Machine Learning

Lecture 11a

Recall how we do linear regression and fit a linear model to a dataset $\mathcal{D} = (X, \mathbf{y})$ where $X = \{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N\}$ is a set of datavectors i.e. points in some space of d-tuples $\mathbb{R}^d$, so each $\underline{x}_i$ is of the form $\underline{x}_i = (x_i^{(1)}, x_i^{(2)}, x_i^{(3)}, \ldots, x_i^{(d)})$ The set $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$ is the set of responses i.e. for each data vector $\underline{x}_i$ we have a response y_i .

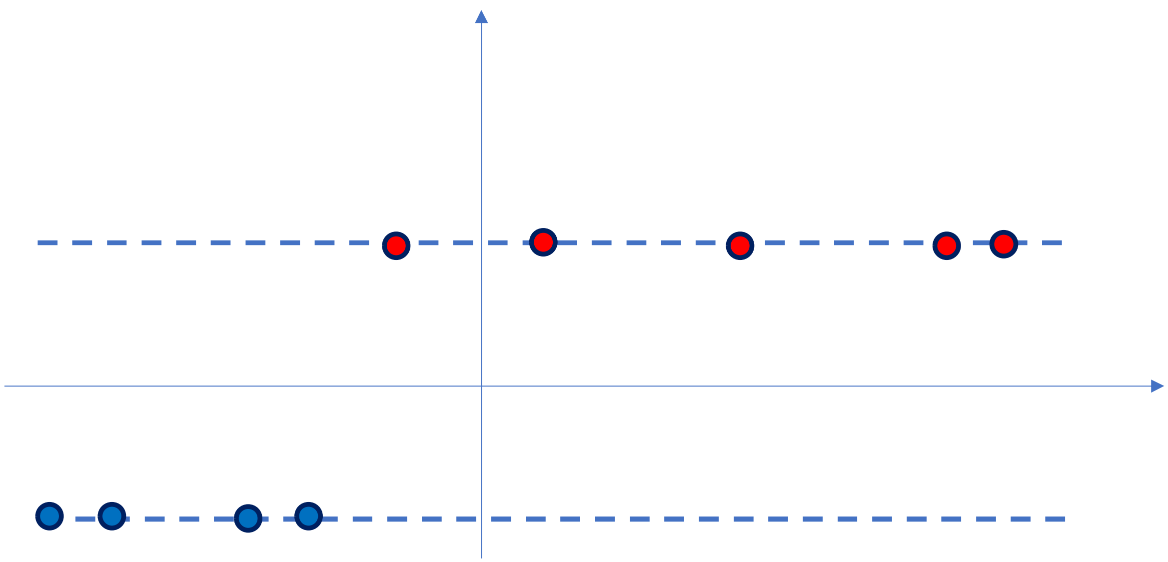
The idea is to find numbers $\beta_0, \beta_1, \beta_2, \ldots, \beta_d$ such that

