Aalto University
School of Electrical
Engineering

Lecture 8: Gradient descent and logistic regression

17.11.2016

Course contents

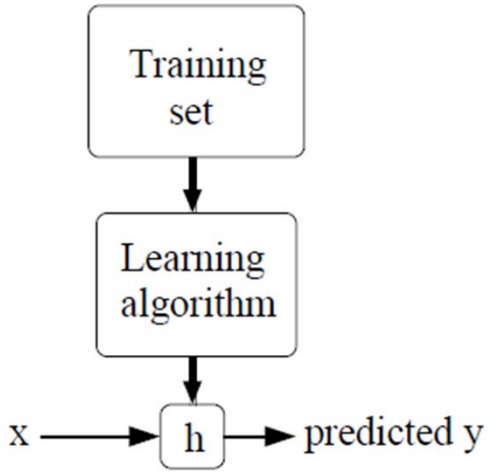
- Lecture 1: Introduction and basic principles
- Lecture 2: Covariance and Gaussianity
- Lecture 3: Multivariate linear regression
- Lecture 4: Principal component analysis
- Lecture 5: Bridging input and output
- Lecture 6: Gaussian Mixture Models (Pedram Daee)
- Lecture 7: Gaussian Process Regression (Pedram Daee)
- Lecture 8: Gradient descent and logistic regression

Mathematical terms

- $x^{(i)}$: input variables , also called input **features**
- $y^{(i)}$: output or **target** variable that we are trying to predict
- A pair ($x^{(i)}$, $y^{(i)}$) is called a **training example**
- The dataset that we'll be using to learn a list of *m* training examples ($x^{(i)}$, $y^{(i)}$) ; *i* ={1, . . .,*m*} is called a **training set**.
- We use *X* denote the space of input values, and *Y* the space of output values.

Model learning

- Our goal is, given a training set, to learn a function $h : X \rightarrow Y$ so that h(x) is a "good" predictor for the corresponding value of y.
- Function *h* is called a **hypothesis**.



Hypothesis function

- For example, if the input has two variables (two-dimensional) R^2 , we should decide how to represent hypotheses *h*
- let's say we decide to approximate y as a linear function of X

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- $\theta_i^{'}$s are parameters or weights

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

- In this course, always include intercept parameter of x_0 and make it equal to 1 : x_0=1 . This deals with offset.

How to learn parameter θ

- make $h(x)$ close to y or minimize the following cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

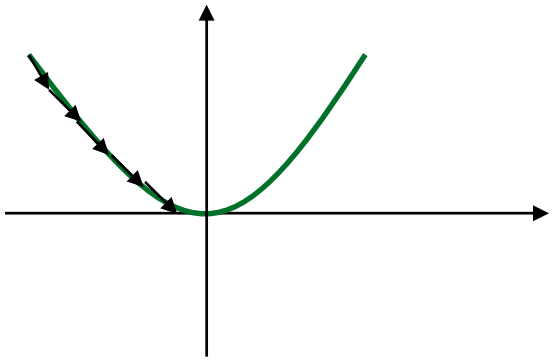
- To do so, let's use a search algorithm that starts with some "initial guess" for θ, and that repeatedly changes θ to make J(θ) smaller
- We need an algorithm starting with an initial θ, repeatedly performs updating until convergence of J

Gradient descent

- A natural algorithm that repeatedly takes step in the direction of steepest decrease of J

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- α is called the learning rate



$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y)\, x_j
\end{aligned}$$

- For a single training example:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

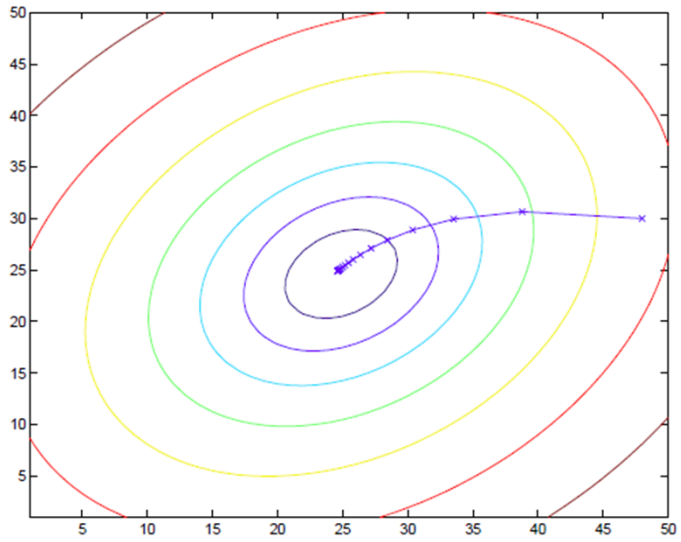
Global minima for linear regression

- Gradient descent can be susceptible to local minima
-
- But, for linear regression it has only one global, and no other local, optima

