \nabla w_t$
- Why not calculate the gradient (∇w_{look_ahead}) at this partially updated value of w ($w_{look\_ahead} = w_t - \gamma \cdot update_{t-1}$) instead of calculating it using the current value w_t

Intuition

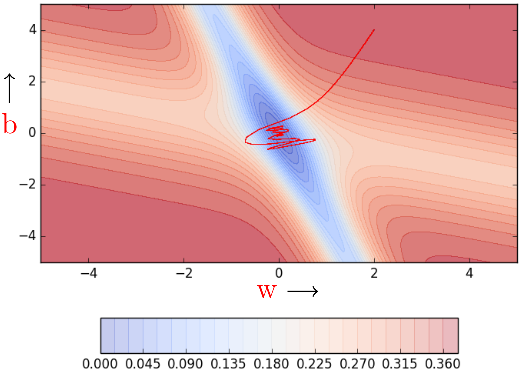
- Look before you leap
- Recall that $update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$
- So we know that we are going to move by at least by $\gamma \cdot update_{t-1}$ and then a bit more by $\eta \nabla w_t$
- Why not calculate the gradient (∇w_{look_ahead}) at this partially updated value of w ($w_{look\_ahead} = w_t - \gamma \cdot update_{t-1}$) instead of calculating it using the current value w_t

Update rule for NAG

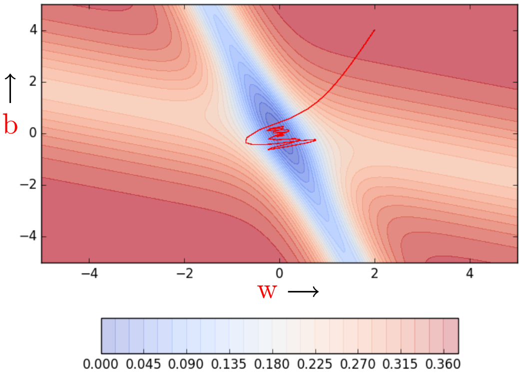
$$
\begin{aligned}
w_{look\_ahead} &= w_t - \gamma \cdot update_{t-1} \\
update_t &= \gamma \cdot update_{t-1} + \eta \nabla w_{look\_ahead} \\
w_{t+1} &= w_t - update_t
\end{aligned}
$$

We will have similar update rule for b_t

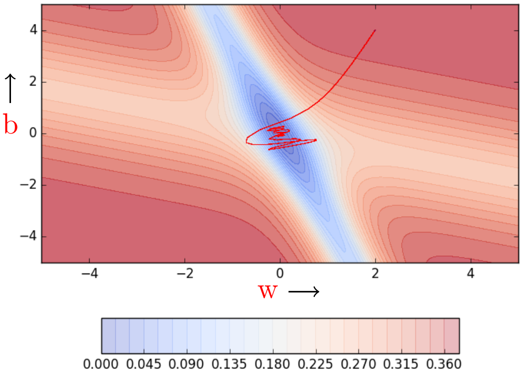
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



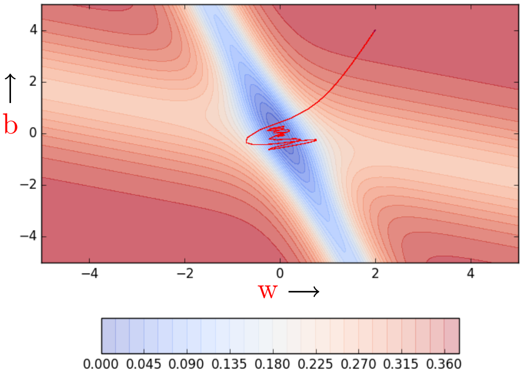
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



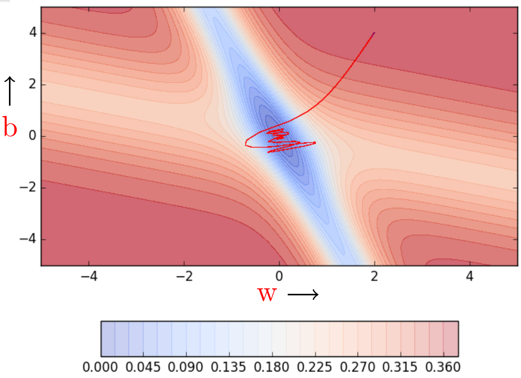
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



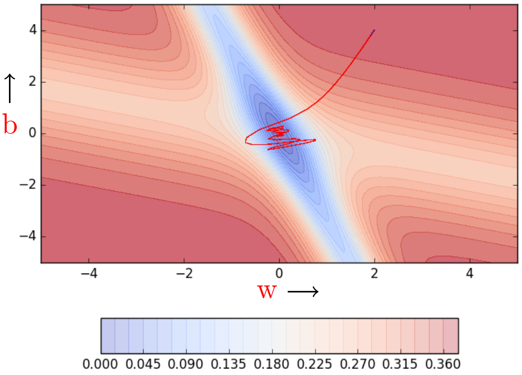
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



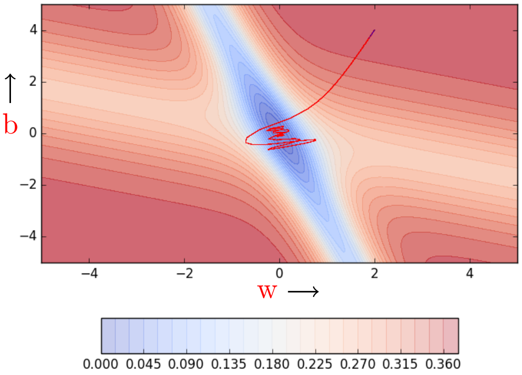
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



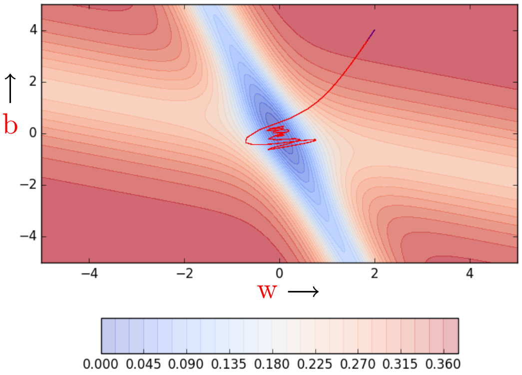
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



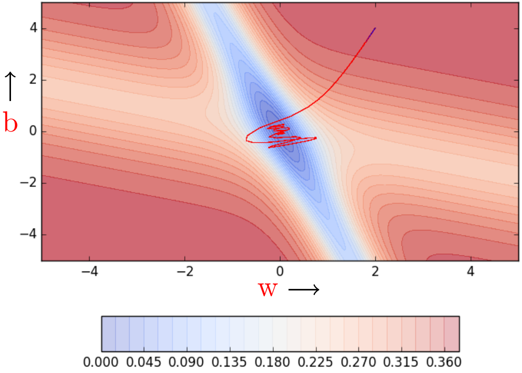
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



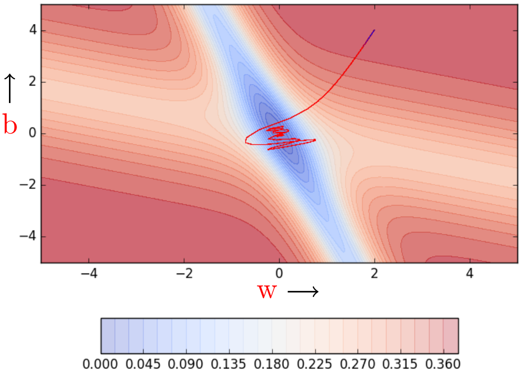
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



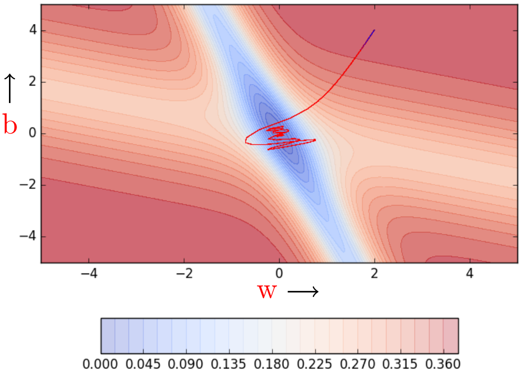
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



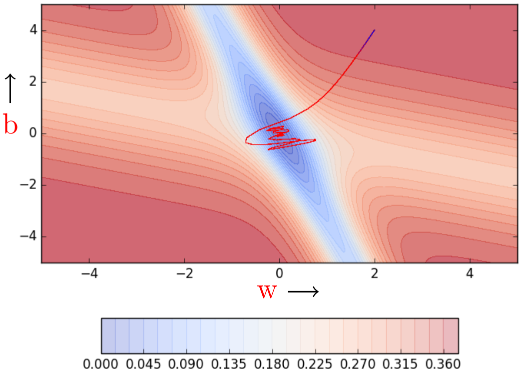
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



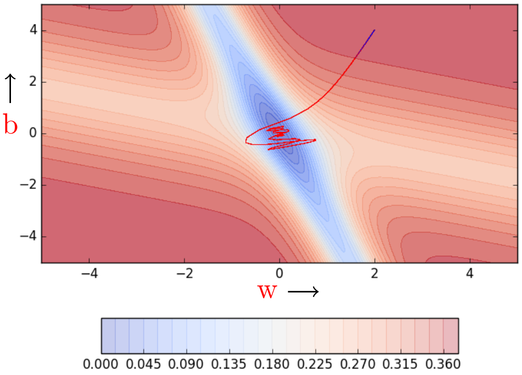
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



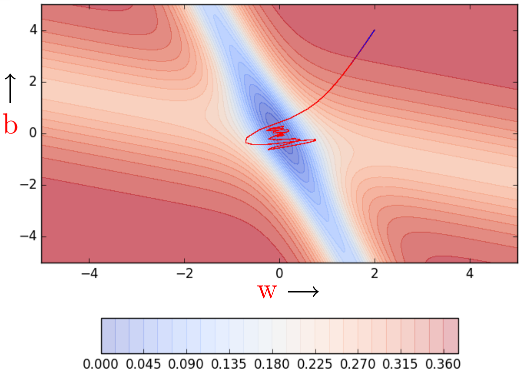
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



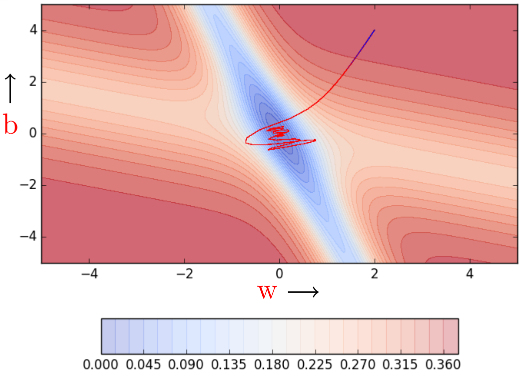
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



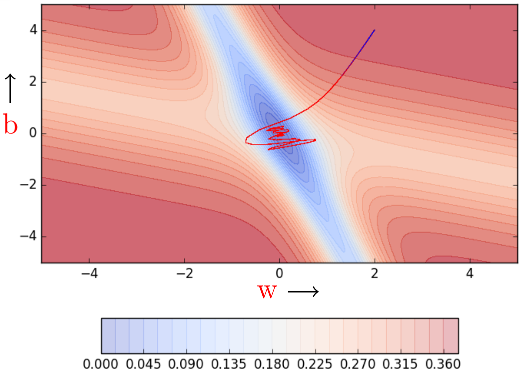
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



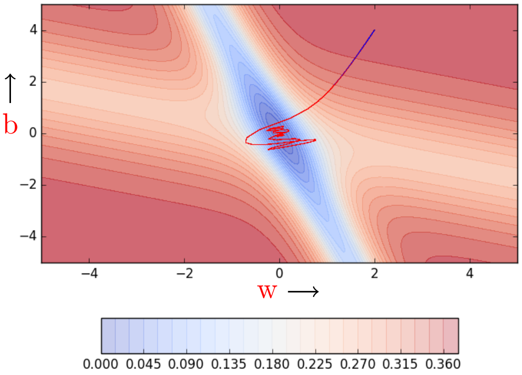
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



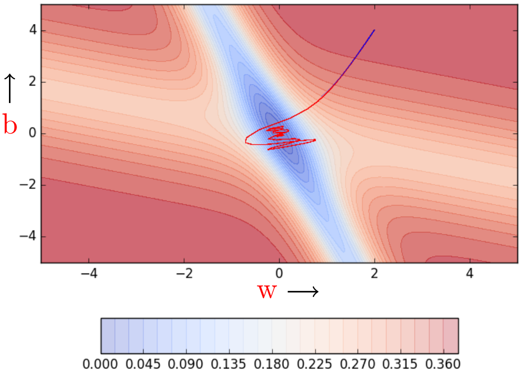
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



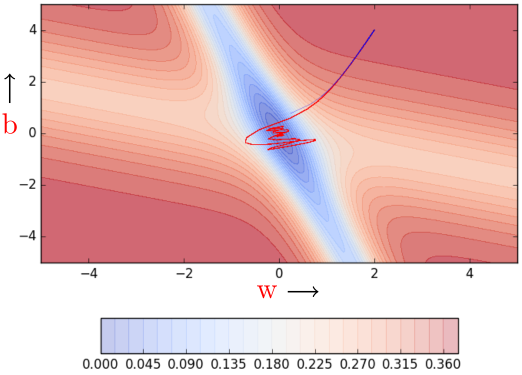
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



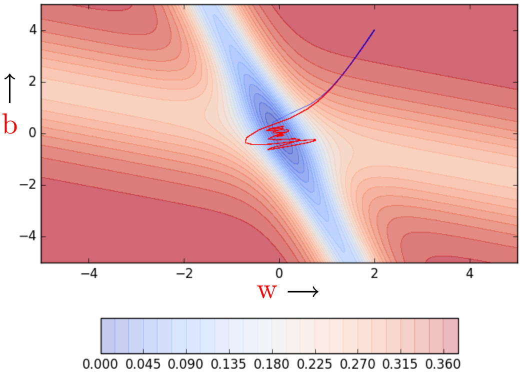
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



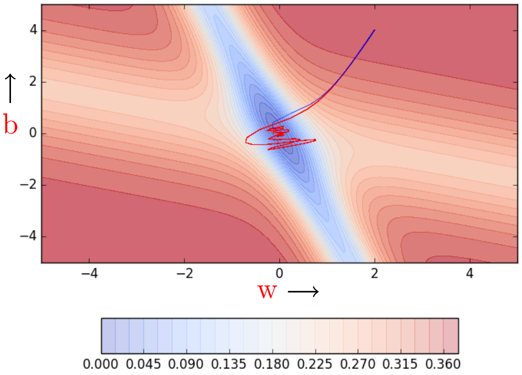
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



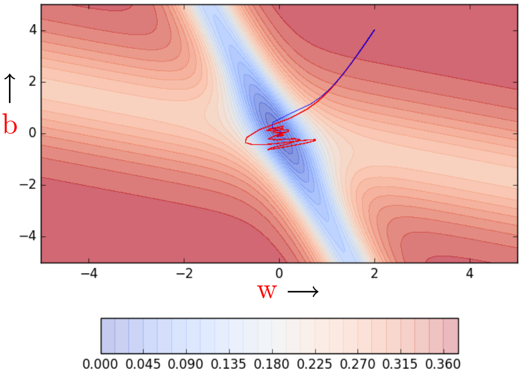
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



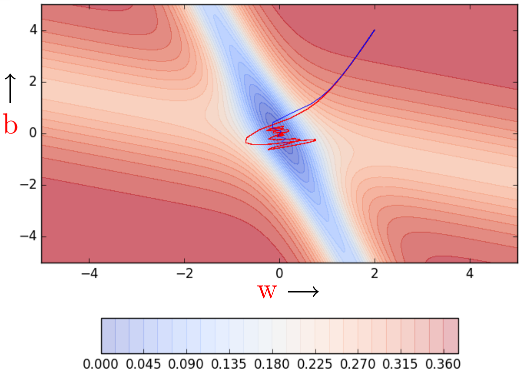
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



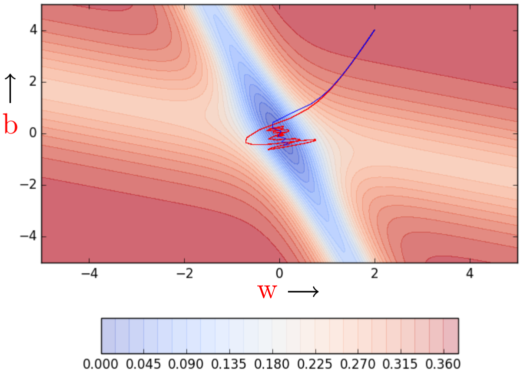
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



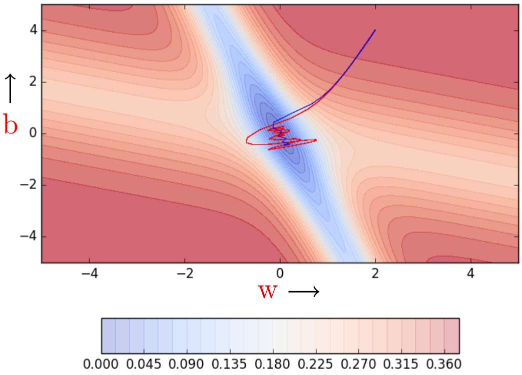
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



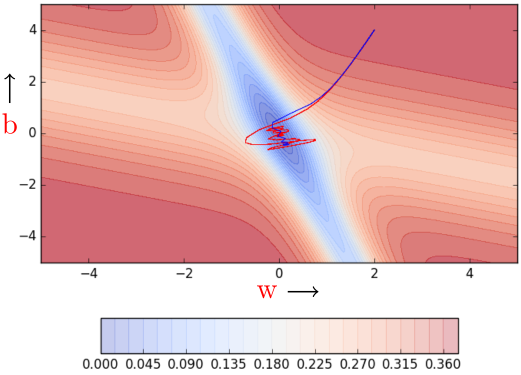
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



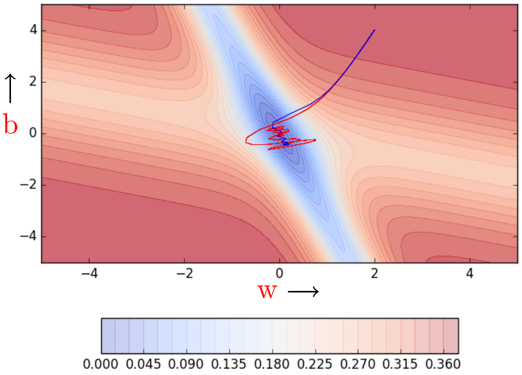
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



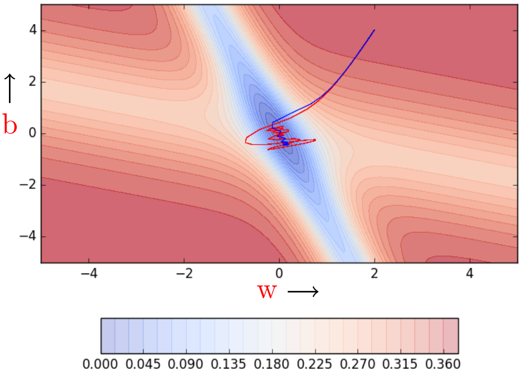
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



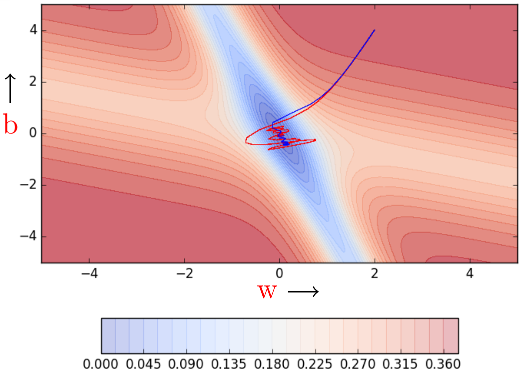
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



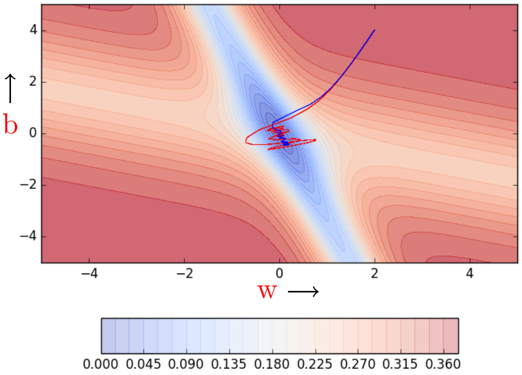
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



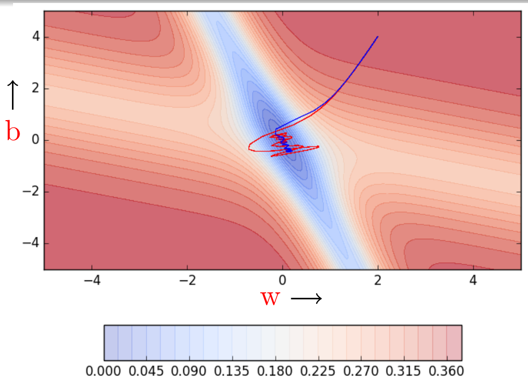
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



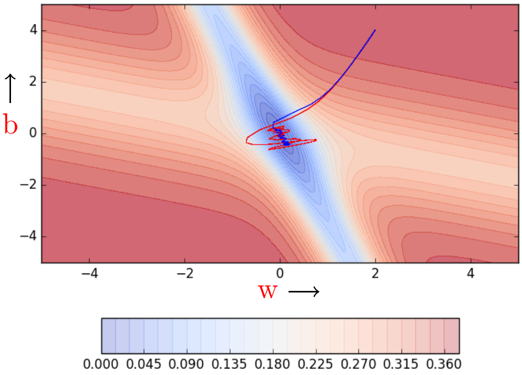
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



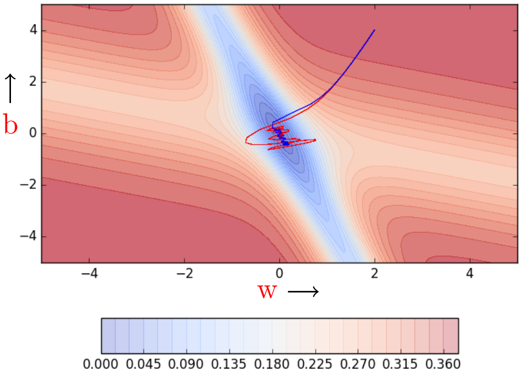
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



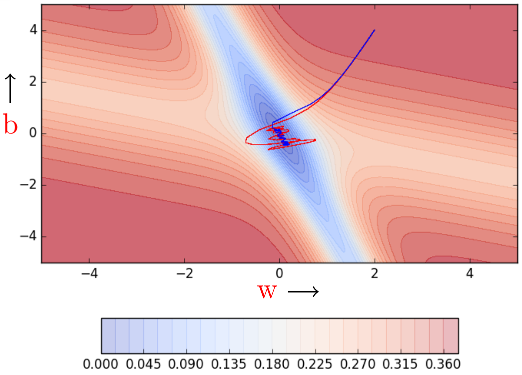
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



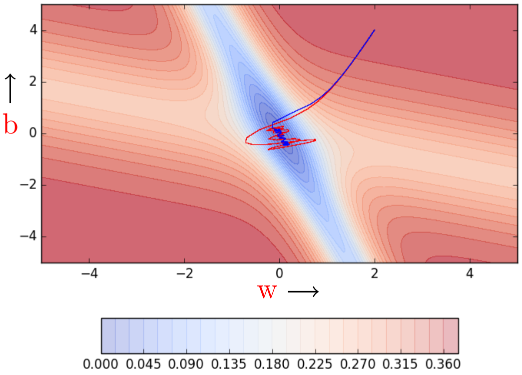
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



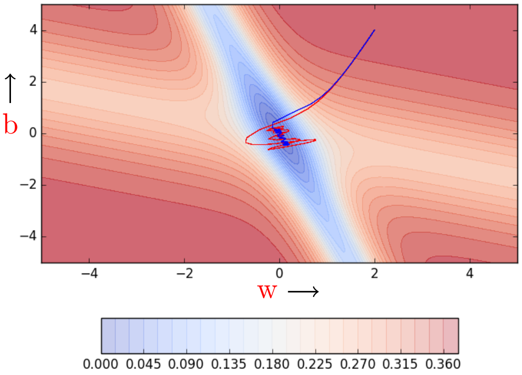
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



