Advanced Concepts in Machine Learning

Kurt Driessens

Overview

- Practical Course Issues
  - Course format
  - Exam format

- What do YOU know about Machine Learning
- Linear/Logistic/Softmax Regression
- Neural Networks

Course Format

- 5 weeks of lectures (+ labs)
  - 4 assignments
    - Should be started at home (put time aside for this!)
    - Each Monday, there is a question asking session (optional)
- 2 last weeks: $4_{(*2)}$ student sessions
  - Review a paper
  - Present that paper
  - In pairs … and in two groups …

Lectures: Title disclaimers

1. Advanced concepts
   - Not only for you, sometimes also for me
   - Selected topics from relatively recent ML developments (and MY view on them!)
   - I expect to get questions I cannot answer (right away)

2. Selection of topics only
   - I get to pick the first weeks
   - You get to pick the last weeks

Materials

Since it is a selection of very recent topics according to (mostly) my preferences

1. I don’t follow a specific book
2. I provide lecture slides
3. I provide reading/studying material
   - I will indicate the parts that are exam material

Assessment Details

1. Written exam (60%)
   - Some **insight** questions about the lectures
   - Some **applications** of what you learned

2. A quartet of Assignments (20%)

3. Review and present a self-selected* paper (20%)

* within a predetermined set, of course

Assignments

Expect to spend about 7-8 hours on these each week.

Yes … I know. Work in pairs and comfort each other.

Expected at the end: SHORT report

Don’t repeat the assignment to me

Do explain how you will solve it/answer it/… it

INTERPRET what you see, try to draw conclusions

This often means running more than 1 experiment

Set of papers to select from

Some of last year's conferences on machine learning

1. ICML
   - http://proceedings.mlr.press/v97/
2. ECML
   - https://ecmlpkdd2019.org/programme/accepted/
   - "Research Track" or maybe "Applied Data Science"
3. Other conference?
   - NIPS, GECCO, ECAI, …
   - Possible, but pick a machine learning oriented paper from an A-level conference from 2019, and check with me! (I'm not a fan of ArXiv, but...)

Search for a paper that tickles your interest

Check EleUM for the paper's availability (first come, first served)

Mail me the paper of your choice, and I will confirm

Paper discussion + presentation

1. Read the paper + related material
   - reading only the paper will probably not be enough!
2. Write a review on the paper (max 4 pages) by the start of week 6!
3. Present the paper and your view on it during week 6 or week 7
   - Showing an interest and what you've learned during other student's presentations will positively influence your grade on this part.

This is part of your academic/scientific training!

Paper Review (incomplete)

- Short Summary of the content
  - Context/motivation (with Related Work)
  - The way it works, brief but understandable
  - Evaluation
- Your view on this
  - What is great? (3 positive points)
  - What could be improved? (3 negative points)
  - What could be done in the future?
  - …
- Treat the course as common knowledge, so relate to the course!

So ...

What do you already know about machine learning?

Some topics to get you started ...

- Lin. or Log. Regression
- Decision Trees
- Support Vector Machines
- Neural Networks
- Genetic Algorithms
- Reinforcement Learning
- Representational Issues
- Deep Learning
- Gaussian Processes

Linear, Logistic and Softmax Regression

Kurt Driessens

with slides from Andrew Ng, Hendrik Blockeel, Haitham Bou Ammar

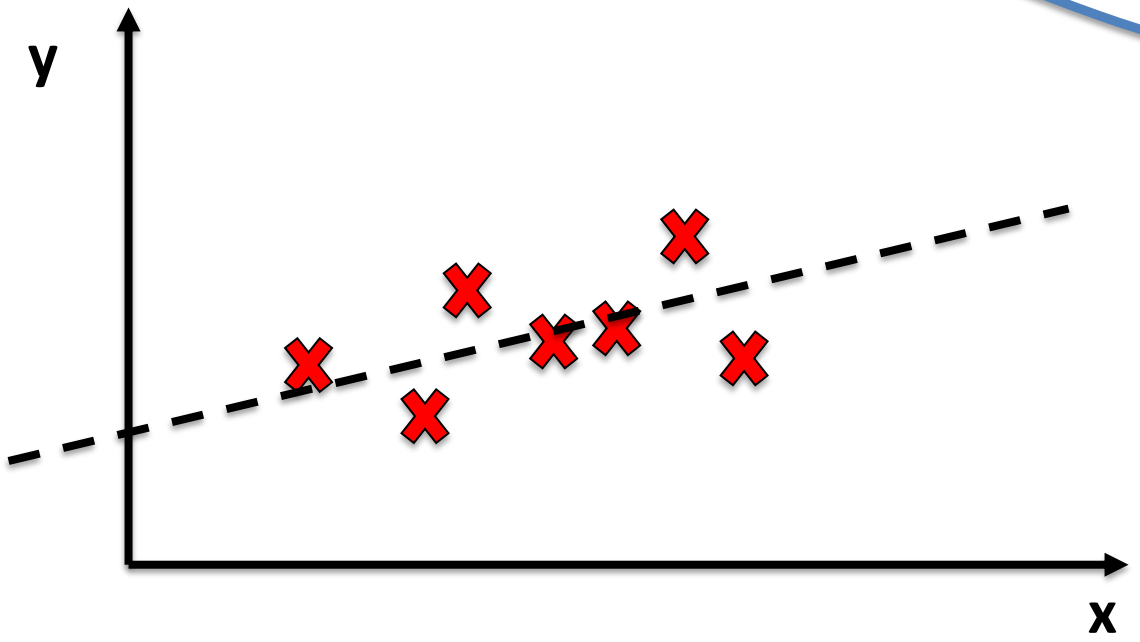
Linear Regression

Given data points $\langle \mathbf{x}_i, y_i \rangle$

Find the weights θ such that $\mathbf{x}_i^T\theta = y_i$

– e.g. for the 1-dim case below: $y = \theta_0 + \theta_1 x$

$$\mathbf{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$$



Data likelihood for $f_\theta(\mathbf{x})$

Assuming that any deviations of $f_\theta(\mathbf{x})$ from y are independent and identically distributed (i.e. Gaussian), the joint (for all examples) likelihood is:

$$p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta}) = \prod_{i=1}^{m} (2\pi)^{-\frac{n}{2}} |\sigma_n^2 \mathbf{I}|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2} (y^{(i)} - f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))^T \frac{1}{\sigma_n^2} \mathbf{I} (y^{(i)} - f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) \right\}$$