$$y_i \sim \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_i^{(d)}$$

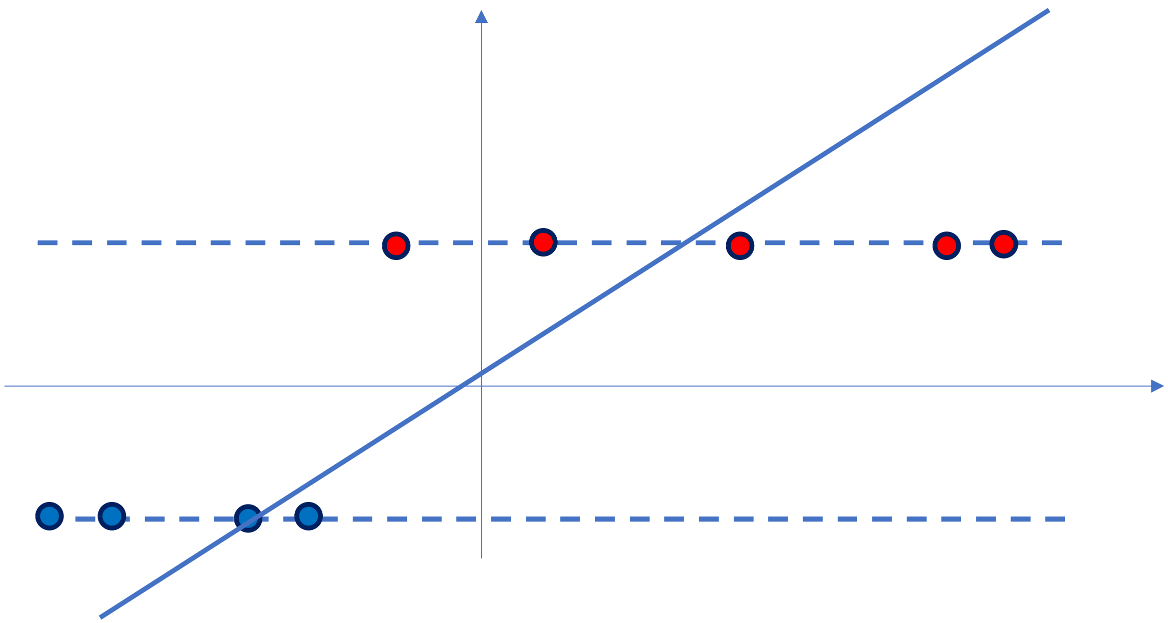
for all $i = 1, 2, 3, \ldots, N$

In the regression problem the y's can take any value.

Suppose now that the y's can only take the values +1 or -1



It is clearly not possible to fit a line through these points with any kind of precision



We will discuss the *Logistic Regression Model.* Even though the name says regression it is in fact a classification model.

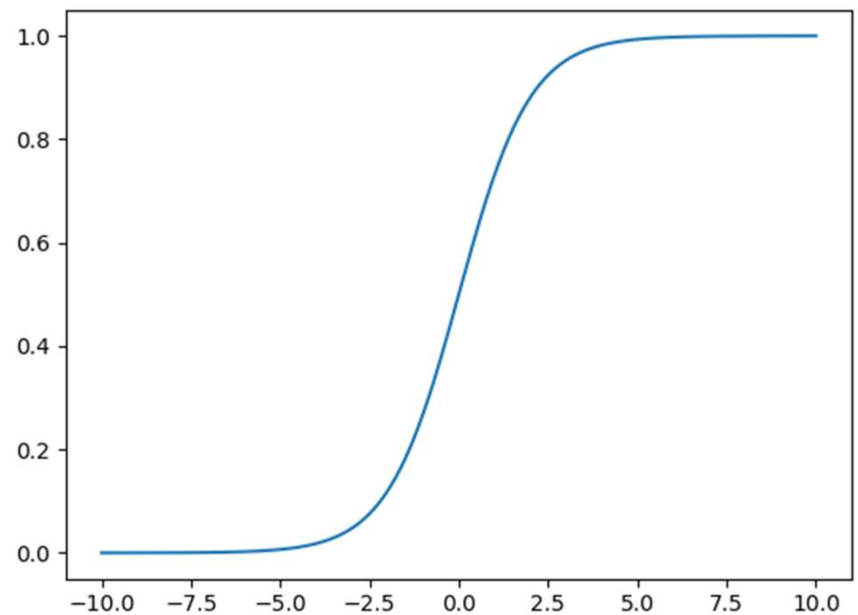
The idea is to assign to each data vector a *probability* i.e. a number between 0 and 1. If this probability is > ½ we associate the label +1 and otherwise -1.

To assign a probability we shall use the logistic function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

```python
def sigma(z):
    return 1./(1.+np.exp(-z))
```

executed in 5ms, finished 11:26:13 2019-05-11



So we need to associate to each data vector $\underline{x}$, a real number z .

Compute the value of the logistic function $\sigma(z)$ and check whether this value is > ½ or < ½.

The way we associate the number z is in fact similar to the linear regression, namely we want to find numbers $\beta_0, \beta_1, \beta_2, \ldots, \beta_d$ such that

$$z = \beta_0 + \beta_1 x^{(1)} + \beta_2 x^{(2)} + \cdots + \beta_d x^{(d)}$$

Remark that whether the data vector is classified as +1 or -1 is determined by the sign of z, if: $z > 0$, $\sigma(z) > 1/2$ so the label is +1

The logistic function at a point z defines a probability distribution over the set $\{-1, +1\}$ by

$$P(+1) = \sigma(z), P(-1) = 1 - \sigma(z)$$

Remark that

$$1 - \sigma(z) = 1 - \frac{1}{1 + \exp(-z)} = \frac{1 + \exp(-z) - 1}{1 + \exp(-z)} = \frac{1}{\exp(z) + 1} = \sigma(-z)$$

So

$$P(-1) = \sigma(-z)$$

The idea of logistic regression is to determine the coefficients $\beta_0, \beta_1, \beta_2, \ldots, \beta_d$ so that for all the data points $\underline{x}_i$ with label $y_i = \pm 1$ the probability distribution over $\{-1, +1\}$ defined by

$$\sigma(z_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_i^{(d)})$$

is 'close' to the true distribution

$$Q(y_i) = 1, Q(-y_i) = 0$$

where we measure 'closeness' by the KL divergence

$$D_{KL}(Q||P) = -\mathbb{E}_Q(\log \frac{P}{Q})$$

In our case this is

$$-Q(1)\log\frac{P(1)}{Q(1)} - Q(-1)\log\frac{P(-1)}{Q(-1)} = -Q(1)\log\frac{\sigma(z)}{Q(1)} - Q(-1)\log\frac{\sigma(-z)}{Q(-1)}$$

