CS5014 Machine Learning

Lecture 3 Linear Regression

Lei Fang

School of Computer Science, University of St Andrews

Spring 2021

University of St Andrews

Topics for today

Linear regression

- matrix notation
- normal equation and closed form solution
  - vector calculus perspective
  - linear algebra perspective: projection
- gradient descent
  - a more general solution

Supervised learning vs unsupervised learning

Supervised learning

- dataset contains both predictors $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ and targets y
- regression: y is continuous
  - e.g. predict your height based your weight: $n = 1$, and x_1 is height, y is weight
- classification: y is categorical
  - e.g. predict adult or child $y = \{A, C\}$ based on height measurement $\boldsymbol{x}$

Unsupervised learning

- dataset formed only with predictors $\boldsymbol{x}$: no targets
- aim: understand the underlying structure of $\boldsymbol{x}$
- typical learning: clustering, dimension reduction etc.

Regression: Catheter dataset

Task: predict a patient's catheter *length* (target) by predictors: *height* and *weight*

height.in	weight.lbs	length.cm
42.8	40.0	37
63.5	93.5	50
37.5	35.5	34
39.5	30.0	36
45.5	52.0	43
38.5	17.0	28
43.0	38.5	37
22.5	8.5	20
37.0	33.0	34
23.5	9.5	30
33.0	21.0	38
58.0	79.0	47

Regression: Catheter dataset

The regression problem can be formed as:

$$y^{(i)} = f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) + e^{(i)}$$

- f is a model that predict $y^{(i)}$ from $\boldsymbol{x}^{(i)}$
  - $i = 1, \ldots, m$: index of data samples (row index),
  - m is the total training size
- $\boldsymbol{x}^{(i)} = [x_1^{(i)}, \ldots, x_n^{(i)}]^T$ is a $n \times 1$ vector:
  - n: number of predictors (columns)
- e.g. $y^{(1)} = 37$ and $\boldsymbol{x}^{(1)} = [42.8, 40]^T$
- $\boldsymbol{\theta}$ is the model parameter
- $e^{(i)}$ is the prediction difference of the i-th entry

Linear regression

If we further assume the relationship is linear, i.e.

$$\begin{aligned} f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) &= \theta_0 + \theta_1 x_1^{(i)} + \ldots + \theta_n x_n^{(i)} \\ &= [\theta_0, \theta_1, \ldots, \theta_n] \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} = \boldsymbol{\theta}^T \boldsymbol{x}^{(i)} \end{aligned}$$

the regression is called **linear regression**

- a dummy predictor $x_0^{(1)} = 1$ is added to $\boldsymbol{x}^{(i)}$

Linear regression: least squared error

The prediction error is

$$e^{(i)} = y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) = y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)}$$

The sum of squared errors is

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{m} (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2$$

Learning objective is then to minimise the cost function

$$\hat{\theta} = \operatorname*{argmin}_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \{\boldsymbol{x}^{(i)}, y^{(i)}\}_1^m)$$

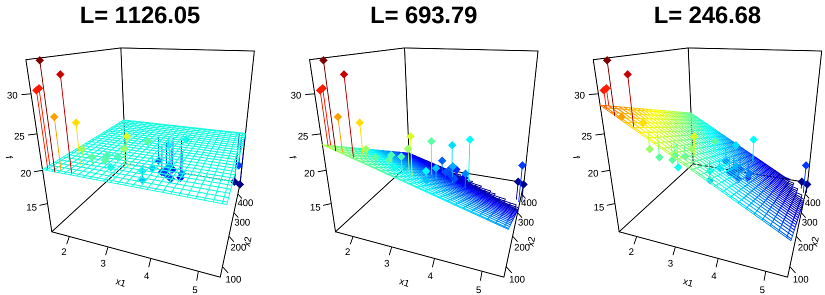
Linear models and hyperplane

Geometrically, linear function

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \boldsymbol{\theta}^T \boldsymbol{x}$$

is a hyperplane

- $\boldsymbol{\theta}$ is the gradient vector $\nabla_{\boldsymbol{x}} f$: the greatest ascent direction of f
- minising L means to find a hyperplane that *fits* the data best



L= 1126.05

L= 693.79

L= 246.68

How to optimise $L(\boldsymbol{\theta})$?

