

**Supervised learning**  $\rightarrow$  data set and already know what the correct output should look like; there is a relationship between the input and the output

- Classification problem**  $\rightarrow$  predict in a discrete output  
"1 or 0"

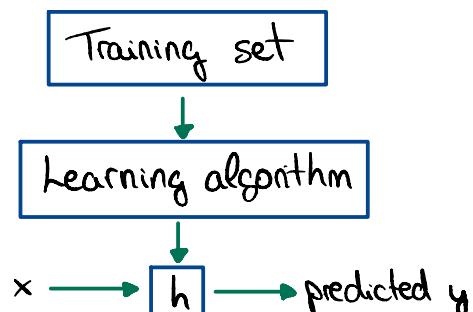
- Regression problem**  $\rightarrow$  predict within a continuous value  
"1000's of values"

**Unsupervised learning**  $\rightarrow$  problems with little or no idea what the result should look like; can derive structure from data where we don't necessarily know the effect of the variables.

- Clustering**  $\rightarrow$  derive the data based on relationships among the variables in the data.
- Cocktail party**  $\rightarrow$  find structure in a chaotic environment

## Model representation (2D idea)

$m \rightarrow$  number of training examples  
 $h \rightarrow$  hypothesis  $h_{\theta}(x) = \theta_0 + \theta_1 x$   
 $\theta_0, \theta_1$  } parameters (to determine)

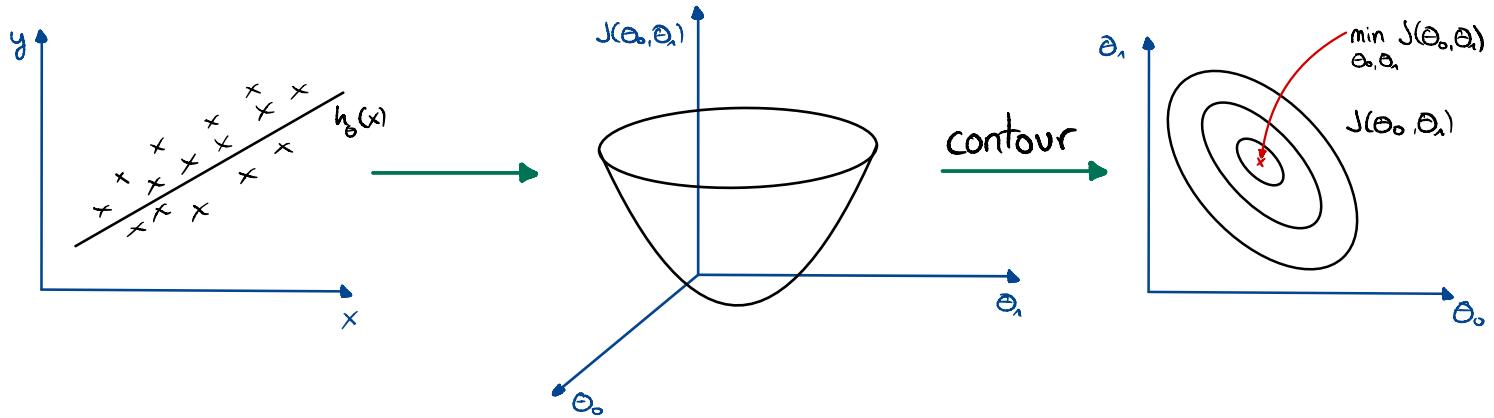


## Cost function

Is used to measure the accuracy of the hypothesis function.  
 Takes an average difference of all results of the hypothesis with inputs from x's and the actual output y's

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1), \text{ where } J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$$

Squared error function  
 Mean squared function



## Parameter learning

Gradient descent  $\rightarrow$  take the derivative (the tangential line to a function) of the cost function

Gd algorithm:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \leftarrow \text{repeat until convergence}$$

$\alpha \rightarrow$  "learning rate"  $\rightarrow$  size of each step       $\alpha \uparrow \uparrow \rightarrow$  small step  
 $\alpha \uparrow \uparrow \rightarrow$  large step

At each iteration  $j$ , one should simultaneously update the parameters  $\theta_0, \theta_1$ . Updating a specific parameter prior to calculating another one on the  $j^{(th)}$  iteration would yield to a wrong implementation

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp0}$$

$$\theta_1 := \text{temp1}$$

At the specific case of linear regression:

$$\text{temp0} := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x_i) - y_i)$$

$$\text{temp1} := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x_i) - y_i) x_i$$

$$\theta_0 := \text{temp0}$$

$$\theta_1 := \text{temp1}$$

## Multivariate linear regression

$x_j^{(i)}$  → value of feature  $j$  in the  $i^{\text{th}}$  training example  
 $x^{(i)}$  → input (features) of the  $i^{\text{th}}$  training example  
 $m$  → the number of training examples  
 $n$  → the number of features

Multivariate form of the hypothesis function:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Using matrix multiplication:

$$h_{\theta}(x) = [\theta_0 \ \theta_1 \ \dots \ \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