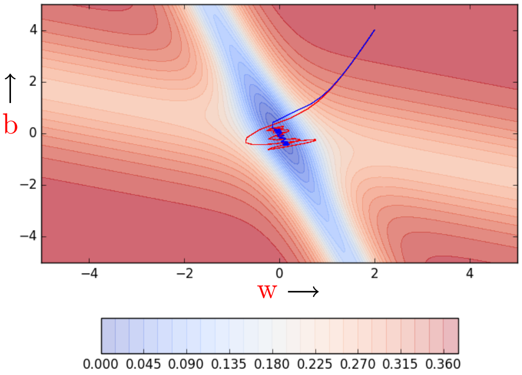
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



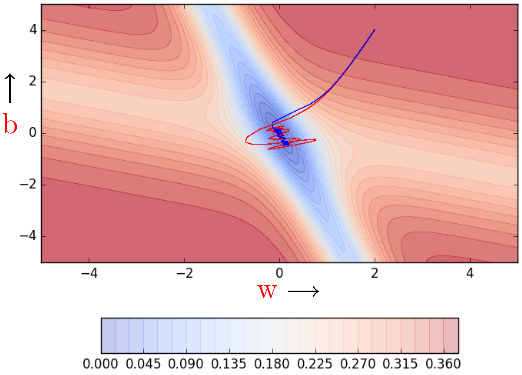
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



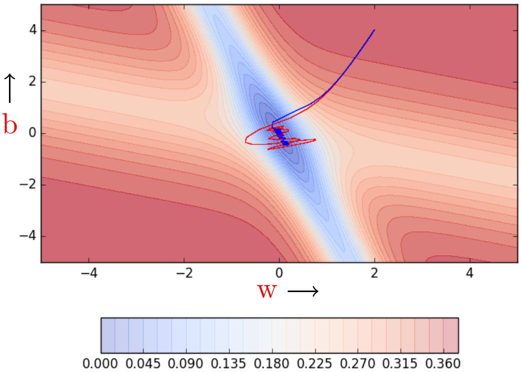
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



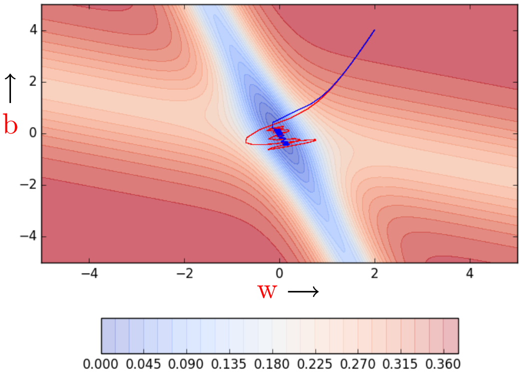
```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



Observations about NAG

- Looking ahead helps NAG in correcting its course quicker than momentum based gradient descent

Observations about NAG

- Looking ahead helps NAG in correcting its course quicker than momentum based gradient descent
- Hence the oscillations are smaller and the chances of escaping the minima valley also smaller

Module 5.6 : Stochastic And Mini-Batch Gradient Descent

Let's digress a bit and talk about the stochastic version of these algorithms...

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why?

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation.

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation. Hence, theoretical guarantees hold (in other words each step guarantees that the loss will decrease)

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation. Hence, theoretical guarantees hold (in other words each step guarantees that the loss will decrease)
- What's the flipside?

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation. Hence, theoretical guarantees hold (in other words each step guarantees that the loss will decrease)
- What's the flipside? Imagine we have a million points in the training data.

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation. Hence, theoretical guarantees hold (in other words each step guarantees that the loss will decrease)
- What's the flipside? Imagine we have a million points in the training data. To make 1 update to w, b the algorithm makes a million calculations.

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation. Hence, theoretical guarantees hold (in other words each step guarantees that the loss will decrease)
- What's the flipside? Imagine we have a million points in the training data. To make 1 update to w, b the algorithm makes a million calculations. Obviously very slow!!

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation. Hence, theoretical guarantees hold (in other words each step guarantees that the loss will decrease)
- What's the flipside? Imagine we have a million points in the training data. To make 1 update to w, b the algorithm makes a million calculations. Obviously very slow!!
- Can we do something better ?

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation. Hence, theoretical guarantees hold (in other words each step guarantees that the loss will decrease)
- What's the flipside? Imagine we have a million points in the training data. To make 1 update to w, b the algorithm makes a million calculations. Obviously very slow!!
- Can we do something better ? Yes, let's look at stochastic gradient descent

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

```python
def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

```python
def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm updates the parameters for every single data point

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)

```python
def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)
- What is the flipside ?

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)
- What is the flipside ? It is an approximate (rather stochastic) gradient

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

- Stochastic because we are estimating the total gradient based on a single data point.

- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)
- What is the flipside ? It is an approximate (rather stochastic) gradient

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

- Stochastic because we are estimating the total gradient based on a single data point. Almost like tossing a coin only once and estimating P(heads).

- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)
- What is the flipside ? It is an approximate (rather stochastic) gradient

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

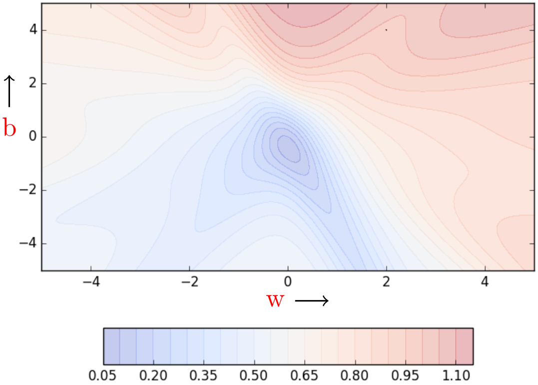
- Stochastic because we are estimating the total gradient based on a single data point. Almost like tossing a coin only once and estimating P(heads).

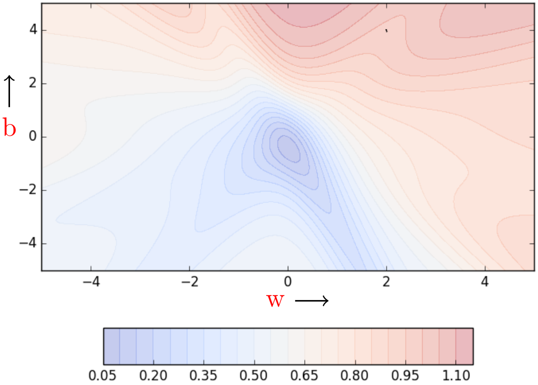
- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)
- What is the flipside ? It is an approximate (rather stochastic) gradient
- No guarantee that each step will decrease the loss

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

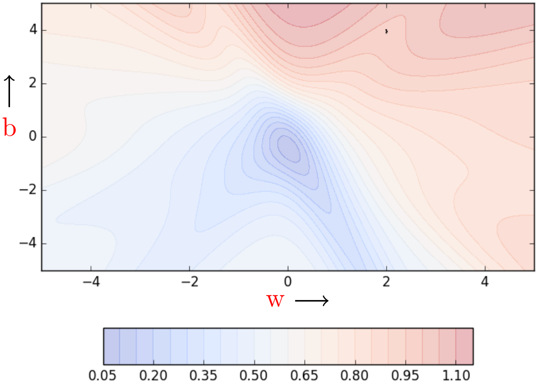
- Stochastic because we are estimating the total gradient based on a single data point. Almost like tossing a coin only once and estimating P(heads).

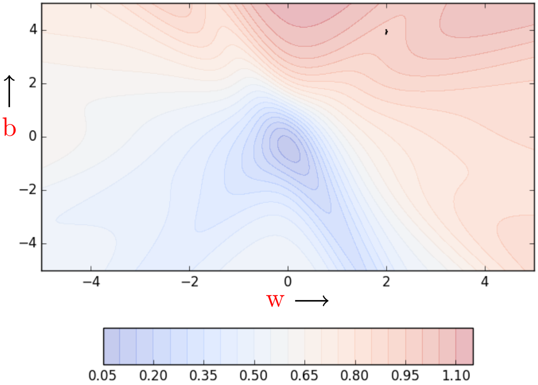
- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)
- What is the flipside ? It is an approximate (rather stochastic) gradient
- No guarantee that each step will decrease the loss
- Let's see this algorithm in action when we have a few data points


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

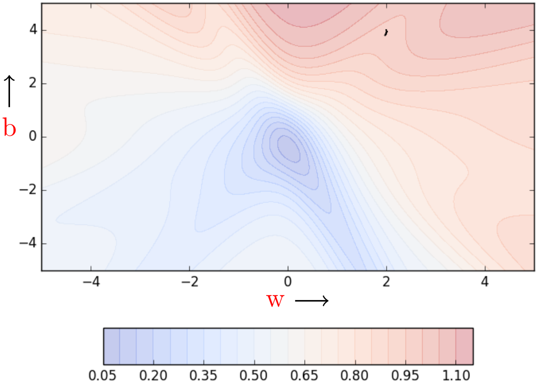


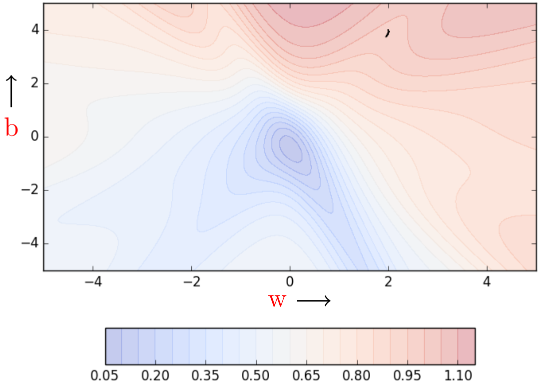
b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

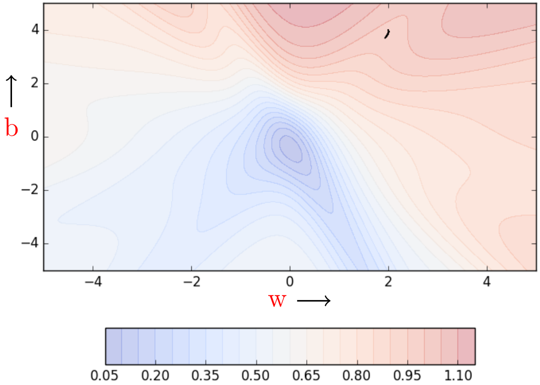


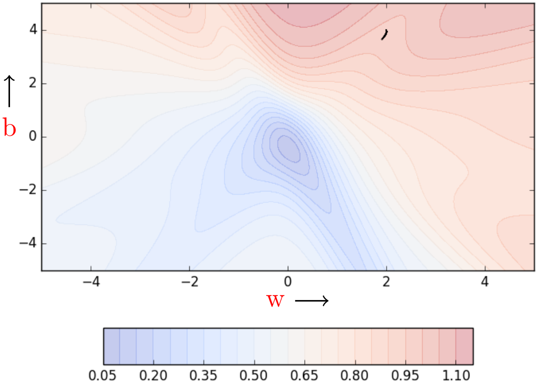
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

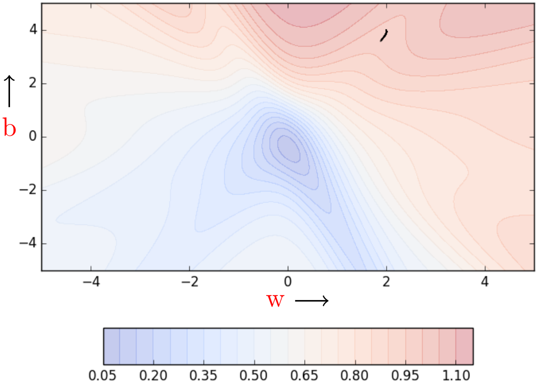

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10



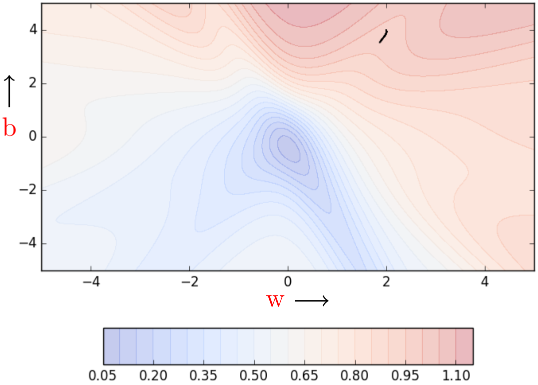
b
w
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

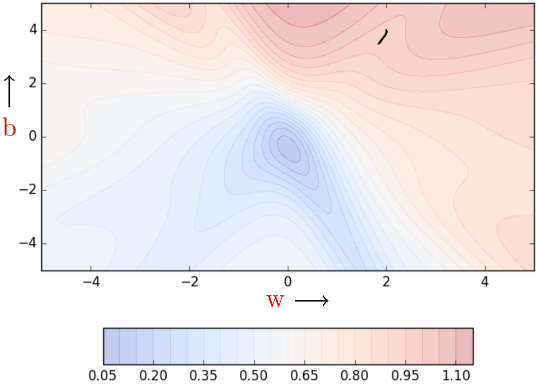

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

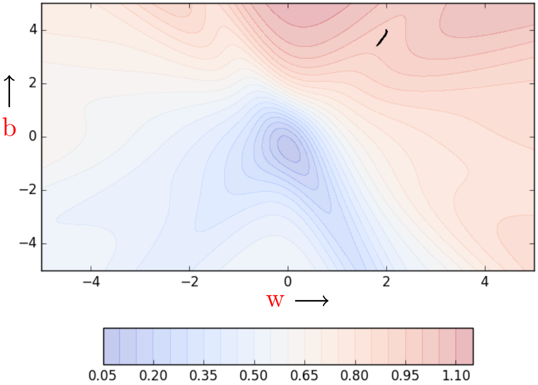


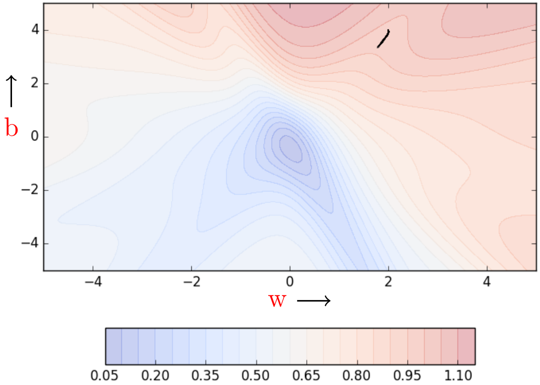
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

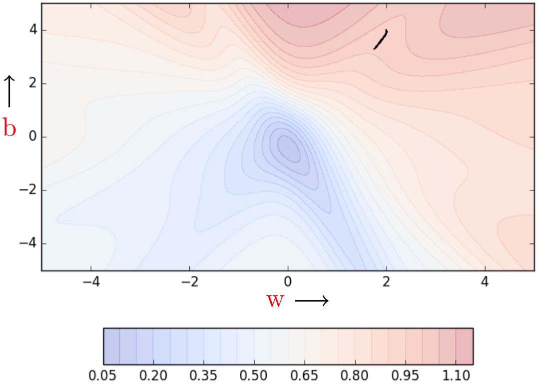

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

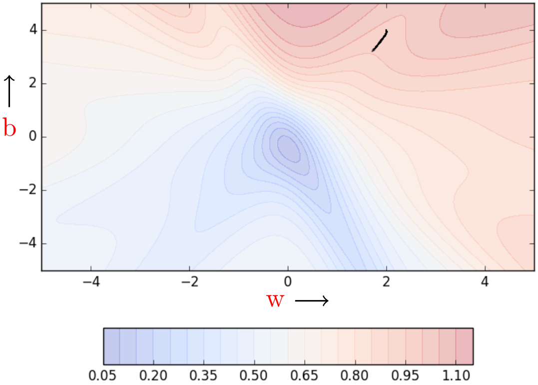


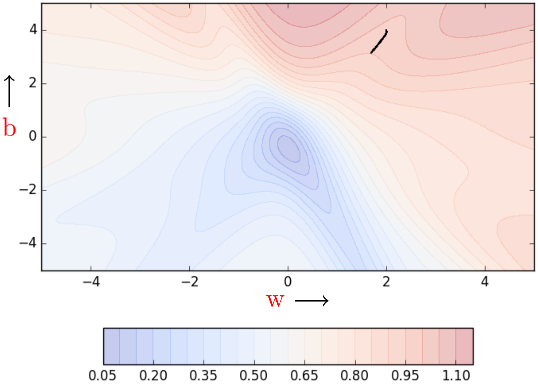
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

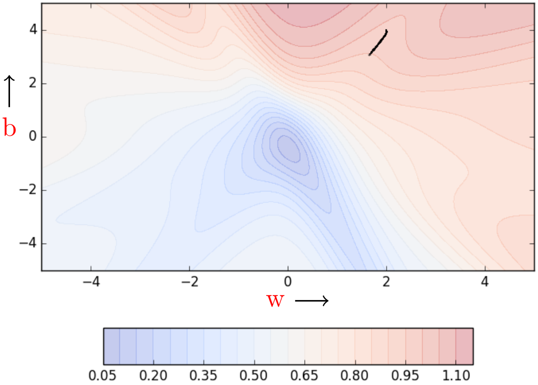

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

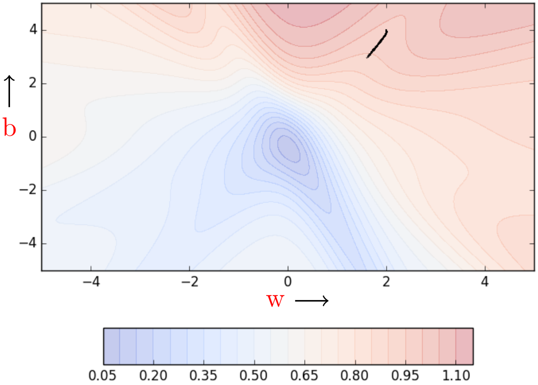

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

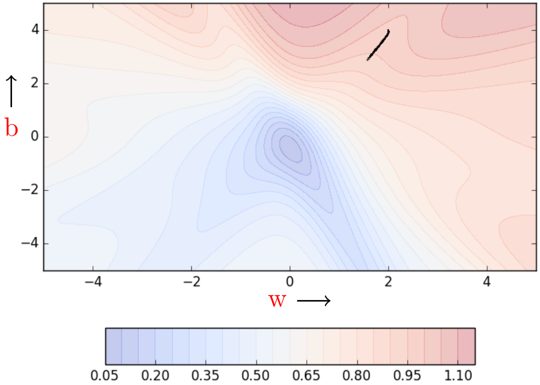


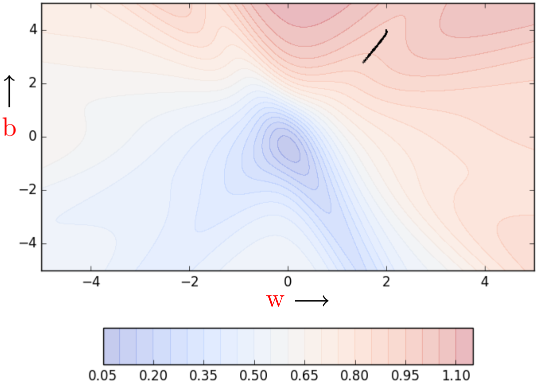
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

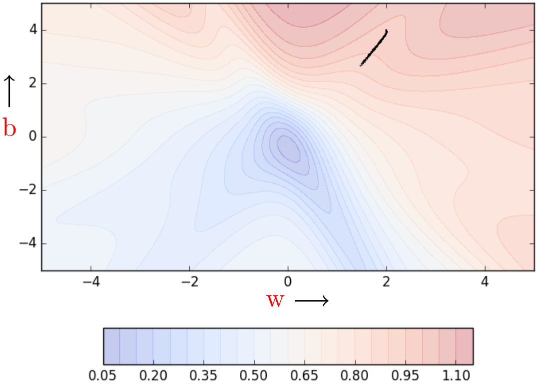

b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10



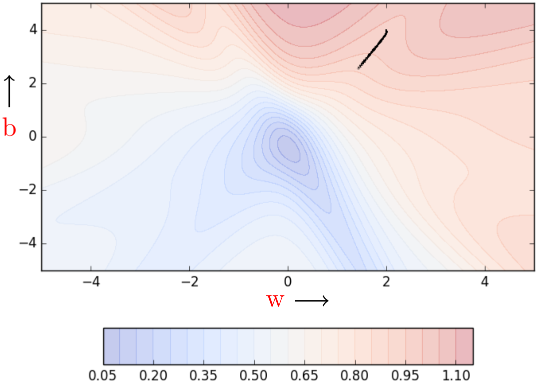
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

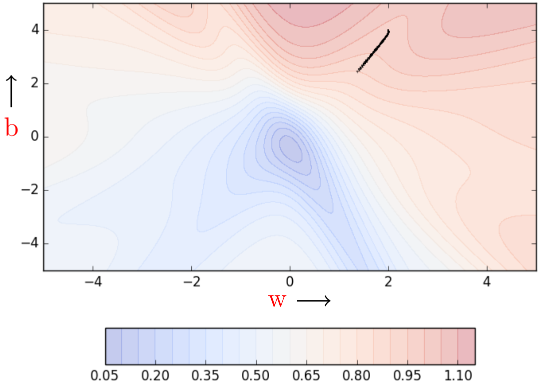

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

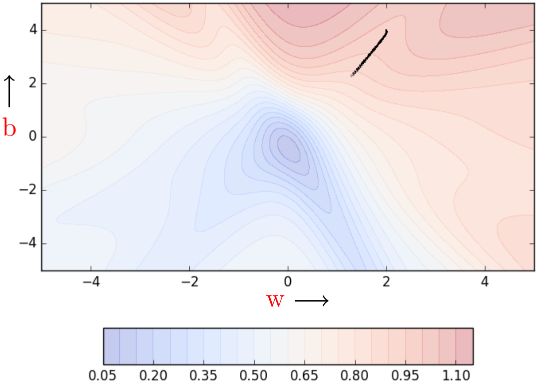


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

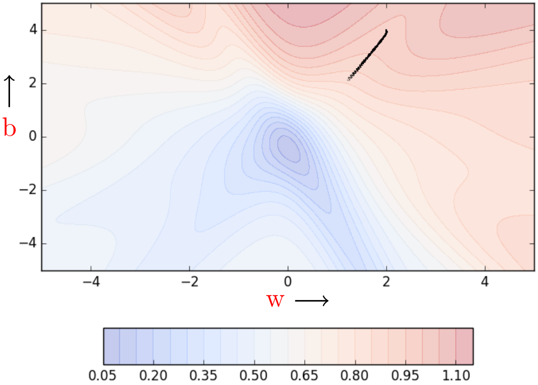


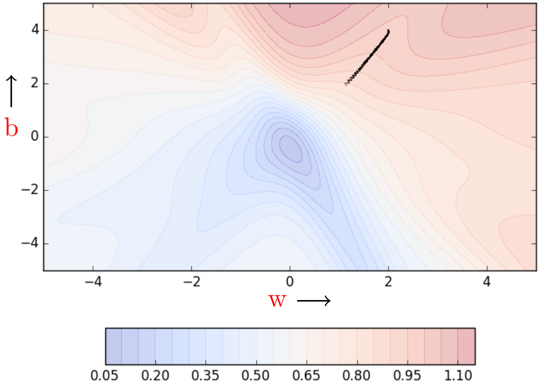
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

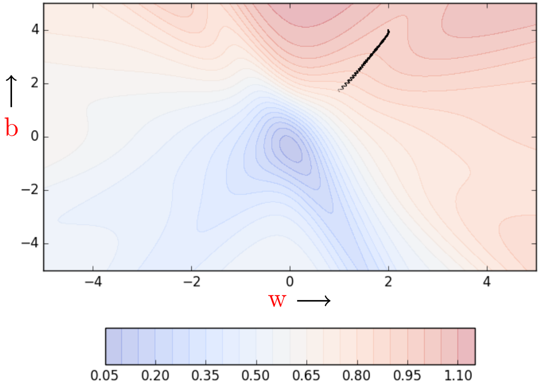

b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

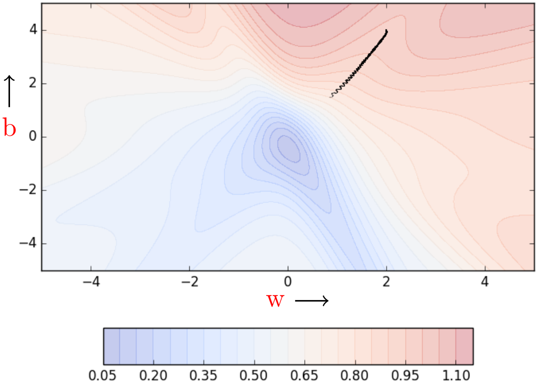


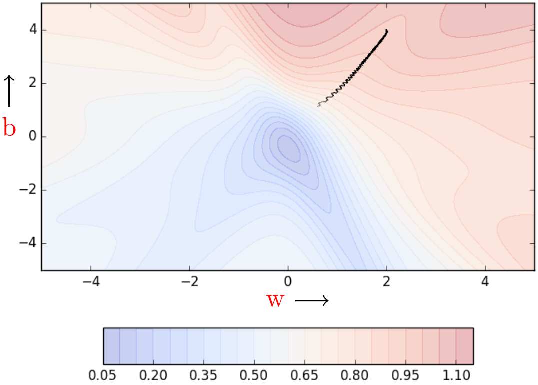
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