Vector calculus is our friend:

- find the gradient $\nabla_{\boldsymbol{\theta}} L$
- set it to zero

In matrix notation, let

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} = \begin{bmatrix} \text{---}(\boldsymbol{x}^{(1)})^T\text{---} \\ \text{---}(\boldsymbol{x}^{(2)})^T\text{---} \\ \vdots \\ \text{---}(\boldsymbol{x}^{(m)})^T\text{---} \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

then

$$\boldsymbol{e} = \begin{bmatrix} e^{(1)} \\ \vdots \\ e^{(m)} \end{bmatrix} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} - \begin{bmatrix} (\boldsymbol{x}^{(1)})^T\boldsymbol{\theta} \\ \vdots \\ (\boldsymbol{x}^{(m)})^T\boldsymbol{\theta} \end{bmatrix} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}$$

Find the gradient: $\nabla_{\boldsymbol{\theta}} L$

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{m} (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) = \boldsymbol{e}^T \boldsymbol{e}$$

- it is a quadratic form (a quadratic form is $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}$: a row vector times a matrix times a column vector, the result is a scalar !)

$$\frac{\partial L}{\partial \boldsymbol{e}} \equiv \nabla_{\boldsymbol{e}} L = \nabla_{\boldsymbol{e}} (\boldsymbol{e}^T \boldsymbol{I} \boldsymbol{e}) = 2(\boldsymbol{I}\boldsymbol{e})^T = 2\boldsymbol{e}^T$$

- but we need $\nabla_{\boldsymbol{\theta}} L$, to apply chain rule we need:

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} = \frac{\partial (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\boldsymbol{X}$$

- finally,

$$\nabla_{\boldsymbol{\theta}} L = \frac{\partial L}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} = 2\boldsymbol{e}^T(-\boldsymbol{X}) = -2(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{X}$$

A few notes on vector derivatives: gradient as row vector

For vector to scalar function $f(\beta) : R^m \rightarrow R$: the gradient

$$\nabla_{\boldsymbol{x}} f = \left[\frac{\partial f}{\partial \beta_1}, \ldots, \frac{\partial f}{\partial \beta_m}\right] \in R^{1\times m}$$

- we adopt the convention: gradients as *row vectors*
- e.g. for $L(\boldsymbol{e}) = \boldsymbol{e}^T\boldsymbol{e}$: $\nabla_{\boldsymbol{e}} L = 2\boldsymbol{e}^T$
  - $\boldsymbol{e}$ is defined as a column vector, its transpose is a row vector

A few notes on vector derivatives: vector valued functions

The convention generalises well to $\boldsymbol{g}(\boldsymbol{\theta}) : R^n \rightarrow R^m$ functions: e.g.

$$\boldsymbol{e} = \boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}$$

- a vector to vector function: $R^n \rightarrow R^m$
- each $e^{(i)} = y^{(i)} - (\boldsymbol{x}^{(i)})^T\boldsymbol{\theta} = y^{(i)} - \sum_{j=1}^{n}(x_j^{(i)})\theta_j$ is $R^n \rightarrow R$
  - its gradient is a row vector (θ_0 and x_0 are dropped here for convenience)

$$\nabla_{\boldsymbol{\theta}} e^{(i)} = \left[\frac{\partial e^{(i)}}{\partial \theta_1}, \ldots, \frac{\partial e^{(i)}}{\partial \theta_n}\right] = \left[-x_1^{(i)}, \ldots, -x_n^{(i)}\right]$$

- the gradient for $\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{\theta})$ is

$$\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{\theta}) = \begin{bmatrix}\nabla_{\boldsymbol{\theta}} e^{(1)} \\ \vdots \\ \nabla_{\boldsymbol{\theta}} e^{(m)}\end{bmatrix} = \begin{bmatrix}-x_1^{(1)}, \ldots, -x_n^{(1)} \\ \vdots \\ -x_1^{(m)}, \ldots, -x_n^{(m)}\end{bmatrix} = -\boldsymbol{X}$$