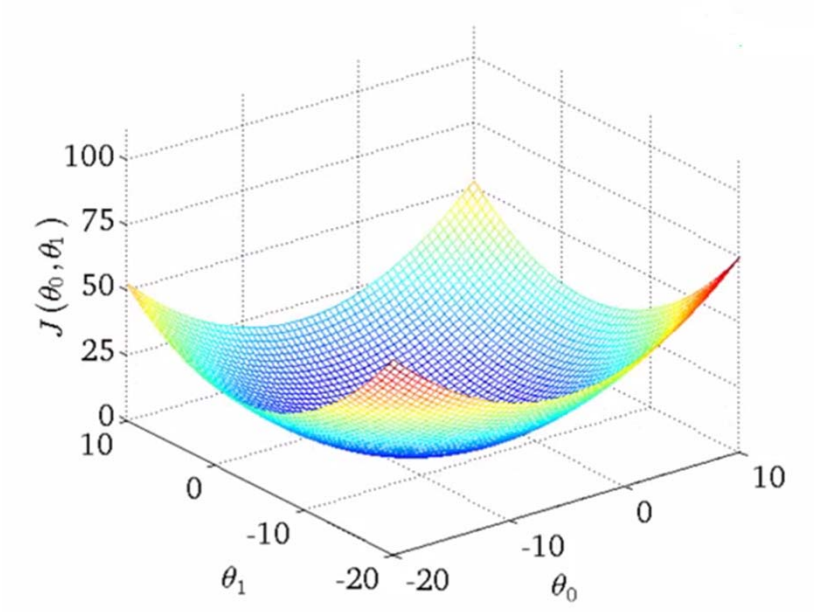
- Because J is a convex quadratic function, thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum

Convex cost function

- For linear regression problem, the defined cost function is convex. Always converges to global optimum.

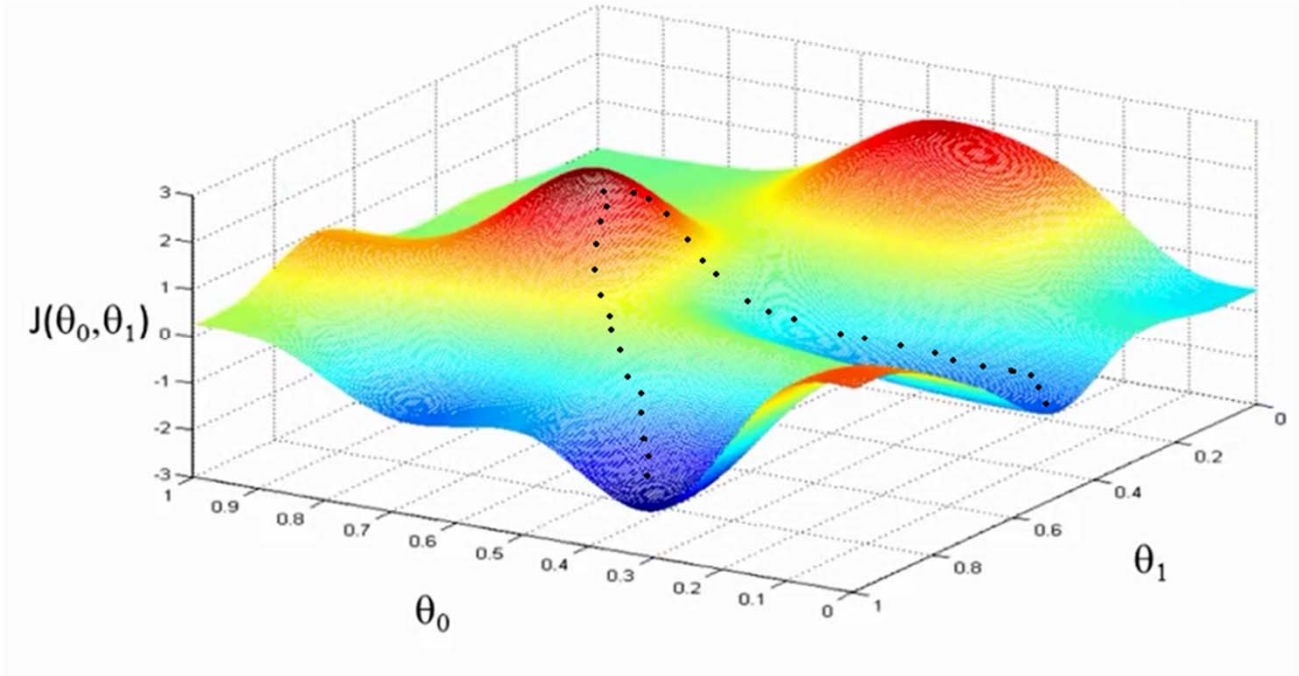


contours of the quadratic function



Non-convex cost function

- Depends on where to start, we might end up to diferent local optimas



Batch and stocastic gradient descents

- Batch:

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}$$

}

- Stocastic:

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}$$

}

}

Batch and stocastic gradient descents

- Batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if *m* is large

- Stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at.