where $\mathbf{y} = [y^{(1)}, \ldots y^{(m)}]$, are all outputs

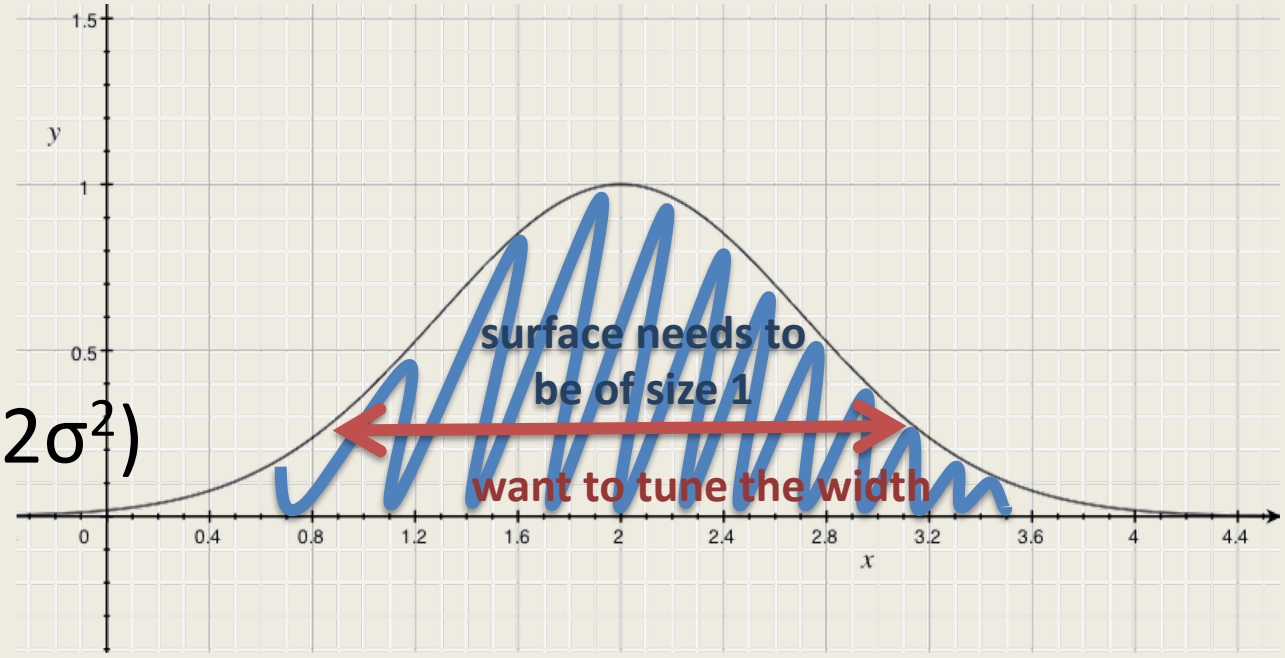
$$\mathbf{X} = \begin{bmatrix} \text{—}(\mathbf{x}_1^T)\text{—} \\ \vdots \\ \text{—}(\mathbf{x}_m^T)\text{—} \end{bmatrix}$$ are all inputs

This is a product of multivariate Gaussians. It looks scary, but is well behaved!

Recap: Gaussians

$\exp(-(x-2)^2)$

$\exp(-(x-2)^2/2\sigma^2)$



$1/(2\pi\sigma^2)^{1/2} * \exp(-(x-2)^2/2\sigma^2)$

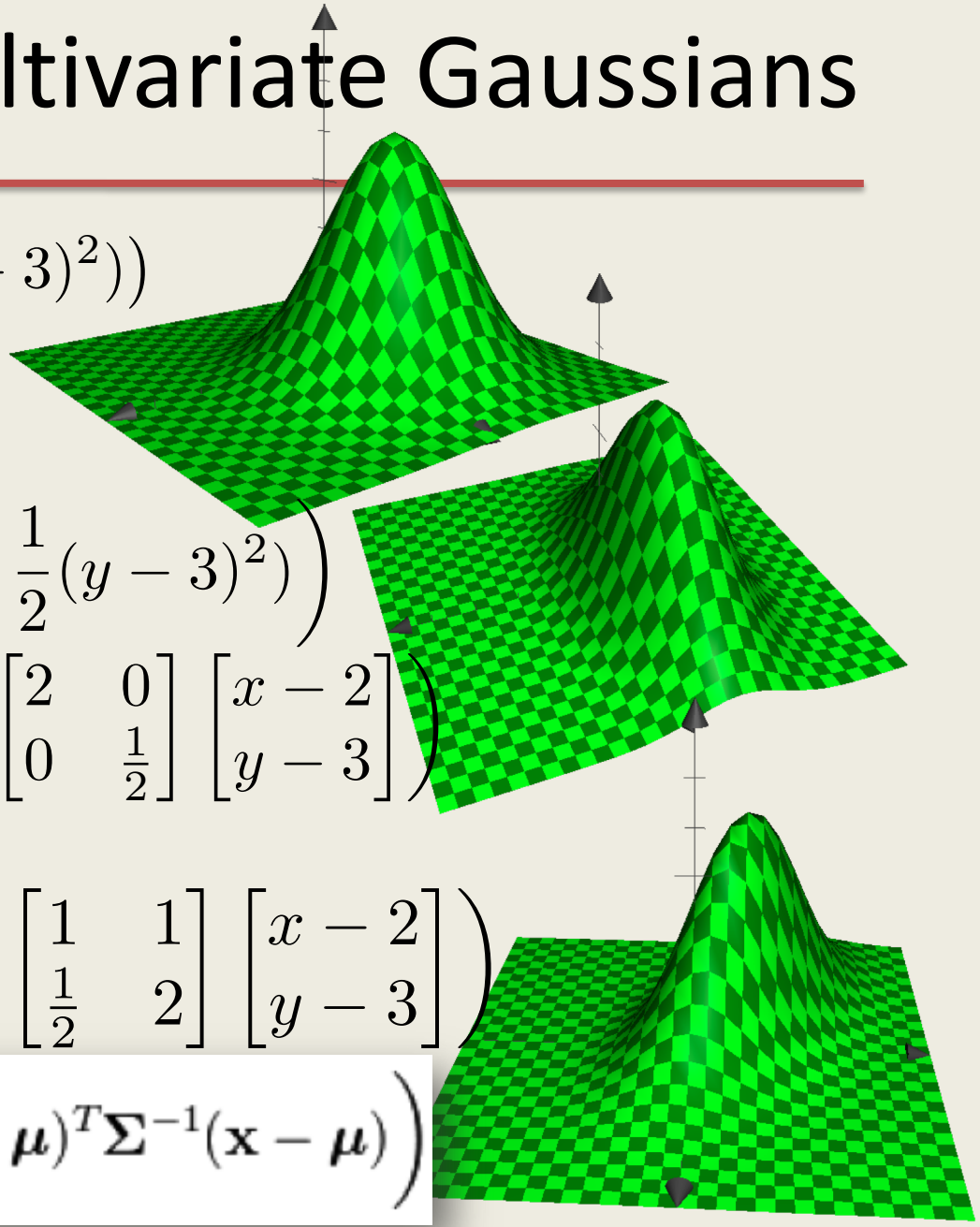
Recap: Multivariate Gaussians

$$exp\left(-((x-2)^2+(y-3)^2)\right)$$

$$exp\left(-(2(x-2)^2+\frac{1}{2}(y-3)^2)\right)$$

$$= exp\left(\begin{bmatrix} x-2 & y-3 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} x-2 \\ y-3 \end{bmatrix}\right)$$