- easier to use chain rule (matrix shapes need to match to multiple!):

$$\nabla_{\boldsymbol{\theta}} L = \frac{\partial L}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} = 2\boldsymbol{e}^T(-\boldsymbol{X}) = -2(\boldsymbol{y} - \boldsymbol{X}\theta)^T \boldsymbol{X}$$

is still a row vector

Some useful gradients

$$\frac{\partial(\boldsymbol{b}+\boldsymbol{A}\boldsymbol{x})}{\partial\boldsymbol{x}} = \boldsymbol{A}; \quad \frac{\partial(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x})}{\partial\boldsymbol{x}} = -\boldsymbol{A}$$

$$\frac{\partial\boldsymbol{x}^T\boldsymbol{a}}{\partial\boldsymbol{x}} = \frac{\partial\boldsymbol{a}^T\boldsymbol{x}}{\partial\boldsymbol{x}} = \boldsymbol{a}^T$$

$$\frac{\partial\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x}}{\partial\boldsymbol{x}} = \boldsymbol{x}^T(\boldsymbol{B}+\boldsymbol{B}^T); \quad \frac{\partial\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{x}}{\partial\boldsymbol{x}} = 2\boldsymbol{x}^T\boldsymbol{W}; \ \boldsymbol{W} \text{ is symmetric}$$

$$\frac{\partial\boldsymbol{x}^T\boldsymbol{x}}{\partial\boldsymbol{x}} = 2\boldsymbol{x}^T$$

$$\frac{\partial(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s})^T\boldsymbol{W}(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s})}{\partial\boldsymbol{s}} = -2(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s})^T\boldsymbol{W}\boldsymbol{A}, \ \boldsymbol{W} \text{ is symmetric}$$

$$\frac{\partial a^T\boldsymbol{X}\boldsymbol{b}}{\partial\boldsymbol{X}} = \boldsymbol{a}\boldsymbol{b}^T$$

Normal equation for linear regression

To find the minimum, set $\nabla_{\boldsymbol{\theta}} L = \mathbf{0}$, we have the **Normal Equations**:

$$\begin{aligned} 2(\boldsymbol{y} - \boldsymbol{X}\theta)^T \boldsymbol{X} = \mathbf{0}^T &\Rightarrow 2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\theta) = \mathbf{0} \\ &\Rightarrow \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} = \boldsymbol{X}^T\boldsymbol{y} \end{aligned}$$

Assuming $\boldsymbol{X}^T\boldsymbol{X}$ is invertible (nonsingular), we have the closed-form solution

$$\boldsymbol{\theta}_{ls} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

- "ls" means least square

$(\boldsymbol{X}^T\boldsymbol{X})$ singular case

$\boldsymbol{X}^T\boldsymbol{X}$ has to be invertible or nonsingular

- otherwise, the matrix is called ill-conditioned
- like dividing a number by 0

Note that $\text{rank}(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}) = \text{rank}(\boldsymbol{X})$

- so $\boldsymbol{X}$ has linearly dependent columns $\Rightarrow$ $\boldsymbol{X}^T\boldsymbol{X}$ singular
- e.g. the same feature but measured in different units, like inch or cm: $\boldsymbol{x}_h = k \times \boldsymbol{x}_i$
- also called highly correlated features (redundant feature for regressing $\boldsymbol{y}$)
- or more general, one of the feature is a linear combination of the rest

Deal with nonsingular $\boldsymbol{X}^T\boldsymbol{X}$

- remove problematic features
- dimension reduction first
- regularization (more on this later)

Normal equation: projection view of col(X)

Derivative is way too complicated! Let's see something cooler :-)

$$\boldsymbol{X\theta} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} = \theta_0 \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + \theta_1 \begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(m)} \end{bmatrix} + \ldots + \theta_n \begin{bmatrix} x_n^{(1)} \\ x_n^{(2)} \\ \vdots \\ x_n^{(m)} \end{bmatrix}$$

$$= \theta_0 \boldsymbol{x}_0 + \theta_1 \boldsymbol{x}_1 + \ldots + \theta_n \boldsymbol{x}_n$$

- linear combination of column vectors of $\boldsymbol{X}$

what does $\boldsymbol{y} = \boldsymbol{X\theta}$ solve ?