Note that  $x_0^{(i)} = 1$  for  $(i \in 1, \dots, m)$

Gradient descend algorithm:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \text{for } j := 0 \dots n$$

repeat until convergence

Feature scaling → divide the input values by the range of the input variable  $\rightarrow -1 \leq x_{c(i)} \leq 1$

↑ max value - min value

$$x_i := \frac{x_i}{s_i}$$

Mean normalization → subtract the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero  $\rightarrow -0.5 \leq x_{c(i)} \leq 0.5$

$$x_i := \frac{x_i - \mu_i}{s_i}$$

$\mu_i$  → average of all the values for feature  $(i)$

$s_i$  → range of values (max-min) (or standard deviation)

## Debugging gradient descend



Linear regression improvement  $\rightarrow$  combine multiple features into one

$$x_3 = x_1 x_2$$

Polynomial regression  $\rightarrow$  change the behavior of the curve

$$\begin{aligned} h_{\theta}(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 \\ h_{\theta}(x) &= \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1} \end{aligned}$$

Normal equation  $\rightarrow$  minimize  $J$  by explicitly taking its derivatives with respect to the  $\theta_j$ 's, and setting them to zero. This allows to find the optimum theta without iteration

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

# There is no need to do feature scaling

### Gradient Descent

- Need to choose alpha
- Needs many iterations
- $O(kn^2)$
- Works well when n is large  
 $n > 10^5$

### Normal Equation

- No need to choose alpha
- No need to iterate
- $O(n^3)$ , need to calculate  $(X^T X)^{-1}$
- Slow if n is very large  
 $n < 10^5$

Use pinv rather than inv  $\rightarrow$  this gives a value of  $\theta$  even if  $X^T X$  not invertible

Common  $X^T X$  not invertible causes

- Redundant features  $\rightarrow$  remove the dependency
- Too many features ( $m \leq n$ )  $\rightarrow$  eq < features

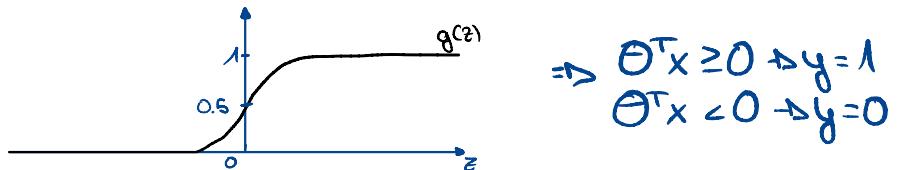
## Classification problem

Linear regression doesn't work well, with the hypothesis  $0 \leq h_{\theta}(x) \leq 1 \rightarrow \Theta^T x$  can be expressed with the Sigmoid function or Logistic function:

$$h_{\theta}(x) = g(\Theta^T x)$$

$$z = \Theta^T x$$

$$g(z) = \frac{1}{1+e^{-z}}$$



in this way  $h_{\theta}(x)$  gives the probability that the output is 1.

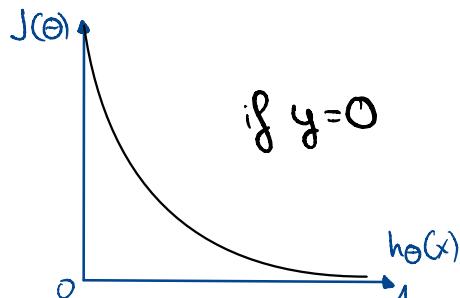
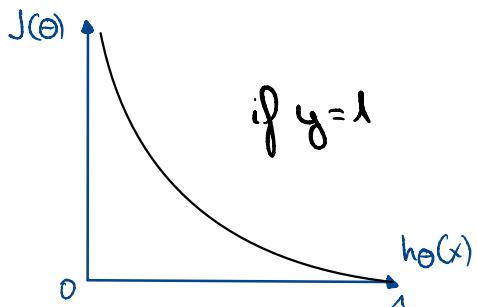
$$h_{\theta}(x) = P(y=1|x; \Theta) = 1 - P(y=0|x; \Theta)$$

Decision boundary  $\rightarrow$  the line that separates the area where  $y=0$  and where  $y=1$  created by the hypothesis function.

Cost function  $\rightarrow$  it is not possible to use the same cost function as linear regression because the Logistic Function will cause the output to be wavy, causing many local optima. Instead, the cost function for logistic regression looks like:

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)}) \text{, where } \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)}) = \frac{1}{2} (h_{\theta}(x_i) - y_i)^2$$

$$\begin{aligned} \text{Cost}(h_{\theta}(x), y) &= -\log(h_{\theta}(x)) && \text{if } y=1 \\ \text{Cost}(h_{\theta}(x), y) &= -\log(1-h_{\theta}(x)) && \text{if } y=0 \end{aligned}$$



$J_{\text{min unc}}$

## Properties

$\text{Cost}(h_{\theta}(x), y) = 0$  if  $h_{\theta}(x) = y$   $\begin{cases} y=0 & h_{\theta}(x)=0 \\ y=1 & h_{\theta}(x)=1 \end{cases}$   $\rightarrow \text{Cost}=0$