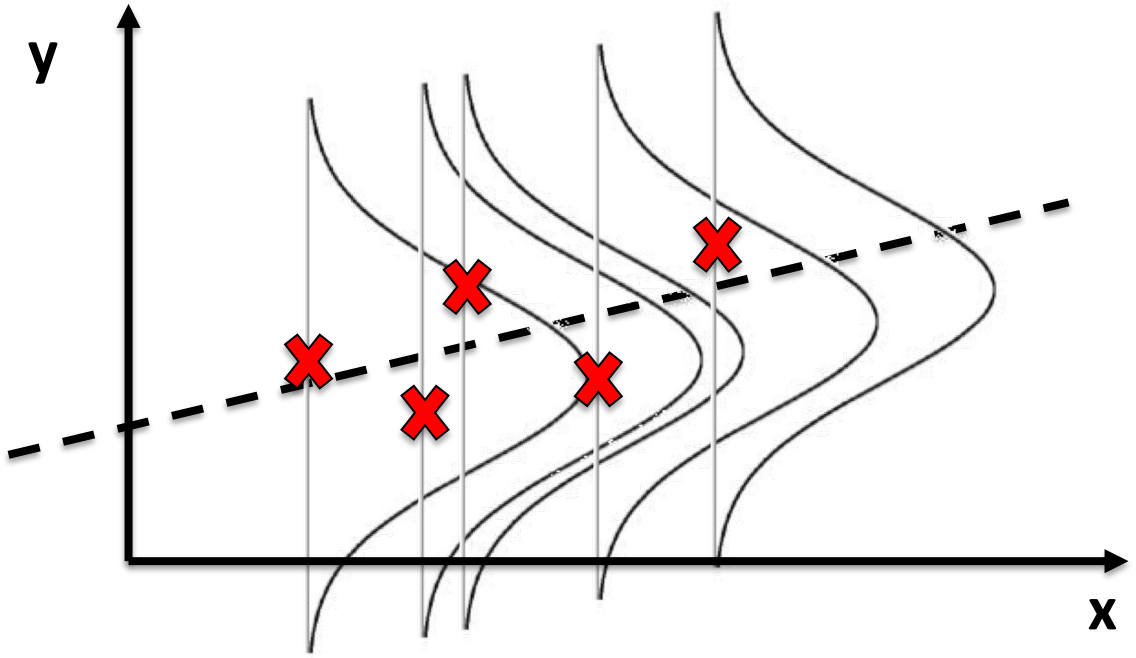
$$exp\left(\begin{bmatrix} x-2 & y-3 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ \frac{1}{2} & 2 \end{bmatrix}\begin{bmatrix} x-2 \\ y-3 \end{bmatrix}\right)$$



$$\frac{1}{(2\pi)^{k/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

Data likelihood for $f_\theta(\mathbf{x})$

$$p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta}) = \prod_{i=1}^{m}(2\pi)^{-\frac{n}{2}}|\sigma_n^2\mathbf{I}|^{-\frac{1}{2}}\exp\left\{-\frac{1}{2}(y^{(i)} - f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))^T\frac{1}{\sigma_n^2}\mathbf{I}(y^{(i)} - f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))\right\}$$



Maximum likelihood

$$\max_{\boldsymbol{\theta}} p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta})$$

= difficult

$$\downarrow$$

$$\max_{\boldsymbol{\theta}} \log\left[p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta})\right]$$

=

$$\max_{\theta} \underbrace{\sum_{i=1}^{m} \log\left[(2\pi)^{-\frac{n}{2}}|\sigma_n^2\mathbf{I}|^{-\frac{1}{2}}\right]}_{\text{= constant}} - \frac{1}{2}\sum_{i=1}^{m}\left[(y^{(i)} - f_\theta(\mathbf{x}^{(i)}))^T \frac{1}{\sigma_n^2}\mathbf{I}(y^{(i)} - f_\theta(\mathbf{x}^{(i)}))\right]$$

= easier

Least Squares Regression*

learning examples

So the problem becomes:

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{m} \left[(y^{(i)} - f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))^T (y^{(i)} - f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) \right]$$

or in matrix form:

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \left[\mathbf{y} - \mathbf{F}_{\boldsymbol{\theta}}(\mathbf{X}) \right]^T \left[\mathbf{y} - \mathbf{F}_{\boldsymbol{\theta}}(\mathbf{X}) \right]$$

For the linear case with 1-dimensional y

$$f_\theta(\mathbf{x}) = \mathbf{x}^T\theta \Rightarrow \min_\theta \frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)} - \theta^T\mathbf{x}^{(i)}\right)^2$$

or in matrix form:

$$\min_{\boldsymbol{\theta}} \frac{1}{2m}[\mathbf{X}\boldsymbol{\theta} - \mathbf{y}]^T[\mathbf{X}\boldsymbol{\theta} - \mathbf{y}]$$

= cost function $J(\theta)$

Cost function intuition

Linear Regression in 1-dim

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Simplified to 1 parameter

$$h_\theta(x) = \theta_1 x$$

$$\theta_1$$

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

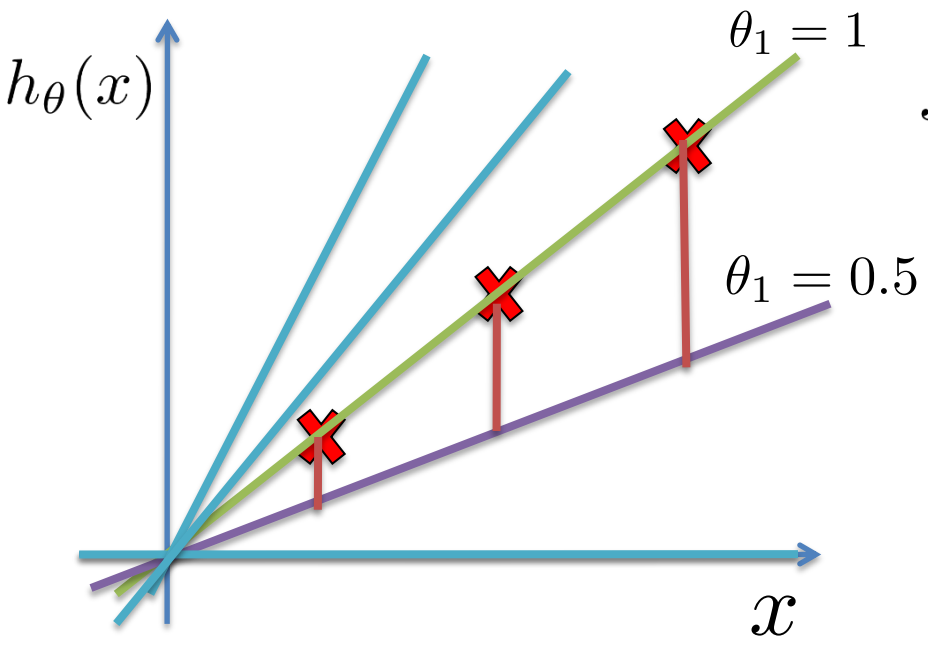
Cost function intuition (2)