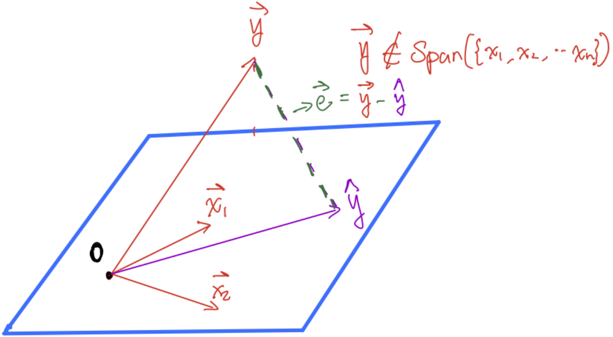
- whether $\boldsymbol{y}$ can be represented as a linear combination of column vectors of $\boldsymbol{X}$
- or $\boldsymbol{y}$ lives in the column space or not: $\boldsymbol{y} \in ?\text{span}(\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\})$

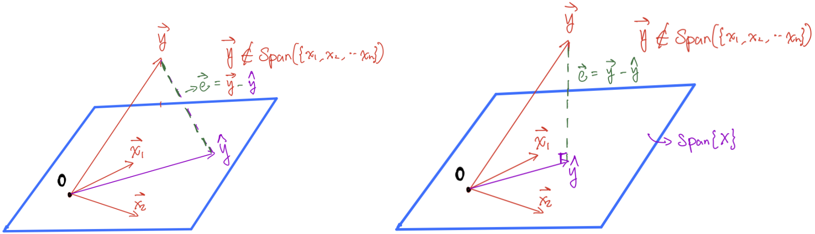
$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta}$ is over determined: $m > n$

- usually $\boldsymbol{y} \notin \text{span}(\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\})$
- but we can find its best approximation in that span:

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\theta} \in \text{span}(\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\})$$

- and minimise $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$


$\vec{y}$
$\vec{y} \notin \text{Span}(\{x_1, x_2, \cdots x_n\})$
$\rightarrow \vec{e} = \vec{y} - \hat{y}$
$\vec{x}_1$
$\hat{y}$
0
$\vec{x}_2$


$\vec{y}$
$\vec{y} \notin Span(\{x_1, x_2, \cdots x_n\})$
$\vec{e} = \vec{y} - \hat{y}$
$\vec{x}_1$
$\vec{x}_2$
$\hat{y}$
0
$\vec{y}$
$\vec{y} \notin Span(\{x_1, x_2, \cdots x_n\})$
$\vec{e} = \vec{y} - \hat{y}$
$\vec{x}_1$
$\vec{x}_2$
$\hat{y}$
0
Span{X}

$\boldsymbol{e}$ is minimised when $\hat{\boldsymbol{y}}$ is $\boldsymbol{y}$'s projection in $\boldsymbol{span}(\{\boldsymbol{x}\})$, or

$$\boldsymbol{e} \perp \text{span}(\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}) \text{ or}$$

$$\begin{cases} \boldsymbol{x}_0^T \boldsymbol{e} = 0 \\ \boldsymbol{x}_1^T \boldsymbol{e} = 0 \\ \ldots \\ \boldsymbol{x}_n^T \boldsymbol{e} = 0 \end{cases} \Rightarrow \boldsymbol{X}^T \boldsymbol{e} = \boldsymbol{0} \Rightarrow \boldsymbol{X}^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) = \boldsymbol{0}$$

Hat matrix

The projected vector :

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\theta} = \underbrace{\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T}_{\text{hat matrix}}\boldsymbol{y}$$

- "it gives $\boldsymbol{y}$ a hat": so given this name
- it is also a projection matrix: it projects $\boldsymbol{y}$ to its projection $\hat{\boldsymbol{y}}$
- note that for all projection matrix $\boldsymbol{P}$, $\boldsymbol{P}\boldsymbol{P} = \boldsymbol{P}$:

$$(\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)(\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T) = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$$

- $\boldsymbol{P}\boldsymbol{P}\ldots\boldsymbol{P} = \boldsymbol{P}$
- $\boldsymbol{P}\boldsymbol{P}\ldots\boldsymbol{P}\boldsymbol{x} = \boldsymbol{P}\boldsymbol{x}$ as expected: further projections have no effect

Gradient descent

For most models, $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \mathbf{0}$ has no closed form solution

- linear regression is probably the only exception

Gradient descent provides a more general algorithm

Remember what gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ is ?