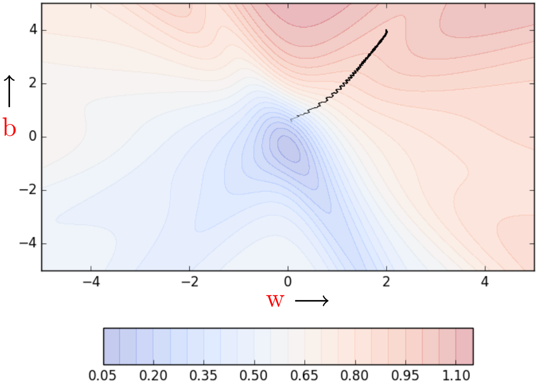

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

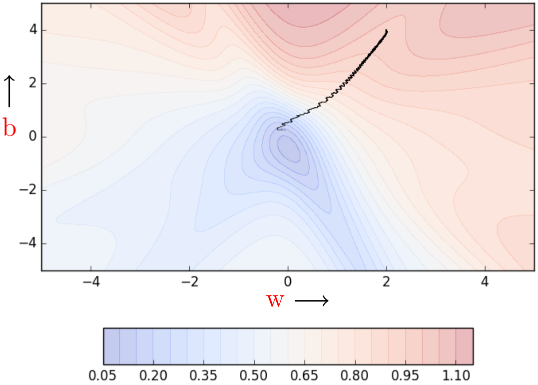

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

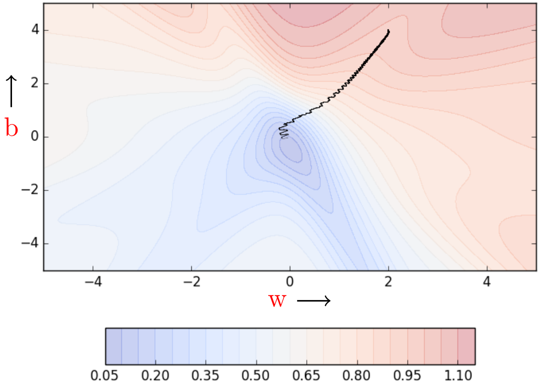

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

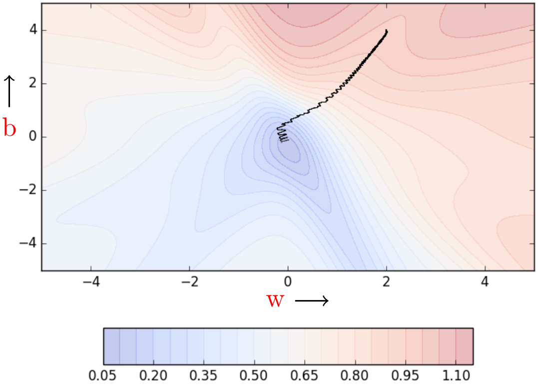


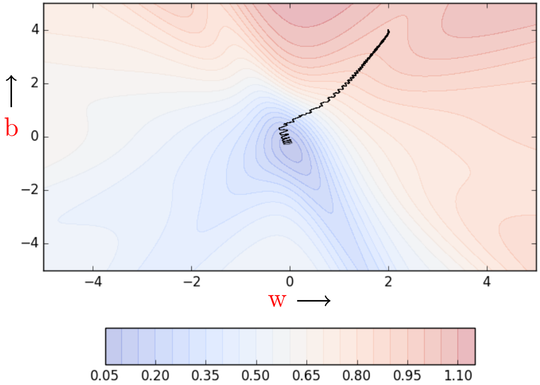
b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

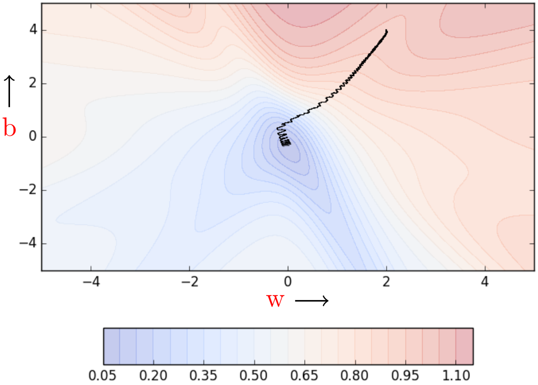


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

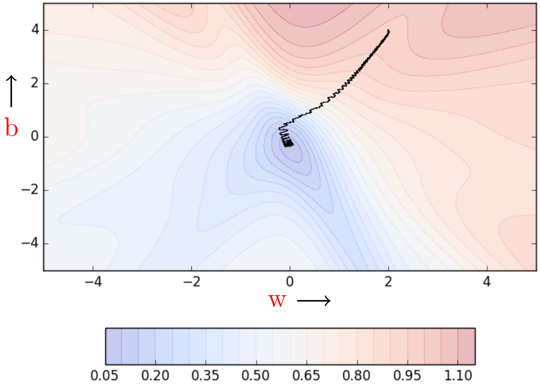

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


b
w
4
2
0
−2
−4
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

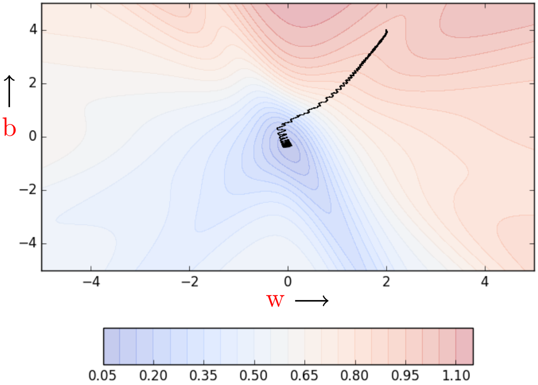


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

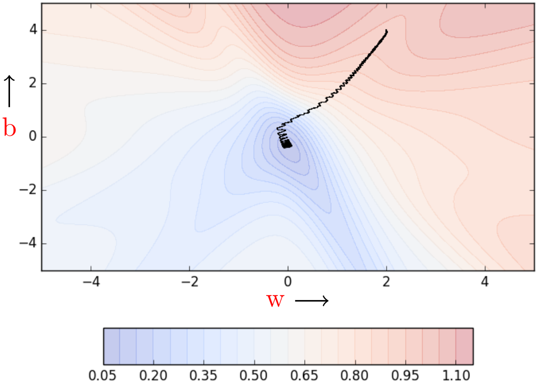


4
2
0
−2
−4
−4
−2
0
2
4
b
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

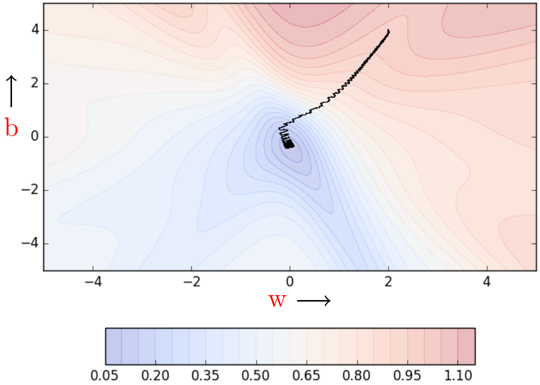
- We see many oscillations. Why ?



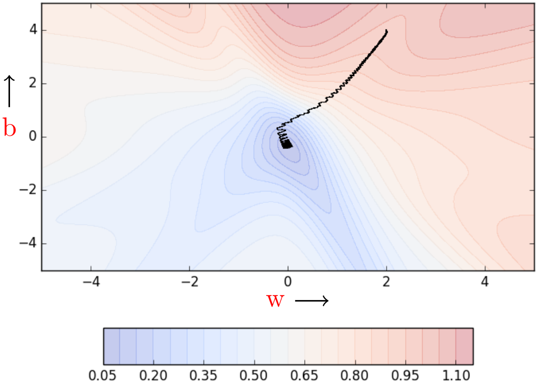
- We see many oscillations. Why ? Because we are making greedy decisions.



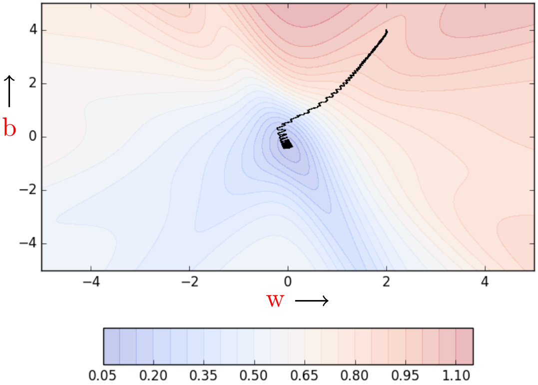
- We see many oscillations. Why ? Because we are making greedy decisions.
- Each point is trying to push the parameters in a direction most favorable to it



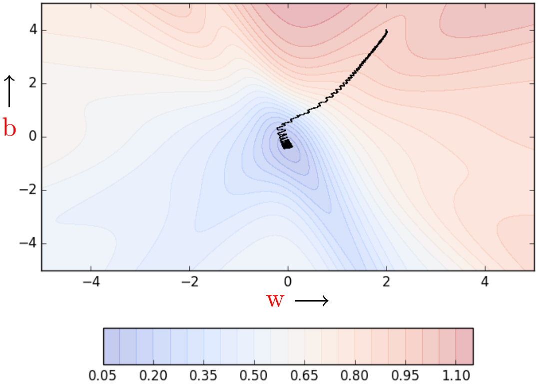
- We see many oscillations. Why ? Because we are making greedy decisions.
- Each point is trying to push the parameters in a direction most favorable to it (without being aware of how this affects other points)



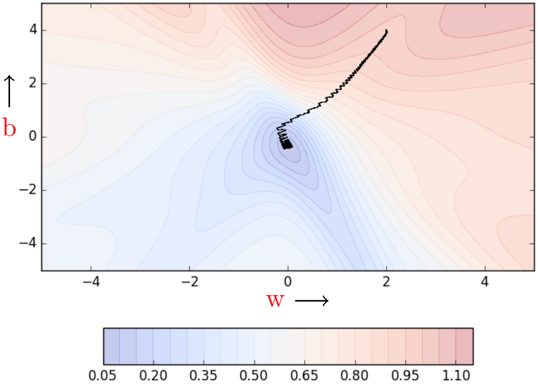
- We see many oscillations. Why ? Because we are making greedy decisions.
- Each point is trying to push the parameters in a direction most favorable to it (without being aware of how this affects other points)
- A parameter update which is locally favorable to one point may harm other points



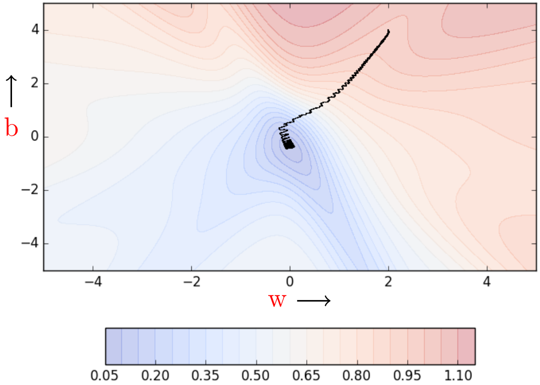
- We see many oscillations. Why ? Because we are making greedy decisions.
- Each point is trying to push the parameters in a direction most favorable to it (without being aware of how this affects other points)
- A parameter update which is locally favorable to one point may harm other points (its almost as if the data points are competing with each other)



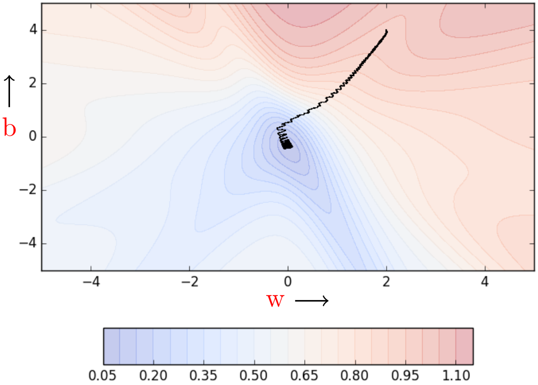
- We see many oscillations. Why ? Because we are making greedy decisions.
- Each point is trying to push the parameters in a direction most favorable to it (without being aware of how this affects other points)
- A parameter update which is locally favorable to one point may harm other points (its almost as if the data points are competing with each other)
- Can we reduce the oscillations by improving our stochastic estimates of the gradient



- We see many oscillations. Why ? Because we are making greedy decisions.
- Each point is trying to push the parameters in a direction most favorable to it (without being aware of how this affects other points)
- A parameter update which is locally favorable to one point may harm other points (its almost as if the data points are competing with each other)
- Can we reduce the oscillations by improving our stochastic estimates of the gradient (currently estimated from just 1 data point at a time)



- We see many oscillations. Why ? Because we are making greedy decisions.
- Each point is trying to push the parameters in a direction most favorable to it (without being aware of how this affects other points)
- A parameter update which is locally favorable to one point may harm other points (its almost as if the data points are competing with each other)
- Can we reduce the oscillations by improving our stochastic estimates of the gradient (currently estimated from just 1 data point at a time)
- Yes, let's look at mini-batch gradient descent



```python
def do_mini_batch_gradient_descent() :
    w, b, eta =-2, -2, 1.0
    mini_batch_size, num_points_seen = 2, 0
    for i in range(max_epochs) :
        dw, db, num_points = 0, 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points_seen +=1

            if num_points_seen % mini_batch_size == 0 :
                # seen one mini_batch
                w = w - eta * dw
                b = b - eta * db
                dw, db = 0, 0 #reset gradients
```

```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

- Notice that the algorithm updates the parameters after it sees *mini_batch_size* number of data points

```python
def do_mini_batch_gradient_descent() :
    w, b, eta =-2, -2, 1.0
    mini_batch_size, num_points_seen = 2, 0
    for i in range(max_epochs) :
        dw, db, num_points = 0, 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points_seen +=1

            if num_points_seen % mini_batch_size == 0 :
                # seen one mini_batch
                w = w - eta * dw
                b = b - eta * db
                dw, db = 0, 0 #reset gradients
```

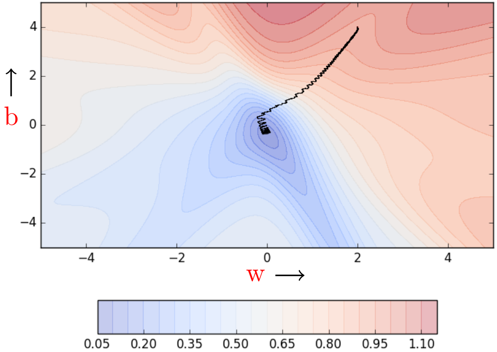
```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

- Notice that the algorithm updates the parameters after it sees *mini_batch_size* number of data points
- The stochastic estimates are now slightly better

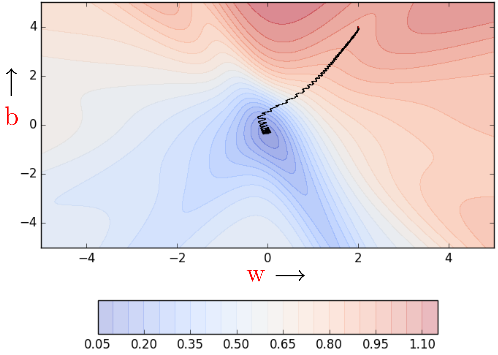
```python
def do_mini_batch_gradient_descent() :
    w, b, eta =-2, -2, 1.0
    mini_batch_size, num_points_seen = 2, 0
    for i in range(max_epochs) :
        dw, db, num_points = 0, 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points_seen +=1

            if num_points_seen % mini_batch_size == 0 :
                # seen one mini_batch
                w = w - eta * dw
                b = b - eta * db
                dw, db = 0, 0 #reset gradients
```