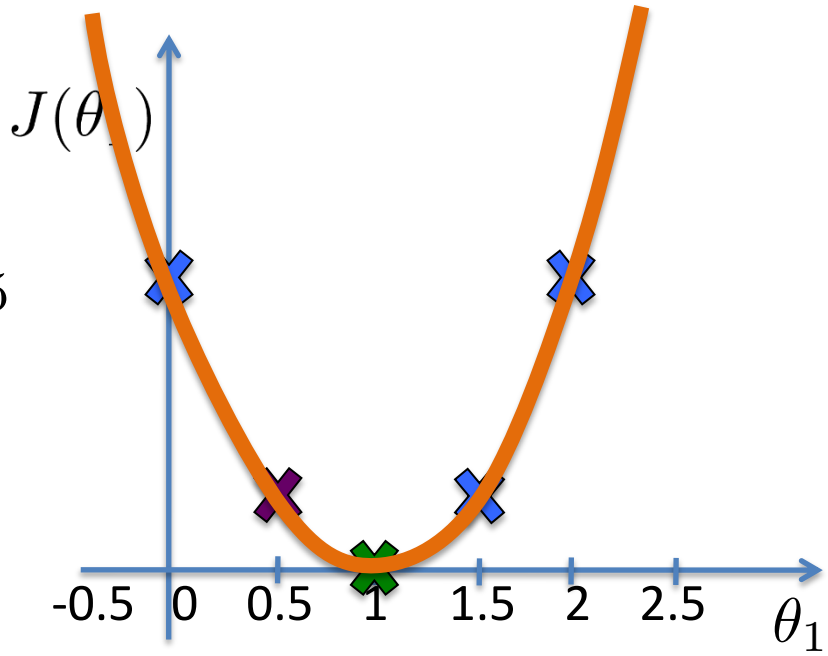
$h_\theta(x)$

for fixed θ, this is a function of x

$J(\theta_1)$

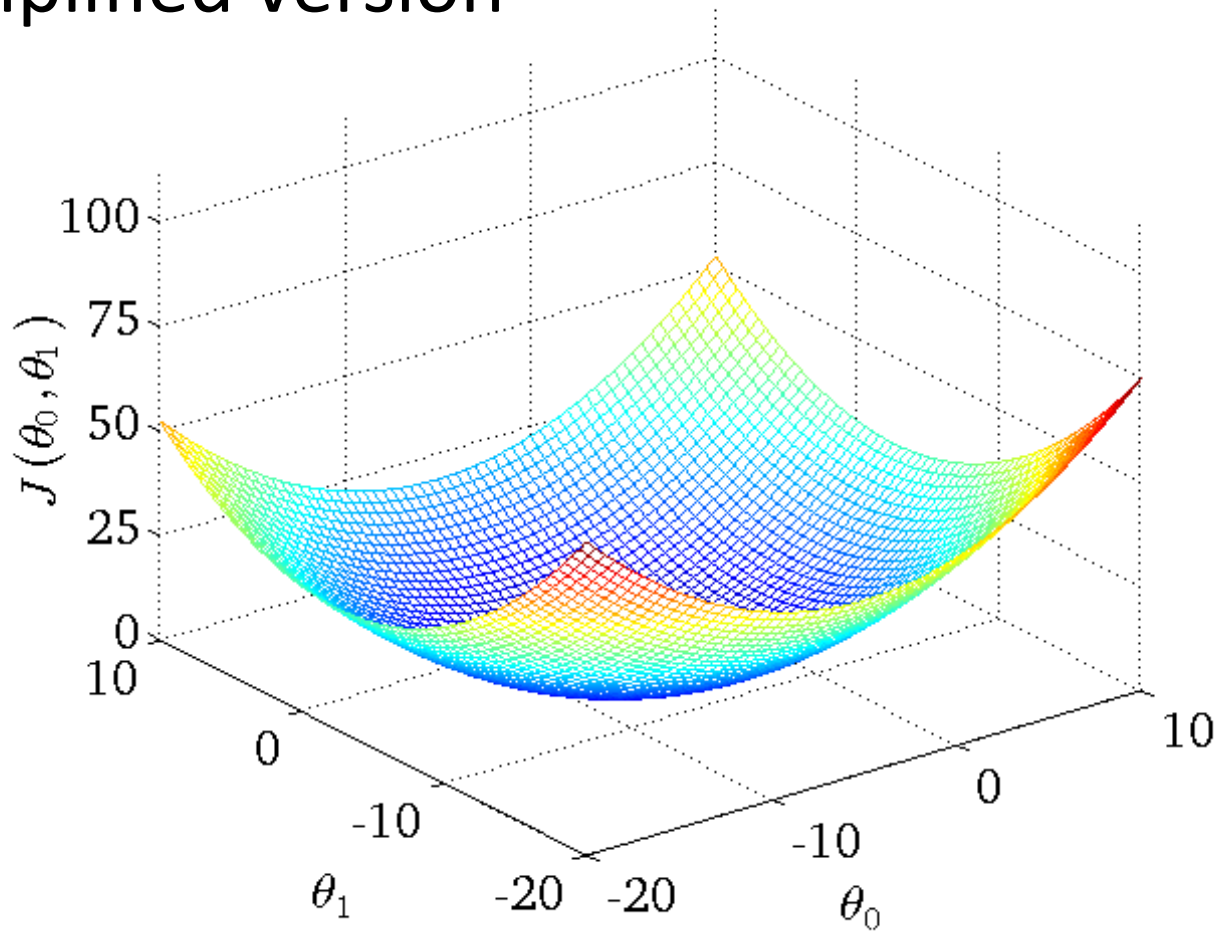
this is a function of θ_1





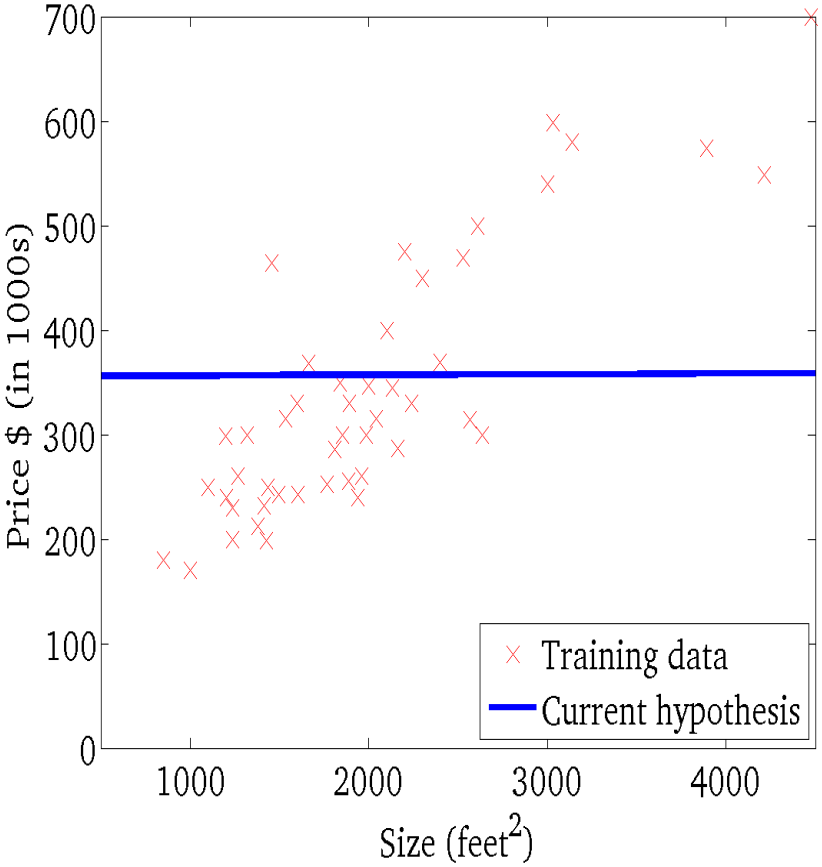
Cost function intuition (3)