In case $y = +1$ this is $-\log\sigma(z)$ and if $y = -1$ it is $-\log\sigma(-z)$ and so in any case

$$D_{KL}(Q||P) = -\log\sigma(yz)$$

It follows that we want to find the coefficients $\beta_0, \beta_1, \ldots, \beta_d$ that minimizes

$$\mathcal{L} = -\frac{1}{N}\sum\log\sigma(y_i(\beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_i^{(d)}))$$

To minimize the loss function we can again use gradient descent i.e. we start with some $\underline{\beta} = (\beta_0, \beta_1, \beta_2, \ldots, \beta_d)$ and then we iterate the sequence

$$\underline{\beta}_{new} = \underline{\beta}_{old} - \alpha \nabla_{\underline{\beta}} \mathcal{L}(\underline{\beta}_{old})$$

So we need to compute the gradient i.e. the partial derivatives

$$\frac{\partial}{\partial \beta_k} \mathcal{L} = -\frac{1}{N} \sum_{i=1,2,\ldots,N} \frac{\partial}{\partial \beta_k} \log \sigma(y_i(\beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_k x_i^{(k)} + \cdots + \beta_d x_i^{(d)}))$$

Now

$$\log \sigma(z) = -\log(1 + \exp(-z))$$

and using the chain rule we get

$$\frac{\partial}{\partial \beta_k} \log \sigma(y_i z_i) = \boxed{\frac{\partial}{\partial z_i}(-\log(1 + \exp(-y_i z_i))} \frac{\partial}{\partial \beta_k} z_i$$

The first factor is (again using the chain rule)

$$\begin{aligned} & -\frac{1}{(1 + \exp(-y_i z_i))} \frac{\partial}{\partial z_i}(1 + \exp(-y_i z_i)) \\ &= -\frac{1}{(1 + \exp(-y_i z_i))} \exp(-y_i z_i)(-y_i) \\ &= \boxed{y_i \left(\frac{\exp(-y_i z_i)}{1 + \exp(-y_i z_i)} \right)} \end{aligned}$$

The second factor is much simpler

$$z_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_i^{(d)}$$

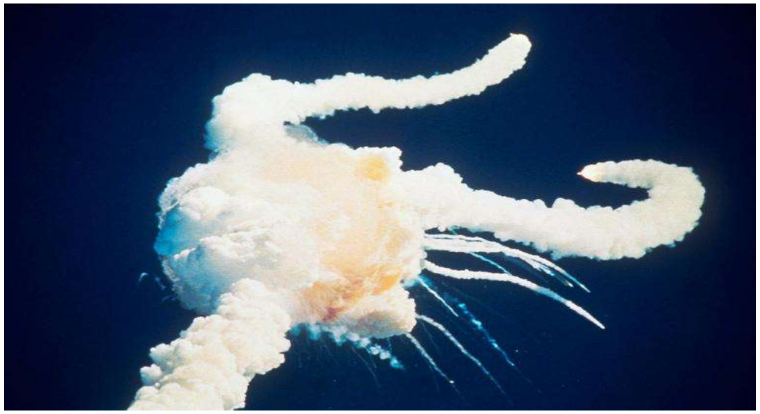
so

$$\frac{\partial}{\partial \beta_k} z_i = \begin{cases} 1 \text{ if } k = 0 \\ x_i^{(k)} \text{ otherwise} \end{cases}$$

finally we get

$$\frac{\partial}{\partial \beta_k} \mathcal{L} = \begin{cases} -\frac{1}{N} \sum_{i=1,2,\ldots,N} y_i \frac{\exp(-y_i z_i)}{1+\exp(-y_i z_i)} = y_i \sigma(-y_i z_i) \text{ if } k = 0 \\ -\frac{1}{N} \sum_{i=1,2,\ldots,N} y_i \frac{\exp(-y_i z_i)}{1+\exp(-y_i z_i)} x_i^{(k)} = y_i \sigma(-y_i z_i) x_i^{(k)} \text{ if } k > 0 \end{cases}$$