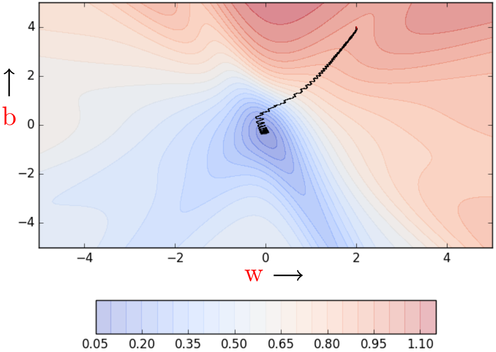
```python
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

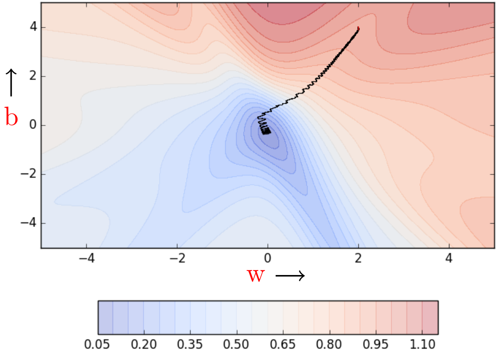
- Notice that the algorithm updates the parameters after it sees *mini_batch_size* number of data points
- The stochastic estimates are now slightly better
- Let's see this algorithm in action when we have k = 2

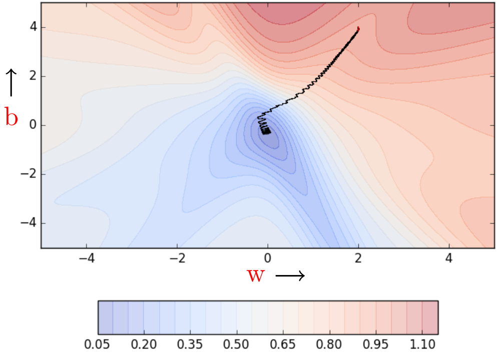


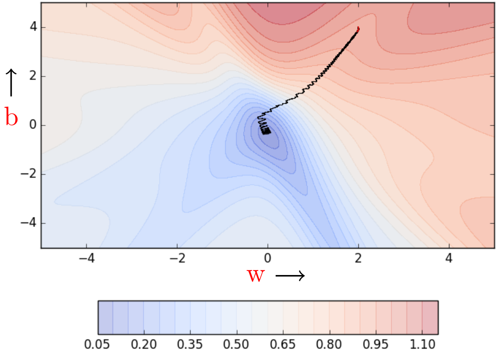
b
w
4
2
0
−2
−4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

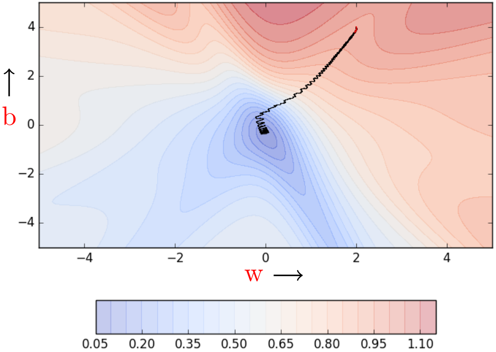

4
2
0
−2
−4
−4
−2
0
2
4
b
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

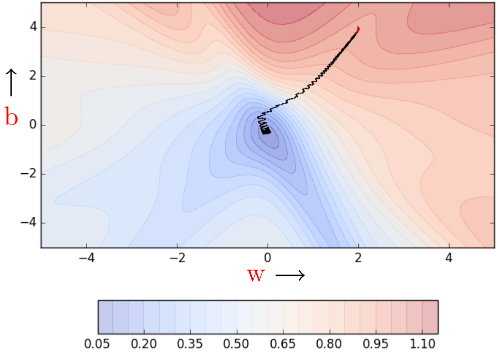

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

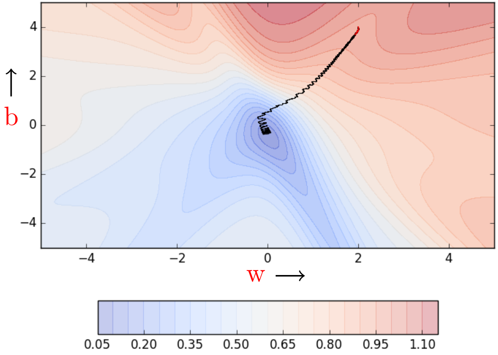

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10



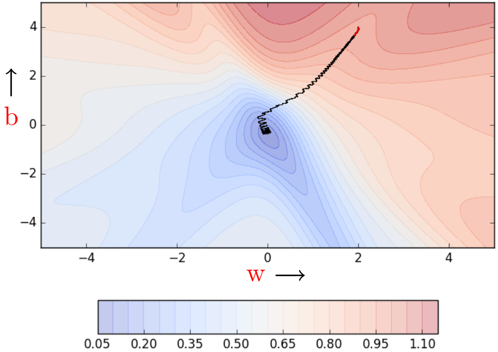
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

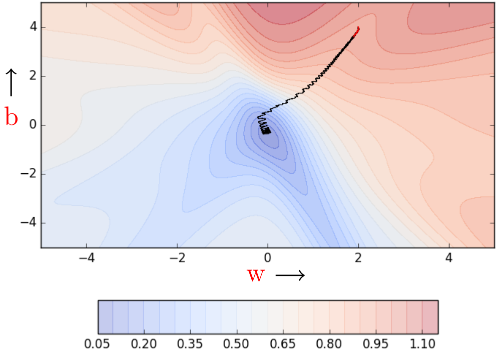


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

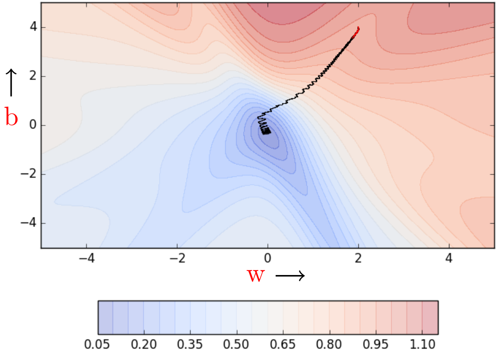


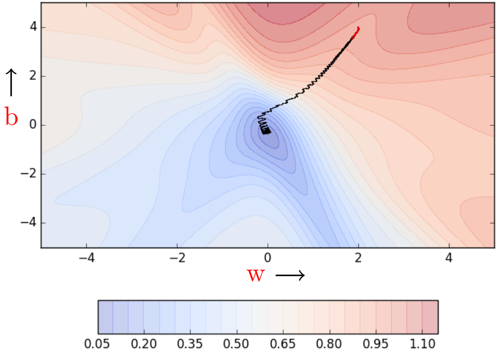
b
4
2
0
−2
−4
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

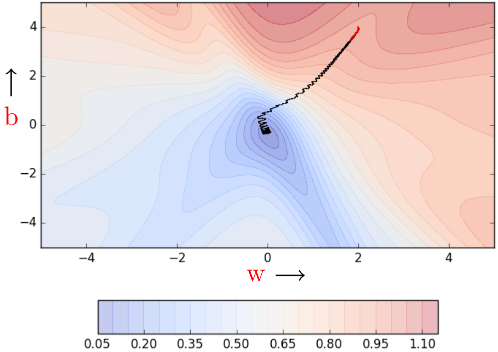

4
2
0
−2
−4
−4
−2
0
2
4
b
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

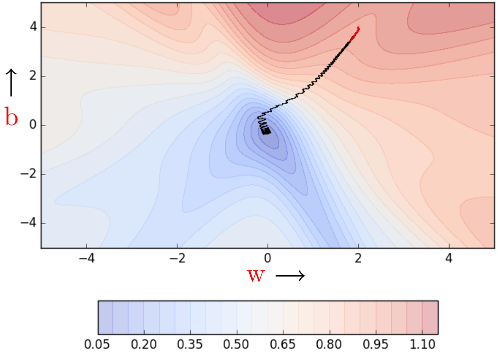


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

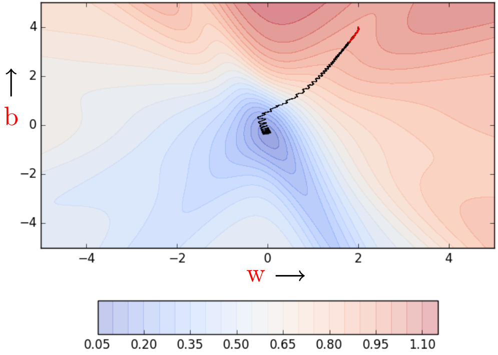

4
2
0
−2
−4
−4
−2
0
2
4
b
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

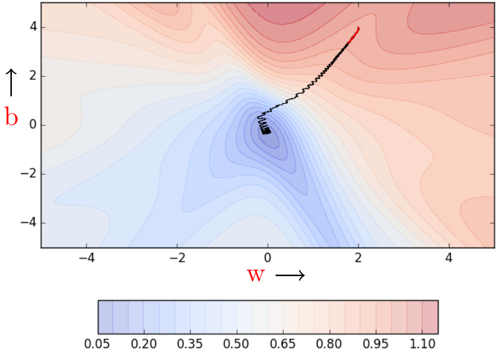

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

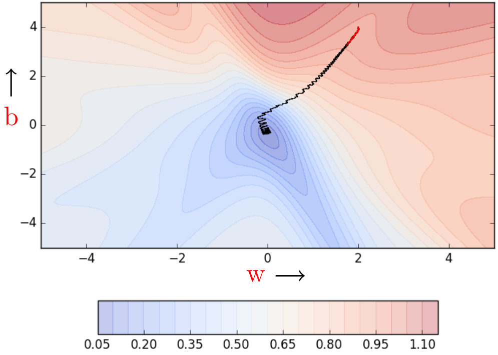

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10



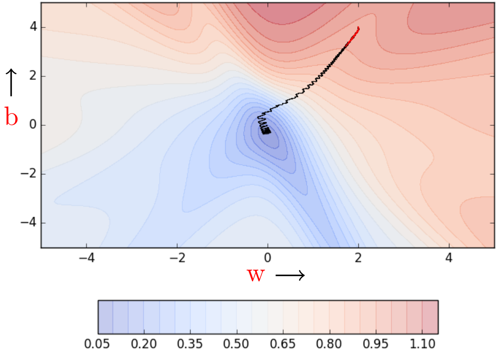
b
w
−4
−2
0
2
4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

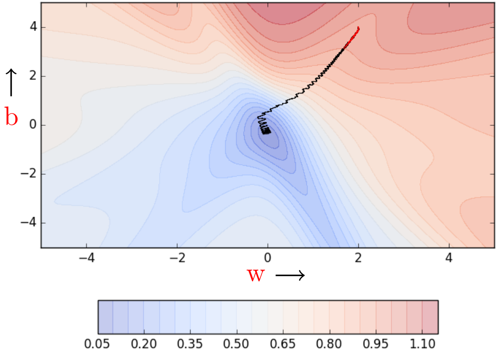

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

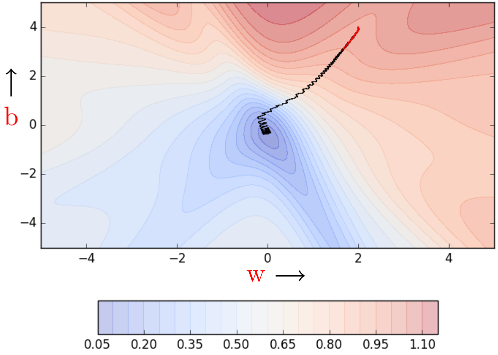

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

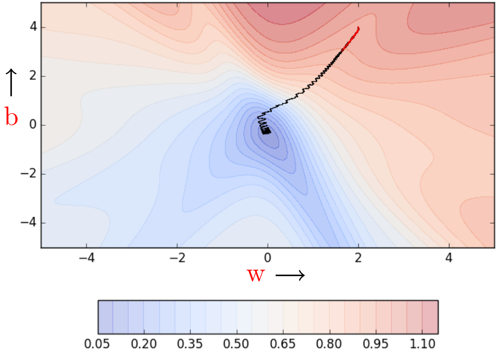


4
2
0
−2
−4
−4
−2
0
2
4
b
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

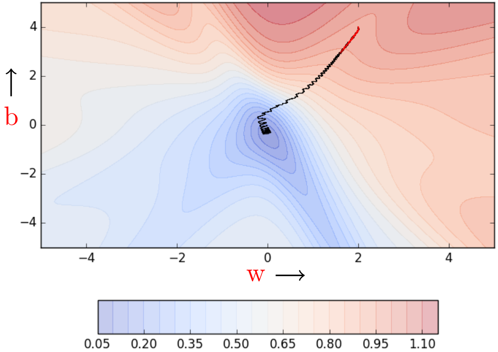

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

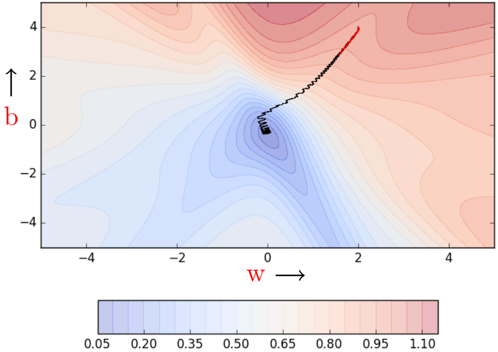

4
2
0
−2
−4
−4
−2
0
2
4
b
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

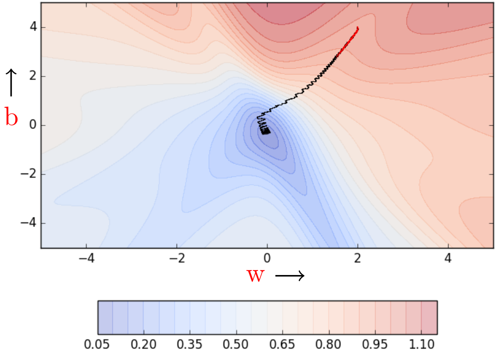


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

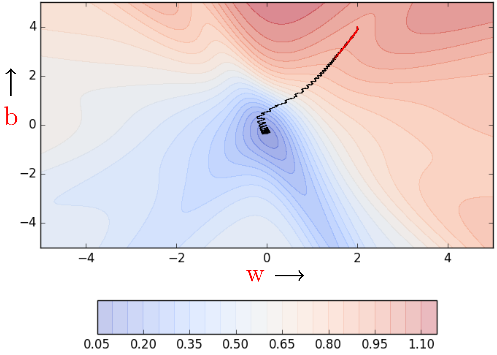


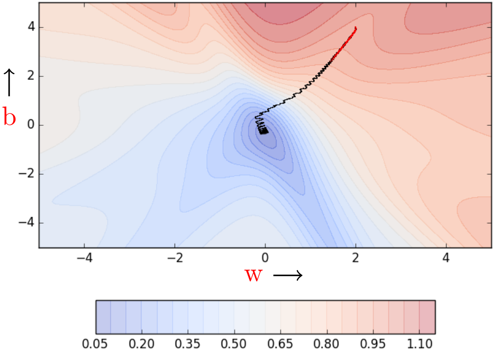
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

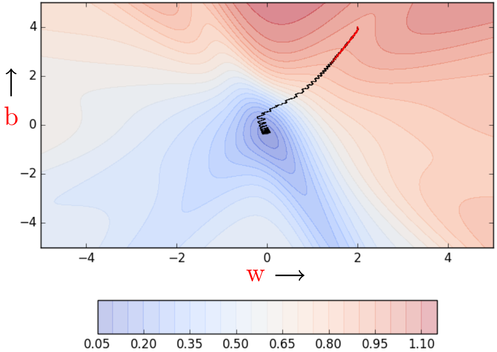

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

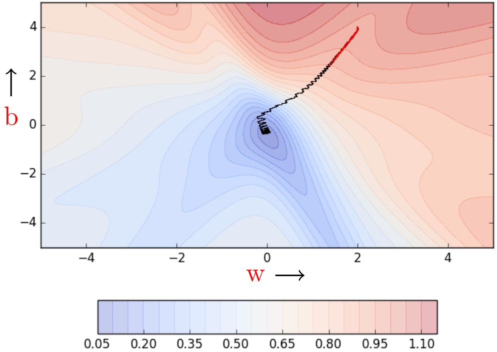


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

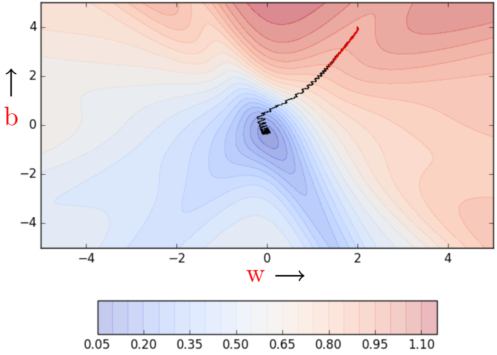

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

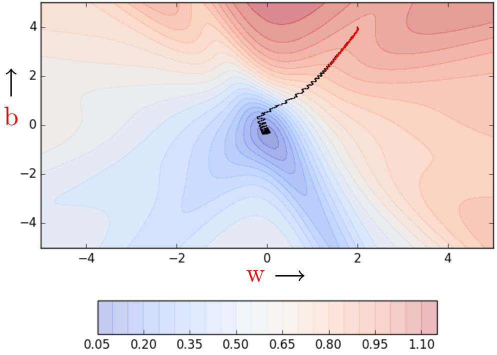

b
4
2
0
−2
−4
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

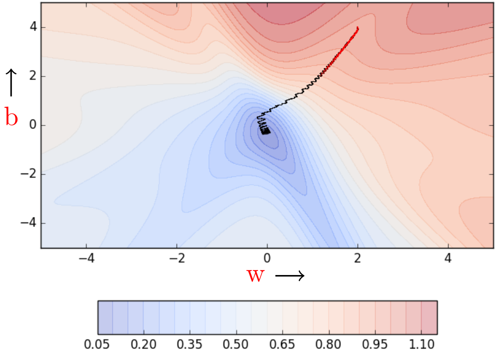


4
2
0
−2
−4
−4
−2
0
2
4
b
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

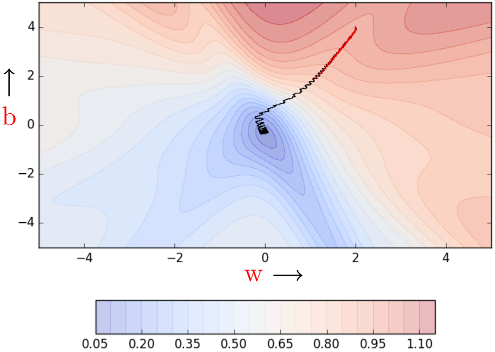


b
w
4
2
0
−2
−4
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

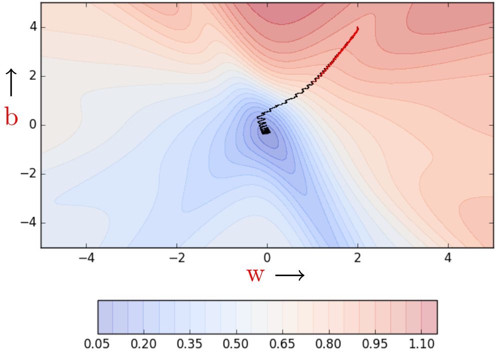

4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10



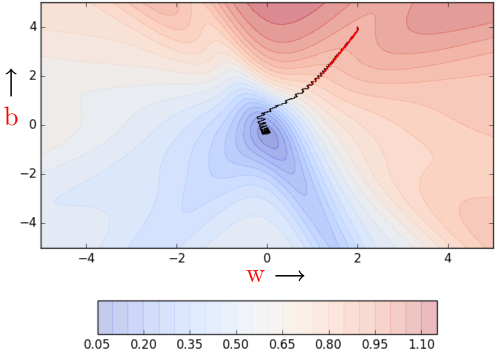
4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10


4
2
0
−2
−4
b
−4
−2
0
2
4
w
0.05
0.20
0.35
0.50
0.65
0.80
0.95
1.10

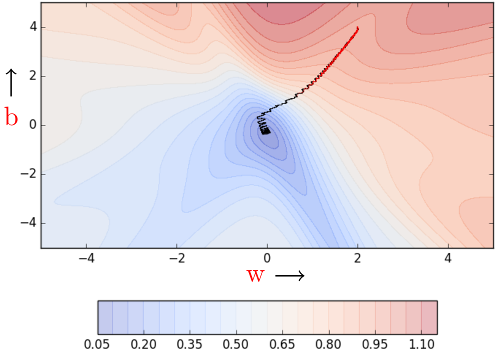
- Even with a batch size of k=2 the oscillations have reduced slightly.



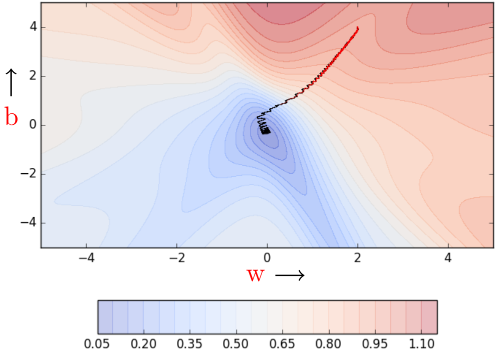
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?



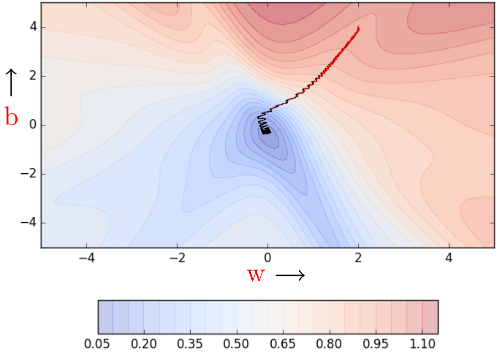
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient



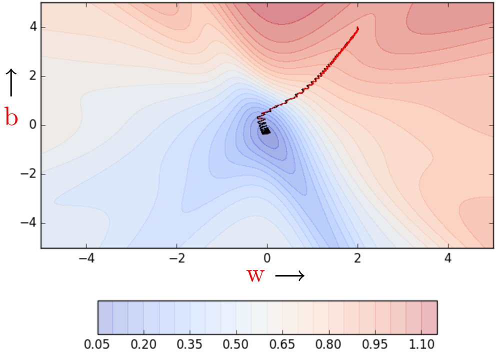
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]



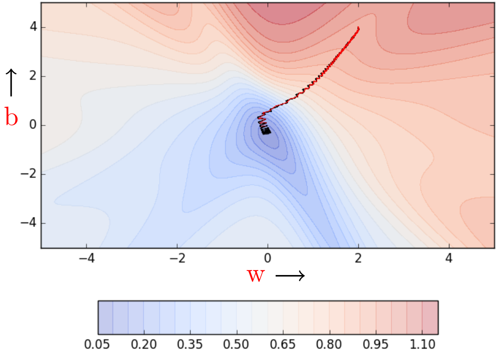
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]
- The higher the value of k the more accurate are the estimates



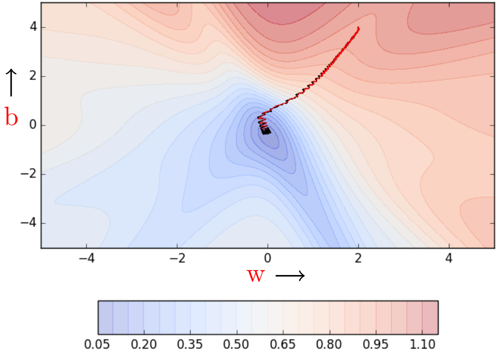
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]
- The higher the value of k the more accurate are the estimates
- In practice, typical values of k are 16, 32, 64



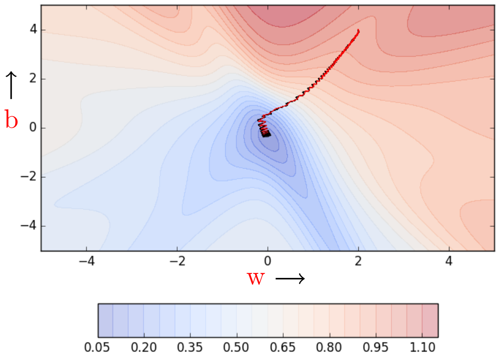
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]
- The higher the value of k the more accurate are the estimates
- In practice, typical values of k are 16, 32, 64
- Of course, there are still oscillations and they will always be there as long as we are using an approximate gradient as opposed to the true gradient



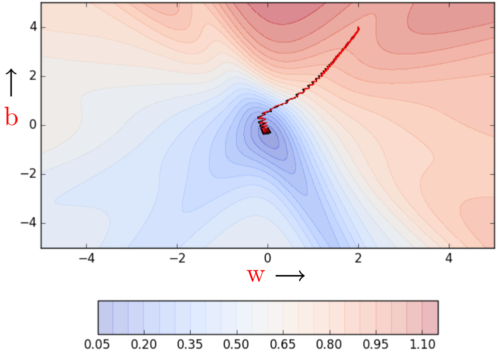
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]
- The higher the value of k the more accurate are the estimates
- In practice, typical values of k are 16, 32, 64
- Of course, there are still oscillations and they will always be there as long as we are using an approximate gradient as opposed to the true gradient



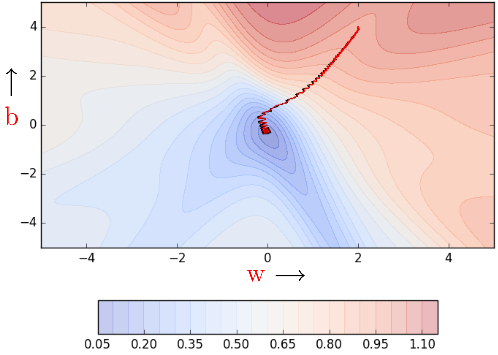
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]
- The higher the value of k the more accurate are the estimates
- In practice, typical values of k are 16, 32, 64
- Of course, there are still oscillations and they will always be there as long as we are using an approximate gradient as opposed to the true gradient



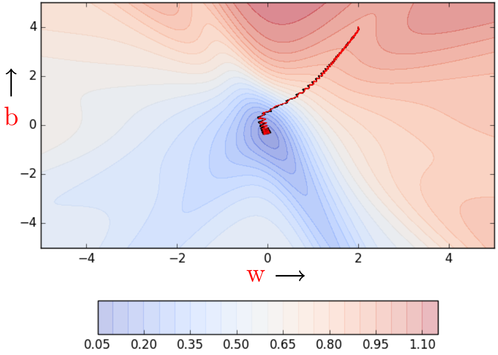
- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]
- The higher the value of k the more accurate are the estimates
- In practice, typical values of k are 16, 32, 64
- Of course, there are still oscillations and they will always be there as long as we are using an approximate gradient as opposed to the true gradient



- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]
- The higher the value of k the more accurate are the estimates
- In practice, typical values of k are 16, 32, 64
- Of course, there are still oscillations and they will always be there as long as we are using an approximate gradient as opposed to the true gradient



- Even with a batch size of k=2 the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin k=2 times to estimate P(heads)]
- The higher the value of k the more accurate are the estimates
- In practice, typical values of k are 16, 32, 64
- Of course, there are still oscillations and they will always be there as long as we are using an approximate gradient as opposed to the true gradient



Some things to remember ....

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	
Stochastic Gradient Descent	
Mini-Batch Gradient Descent	

Some things to remember ....

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	
Mini-Batch Gradient Descent	

Some things to remember ....

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	

Some things to remember ....

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	$\frac{N}{B}$

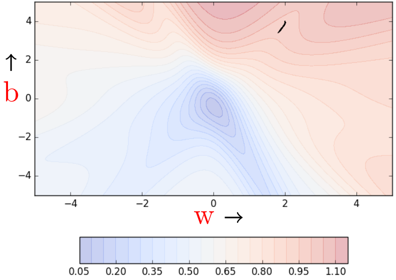
Similarly, we can have stochastic versions of Momentum based gradient descent and Nesterov accelerated based gradient descent

```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```

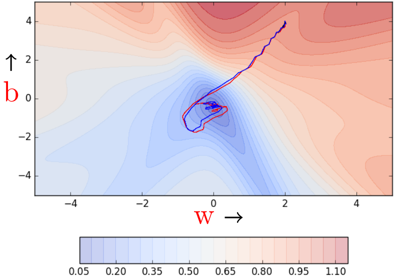
```python
def do_stochastic_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)

            v_w = gamma * prev_v_w + eta* dw
            v_b = gamma * prev_v_b + eta* db
            w = w - v_w
            b = b - v_b
            prev_v_w = v_w
            prev_v_b = v_b
```

```python
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```

```python
def do_nesterov_accelerated_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            #do partial updates
            v_w = gamma * prev_v_w
            v_b = gamma * prev_v_b
            #calculate gradients after partial update
            dw = grad_w(w - v_w, b - v_b, x, y)
            db = grad_b(w - v_w, b - v_b, x, y)