Un-simplified version



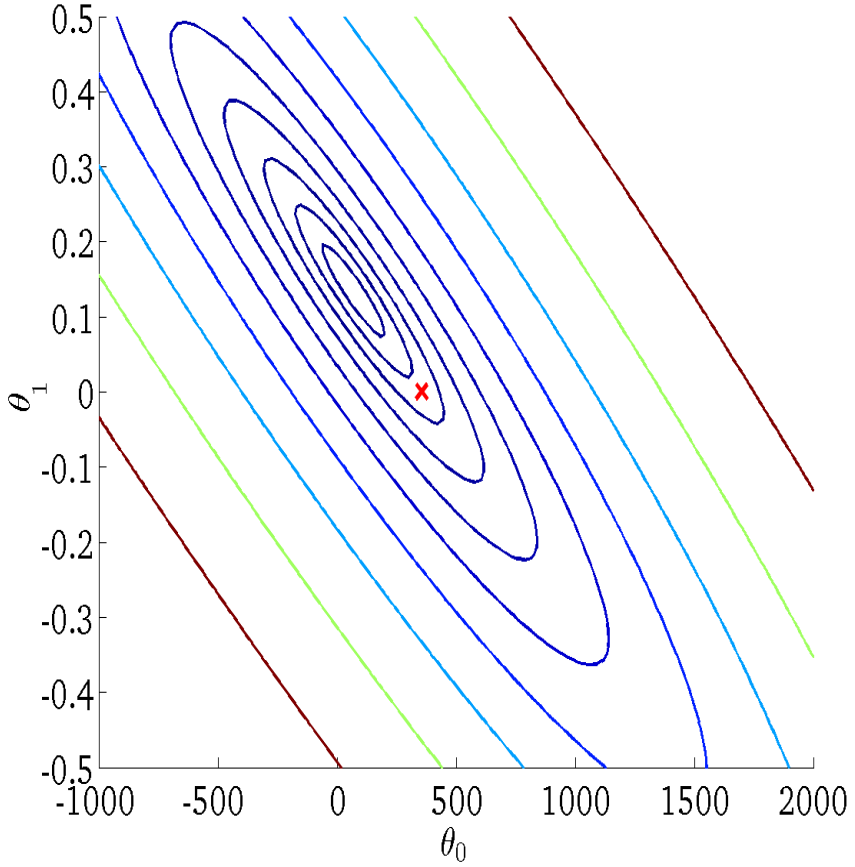
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



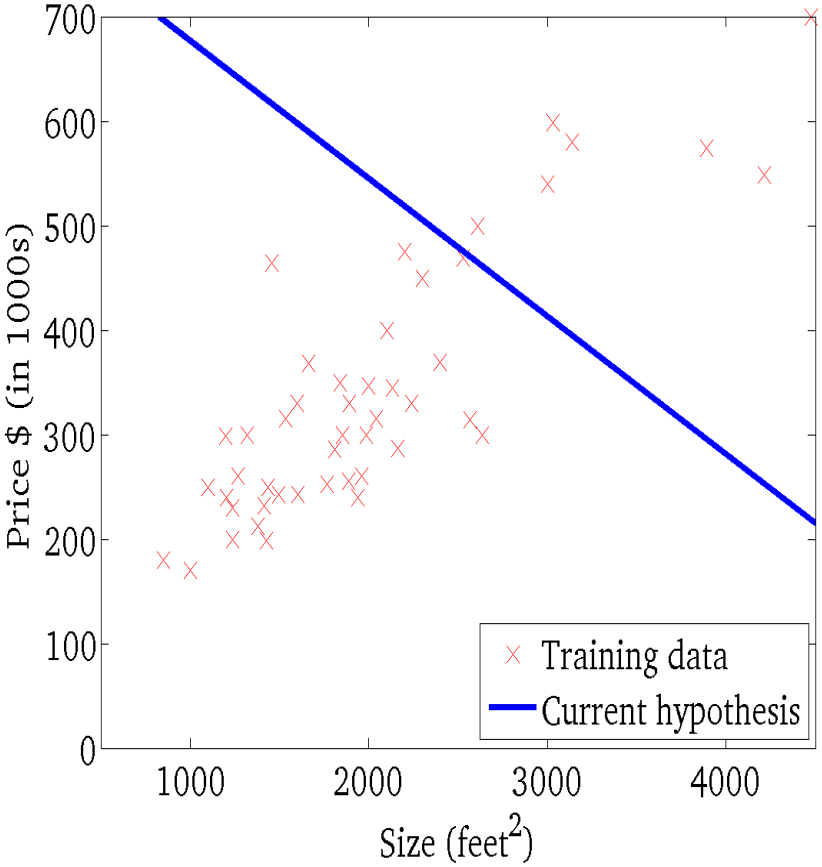
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



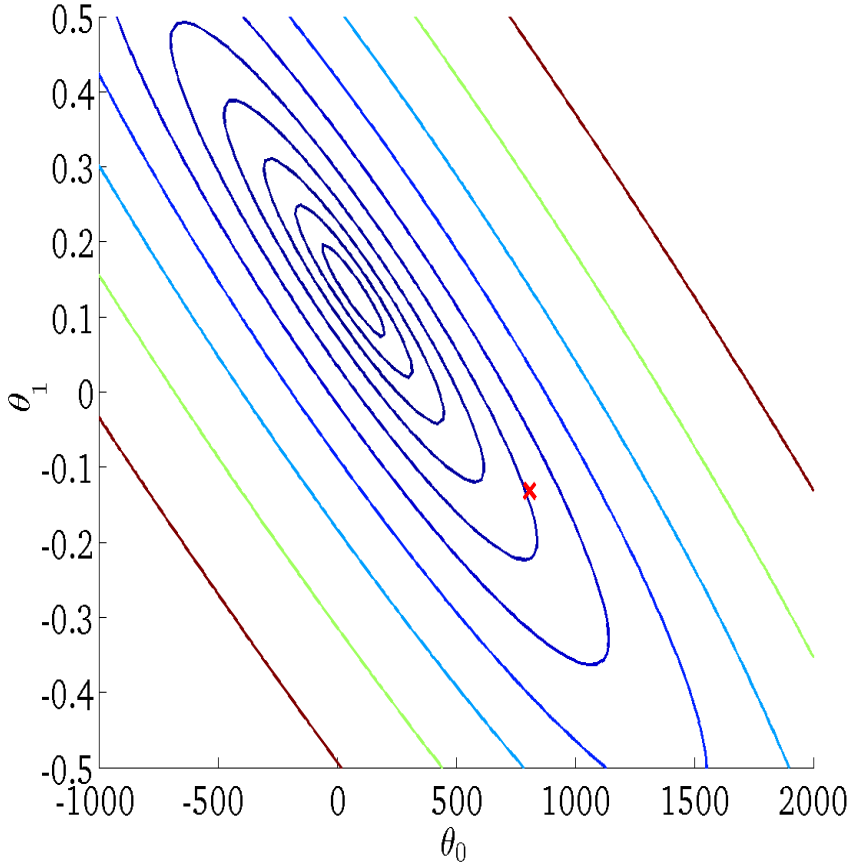
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