Challenger Explosion

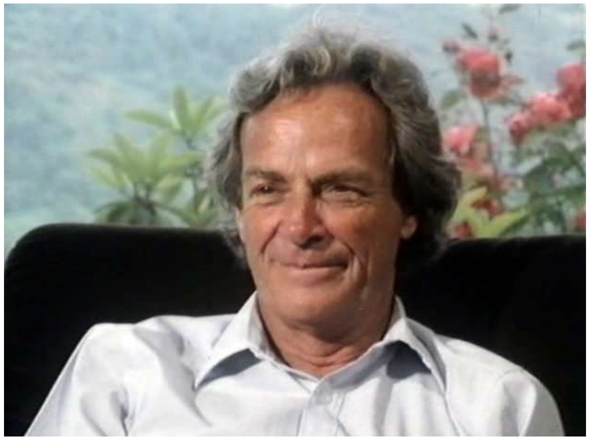


On January 28, 1986 the space shuttle, Challenger, exploded shortly after launch, killing the entire crew. It was later determined that the cause of the disaster was the burn-through of an O-ring seal at a joint in one of the solid-fuel rocket boosters. Of the 24 previous shuttle flights it was determined that in 7 of those flights there was damage to the O-ring seals. In one case the solid fuel booster rockets were not recovered so there are only data for 23 flights. We want to investigate if the temperature at launch time has an influence on whether the O-rings are damaged during the launch.

Table 1: Temperature data from space shuttle launches

Temp	Damage(No = 0,Yes = 1)
66	0
70	1
69	0
68	0
67	0
72	0
73	0
70	0
57	1
63	1
70	1
78	0
67	0
75	0
70	0
81	0
76	0
79	0
75	1
76	0
58	0

During the investigation of the disaster, Richard Feynman, a member of the investigative panel, demonstrated that the O-rings would lose their elasticity under low temperature conditions. The O-rings were designed to expand to completely seal the joints and so the theory was that because of the unusually cold temperatures at launch time of the Challenger mission the O-rings had lost their elasticity and were unable to expand and seal the joints



We are going to use Logistic Regression to investigate whether the data support the theory that cold weather had an impact on damage to the O-ring seals.

We are first going to replace the 0 labels with -1 labels and so we have the dataset

```
import numpy as np
import matplotlib.pyplot as plt
```

```
temp = np.array([66.,70.,69.,68.,67.,72.,73.,70.,57.,
        63.,70.,78.,67.,53.,67.,75.,70.,81.,
        76.,79.,75.,76.,58.])

damage = [-1,1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1]
```

A -1 indicates no damage to the O-rings while a 1 indicates damage

We first write the logistic function and the loss function and the gradient of the loss function

```
def sigma(z):
    return 1./(1.+np.exp(-z))
```

executed in 5ms, finished 11:26:13 2019-05-11

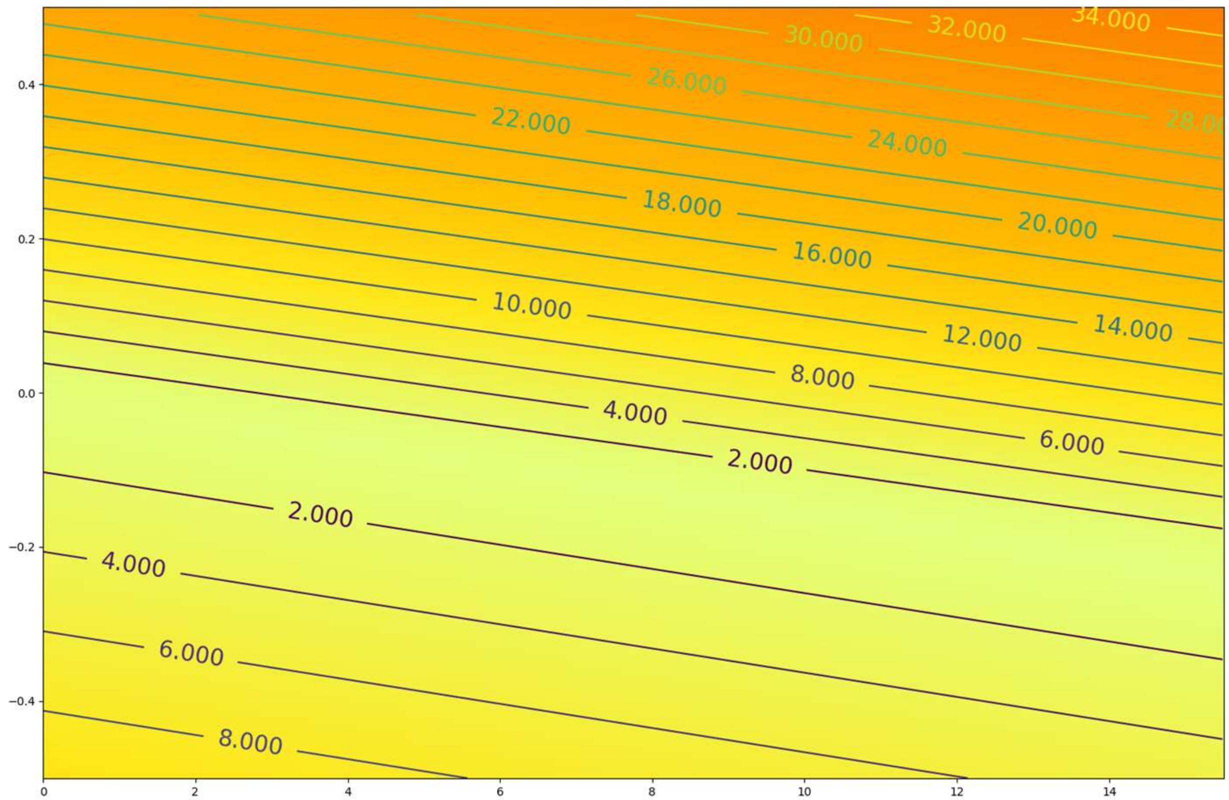
```
def loss_function(b0,b1):
    return np.mean([-np.log(sigma(y*(b0 + b1 * t))) for y,t in zip(damage,temp)])
```