            v_w = gamma * prev_v_w + eta * dw
            v_b = gamma * prev_v_b + eta * db
            w = w - v_w
            b = b - v_b
            prev_v_w = v_w
            prev_v_b = v_b
```

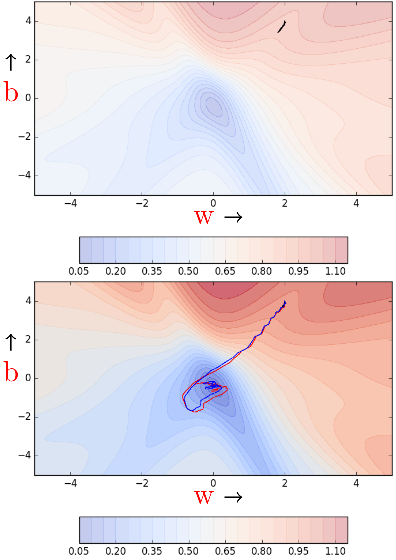


b
w
0.05 0.20 0.35 0.50 0.65 0.80 0.95 1.10

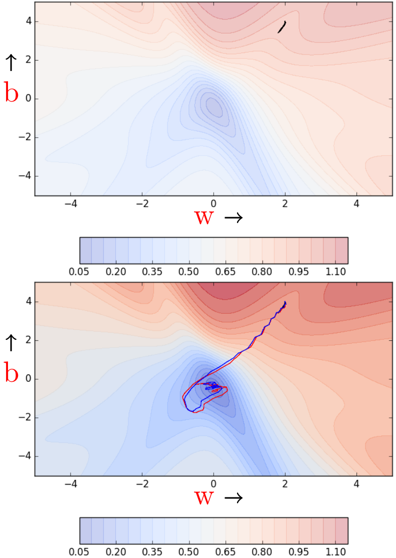


b
w
0.05 0.20 0.35 0.50 0.65 0.80 0.95 1.10

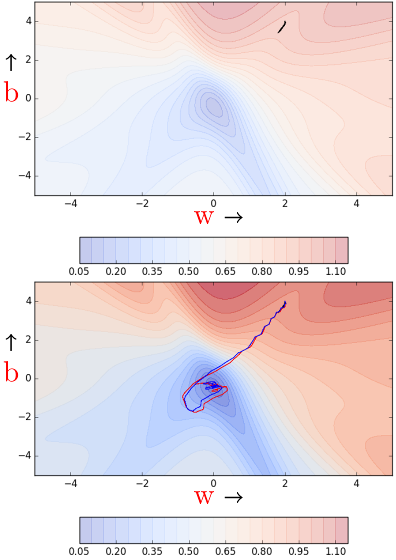
- While the stochastic versions of both Momentum [blue] and NAG [red] exhibit oscillations the relative advantage of NAG over Momentum still holds



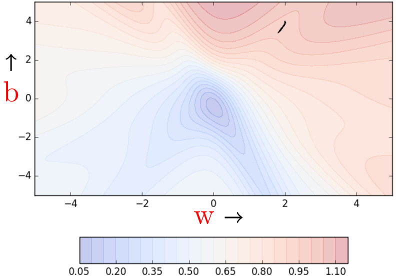
- While the stochastic versions of both Momentum [blue] and NAG [red] exhibit oscillations the relative advantage of NAG over Momentum still holds (i.e., NAG takes relatively shorter u-turns)

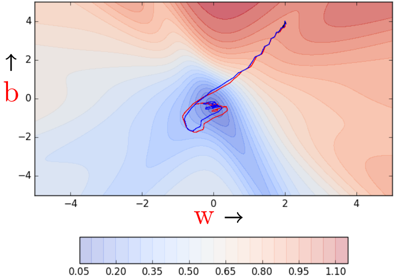


- While the stochastic versions of both Momentum [blue] and NAG [red] exhibit oscillations the relative advantage of NAG over Momentum still holds (i.e., NAG takes relatively shorter u-turns)
- Further both of them are faster than stochastic gradient descent



- While the stochastic versions of both Momentum [blue] and NAG [red] exhibit oscillations the relative advantage of NAG over Momentum still holds (i.e., NAG takes relatively shorter u-turns)
- Further both of them are faster than stochastic gradient descent (after 60 steps, stochastic gradient descent [black - top figure] still exhibits a very high error whereas NAG and Momentum are close to convergence)





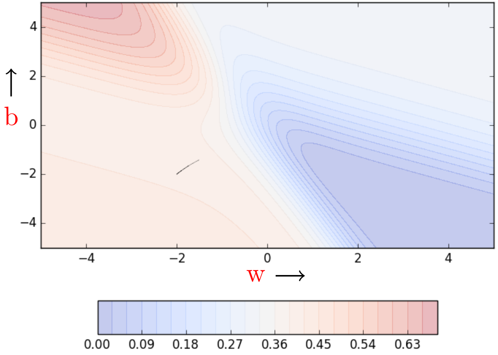
And, of course, you can also have the mini batch version of Momentum and NAG...

And, of course, you can also have the mini batch version of Momentum and NAG...I leave that as an exercise :-)

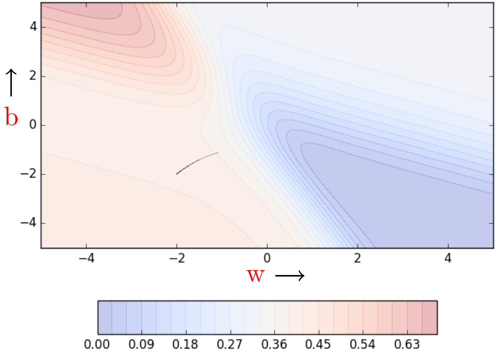
Module 5.7 : Tips for Adjusting learning Rate and Momentum

Before moving on to advanced optimization algorithms let us revisit the problem of learning rate in gradient descent

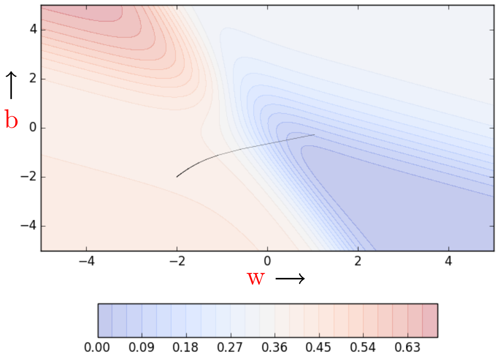
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)



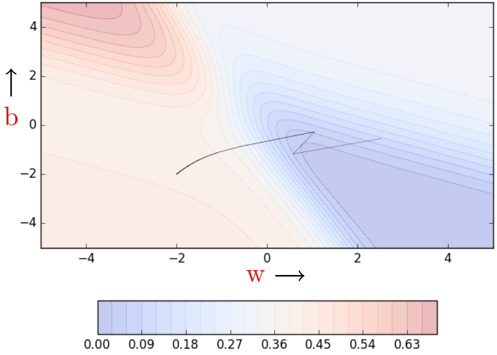
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



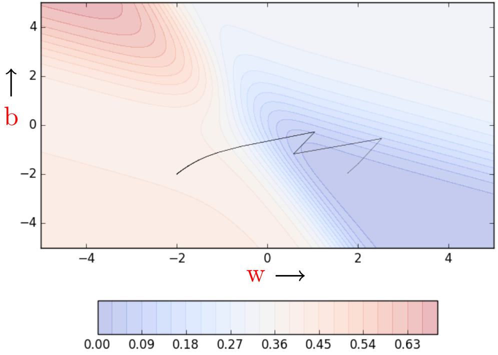
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



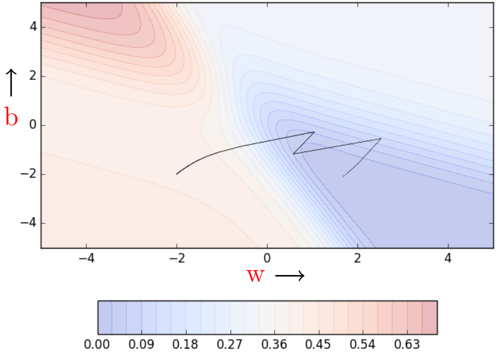
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



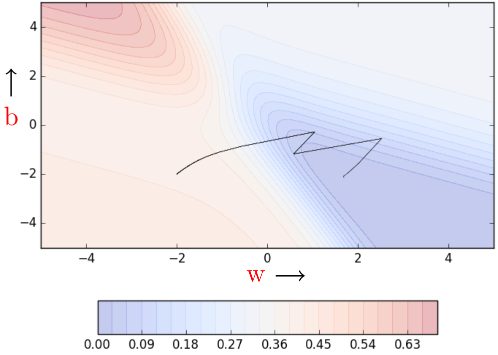
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



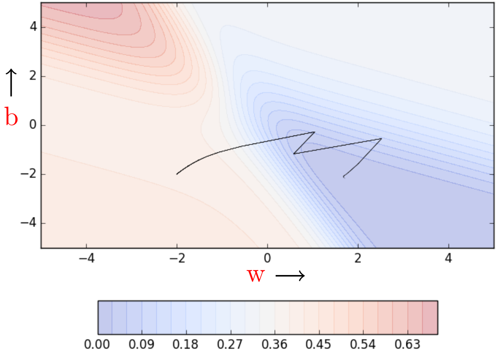
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



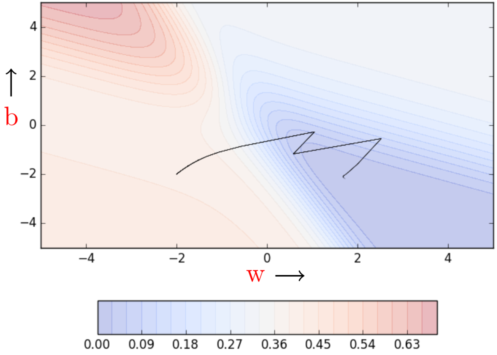
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



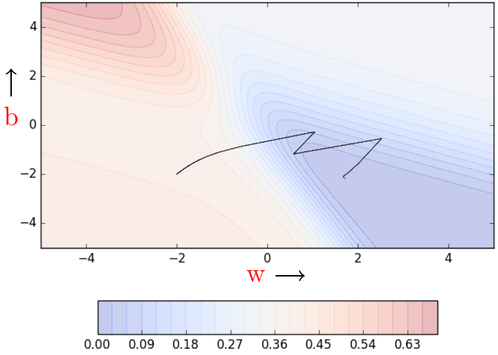
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



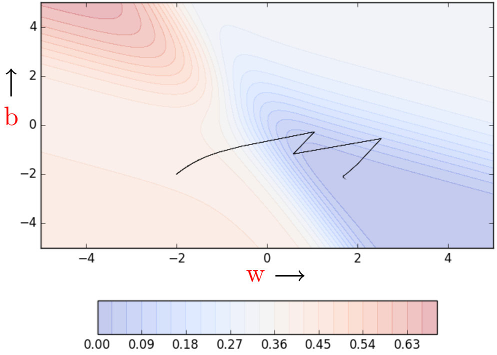
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



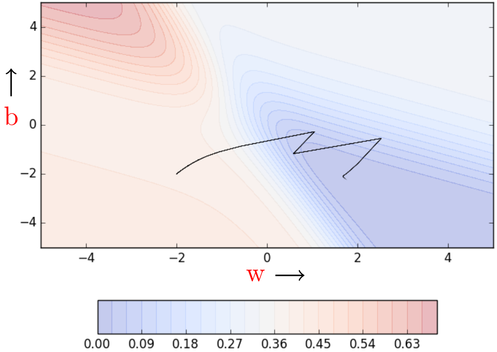
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



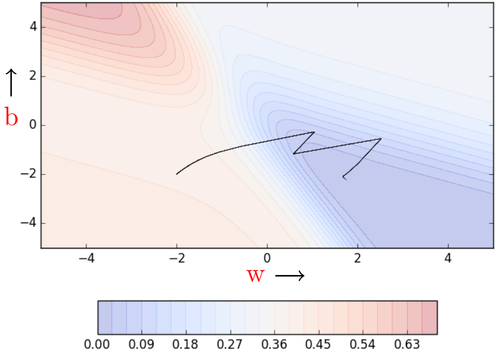
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



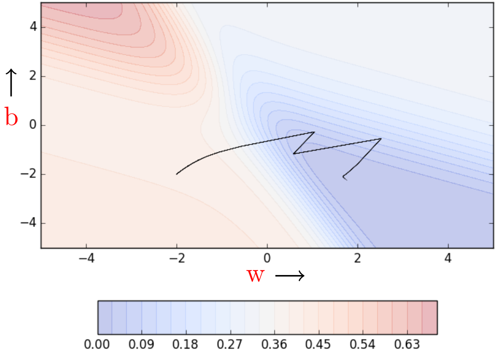
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



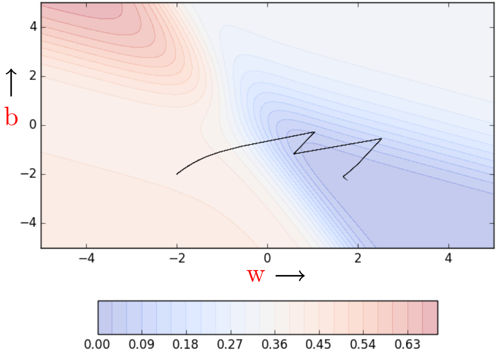
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



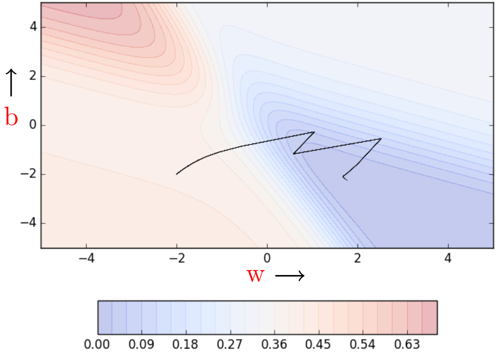
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



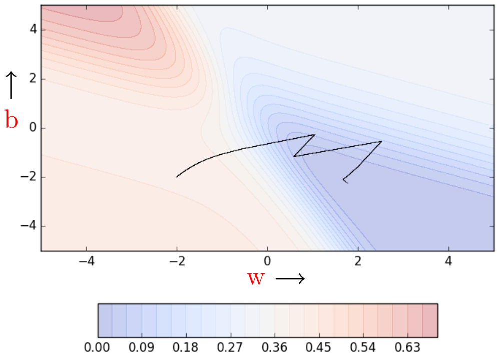
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



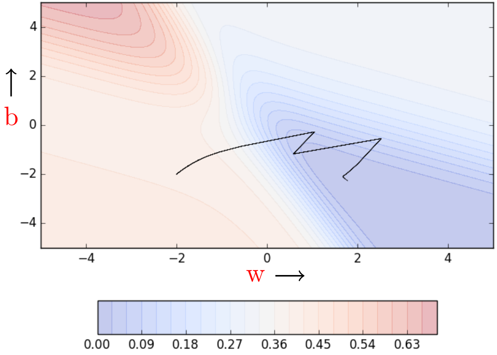
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



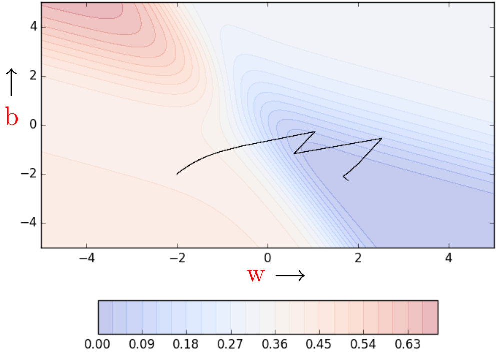
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10



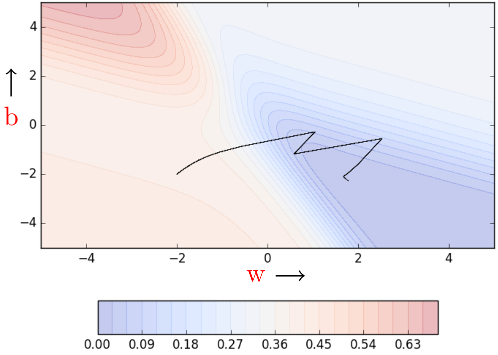
- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10
- On the regions which have a steep slope, the already large gradient blows up further



- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10
- On the regions which have a steep slope, the already large gradient blows up further
- It would be good to have a learning rate which could adjust to the gradient ...



- One could argue that we could have solved the problem of navigating gentle slopes by setting the learning rate high (i.e., blow up the small gradient by multiplying it with a large η)
- Let us see what happens if we set the learning rate to 10
- On the regions which have a steep slope, the already large gradient blows up further
- It would be good to have a learning rate which could adjust to the gradient ... we will see a few such algorithms soon



Tips for initial learning rate ?

Tips for initial learning rate ?

- Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1. 1.0]

Tips for initial learning rate ?

- Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1. 1.0]
- Run a few epochs with each of these and figure out a learning rate which works best

Tips for initial learning rate ?

- Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1. 1.0]
- Run a few epochs with each of these and figure out a learning rate which works best
- Now do a finer search around this value [for example, if the best learning rate was 0.1 then now try some values around it: 0.05, 0.2, 0.3]

Tips for initial learning rate ?

- Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1. 1.0]
- Run a few epochs with each of these and figure out a learning rate which works best
- Now do a finer search around this value [for example, if the best learning rate was 0.1 then now try some values around it: 0.05, 0.2, 0.3]
- Disclaimer: these are just heuristics ... no clear winner strategy

Tips for annealing learning rate

Tips for annealing learning rate

- **Step Decay:**

Tips for annealing learning rate

- **Step Decay:**
  - Halve the learning rate after every 5 epochs or

Tips for annealing learning rate

- **Step Decay:**
  - Halve the learning rate after every 5 epochs or
  - Halve the learning rate after an epoch if the validation error is more than what it was at the end of the previous epoch

Tips for annealing learning rate

- **Step Decay:**
  - Halve the learning rate after every 5 epochs or
  - Halve the learning rate after an epoch if the validation error is more than what it was at the end of the previous epoch
- **Exponential Decay:** $\eta = \eta_0^{-kt}$ where η_0 and k are hyperparameters and t is the step number

Tips for annealing learning rate

- **Step Decay:**
  - Halve the learning rate after every 5 epochs or
  - Halve the learning rate after an epoch if the validation error is more than what it was at the end of the previous epoch
- **Exponential Decay:** $\eta = \eta_0^{-kt}$ where η_0 and k are hyperparameters and t is the step number
- **1/t Decay:** $\eta = \frac{\eta_0}{1+kt}$ where η_0 and k are hyperparameters and t is the step number

Tips for momentum

- The following schedule was suggested by Sutskever *et. al.*, 2013

$$\mu_t = min(1 - 2^{-1-log_2(\lfloor t/250 \rfloor+1)}, \mu_{max})$$

where, μ_{max} was chosen from $\{0.999, 0.995, 0.99, 0.9, 0\}$

Module 5.8 : Line Search

Just one last thing before we move on to some other algorithms ...

- In practice, often a line search is done to find a relatively better value of η

- In practice, often a line search is done to find a relatively better value of η

```
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```

- In practice, often a line search is done to find a relatively better value of η
- Update w using different values of η

```python
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```

- In practice, often a line search is done to find a relatively better value of η
- Update w using different values of η
- Now retain that updated value of w which gives the lowest loss

```
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```

- In practice, often a line search is done to find a relatively better value of η
- Update w using different values of η
- Now retain that updated value of w which gives the lowest loss
- Esentially at each step we are trying to use the best η value from the available choices

```python
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```

- In practice, often a line search is done to find a relatively better value of η
- Update w using different values of η
- Now retain that updated value of w which gives the lowest loss
- Esentially at each step we are trying to use the best η value from the available choices
- What's the flipside?

```python
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```

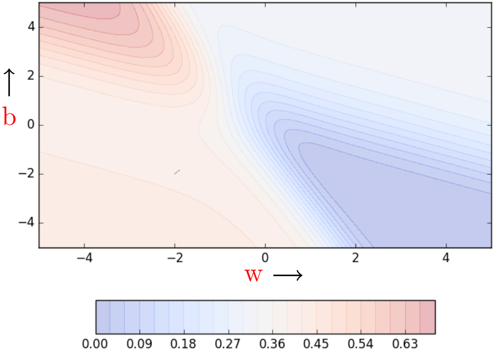
- In practice, often a line search is done to find a relatively better value of η
- Update w using different values of η
- Now retain that updated value of w which gives the lowest loss
- Esentially at each step we are trying to use the best η value from the available choices
- What's the flipside? We are doing many more computations in each step

```python
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```

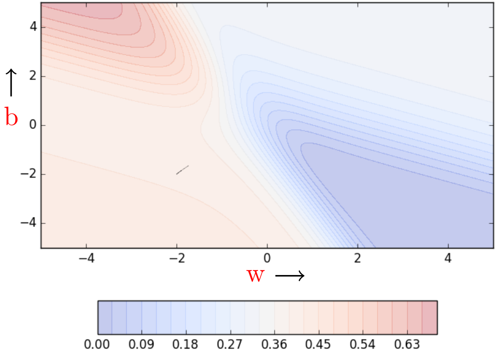
- In practice, often a line search is done to find a relatively better value of η
- Update w using different values of η
- Now retain that updated value of w which gives the lowest loss
- Esentially at each step we are trying to use the best η value from the available choices
- What's the flipside? We are doing many more computations in each step
- We will come back to this when we talk about second order optimization methods

```python
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```

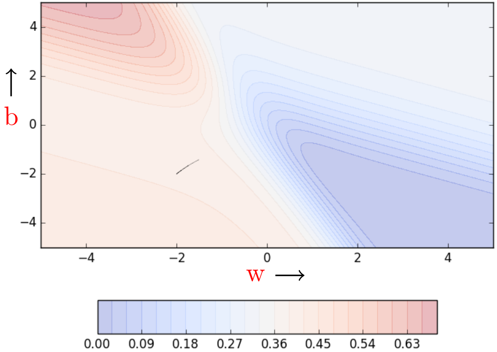
- Let us see line search in action



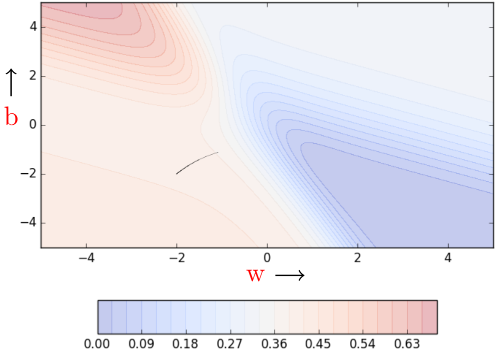
- Let us see line search in action



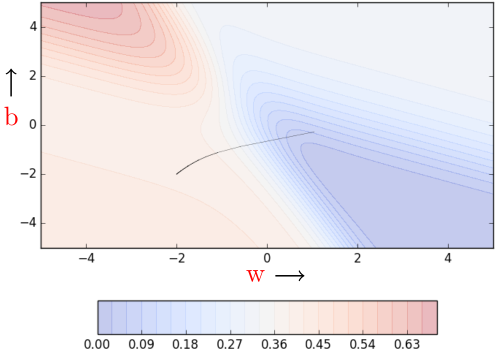
- Let us see line search in action



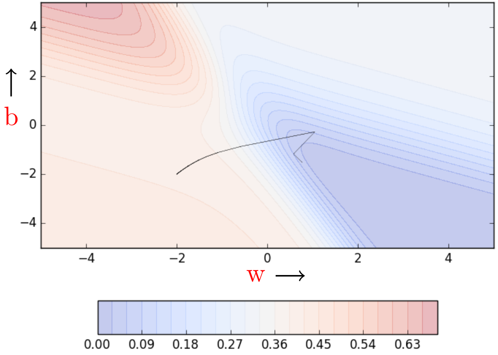
- Let us see line search in action



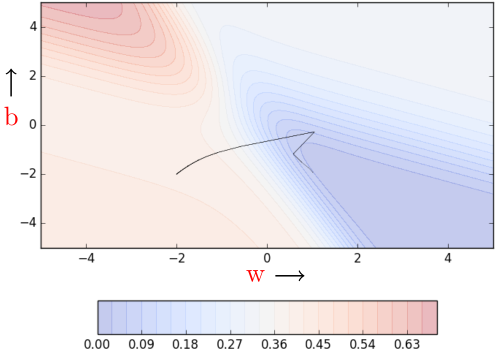
- Let us see line search in action



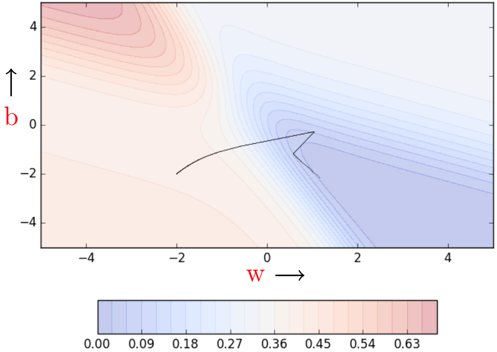
- Let us see line search in action



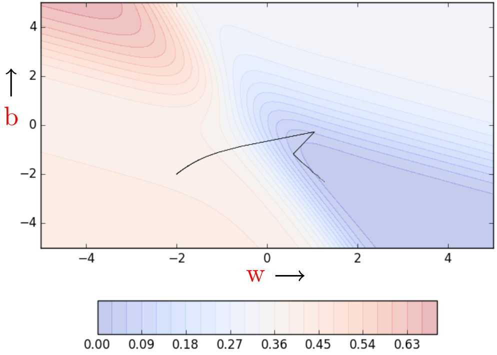
- Let us see line search in action



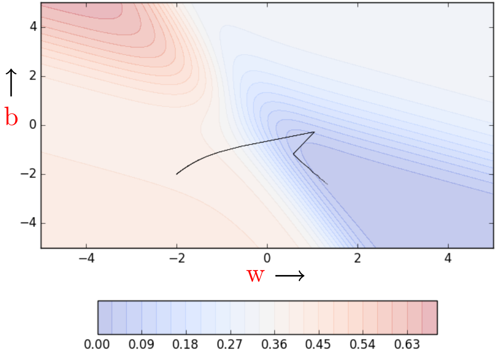
- Let us see line search in action



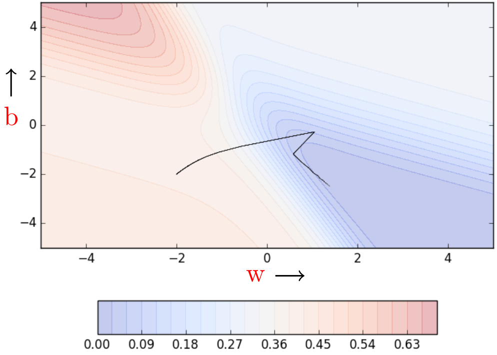
- Let us see line search in action



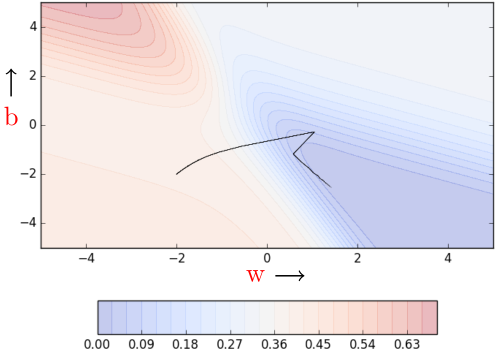
- Let us see line search in action



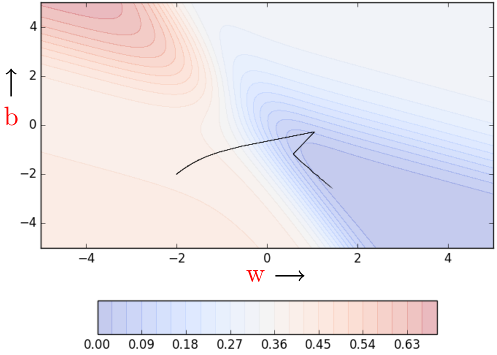
- Let us see line search in action



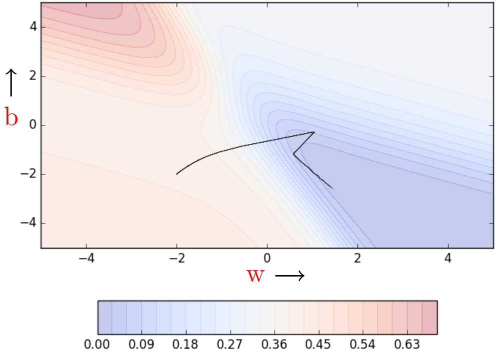
- Let us see line search in action



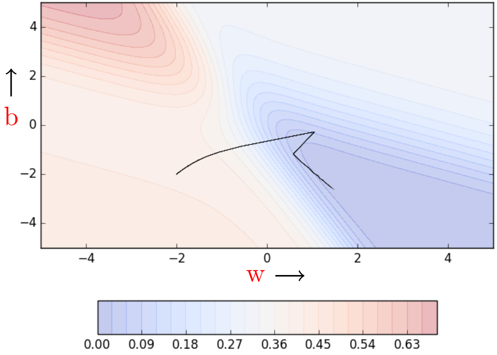
- Let us see line search in action



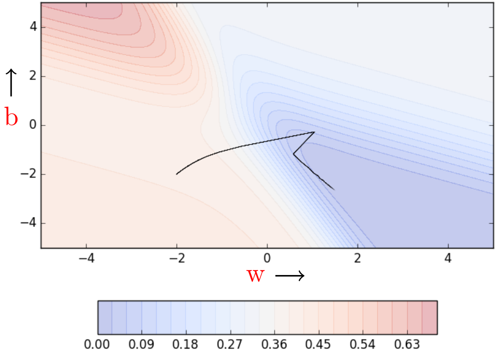
- Let us see line search in action



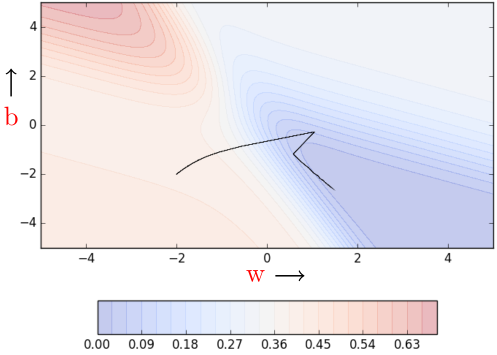
- Let us see line search in action



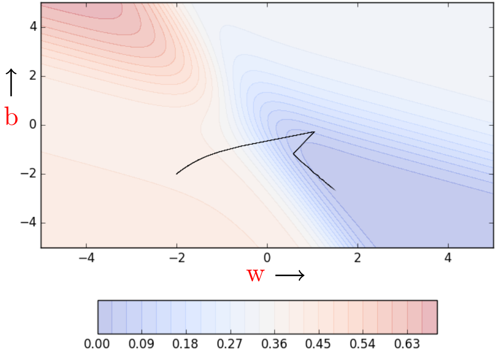
- Let us see line search in action



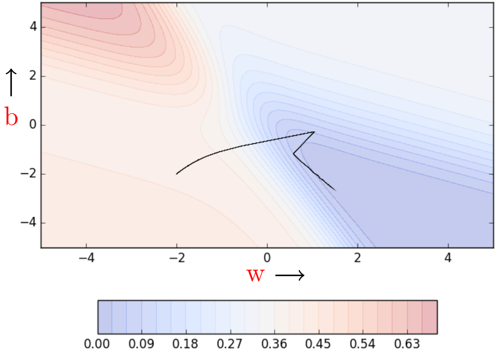
- Let us see line search in action
- Convergence is faster than vanilla gradient descent