- When the training set is large, stochastic gradient descent is often preferred over batch gradient descent

Some points before implementation

- Make sure features are on a similar scale.
- It makes the counour plots look more circular and faster to reach to the global optima
- Scaling can be done using variance or standard deviation of the variable

Correct implementation

Correct: Simultaneous update

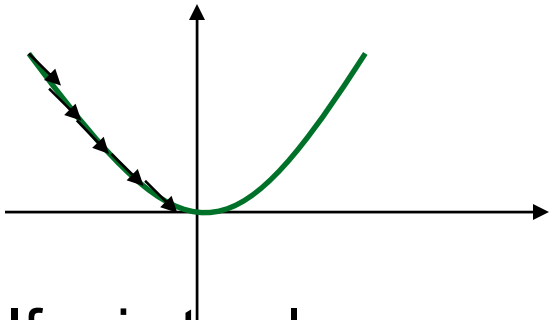
$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

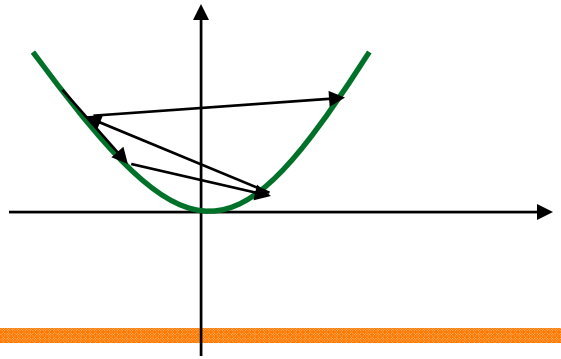
$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Selecting learning rate