executed in 4ms, finished 11:46:06 2019-05-11

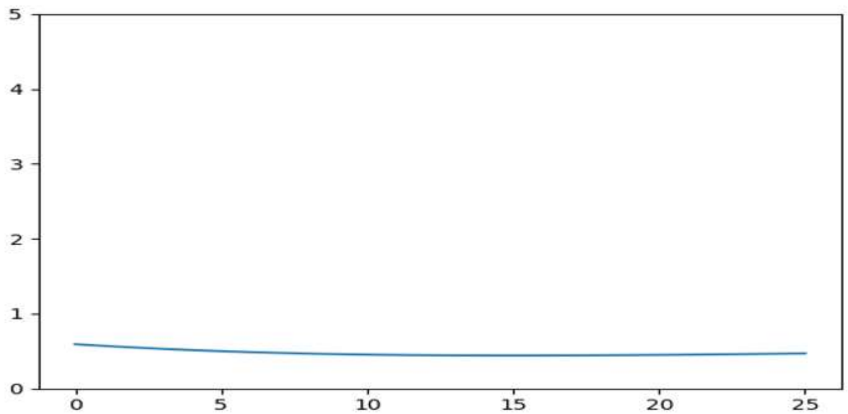
```
def gradient(b0,b1):
    return np.mean([np.array([-y * sigma(-y * (b0 + b1 * t)),
            -y * sigma(-y * (b0 + b1 * t))*t]) for y,t in zip(damage,temp)],axis=0)
```

executed in 15ms, finished 13:10:41 2019-05-11

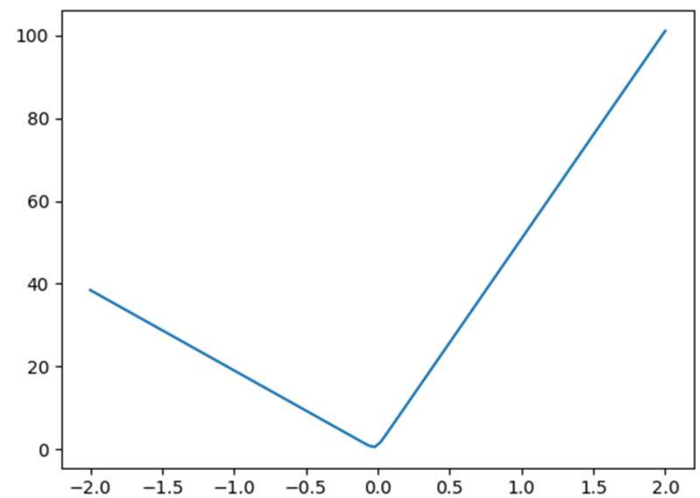
We can visualize the loss function since in this case it is a function of only 2 variables. The minimum is somewhere in the lightest area. We can see that there is a long flat valley so the gradient descent will be very slow when we move along the valley because the gradient will be close to 0



This graph shows the loss function along the bottom of the valley. We see how flat it is so the magnitude of the gradient will be very small = slow convergence



The other cross-section shows how the loss function drops off as we approach the valley



This gives us a problem: if we use a large learning rate to speed up the convergence, the second coordinate will not converge because it will jump from one side of the valley to the other

Learning rate = 0.01, no convergence

```
B = np.array([0,0.])
for _ in range(1000):
    B = B - 0.01 * gradient(B[0],B[1])
    print(B)

```

executed in 481ms, finished 12:40:12 2019-05-12

```
[ 0.24694358 -0.20314444]
[ 0.24998703 -0.00923311]
[ 0.24899799 -0.09478147]
[0.25201923 0.09770133]
[ 0.2450739  -0.40332013]
[ 0.24811738 -0.20940708]
[ 0.25116084 -0.01549522]
[ 0.25115638 -0.03203237]
[0.25296357 0.07760455]
```

