Today : general notions

linear regression

→ Ordinary least squares

→ gradient descent

→ Normal equation

Supervised learning

data set $\mathcal{D} = \{ x^{(i)}, t^{(i)} \}_{i=1}^{N}$ $\quad N$ = number of training examples

$x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_D^{(i)}) \in \mathbb{R}^D$ $\quad D$ = number of features

feature vectors / input variables
prototypes

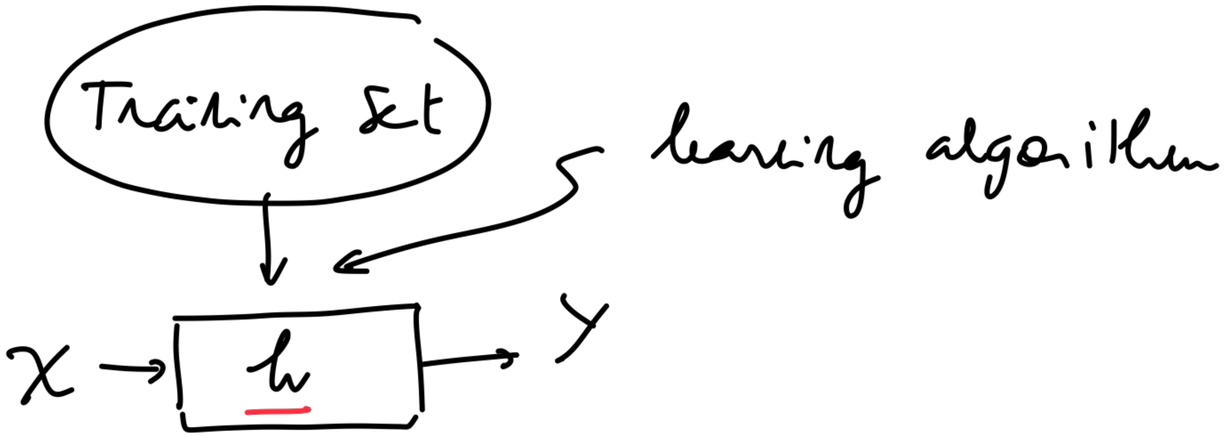
$t^{(i)} \in \mathbb{R}$ labels / annotations / output variables

$(x^{(i)}, t^{(i)})$ = training sample

the set $\{ x^{(i)}, t^{(i)} \}_{i=1}^{N}$ is called the training set

$\mathcal{X}$: input space $\subseteq \mathbb{R}^D$
$\mathcal{Y}$: output space $\subseteq \mathbb{R}$

General idea in supervised learning: learn a mapping h from X to Y such that $h(x^{(i)})$ is a good prediction for $t^{(i)}$

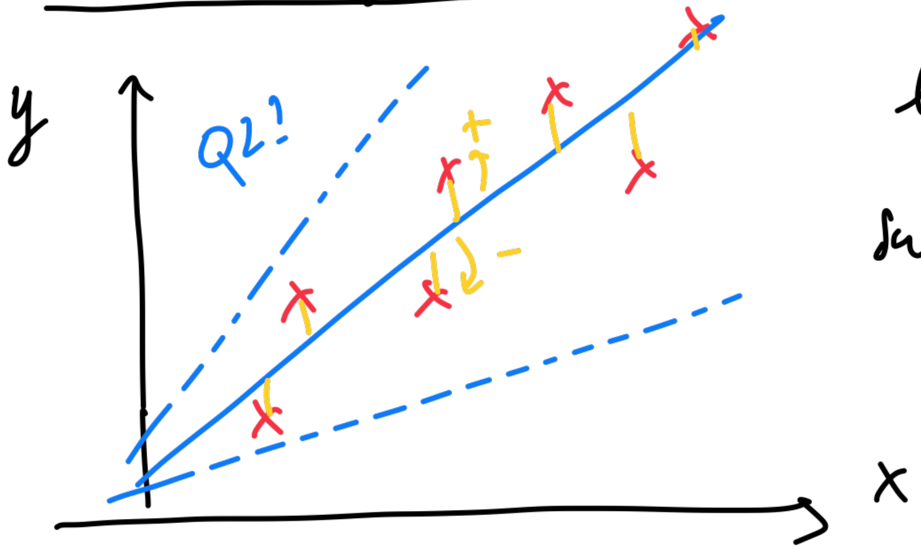


h : hypothesis

→ Question how to represent and store the hypotheses?

→ Reasonable idea: limit ourselves to finite num of parameters.

I linear regression



How to learn a good model h on $(x^{(i)}, t^{(i)})_{i=1}^{N}$ such that $h(x^{(i)}) \approx t^{(i)}$

Question 1 : What would be a good representation for h?

Question 2 : How can we learn the resulting representation?

Q1: How about we take h to be a linear combination of the features :

$$h(x^{(i)}) = \beta_0 + \beta_1 x^{(i)} \quad (D=1)$$

$$h(x^{(i)}) = \beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \dots + \beta_D x_D^{(i)}$$

(linear model)

$$h(x^{(i)}) = \underbrace{\beta_0} + \sum_{j=1}^{D} x_j^{(i)} \beta_j$$

→ To get a more compact representation, we consider the notation $\tilde{x}^{(i)} = [1, x^{(i)}]$

thanks to this notation, one can now write $h(x^{(i)})$ as

$$h(x^{(i)}) = \beta^T \tilde{x}^{(i)} = \beta_0 \cdot 1 + \beta_1 \tilde{x}_1^{(i)} + \dots + \beta_D \tilde{x}_D^{(i)}$$

$$= \beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_D x_D^{(i)}$$

$$= \langle \beta, \tilde{x}^{(i)} \rangle = \sum_{j=1}^{D} \beta_j \tilde{x}_j^{(i)}$$