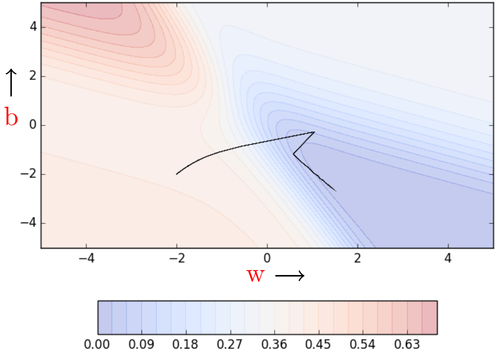
- Let us see line search in action
- Convergence is faster than vanilla gradient descent
- We see some oscillations,



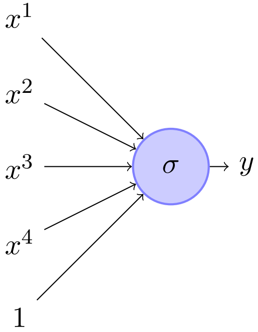
- Let us see line search in action
- Convergence is faster than vanilla gradient descent
- We see some oscillations, but note that these oscillations are different from what we see in momentum and NAG



- Let us see line search in action
- Convergence is faster than vanilla gradient descent
- We see some oscillations, but note that these oscillations are different from what we see in momentum and NAG



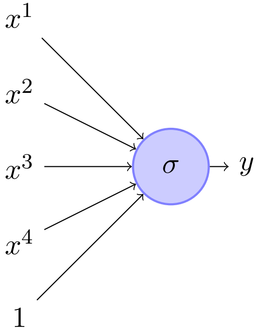
Module 5.9 : Gradient Descent with Adaptive Learning Rate



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

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

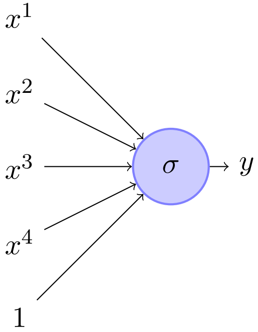


$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point $(\mathbf{x}, y)$...

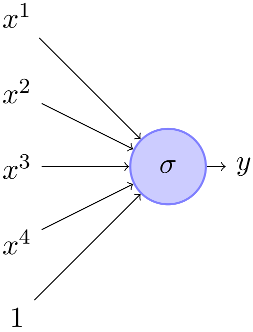


$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point $(\mathbf{x}, y)$...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$ ... so on

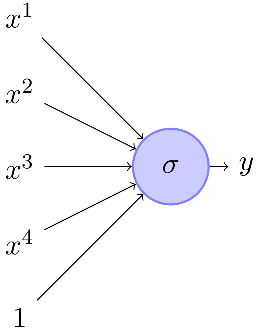


$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point $(\mathbf{x}, y)$...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$ ... so on
- If there are n points, we can just sum the gradients over all the n points to get the total gradient

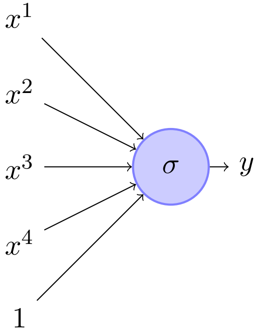


$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point $(\mathbf{x}, y)$...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$ ... so on
- If there are n points, we can just sum the gradients over all the n points to get the total gradient
- What happens if the feature x^2 is very sparse?

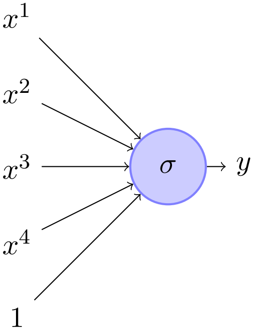


$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point $(\mathbf{x}, y)$...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$ ... so on
- If there are n points, we can just sum the gradients over all the n points to get the total gradient
- What happens if the feature x^2 is very sparse? (*i.e.*, if its value is 0 for most inputs)

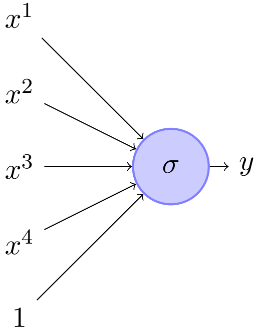


$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point $(\mathbf{x}, y)$...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$ ... so on
- If there are n points, we can just sum the gradients over all the n points to get the total gradient
- What happens if the feature x^2 is very sparse? (*i.e.*, if its value is 0 for most inputs)
- ∇w^2 will be 0 for most inputs (see formula) and hence w^2 will not get enough updates

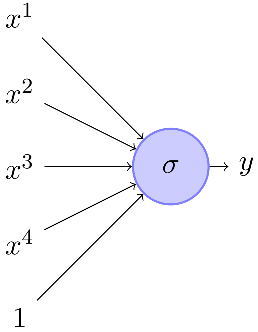


$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point $(\mathbf{x}, y)$...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$ ... so on
- If there are n points, we can just sum the gradients over all the n points to get the total gradient
- What happens if the feature x^2 is very sparse? (*i.e.*, if its value is 0 for most inputs)
- ∇w^2 will be 0 for most inputs (see formula) and hence w^2 will not get enough updates
- If x^2 happens to be sparse as well as important we would want to take the updates to w^2 more seriously



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

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point $(\mathbf{x}, y)$...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$ ... so on
- If there are n points, we can just sum the gradients over all the n points to get the total gradient
- What happens if the feature x^2 is very sparse? (*i.e.*, if its value is 0 for most inputs)
- ∇w^2 will be 0 for most inputs (see formula) and hence w^2 will not get enough updates
- If x^2 happens to be sparse as well as important we would want to take the updates to w^2 more seriously
- Can we have a different learning rate for each parameter which takes care of the frequency of features ?

Intuition

- Decay the learning rate for parameters in proportion to their update history

Intuition

- Decay the learning rate for parameters in proportion to their update history (more updates means more decay)

Intuition

- Decay the learning rate for parameters in proportion to their update history (more updates means more decay)

Update rule for Adagrad

$$v_t = v_{t-1} + (\nabla w_t)^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

... and a similar set of equations for b_t

- To see this in action we need to first create some data where one of the features is sparse

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ?

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ? Take some time to think about it

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ? Take some time to think about it
- Well, our network has just two parameters (w and b).

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ? Take some time to think about it
- Well, our network has just two parameters (w and b). Of these, the input/feature corresponding to b is always on

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ? Take some time to think about it
- Well, our network has just two parameters (w and b). Of these, the input/feature corresponding to b is always on (so can't really make it sparse)

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ? Take some time to think about it
- Well, our network has just two parameters (w and b). Of these, the input/feature corresponding to b is always on (so can't really make it sparse)
- The only option is to make x sparse

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

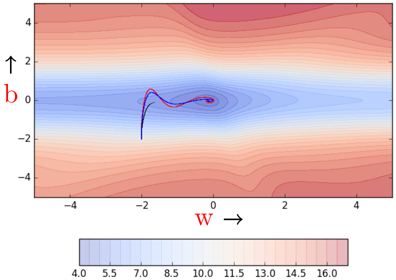
- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ? Take some time to think about it
- Well, our network has just two parameters (w and b). Of these, the input/feature corresponding to b is always on (so can't really make it sparse)
- The only option is to make x sparse
- **Solution:** We created 100 random (x, y) pairs and then for roughly 80% of these pairs we set x to 0