Learning rate = 0.001, very slow convergence after 100,000 steps not close to the minimum

```
B = np.array([0,0.])
for _ in range(100000):
    B = B - 0.001 * gradient(B[0],B[1])
    print(B)

```

executed in 50.0s, finished 12:42:15 2019-05-12

```
[ 2.14228488 -0.04428776]
[ 2.14230417 -0.04428804]
[ 2.14232347 -0.04428832]
[ 2.14234276 -0.04428859]
[ 2.14236205 -0.04428887]
[ 2.14238134 -0.04428915]
[ 2.14240064 -0.04428943]
[ 2.14241993 -0.0442897 ]
[ 2.14243922 -0.04428998]
```

If we use these values $\beta_0 = 2.142438, \beta_1 = -0.04428$

$$Prob(damage|t) = \sigma(2.142438 - 0.0442899\, t)$$

At launch time the temperature was $t = 31$ so

$$Prob(damage|t) = \sigma(2.142438 - 0.0442899 \cdot 31) = 0.6834$$

so about a 2/3 probability of damage, maybe this might have been an acceptable risk

There are many ways to construct sequences converging to a minimum of the loss function. The gradient descent sequence can be very slow as the example shows. One of the fastest converging methods is Newton's method. While it constructs a sequence that converges fast it is also somewhat more complicated.

The gradient descent sequence can be derived from approximating a function with a linear function

$$f(x - \Delta x) \sim f(x) - \nabla f(x) \cdot \Delta x$$

Here we view x as being fixed and we view both sides as functions of Δx

The largest decline

$$|f(x - \Delta x) - f(x)| = |\nabla f(x) \cdot \Delta x|$$

is when absolute value of the dot-product $|\nabla f(x) \cdot \Delta x|$ is maximal and that is when $\nabla f(x)$ and Δx are parallel (Cauchy-Schwartz)

In Newton's method we use a quadratic approximation to f:

$$f(x+\Delta x) \sim f(x) + \nabla f(x) \cdot \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x) \Delta x$$

where $\nabla^2 f(x)$ is the matrix of double derivatives, this matrix is called the Hessian

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial {x^{(1)}}^2} & \frac{\partial^2 f(x)}{\partial x^{(1)} \partial x^{(2)}} & \cdots & \frac{\partial^2 f(x)}{\partial x^{(1)} \partial x^{(d)}} \\ \frac{\partial^2 f(x)}{\partial x^{(2)} \partial x^{(1)}} & \frac{\partial^2 f(x)}{\partial {x^{(2)}}^2} & \cdots & \frac{\partial^2 f(x)}{\partial x^{(2)} \partial x^{(d)}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f(x)}{\partial x^{(1)} \partial x^{(d)}} & \frac{\partial^2 f(x)}{\partial x^{(2)} \partial x^{(d)}} & \cdots & \frac{\partial^2 f(x)}{\partial {x^{(d)}}^2} \end{pmatrix}$$

The idea is that for small Δx , the quadratic function is a good approximation to f and so the minimum of the quadratic function will approximate the minimum of f