- If α is small, gradient descent can be slow

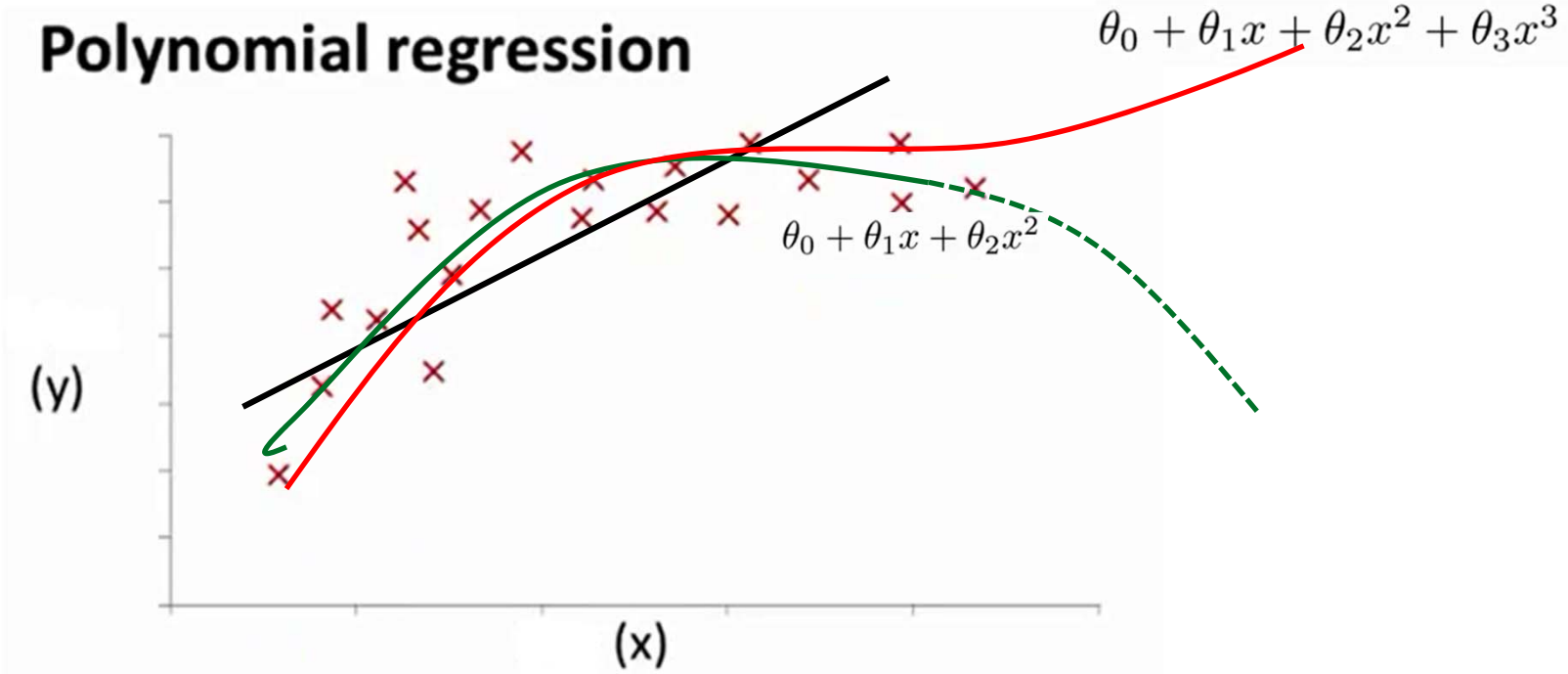


- If α is too lrage, gradient descent might overshoot the minimum



Polynomial fitting

- Adding features to the model can make better fit



- Overfitting is problem
- Cross validation is one way to avoid overfitting

Gradient descent versus normal equation

Gradient Descent	Normal Equation
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$	$F_{\text{MLR}} = \left(X^T X \right)^{-1} X^T Y.$
Multiple iterations and many steps to reach to global optimum	One step to get to the optimal value
Need to choose α	No need to choose α
Works fine for large number of features	Slow if features are large
Time complexity is O(n)	Time complexity is $O(n^3)$ for $(X^TX)^{-1}$
Not closed solution for all error measures: |y-f(x)| or any non-differentiable term	Not motivated when X^TX is not invertible. Pseudo inverse can help to some extend

Classification

- Binary classification

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

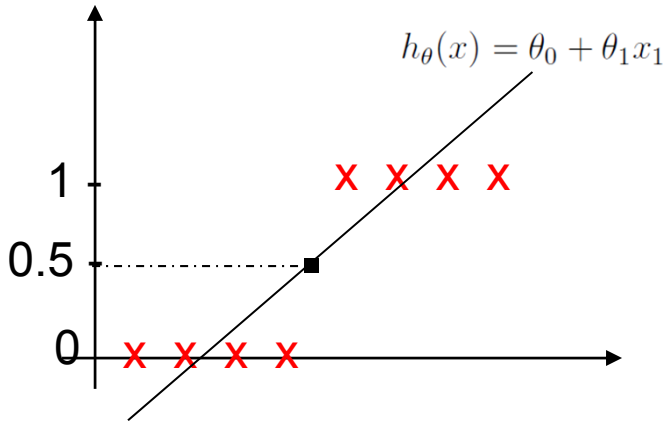
0: "Negative Class" (e.g., benign tumor)
1: "Positive Class" (e.g., malignant tumor)

- Multi-class classification
  - y ∈ {0,1,2,3,…}

❑ A threshold is defined to classify

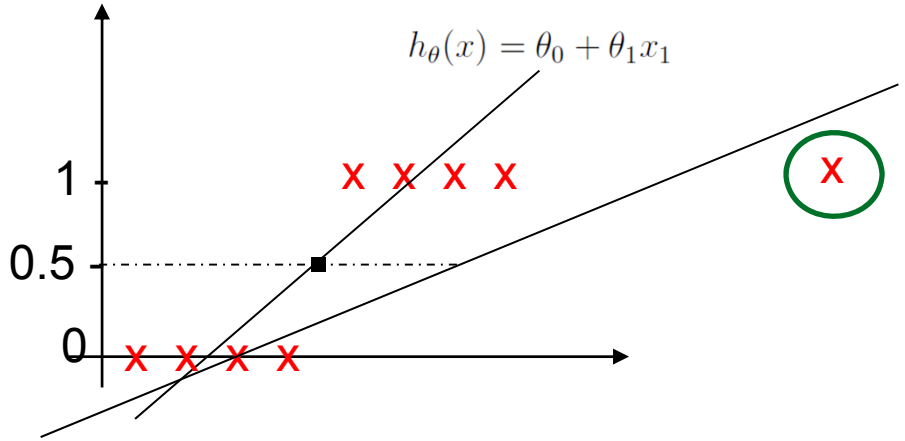
If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"