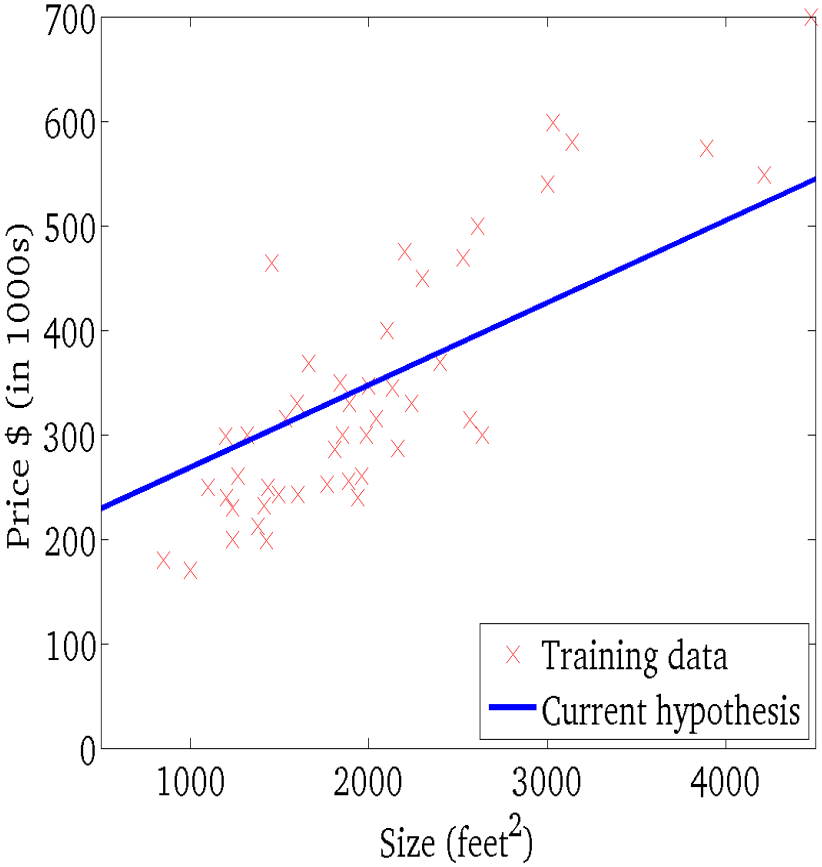
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



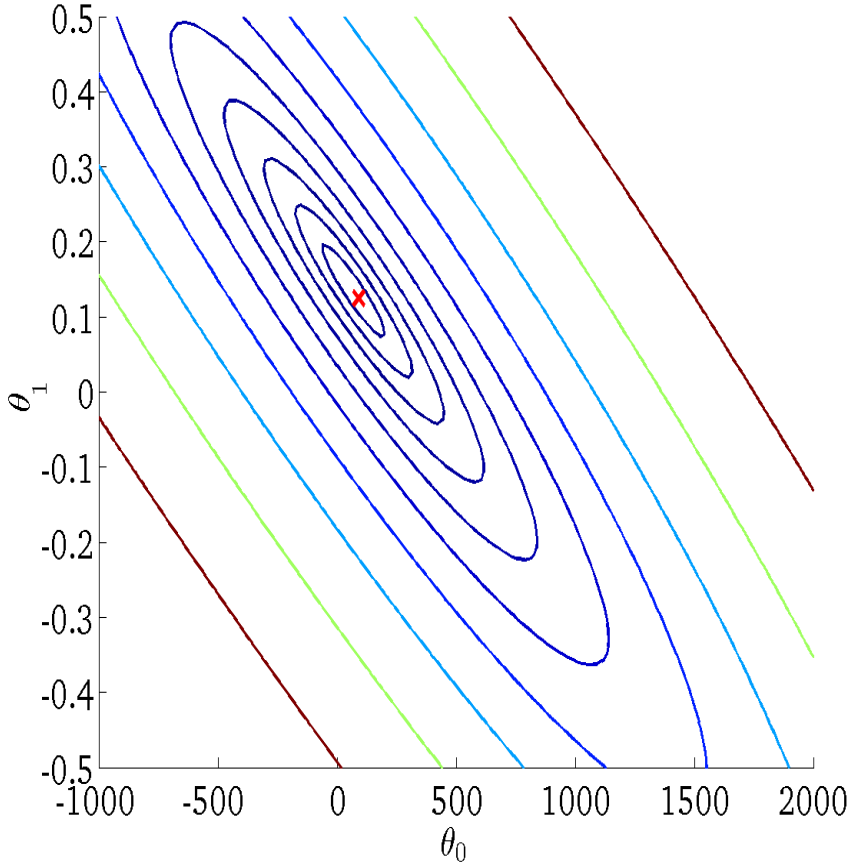
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Normal equation

= method to solve for θ analytically

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \left[\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\right]^T \left[\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\right]$$

$$\nabla_{\boldsymbol{\theta}} \frac{1}{2} \left[\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\right]^T \left[\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\right]$$

$$= \ldots$$

$$= \mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - \mathbf{X}^T\mathbf{y}$$

Setting the gradient to zero:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^T\mathbf{y}$$