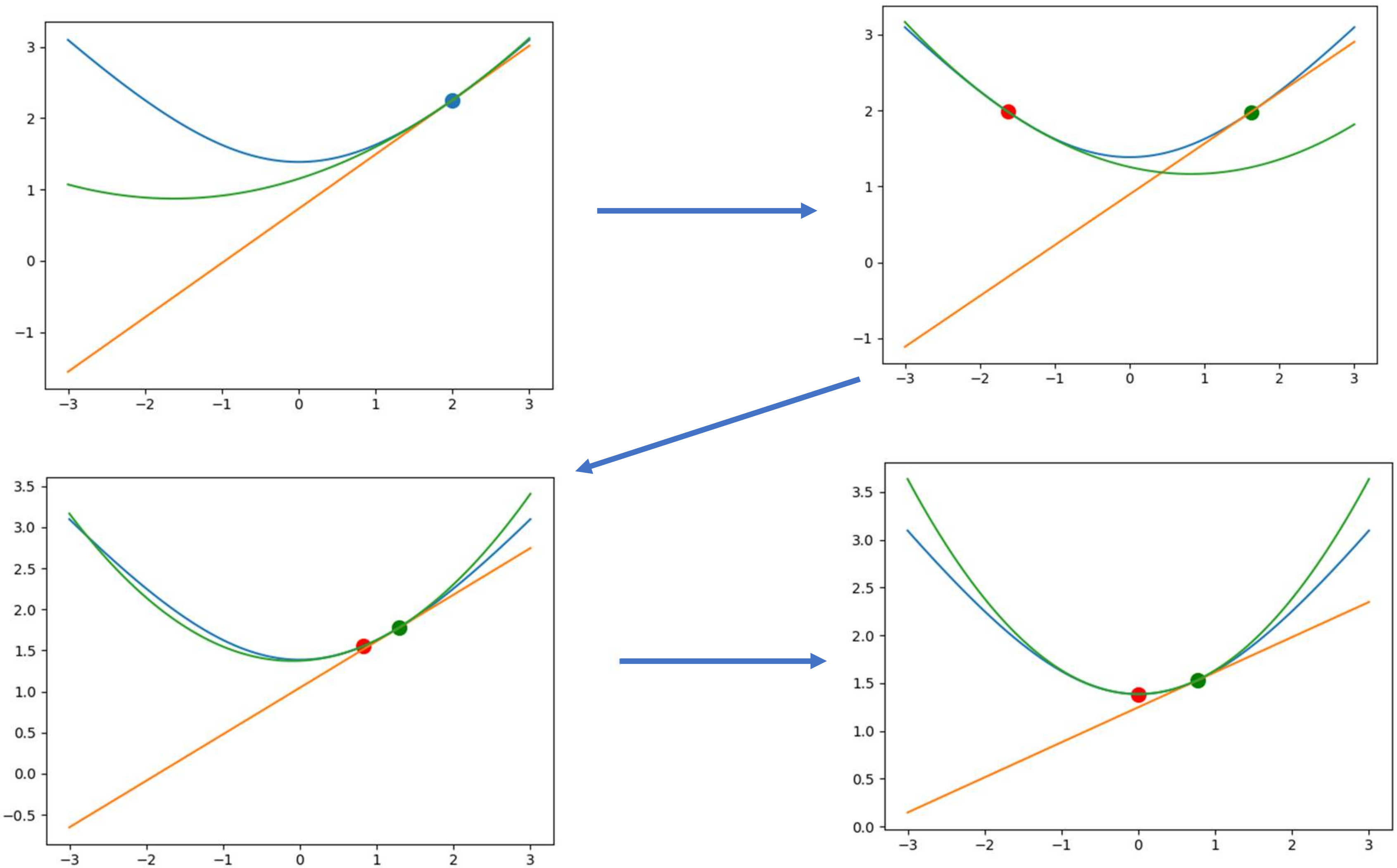
Taking derivative with respect to Δx and setting the derivative equal to 0, we get

$$\nabla f(x) + \nabla^2 f(x)\Delta x = 0$$

so

$$\Delta x = -\nabla^2 f(x)^{-1}\nabla f(x)$$

This is the increment in the Newton sequence



−3 −2 −1 0 1 2 3
−1 0 1 2 3
−3 −2 −1 0 1 2 3
−1 0 1 2 3
−3 −2 −1 0 1 2 3
−0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5
−3 −2 −1 0 1 2 3
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5

```
def Hessian(b0,b1):

    return np.array([[[sigma(y * (b0 + b1 *t)) * sigma(-y * (b0 + b1 *t)),
                      sigma(y * (b0 + b1 *t))* sigma(-y * (b0 + b1 *t)) *t],
                     [sigma(y * (b0 + b1 *t))* sigma(-y * (b0 + b1 *t)) *t,
                     sigma(y * (b0 + b1 *t))* sigma(-y * (b0 + b1 *t)) *t**2]] for
                    y,t in zip(damage,temp)]).mean(axis=0)
```

executed in 31ms, finished 13:34:58 2019-05-11

```python
B = np.array([0.,0.])
for _ in range(10):
    B = B - np.linalg.inv(Hessian(B[0],B[1])).dot(gradient(B[0],B[1]))
    print(B)
```

executed in 31ms, finished 13:43:34 2019-05-11

```
[ 9.61904762 -0.14952381]
[13.65573791 -0.21124698]
[14.93828914 -0.2306001 ]
[15.04229114 -0.23215369]
[15.04290163 -0.23216274]
[15.04290165 -0.23216274]
[15.04290165 -0.23216274]
```

Convergence in 6 iterations

At launch time of the Challenger mission the temperature was 31F.

Using the minimum of the loss function we found we can compute the prediction of the model

```
sigma(15.04290165 -0.23216274 * 31)
```

executed in 18ms, finished 13:10:49 2019-05-12

```
0.9996087829369722
```

So the model predicts with almost certainty that there would be damage to the O-ring seals

The other concept we need is *regularization* more specifically *L2 - regularization*

It is a way to put bounds on the size of the parameters when minimizing a loss function. It is a good tool to avoid overfitting.