Linear regression for classification

- Applying linear regression for classification is often not useful



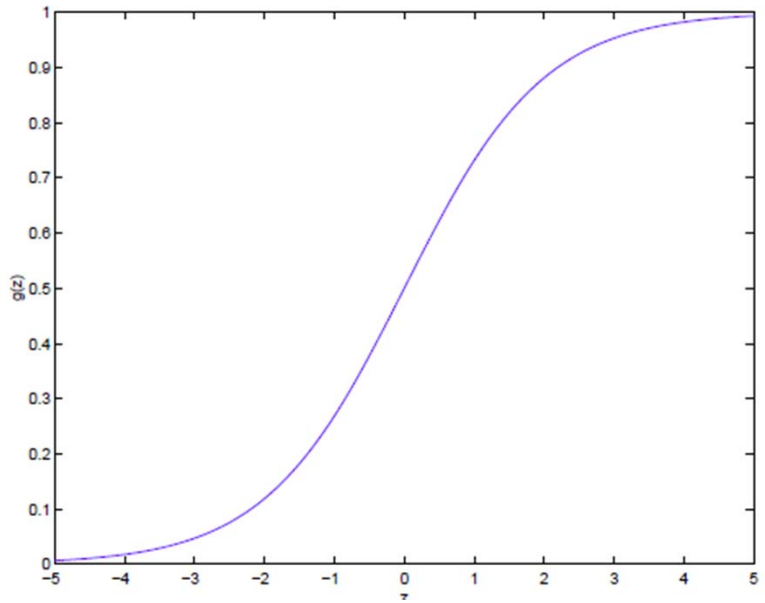
- $h_\theta(x)$ can be a large positive or negative value while y is 0 or 1
- Logistic regression : $0 \leq h_\theta(x) \leq 1$
  - A classification problem not regression despite the name

Change of hypothesis

- Logistic or sigmoid function

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

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



- Derivative of sigmoid:

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\ &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

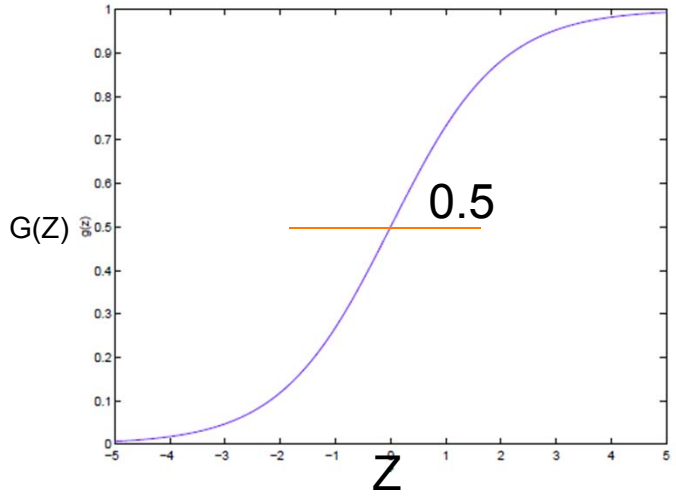
Logistic regression

- Assume:

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

- It can be written as:

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$



- We can see: g(z)= $h_\theta(x) = g(\theta^T x) \geq 0.5$ if $Z \geq 0$
  And g(z) < 0.5 if Z< 0

Cost Function and optimization

- Linear regression cost function was convex

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- The same cost function for logestic regression is non-convex because of nonlinear sigmoid function

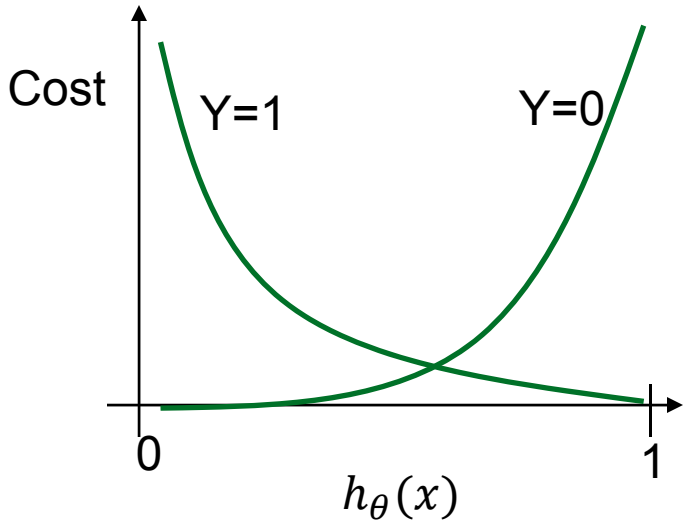
$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

- We define logistic regression cost function as :

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

Convex cost function for logistic regression

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



- If h goes to zero and Cost also goes to zero, Class 0 is selected
- If h goes to 1 and Cost goes to zero, class 1 is selected

Cost function for logistic regression

$$
\begin{aligned}
P(y = 1 \mid x; \theta) &= h_\theta(x) \\
P(y = 0 \mid x; \theta) &= 1 - h_\theta(x)
\end{aligned}
$$

- It can be written as : $p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$
- For m training example,the likelihood of the parameters:

$$
\begin{aligned}
L(\theta) &= p(\vec{y} \mid X; \theta) \\
&= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\
&= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}
\end{aligned}
$$

- It will be easier to work on the log likelihood and instead of minimizing the cost function we will maximize the log likelihood function:

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$

- Given: $g'(z) = g(z)(1 - g(z))$

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y(1 - g(\theta^T x)) - (1 - y) g(\theta^T x)\right) x_j \\ &= (y - h_\theta(x))\, x_j\end{aligned}$$