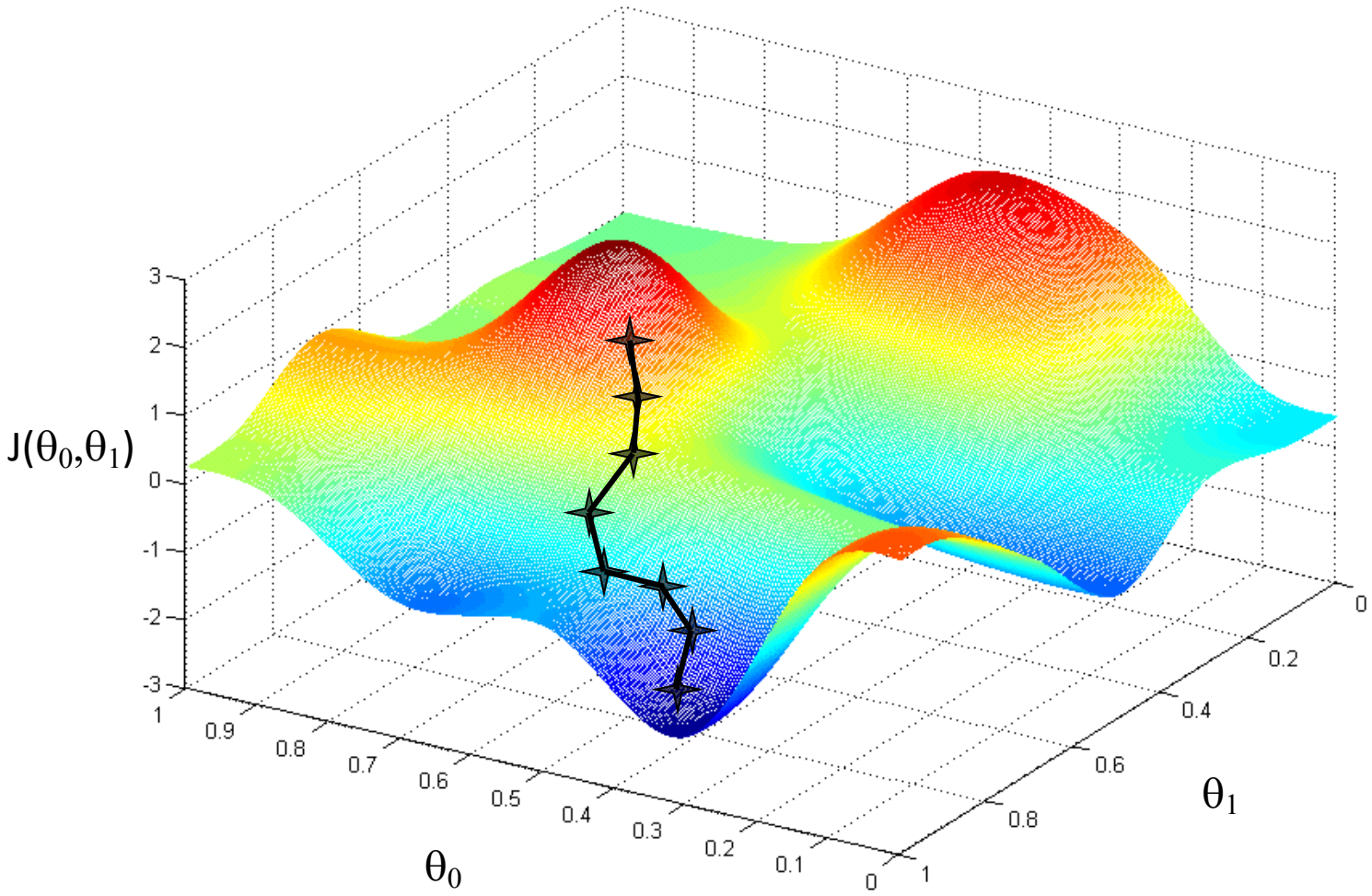
$$\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