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

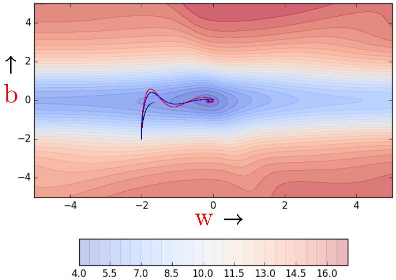
- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ? Take some time to think about it
- Well, our network has just two parameters (w and b). Of these, the input/feature corresponding to b is always on (so can't really make it sparse)
- The only option is to make x sparse
- **Solution:** We created 100 random (x, y) pairs and then for roughly 80% of these pairs we set x to 0 thereby, making the feature for w sparse

```python
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

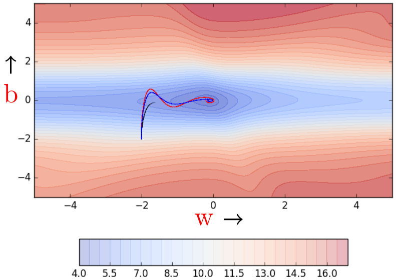
- GD (black), momentum (red) and NAG (blue)



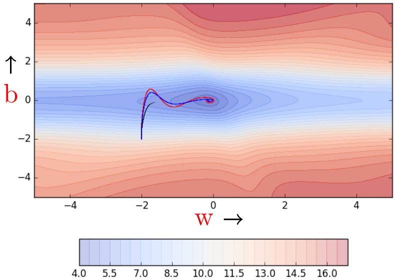
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset.



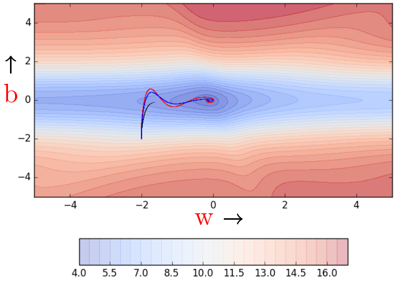
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?



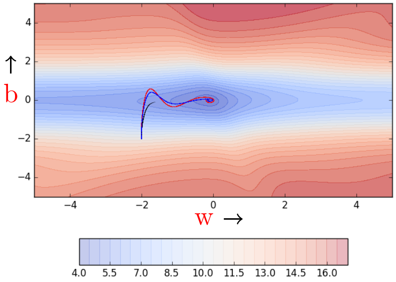
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical (b) axis and there is very little movement along the horizontal (w) axis



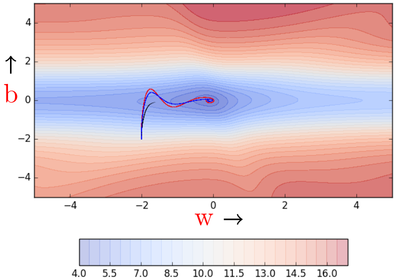
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical (b) axis and there is very little movement along the horizontal (w) axis
- Why?



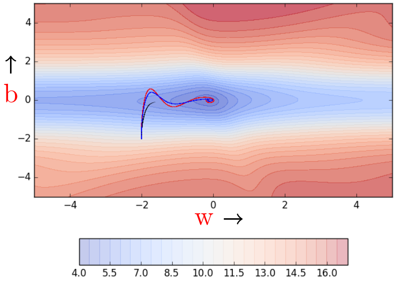
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical (b) axis and there is very little movement along the horizontal (w) axis
- Why? Because in our data, the feature corresponding to w is sparse and hence w undergoes very few updates



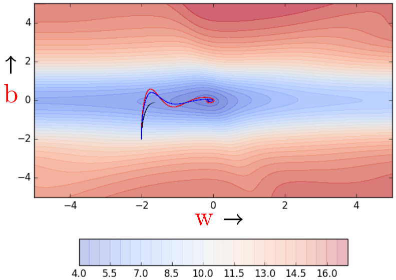
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical (b) axis and there is very little movement along the horizontal (w) axis
- Why? Because in our data, the feature corresponding to w is sparse and hence w undergoes very few updates ...on the other hand b is very dense and undergoes many updates



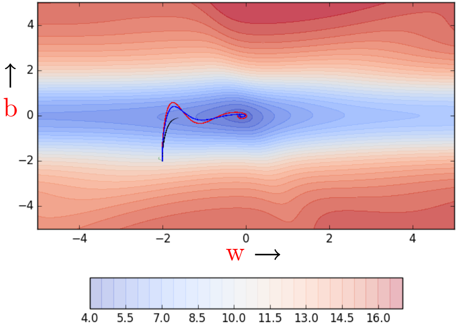
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical (b) axis and there is very little movement along the horizontal (w) axis
- Why? Because in our data, the feature corresponding to w is sparse and hence w undergoes very few updates ...on the other hand b is very dense and undergoes many updates
- Such sparsity is very common in large neural networks containing $1000s$ of input features and hence we need to address it



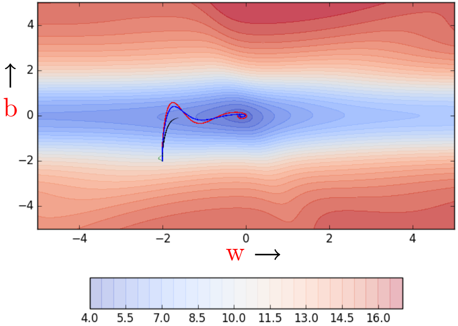
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical (b) axis and there is very little movement along the horizontal (w) axis
- Why? Because in our data, the feature corresponding to w is sparse and hence w undergoes very few updates ...on the other hand b is very dense and undergoes many updates
- Such sparsity is very common in large neural networks containing $1000s$ of input features and hence we need to address it



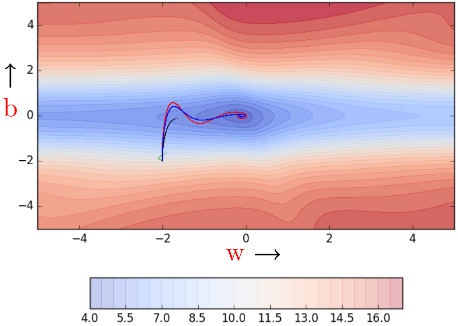
- Let's see what Adagrad does....



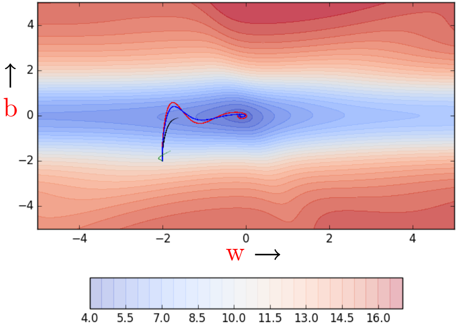
4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

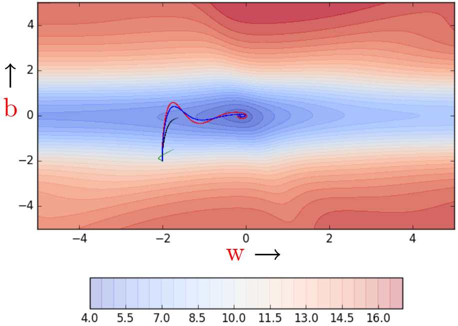


b
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

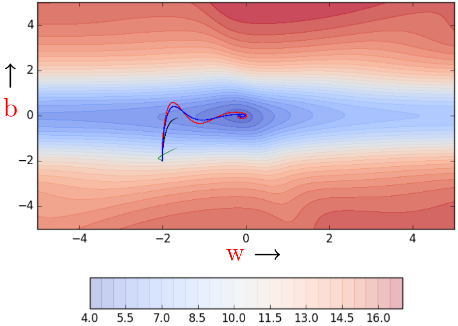


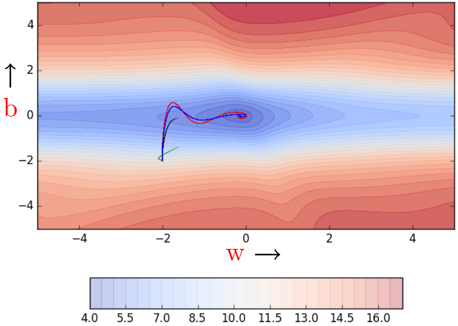
b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0


4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

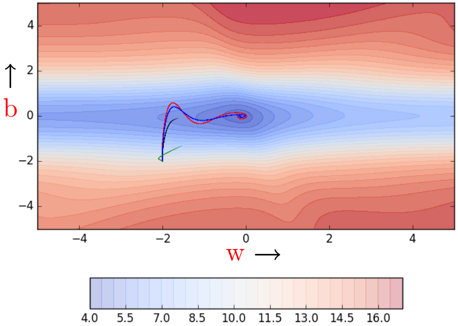


b
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

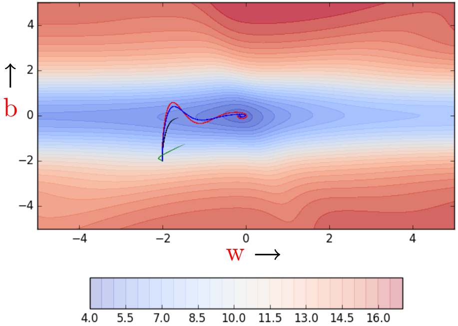

4
2
0
−2
−4
−4
−2
0
2
4
b
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

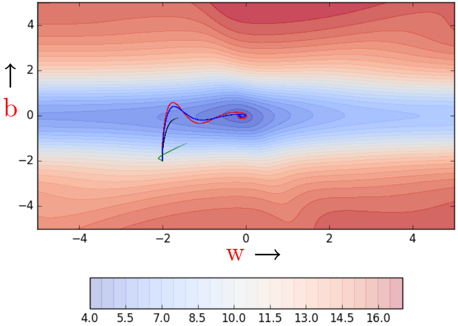


4
2
0
−2
−4
−4
−2
0
2
4
b
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

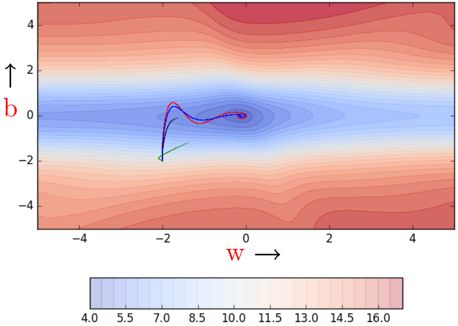


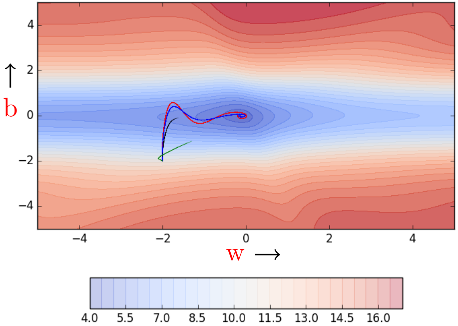
b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

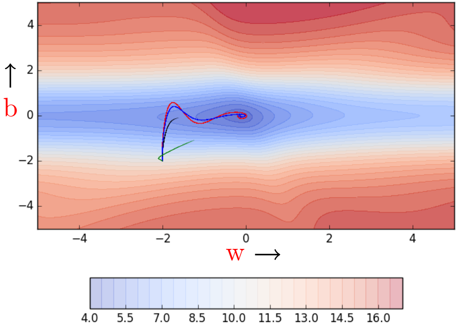

b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0 5.5 7.0 8.5 10.0 11.5 13.0 14.5 16.0

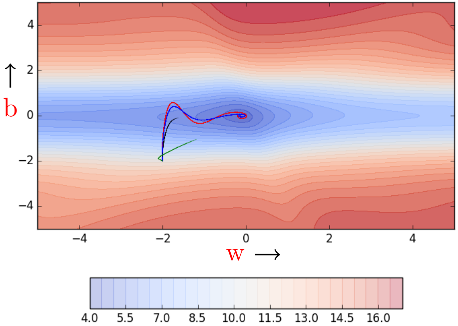


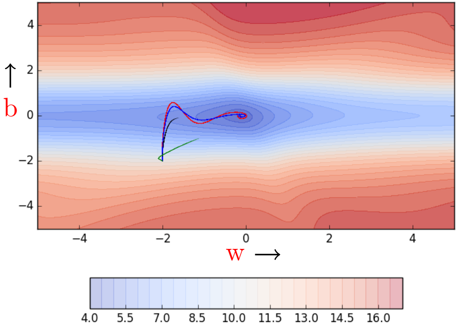
4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

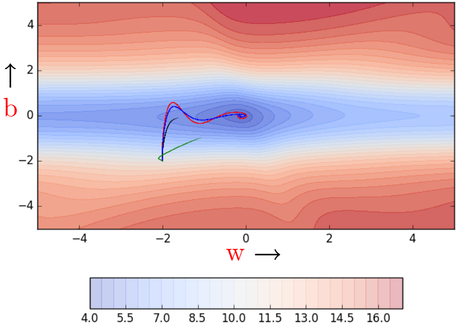

4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0


b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

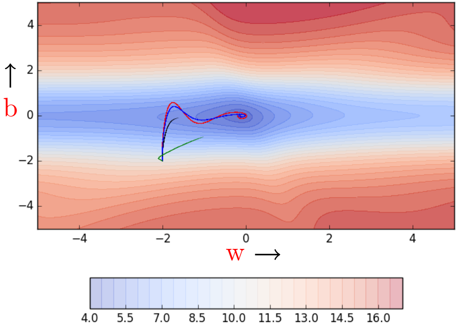

4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0


4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

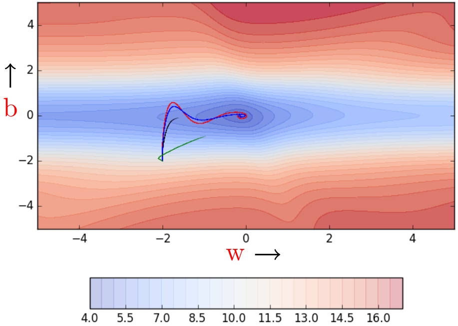

4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

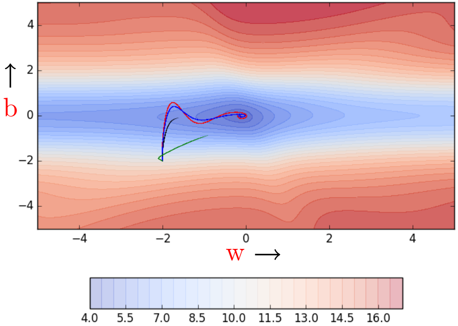


4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

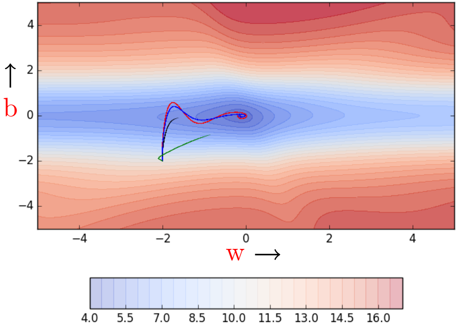


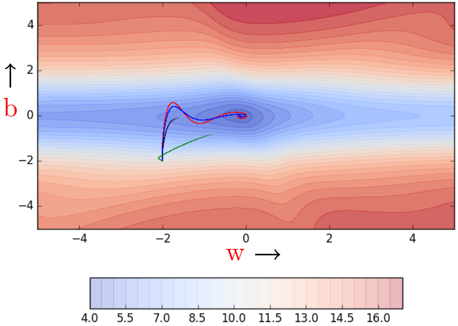
b
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0


4
2
0
−2
−4
−4
−2
0
2
4
b
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

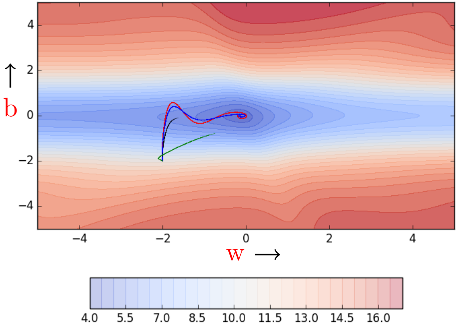


b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

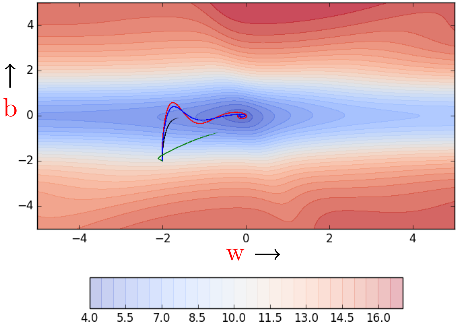

b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

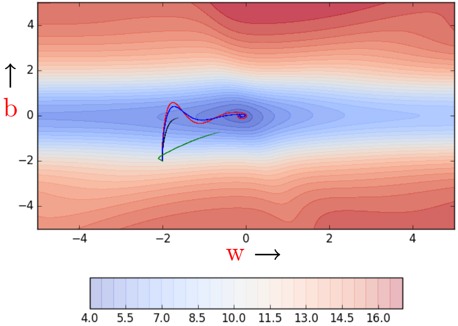


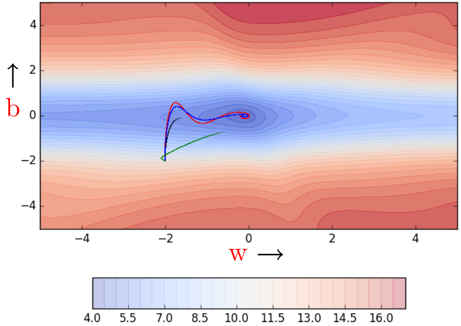
4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

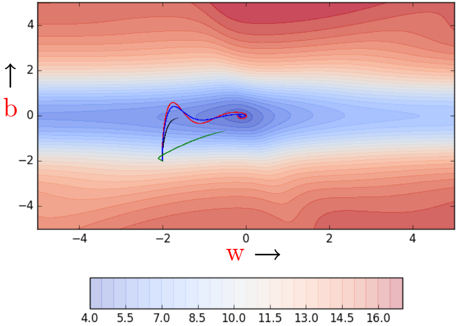


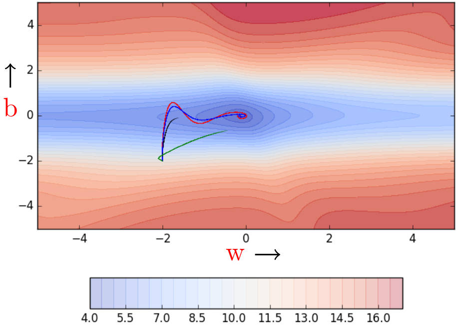
4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0


b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

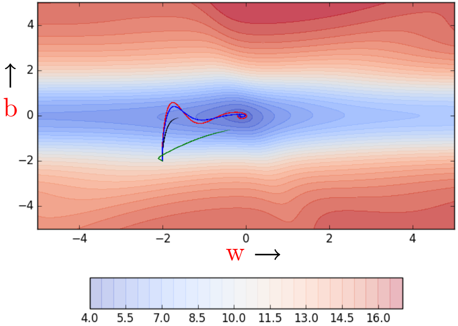

4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0


4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

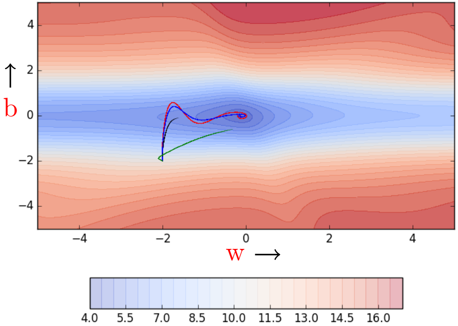

4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0



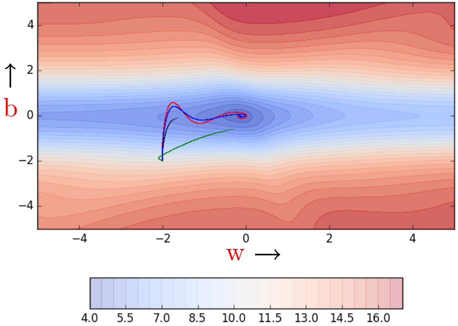
b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

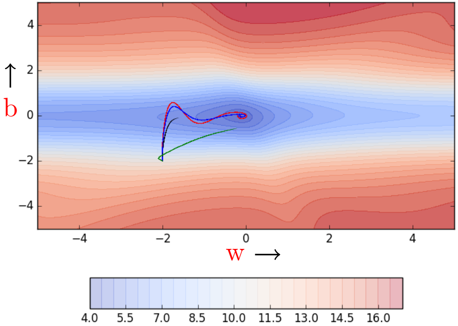


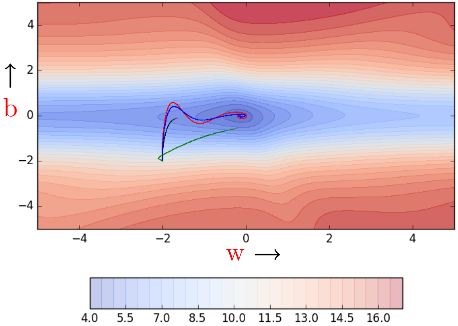
4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0



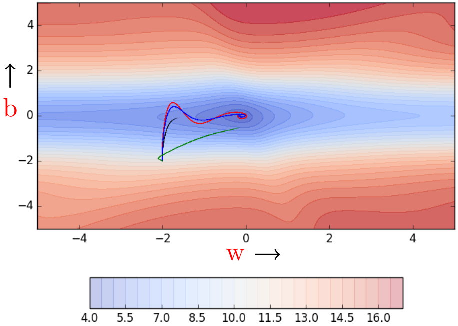
4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0


4
2
0
−2
−4
b
−4
−2
0
2
4
w
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0


b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

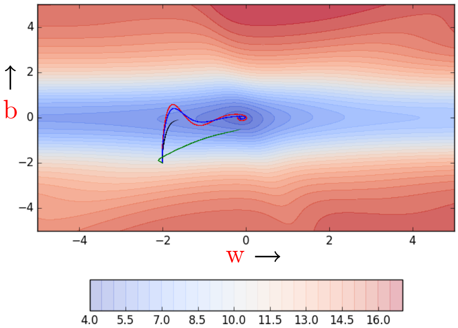


b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

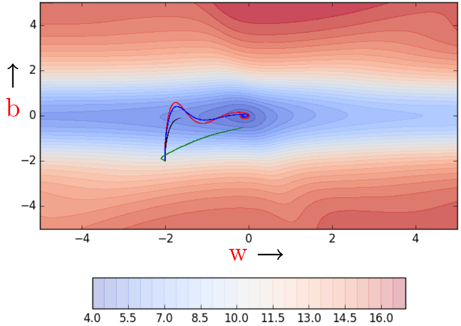


b
w
4
2
0
−2
−4
−4
−2
0
2
4
4.0
5.5
7.0
8.5
10.0
11.5
13.0
14.5
16.0

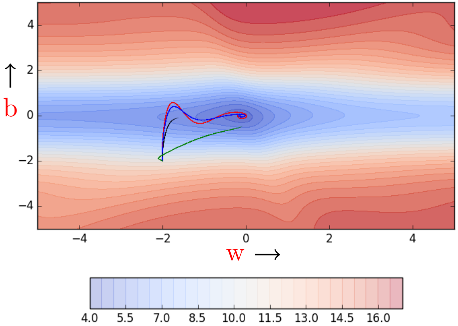
- By using a parameter specific learning rate it ensures that despite sparsity w gets a higher learning rate and hence larger updates



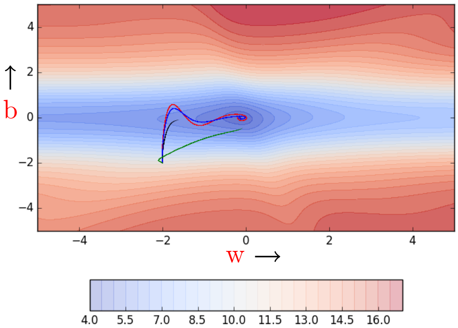
- By using a parameter specific learning rate it ensures that despite sparsity w gets a higher learning rate and hence larger updates
- Further, it also ensures that if b undergoes a lot of updates its effective learning rate decreases because of the growing denominator



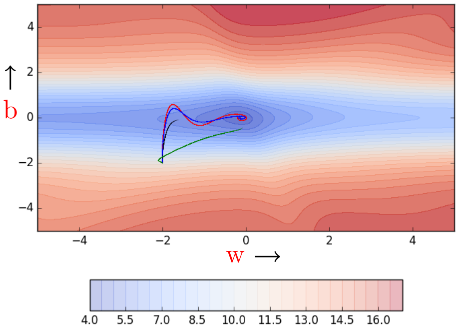
- By using a parameter specific learning rate it ensures that despite sparsity w gets a higher learning rate and hence larger updates
- Further, it also ensures that if b undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator



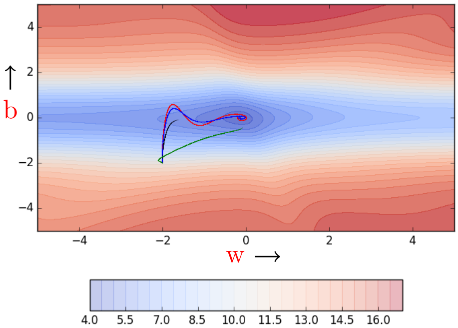
- By using a parameter specific learning rate it ensures that despite sparsity w gets a higher learning rate and hence larger updates
- Further, it also ensures that if b undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)



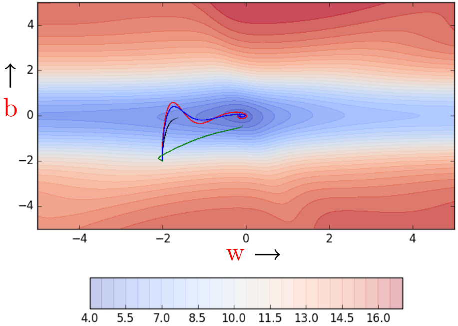
- By using a parameter specific learning rate it ensures that despite sparsity w gets a higher learning rate and hence larger updates
- Further, it also ensures that if b undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)
- What's the flipside?



- By using a parameter specific learning rate it ensures that despite sparsity w gets a higher learning rate and hence larger updates
- Further, it also ensures that if b undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)
- What's the flipside? over time the effective learning rate for b will decay to an extent that there will be no further updates to b



- By using a parameter specific learning rate it ensures that despite sparsity w gets a higher learning rate and hence larger updates
- Further, it also ensures that if b undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)
- What's the flipside? over time the effective learning rate for b will decay to an extent that there will be no further updates to b
- Can we avoid this?



Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)

Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)
- As a result after a while the frequent parameters will start receiving very small updates because of the decayed learning rate

Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)
- As a result after a while the frequent parameters will start receiving very small updates because of the decayed learning rate
- To avoid this why not decay the denominator and prevent its rapid growth

Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)
- As a result after a while the frequent parameters will start receiving very small updates because of the decayed learning rate
- To avoid this why not decay the denominator and prevent its rapid growth

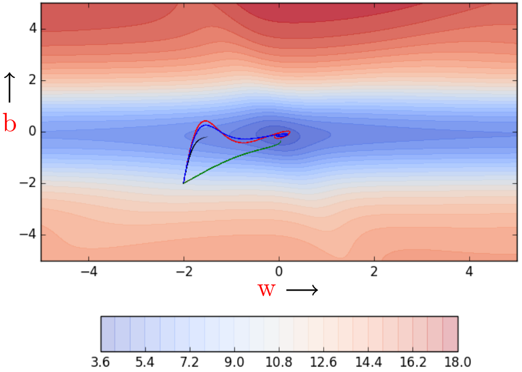
Update rule for RMSProp

$$v_t = \beta * v_{t-1} + (1 - \beta)(\nabla w_t)^2$$

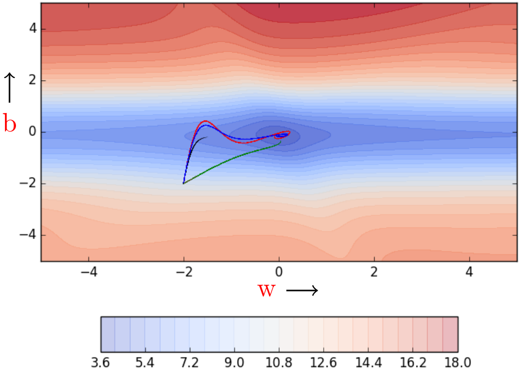
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

... and a similar set of equations for b_t

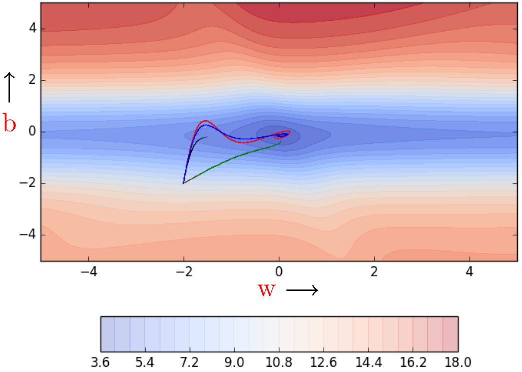
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



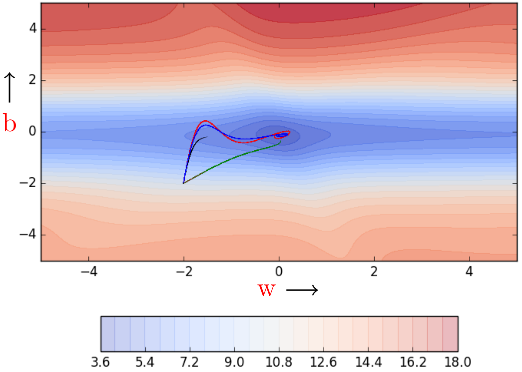
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



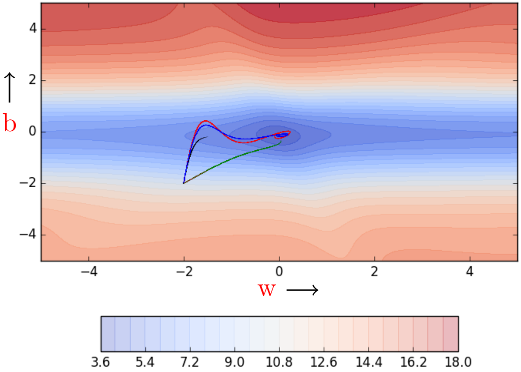
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



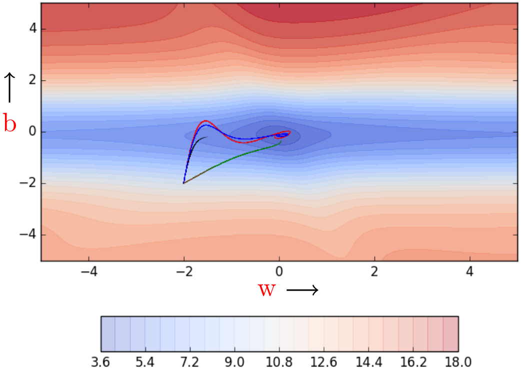
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



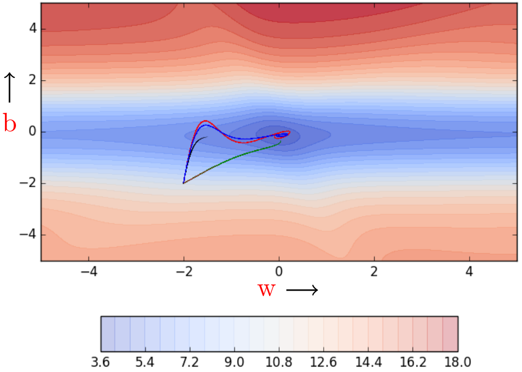
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



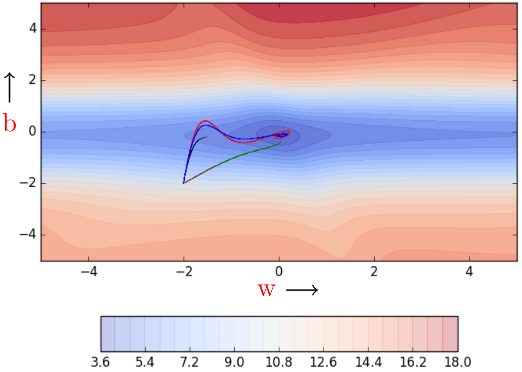
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



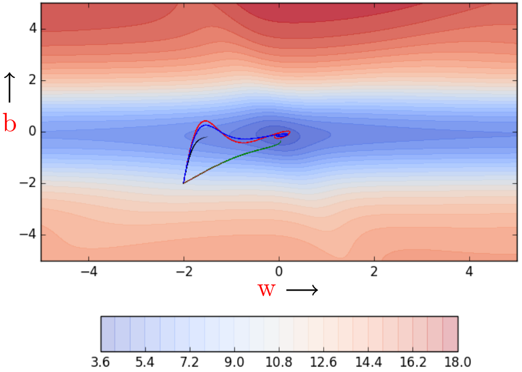
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



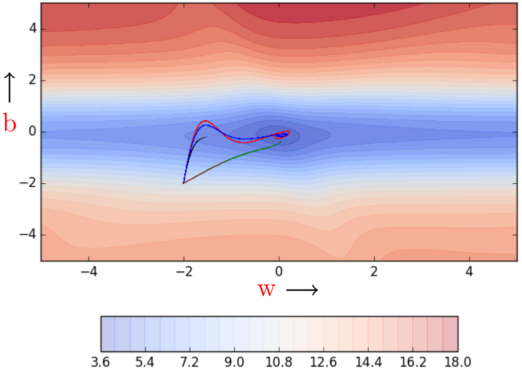
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



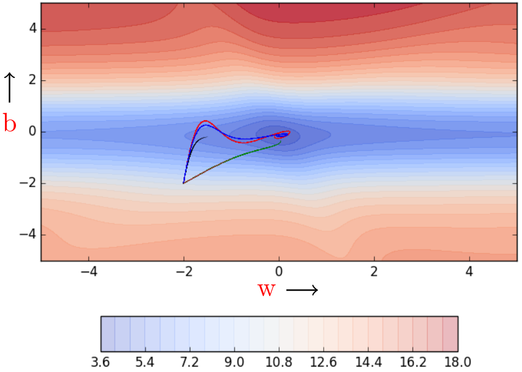
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



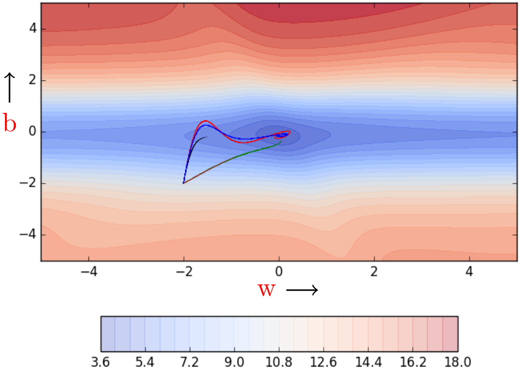
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



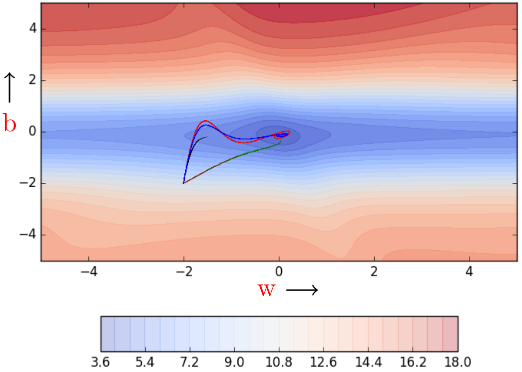
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



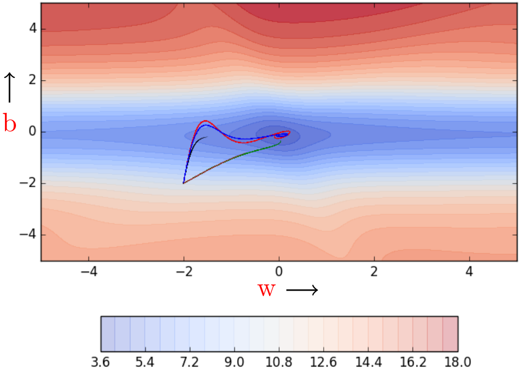
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



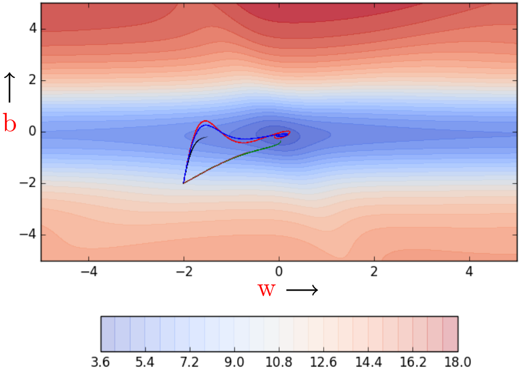
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



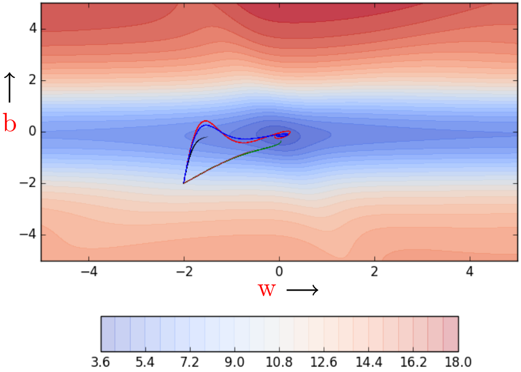
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



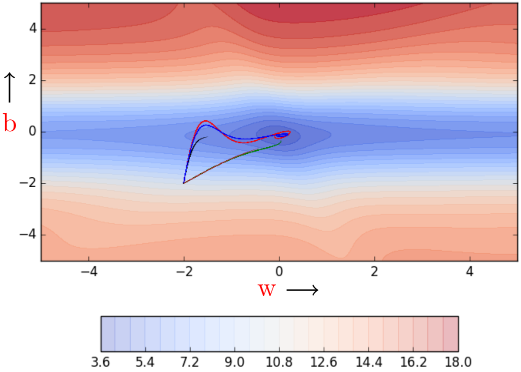
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



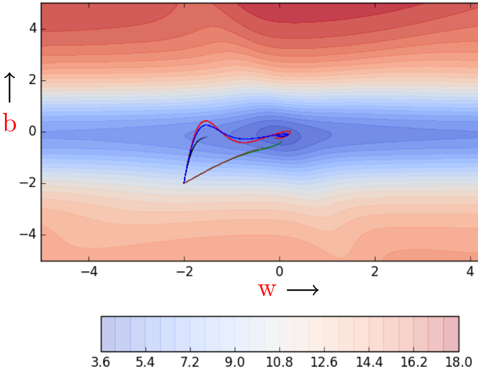
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



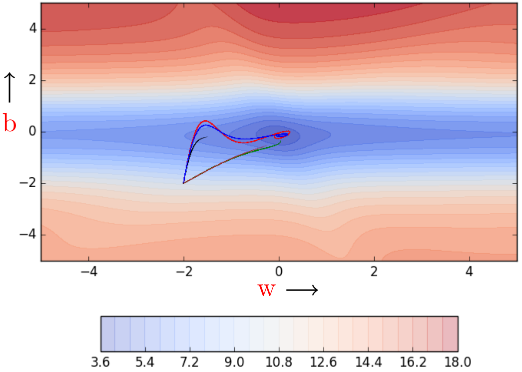
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



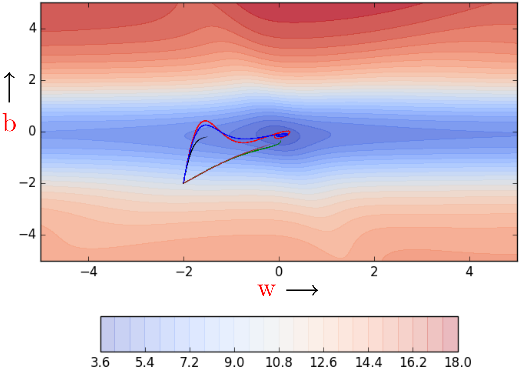
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



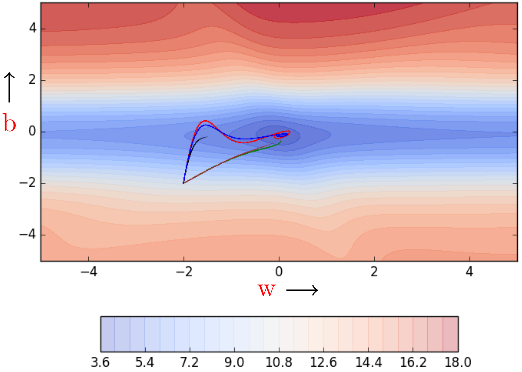
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



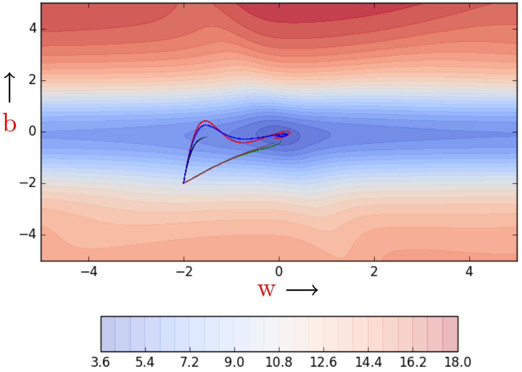
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



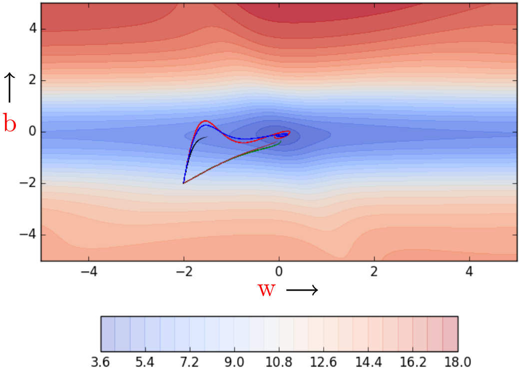
```
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