- Since we are maximizing rather than minimizing, gradient ascent applied for parameter optimization

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta)$$

- The parameter updating algorithm:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

- Looks similar to Least-Mean squared errors
- However, they are different due to the fact that $\theta^T x^{(i)}$ is a nonlinear function of $h_\theta(x^{(i)})$
- But the updating rules of the parameters are the same

Another algorithm for optimization

- Gradient descent takes many steps iteratively to reach to the optima
- The parameter α should be manually set
- There are other lagorithms converging faster than gradient descent with no need to pick α
- However, they are more complex than gradient descent
- We know newton methods for finding zeros of a function:

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

Optimization using newton's method

- In fact, newton method is approximating finction f using a linear function f ' , which is the tangent of f at the current guess of parameter θ
- It solves until function f equals to zero.
- Newton method is a way of finding zeros. What about finding the maxima of a function ℓ ?
- Maxima of a function occurs where its derivative is zero:
  - $\ell'(\theta) = 0$
- Therefore, in newton method, by replacing $f(\theta) = \ell'(\theta)$
- The same algorithm can be sued:

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

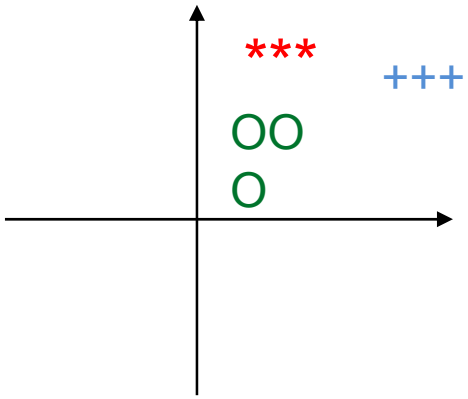
Newton- Raphson

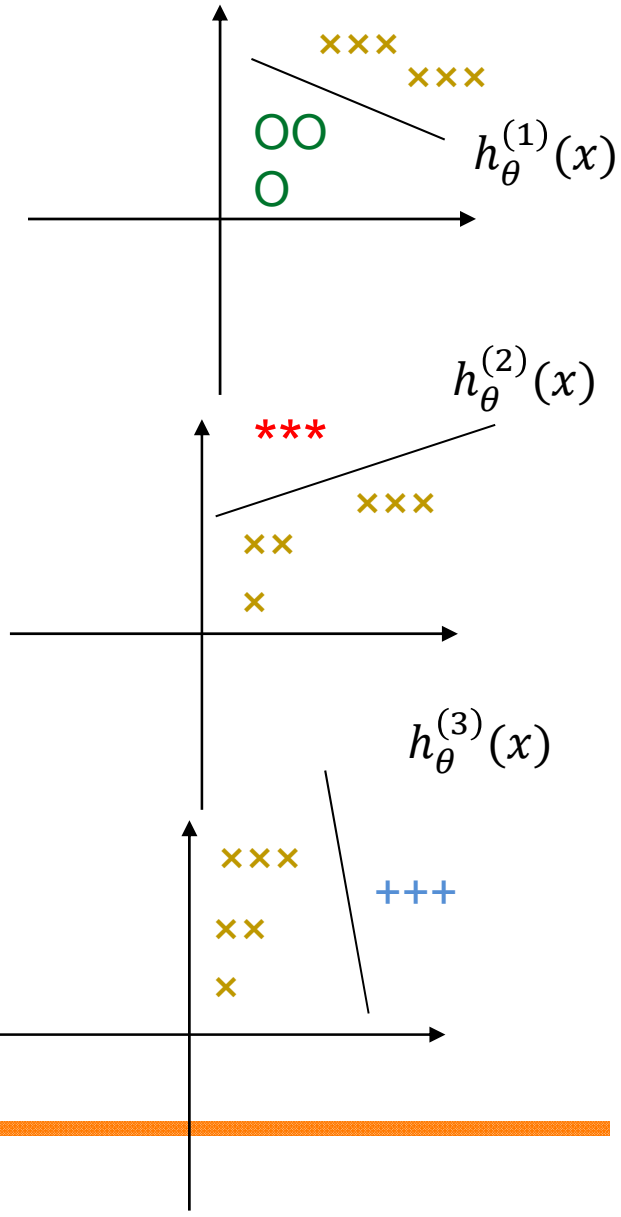
- Generalization of Newton method to multi-dimensional set is called Newton Raphson:
- $\theta := \theta - H^{-1}\nabla_\theta \ell(\theta)$ where H is the hessian: $H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}$
- Newton's method usually converges faster than gradient descent when maximizing logistic regression log likelihood.
- Each iteration is more expensive than gradient descent because of calculating inverse of Hessian
- As long as data points are not very large, Newton's methods are preferred.