Question 2 : In order to assess the quality of a model, we need a cost function (= loss function of our learning algorithm)

→ One approach is to penalize deviations between $h(x^{(i)})$ and $t^{(i)}$ for example by considering the sum of the squares of these deviations

$\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \dots + \beta_D x_D^{(i)}$

loss

$$J(\beta) = \frac{1}{2N} \sum_{i=1}^{N} \left(t^{(i)} - h(x^{(i)}) \right)^2$$

In this case in particular $J(\beta)$ is known as the ordinary least squares (OLS) loss. It is also sometimes known as Residual Sum of Squares loss (RSS)

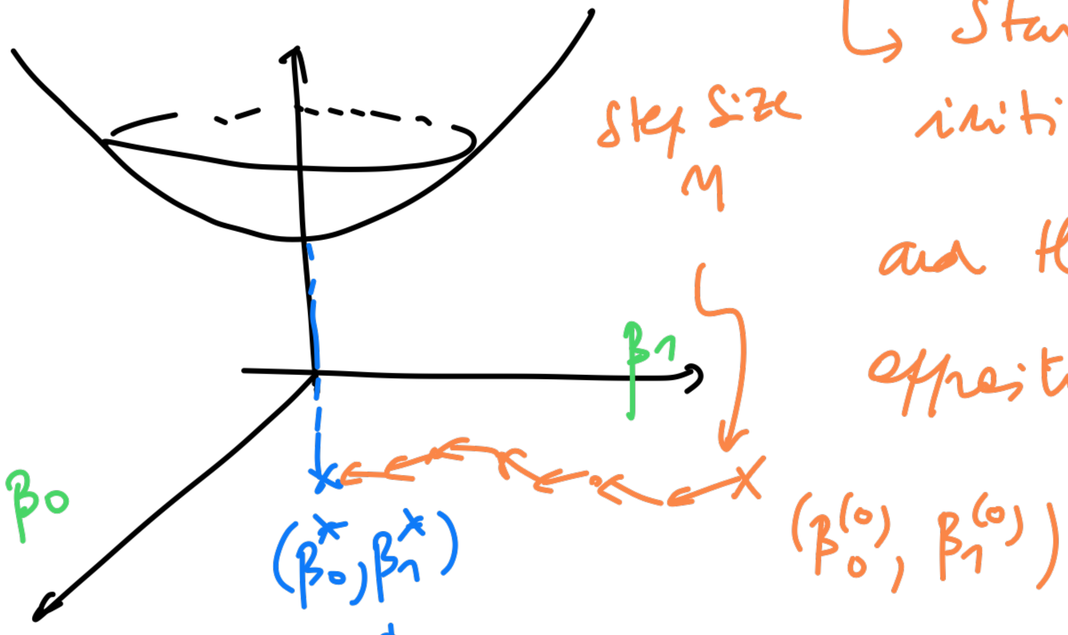
Question 2b: How can we use $J(\beta)$ in order to learn h?

<u>Approach I</u> : Minimize $J(\beta)$ with respect to $\beta_0, \beta_1, \beta_2, \dots, \beta_D$

→ To achieve this minimization we can turn to the

<u>Gradient descent algorithm</u>

↳ Start from an arbitrary initial guess $(\beta_0^{(0)}, \beta_1^{(0)})$ and then move in the direction opposite to the gradient (steepest descent direction)



optimal unknown model

Going back to $J(\beta)$ how can we apply this idea?

Step 1 generate initial value for $\beta_0, \beta_1, \dots, \beta_D$

Step 2 find the derivative of $J(\beta)$ w.r.t $\beta_0, \beta_1, \dots, \beta_D$

$$\frac{\partial J}{\partial \beta_k} = \frac{\partial}{\partial \beta_k} \frac{1}{2N} \sum_{i=1}^{N} \left(t^{(i)} - \left(\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \dots + \beta_D x_D^{(i)} \right) \right)^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \left(t^{(i)} - \left(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_D x_D^{(i)} \right) \right) \cdot \left(-\tilde{x}_k^{(i)} \right)$$

Step 3 Define the gradient as the vector of all $\frac{\partial J}{\partial \beta_k}$

$$\text{grad } J = \left[\frac{\partial J}{\partial \beta_0}, \frac{\partial J}{\partial \beta_1}, \dots, \frac{\partial J}{\partial \beta_D} \right]$$

Step 4 Move one step in the direction opposite to the gradient

$$\beta^{(\ell+1)} \leftarrow \beta^{(\ell)} - \eta \cdot \text{grad}_\beta J \qquad (*)$$

learning rate
= step size of the gradient descent algorithm

→ Common notation used = greek letter "eta" η

Repeat

Update (*) is known as the LMS (least mean square) or Widrow Hoff update.

When implementing gradient descent, there are 2 main approaches:

Batch Gradient descent

1) Carry out each step by processing the whole set of training examples $\{t^{(i)}, x^{(i)}\}_{i=1}^{N}$ (convergence more accurate but more costly)

Stochastic gradient descent

2) Only use one example at each gradient step.

Batch Gradient descent : For iter < MaxIter

uses the all N Samples $\longrightarrow$ $$\beta \leftarrow \beta - \eta \sum_{i=1}^{N} \left(t^{(i)} - h_\beta(x^{(i)}) \right) (-\tilde{x}^{(i)})$$

Stochastic gradient descent : For epoch < MaxNumEpochs

reshuffle the set $\{t^{(i)}, x^{(i)}\}_{i=1}^{N}$

For i in $[1, .., N]$

Only uses the current sample $x^{(i)}$ $\longleftarrow$ $$\beta \leftarrow \beta - \eta \left(t^{(i)} - h_\beta(x^{(i)}) \right) (-\tilde{x}^{(i)})$$