$\text{Cost}(h_{\theta}(x), y) \rightarrow \infty$  if  $y=0$  and  $h_{\theta}(x) \rightarrow 1$   $y=0, h_{\theta}(x)=1 \rightarrow \text{cost}=\infty$

$\text{Cost}(h_{\theta}(x), y) \rightarrow \infty$  if  $y=1$  and  $h_{\theta}(x) \rightarrow 0$   $y=1, h_{\theta}(x)=0 \rightarrow \text{cost}=\infty$

Simplified version of the cost function

$$\text{Cost}(h_{\theta}(x), y) = -y \log(h_{\theta}(x)) - (1-y) \log(1-h_{\theta}(x))$$

then:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_{\theta}(x^{(i)})) - (1-y^{(i)}) \log(1-h_{\theta}(x^{(i)}))]$$

where, a vectorized implementation is:

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m} [-y^T \log(h) - (1-y)^T \log(1-h)]$$

Gradient descent

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_{j(i)} \quad \text{for } j=0 \dots n$$

repeat until convergence

# in this case  $h_{\theta} = \frac{1}{1+e^{-\theta^T x}}$

where, a vectorized implementation is:

$$\theta := \theta - \alpha \frac{1}{m} X^T (g(X\theta) - y)$$

Advanced optimization:

Conjugate gradient, BFGS, L-BFGS more sophisticated and faster ways to optimize  $\theta$ . To compute this, there are two steps:

1) Provide a function that evaluates  $J(\theta)$  and  $\frac{\partial}{\partial \theta_j} J(\theta)$

function [ $j\backslash \text{Val}$ , gradient] = costFunction(theta)

$j\backslash \text{Val} = [\dots \text{code to compute } J(\theta) \dots]$

gradient =  $[\dots \text{code to compute } \frac{\partial}{\partial \theta_j} J(\theta) \dots]$

end

## 2) Use function fminunc

```
options = optimset ('GradObj', 'on', 'MaxIter', 100)
```

```
initialTheta = zeros (2, 1)
```

```
[OptTheta, functionVal, exitFlag] =
```

```
= fminunc (@costFunction, initialTheta, options)
```

## Multiclass classification : One-vs-all

Since  $y \in \{0, 1, \dots, n\}$ , we divide our problem into  $n+1$  binary classification problems. In each one, we predict the probability that  $y$  is a member of one of our classes.

$$y \in \{0, 1, \dots, n\}$$

$$h_\theta^{(0)}(x) = P(y=0|x; \theta)$$

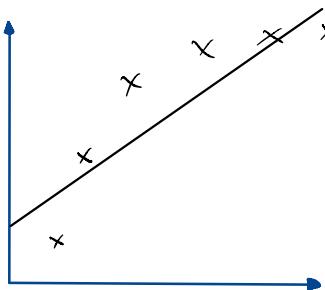
$$h_\theta^{(1)}(x) = P(y=1|x; \theta)$$

$$h_\theta^{(n)}(x) = P(y=n|x; \theta)$$

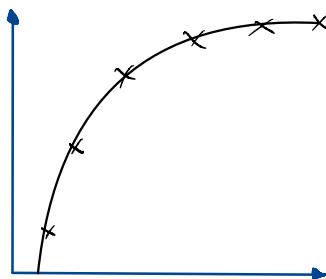
} probability that one of them ( $y=0; y=1; \dots; y=n$ ) compared to the rest is

$$h_\theta^{(n)}(x) = 1$$

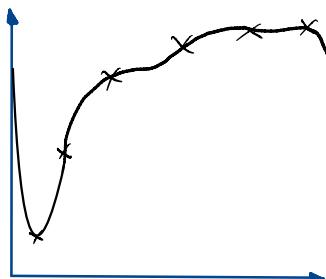
Overfitting → even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor



Underfitting  
"high bias"



It's right



Overfitting  
"high variance"

Terminology for linear and logistic regression.

Main options to address the overfitting:

1) Reduce the number of features

- \* Manually select which features to keep

- \* Use a model selection algorithm

2) Regularization

- \* Keep all the features, but reduce the magnitude of  $\theta$

- \* Works well when there are a lot of slightly useful features.

**Regularization**  $\rightarrow$  Reduce the weight that some of the terms in the function carry by increasing their cost:

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2$$

$\lambda$   $\rightarrow$  "Lambda"  $\rightarrow$  regularization parameter. It determines how much the costs of our theta parameters are inflated. If it is too large, it may smooth out the function too much and cause underfitting.

## Regularized linear regression

### Gradient descent

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)} \\ \theta_j &:= \theta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \text{ for } j = 1 \dots n\end{aligned}$$

repeat until convergence

where  $\frac{\lambda}{m} \theta_j$  performs the regularization

The equation introduced above can also be represented as

$$\theta_j := \theta_j (1 - \frac{\lambda}{m} \alpha) - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

where  $1 - \frac{\lambda}{m} \alpha < 1$

### Normal equation

$$\theta = X^T X + \lambda L^{-1} X^T y$$

$$\text{where } L = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Recall that if  $m < n$ , then  $X^T X$  is non-invertible. However, when the term  $\lambda L$  is added, then  $X^T X + \lambda L$  becomes invertible.