Advanced optimizaiton algorithms

- It is recommended to use the built-in functions or softwares when using advanced optimizaiton algorithms rather than coding the algorithm yourself
- For this course, it is recommended to use advanced optimization function of Matlab : fminunc
- Built-in functions apply other methods of optimization which are faster . E.g. quasi-Newton methods instead of Newton's methods
- Newton's methods calculate H directly. Calculating H numerically involves a large amount of computation. Quasi-Newton methods avoid this by using an approximation to H

Multi-class classification



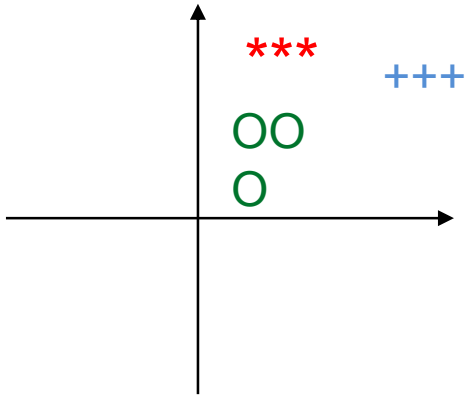


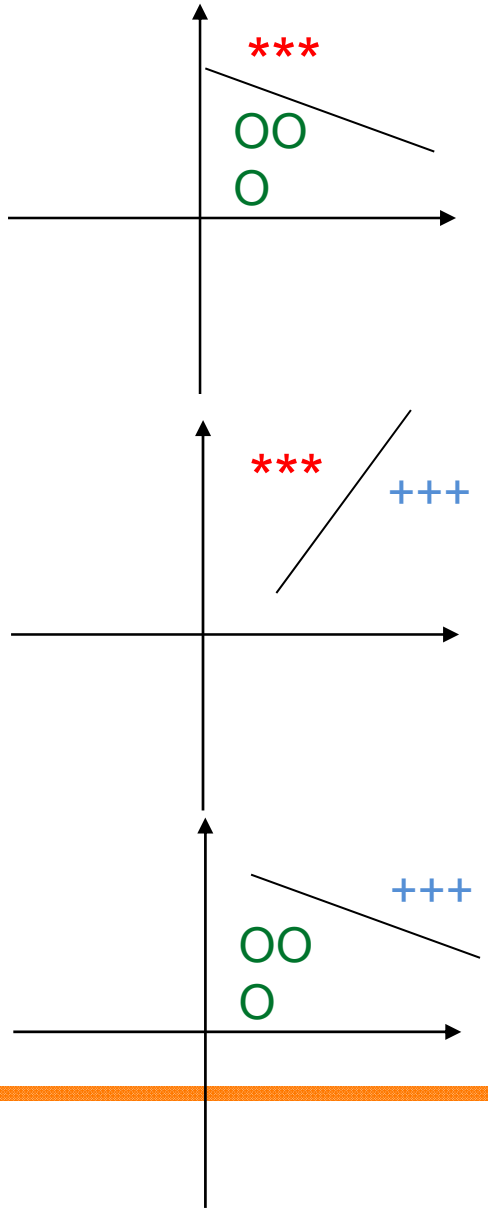
- One-vs-all strategy: working with multiple binary classifications
- We train one logistic regression classifier for each class i to predict the probability that y = i

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$

- For each x, pick the class having highest value of probability $\max_i h_\theta^{(i)}(x)$

One versus one strategy

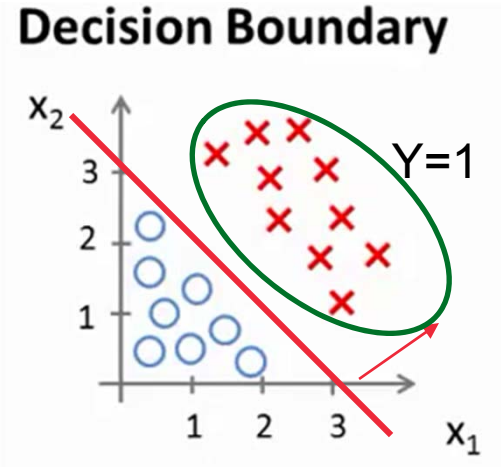




we train $\frac{C\,(C-1)}{2}$ binary classifiers corresponding to every combination of two class classifiers. For the test data, we use all the classifiers to classify the data and then count the number of times that the test data was assigned to each class. The final class is the one with the maximum number of wins.