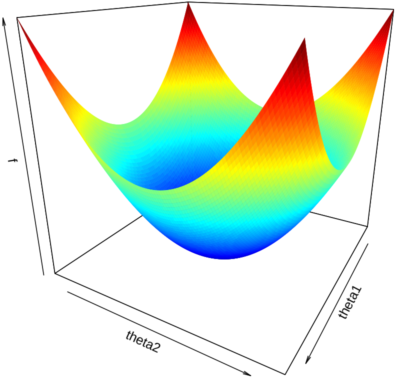
- it points to the greatest ascent direction of L at location $\boldsymbol{\theta}_t$
- gradient descent algorithm is simple
- at each t, we move by the steepest descent direction
- looping until converge:

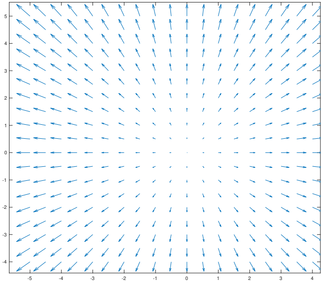
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$$

Gradient recap

For function $L(\boldsymbol{\theta})$

- the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ points to the ascent direction
  - vector field: input a location, output a direction
- the opposite $-\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ points to the steepest descent direction
- $\boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$ moves to a new position in the input space

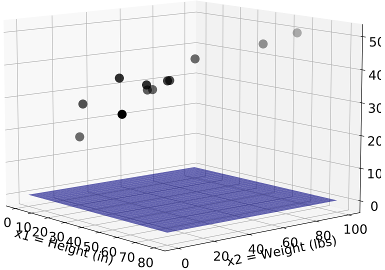




Gradient descent: step by step

Initialisation: $\boldsymbol{\theta}_0 = \mathbf{0}$;

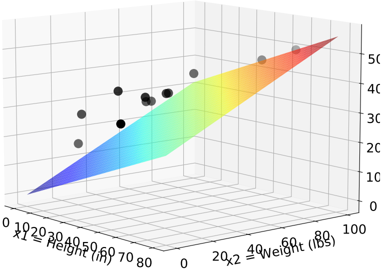
- $L = 1369.33$



Gradient descent: step by step

Step 1: $\boldsymbol{\theta}_1 = [0.007, 0.308, 0.311]$

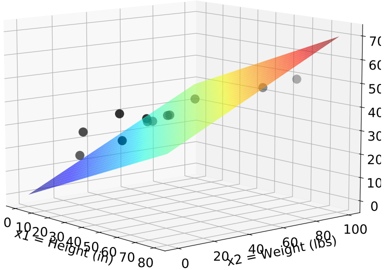
- $L = 168$



Gradient descent: step by step

Step 2: $\boldsymbol{\theta}_2 = [0.010, 0.395, 0.381]$

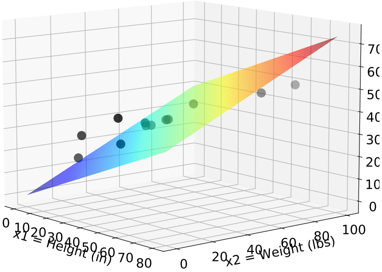
- $L = 89.22$



Gradient descent: step by step

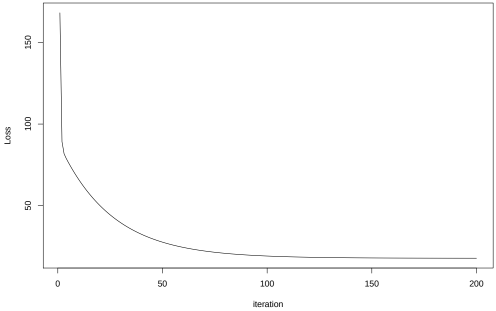
Step 3: $\boldsymbol{\theta}_3 = [0.011, 0.425, 0.391]$

- $L = 81.78$



Gradient descent

The loss function plot:



Next time

- implementation in Python
- Gaussian distribution
- linear regression: maximum likelihood (ML) estimation view
    - why squared error makes sense ?
    - uncertainty of $\boldsymbol{\theta}_{ls}$: its sampling distribution
- logistic regression
    - ML estimation
    - another gradient based optimisation method: Newton's method

Suggested reading

- ESL chapter 3:
  - I find ESL a bit too statistical; but try reading it and see how much you can understand
- ISL chapter 3
  - a bit less technical
  - the hypothesis testing bits are not essential: we are not learning statistics :-)
- Mathematics for ML by Marc Deisenroth et. al, 5.1-5.5; 7.1;
- MLAPP by Kevin Murphy, 7.1-7.3
  - we will discuss the ML view next time
-
- Hands on ML: chapter 4
  - I dont know much about this book