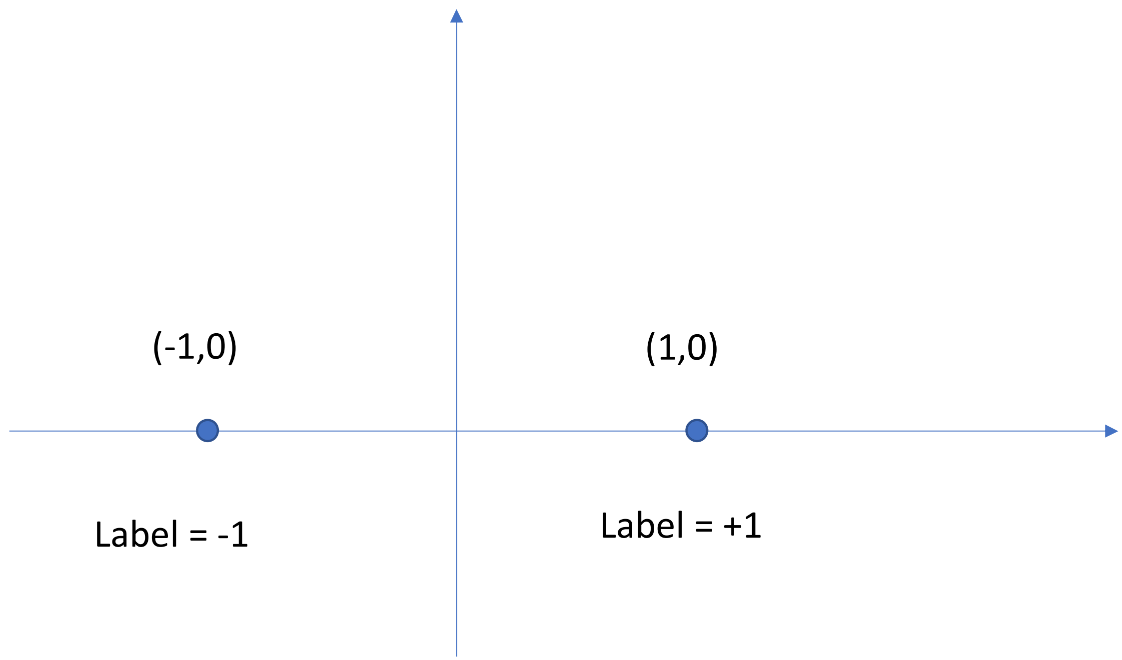
It works by adding a term to the loss function and then minimizing this new function, for instance for regression the regularized loss function becomes

$$\frac{1}{N}\sum\left(y_i - (\beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_d^{(d)})\right)^2 + \lambda(\beta_1^2 + \beta_2^2 + \cdots + \beta_d^2)$$

The parameter λ is called the *regularization parameter,* the larger the regularization parameter, the more the minimization algorithm will focus on the size of the parameters at the expense of fitting to the data.

A regularized loss function will always have a minimum, this is not necessarily the case without regularization

Consider the simplest possible example



The loss function is

$$-\log(\sigma(-(\beta_0 - \beta_1)) - \log(\sigma(\beta_0 + \beta_1))$$

and the partials are

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = \frac{\exp(\beta_0 - \beta_1)}{1 + \exp(\beta_0 - \beta_1))} + \frac{-\exp(-(\beta_0 + \beta_1))}{1 + \exp(-(\beta_0 + \beta_1))}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_1} = \frac{-\exp(\beta_0 - \beta_1)}{1 + \exp(\beta_0 - \beta_1))} + \frac{-\exp(-(\beta_0 + \beta_1))}{1 + \exp(-(\beta_0 + \beta_1))}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_2} = 0$$

Setting these expressions equal to 0 we get equations that have no solutions

Now the regularized loss function is

$$-\log(\sigma(-(\beta_0 - \beta_1)) - \log(\sigma(\beta_0 + \beta_1)) + \lambda(\beta_1^2 + \beta_2^2)$$

with partials

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = \frac{\exp(\beta_0 - \beta_1)}{1 + \exp(\beta_0 - \beta_1))} + \frac{-\exp(-(\beta_0 + \beta_1))}{1 + \exp(-(\beta_0 + \beta_1))}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_1} = \frac{-\exp(\beta_0 - \beta_1)}{1 + \exp(\beta_0 - \beta_1))} + \frac{-\exp(-(\beta_0 + \beta_1))}{1 + \exp(-(\beta_0 + \beta_1))} + 2\lambda\beta_1$$

$$\frac{\partial \mathcal{L}}{\partial \beta_2} = 2\lambda\beta_2$$

These equations are satisfied with $\beta_0 = \beta_2 = 0$ and $\lambda\beta_1 - \sigma(-\beta_1) = 0$ we get the classification function $\sigma(x^{(1)})$ so any point $(x^{(1)}, x^{(2)})$ is classified by $sign(x^{(1)})$