Decision boundary

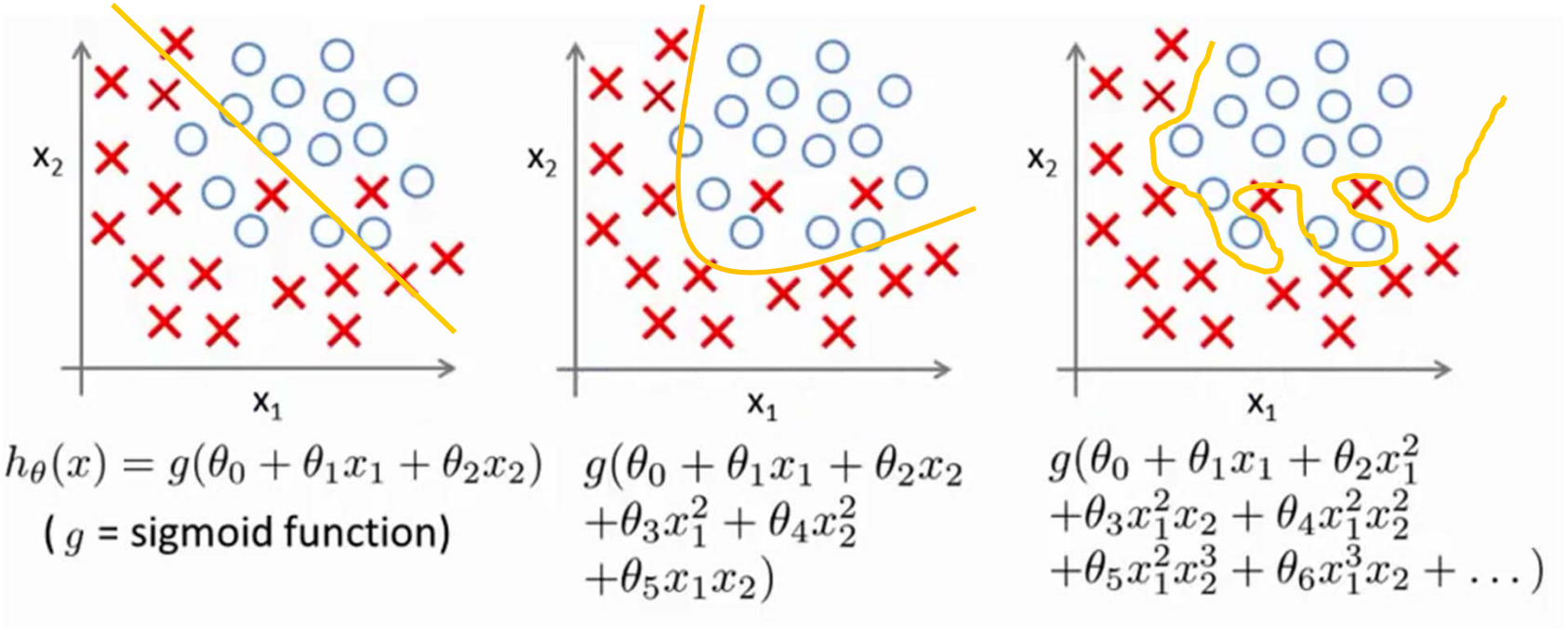
- Example: $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- If θ = [-3 1 1] , then y = 1 if -3+x1+x2 $\geq$ 0 $\rightarrow$ x1+x2 $\geq$ 3



Note:

- decision boundary is the property of the parameters not the data.
- Parameters are the property of data and learned from data
- Decision boundary can be more complex shape if higher order of polynomials applied

Overfitting



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

Uderfit, high bias

Quadratic terms: Good fit

Higher orders: Over-fit, not able to generalize the unseens

How to deal with overfitting

- Seems having higher order of polynomials is good fit, but how to deal with overfitting?
  - Reduce the number of features manually
  - Keep all the features, but apply regularization

- The most common variants in machine learning are L_1 and L_2 regularization
- Minimizing E(X, Y) + α‖w‖, where w is the model's weight vector, ‖·‖ is either the L_1 norm or the squared L_2 norm, and α is a free parameter that needs to be tuned empirically
- Regularization using L_2 norm is called Tikhonov regularization (Ridge regression), using L_1 norm is called Lasso regularization

Regularized linear regression

- Note: Do not include the intercept parameter θ_0 in the regularization

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

- If you consider a very large value for the regularization parameter λ , all the parameters θ are penalized close to zero, and intercept θ_0 is left. The decision boundary will be a straight line

Regularized regression

- Regularized gradient descent

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

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

$$(j = 1, 2, 3, \ldots, n)$$

}

Regularized logistic regression

- Similar to linear regression, the cost function is :

$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_\theta(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log 1 - h_\theta(x^{(i)}) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

- Gradient descent:

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

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

$$(j = 1, 2, 3, \ldots, n)$$

$\}$

- difference is : $h_\theta(x) = g(\theta^T x) = \dfrac{1}{1 + e^{-\theta^T x}}$

Summary

- Gradient descent is a useful optimization technique for both classification and linear regression
- For linear regression the cost function is convex meaning that always converges to golbal optimum
- For non-linear cost function, gradient descent might get stuck in the local optima
- Logistic regression is a widely applied supervised classification technique
- For logistic regression, gradient descent and newton Raphson optimization techniques were explained.