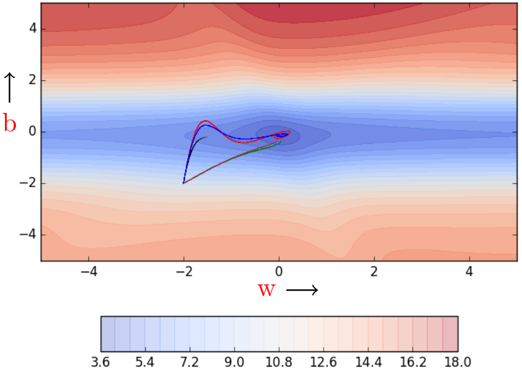
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



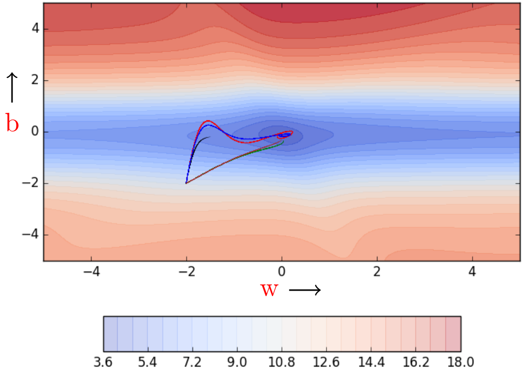
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



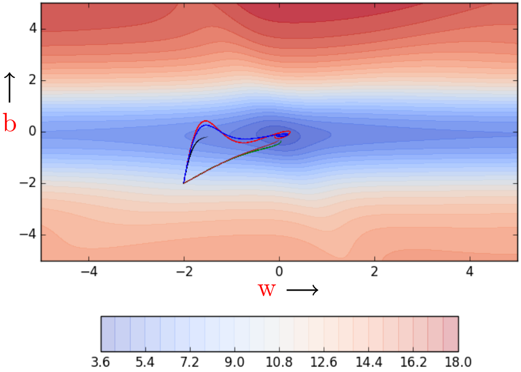
```
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



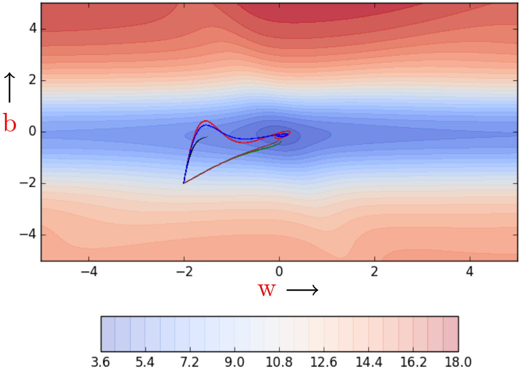
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



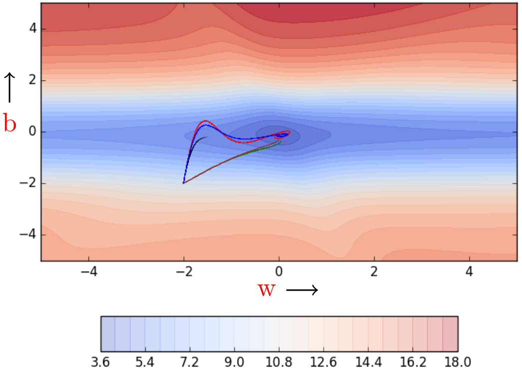
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



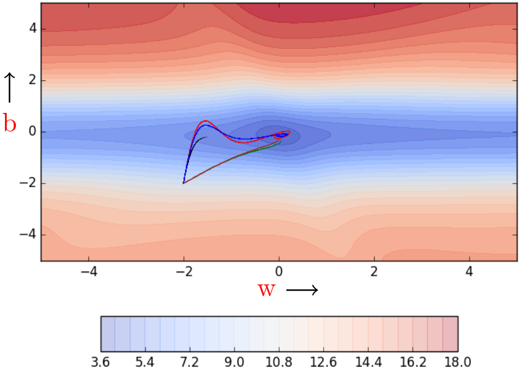
```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

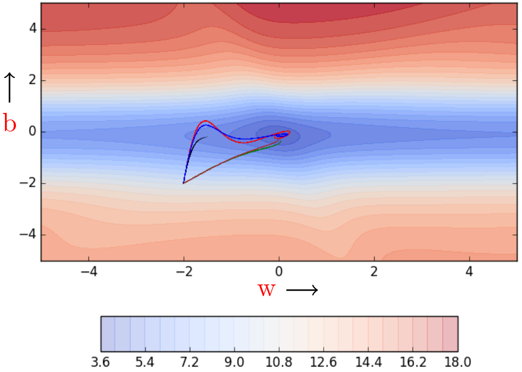


```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



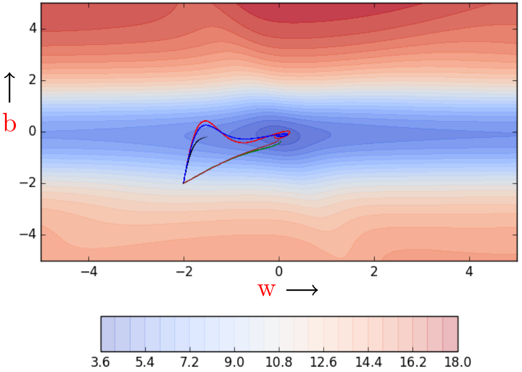
- Adagrad got stuck when it was close to convergence

```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



- Adagrad got stuck when it was close to convergence (it was no longer able to move in the vertical (b) direction because of the decayed learning rate)

```python
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



- Adagrad got stuck when it was close to convergence (it was no longer able to move in the vertical (b) direction because of the decayed learning rate)

- RMSProp overcomes this problem by being less aggressive on the decay

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

Update rule for Adam

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients
- In practice, $\beta_1 = 0.9$ and $\beta_2 = 0.999$

Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients
- In practice, $\beta_1 = 0.9$ and $\beta_2 = 0.999$

Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$$

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients
- In practice, $\beta_1 = 0.9$ and $\beta_2 = 0.999$

Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

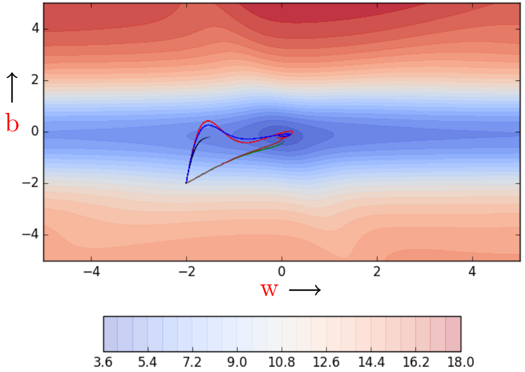
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

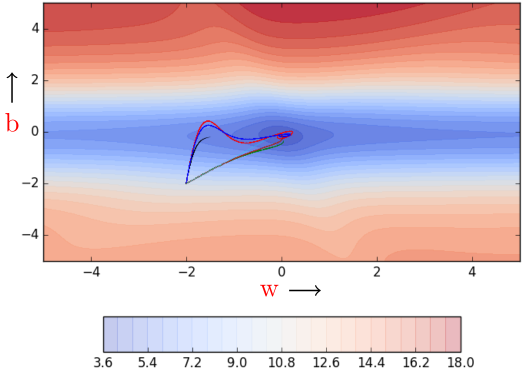
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$$

... and a similar set of equations for b_t

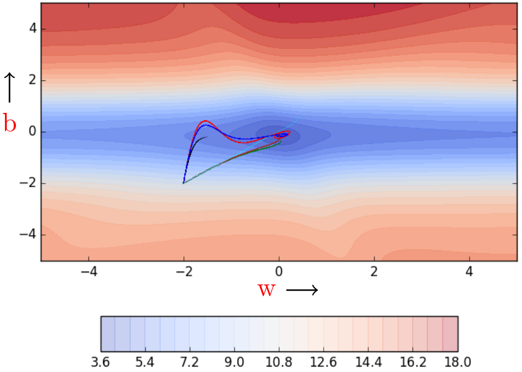
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



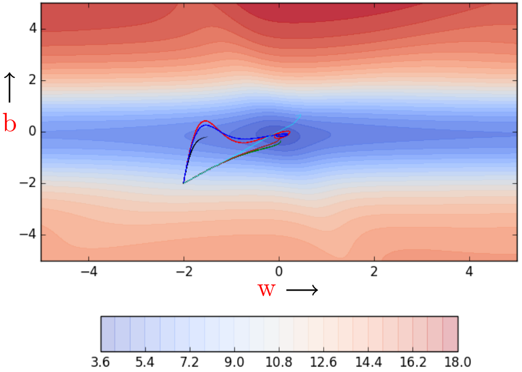
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



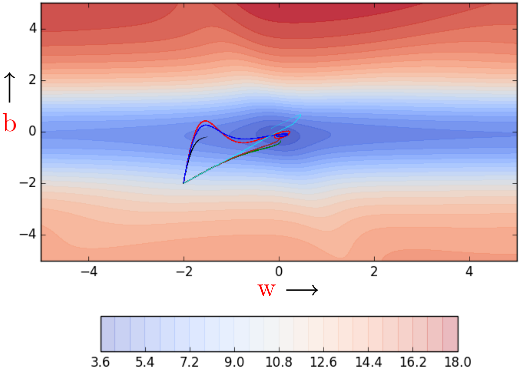
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



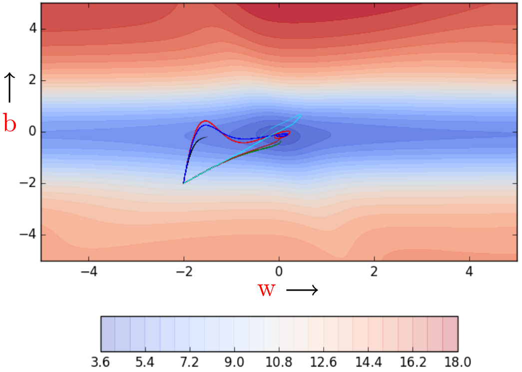
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



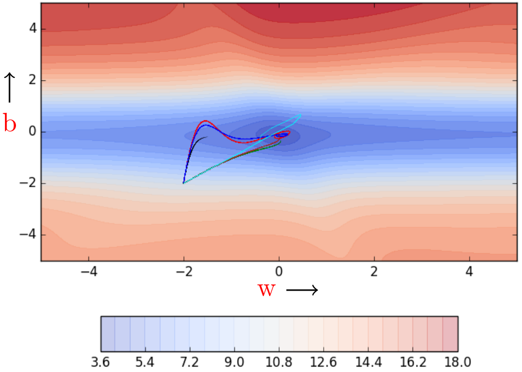
```
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



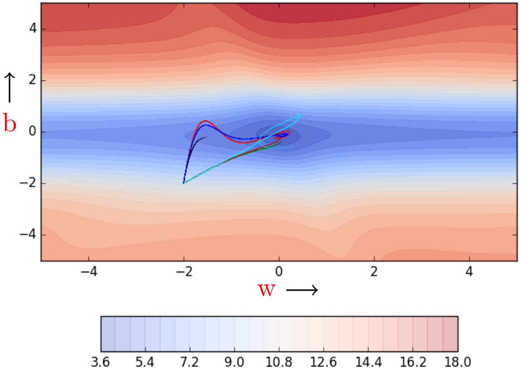
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



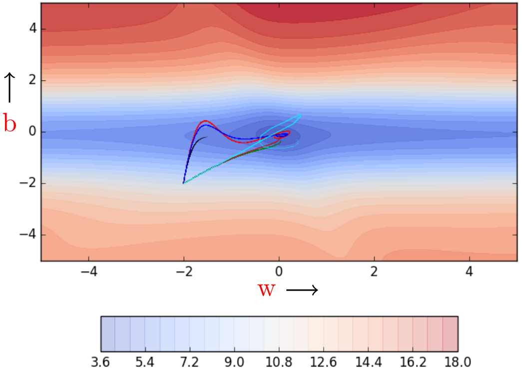
```
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



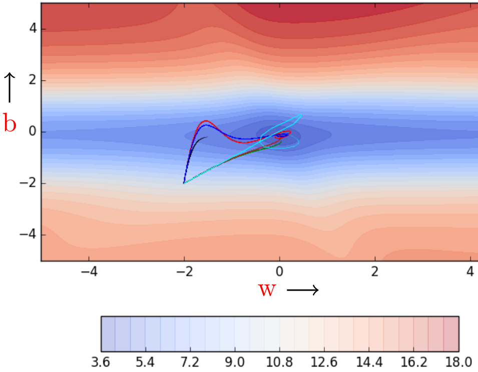
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



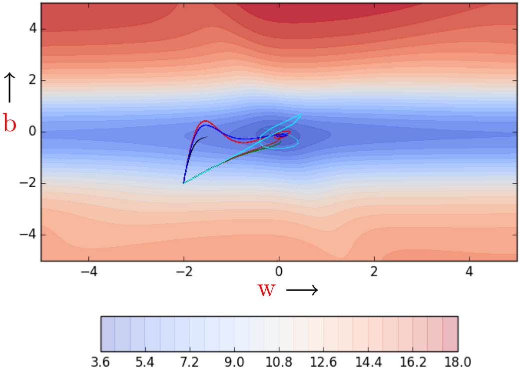
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



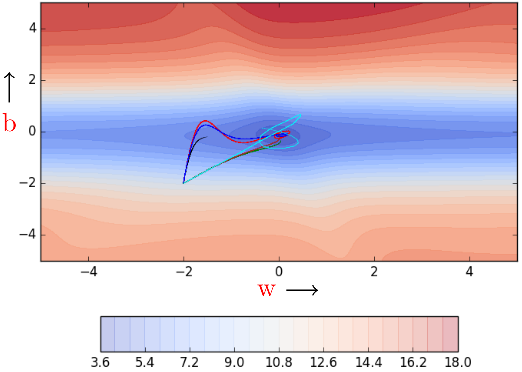
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



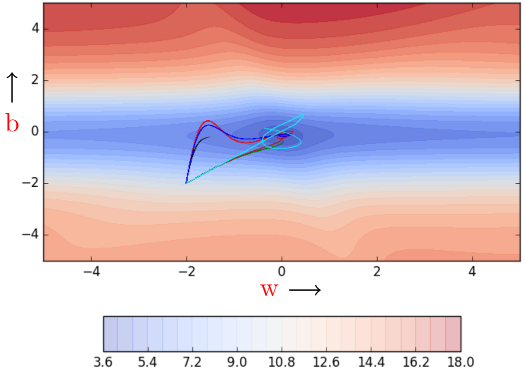
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



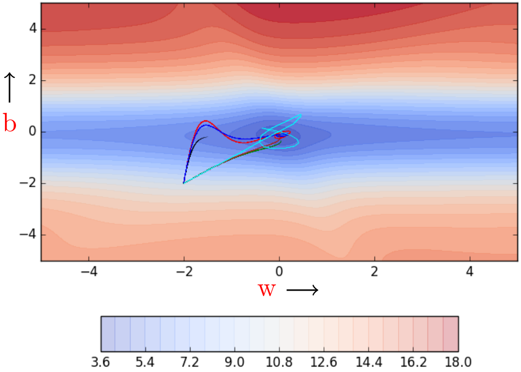
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



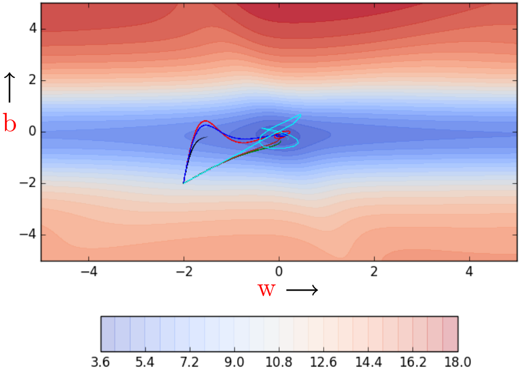
```
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



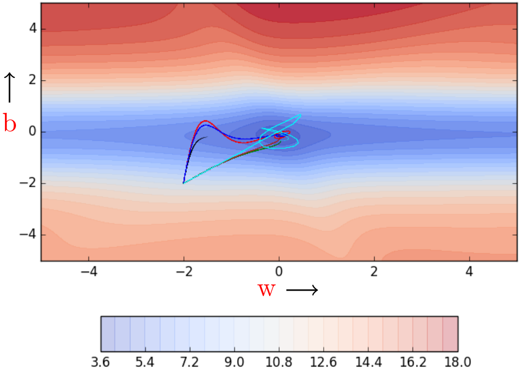
```
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



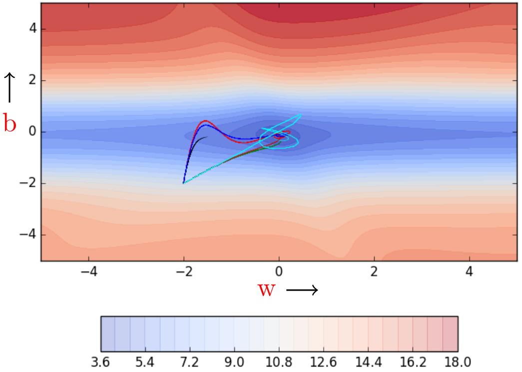
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



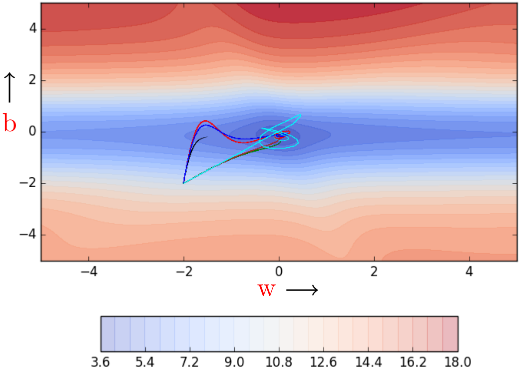
```
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



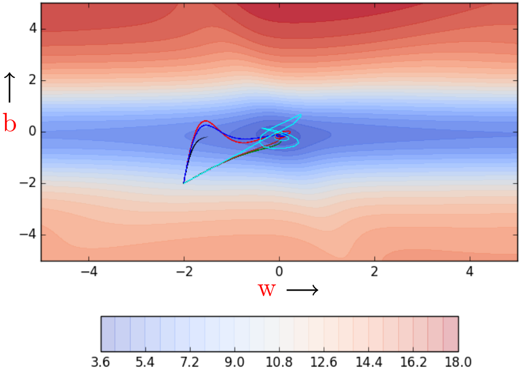
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



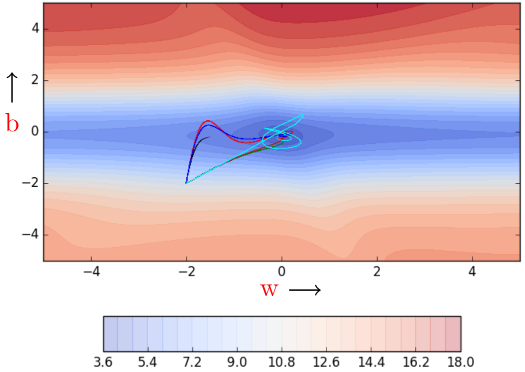
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



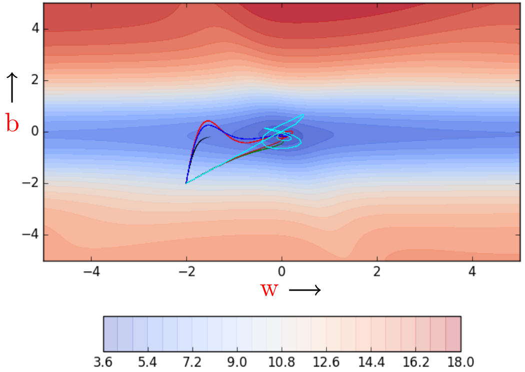
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



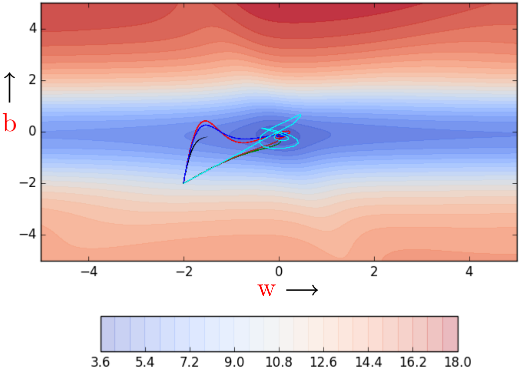
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



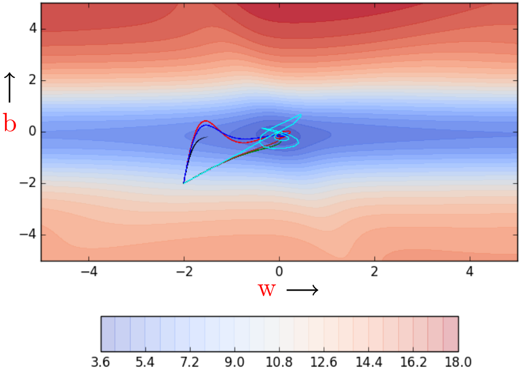
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



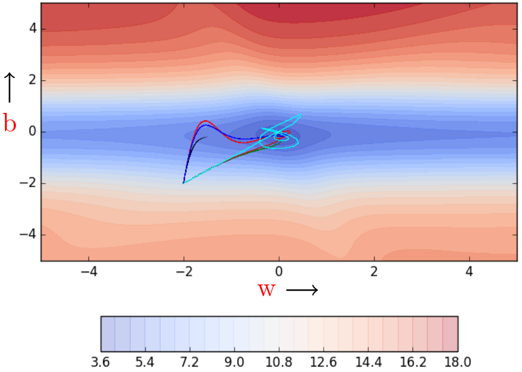
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



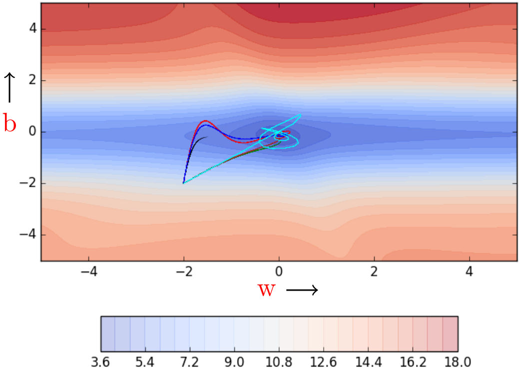
```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



```python
def do_adam() :
    w_b_dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
        ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```



- As expected, taking a cumulative history gives a speed up ...

Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e-8$)

Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e-8$)
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems $\eta = 0.001, 0.0001$ works best

Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e-8$)
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems $\eta = 0.001, 0.0001$ works best
- Having said that, many papers report that SGD with momentum (Nesterov or classical) with a simple annealing learning rate schedule also works well in practice (typically, starting with $\eta = 0.001, 0.0001$ for sequence generation problems)

Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e-8$)
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems $\eta = 0.001, 0.0001$ works best
- Having said that, many papers report that SGD with momentum (Nesterov or classical) with a simple annealing learning rate schedule also works well in practice (typically, starting with $\eta = 0.001, 0.0001$ for sequence generation problems)
- Adam might just be the best choice overall!!

Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e-8$)
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems $\eta = 0.001, 0.0001$ works best
- Having said that, many papers report that SGD with momentum (Nesterov or classical) with a simple annealing learning rate schedule also works well in practice (typically, starting with $\eta = 0.001, 0.0001$ for sequence generation problems)
- Adam might just be the best choice overall!!
- Some recent work suggest that there is a problem with Adam and it will not converge in some cases