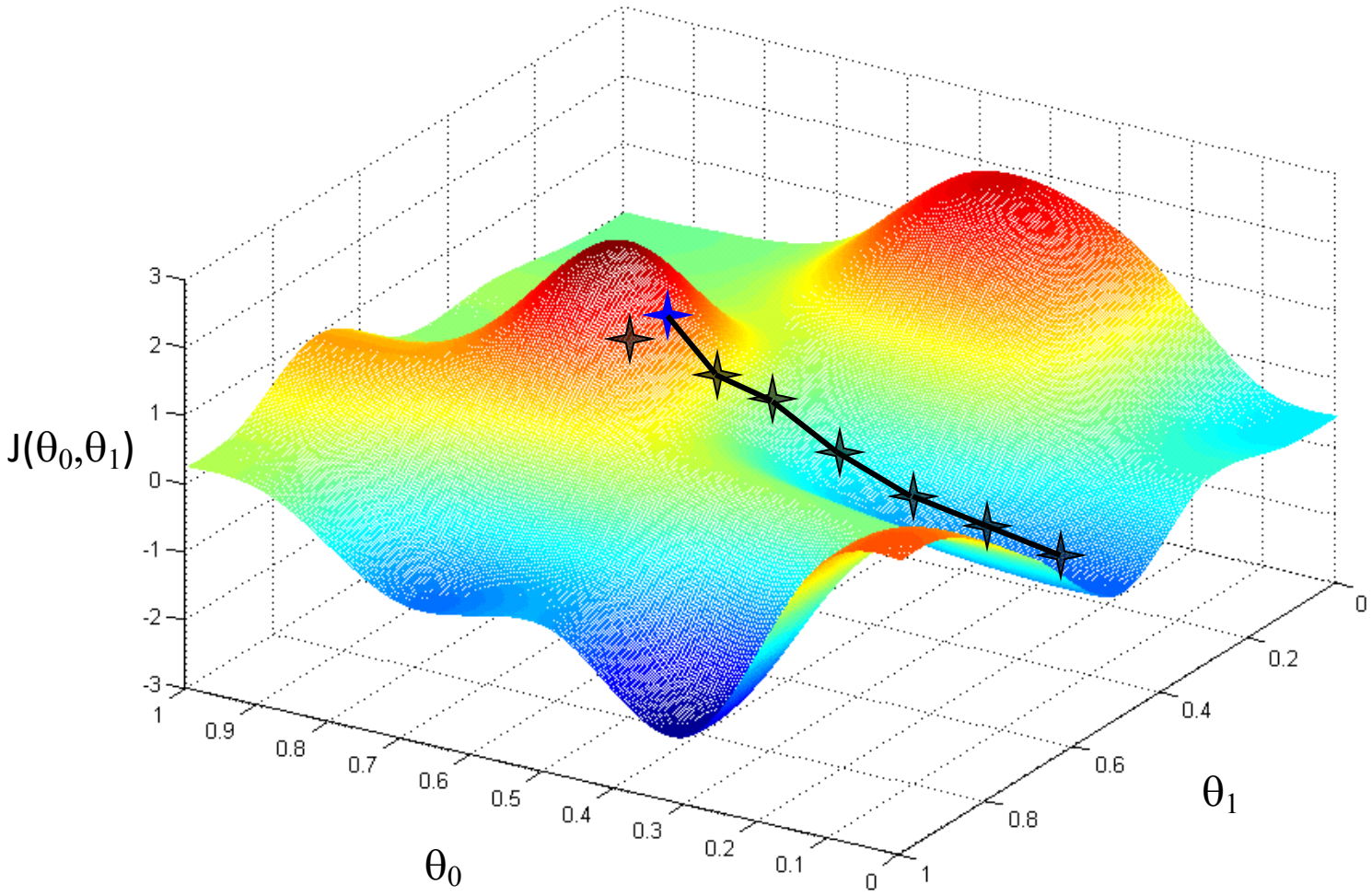
matrix inversion

Gradient Descent

Iterative algorithm for finding min/max of function

1. Start with some θ_0, θ_1
2. Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until you reach a minimum (hopefully)


J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1


J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

learning rate

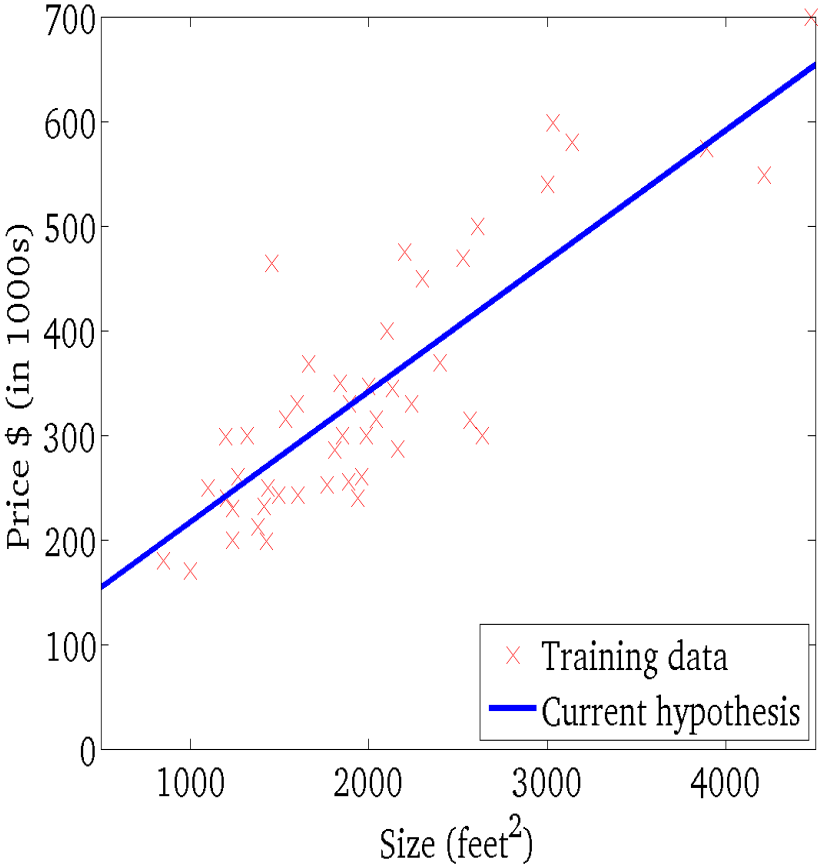
For linear regression: $h_\theta(x) = \theta_0 + \theta_1 x$

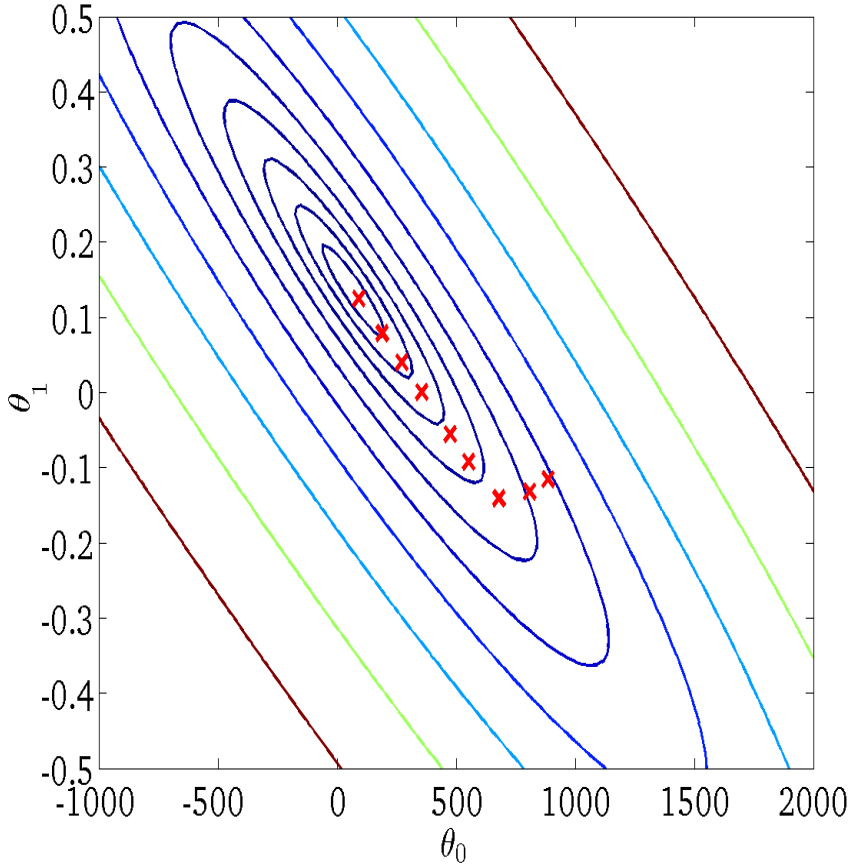
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Important to do simultaneous updates!

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$


Price $ (in 1000s)
Size (feet^2)
Training data
Current hypothesis
0
100
200
300
400
500
600
700
1000
2000
3000
4000


θ_1
θ_0
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1000
-500
0
500
1000
1500
2000

Multiple features

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

with $x_0 = 1$

$$= \theta^T \mathbf{x}$$

Gradient descent for multivariate linear regression

Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

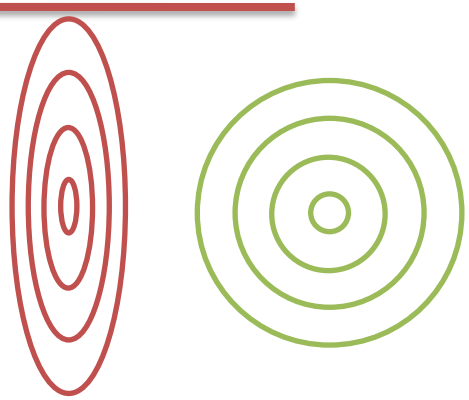
$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$\ldots$

Some issues

Feature scaling

Get all features in the [-1;1] range



Learning rate α

– if α is too small: slow convergence

– if α is too large: gradient steps may overshoot; may not converge

Gradient descent vs Normal Equation

Gradient descent

- need to choose α
- needs many iterations

- works well even when the number of features is large

Normal Equation

- no need for α
- don't need to iterate

- needs to compute $(X^T X)^{-1}$
  - (n+1)*(n+1) matrix (n features)
  - O(n^3)

  - might be non-invertible
    - redundant features -> delete
    - use regularization

Over-fitting

A high number of features means a high number of weights to train and possible over-fitting

1. Reduce number of features
   - Manually select which features to keep
   - Feature/Model selection algorithm
2. Regularization
   - Keep all the features, but reduce magnitude/values of parameters θ_j
   - Works well when we have a lot of features, each of which contributes a bit to predicting y

Regularization

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

extra term

Gradient descent:

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

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

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

Normal equation under regularization

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

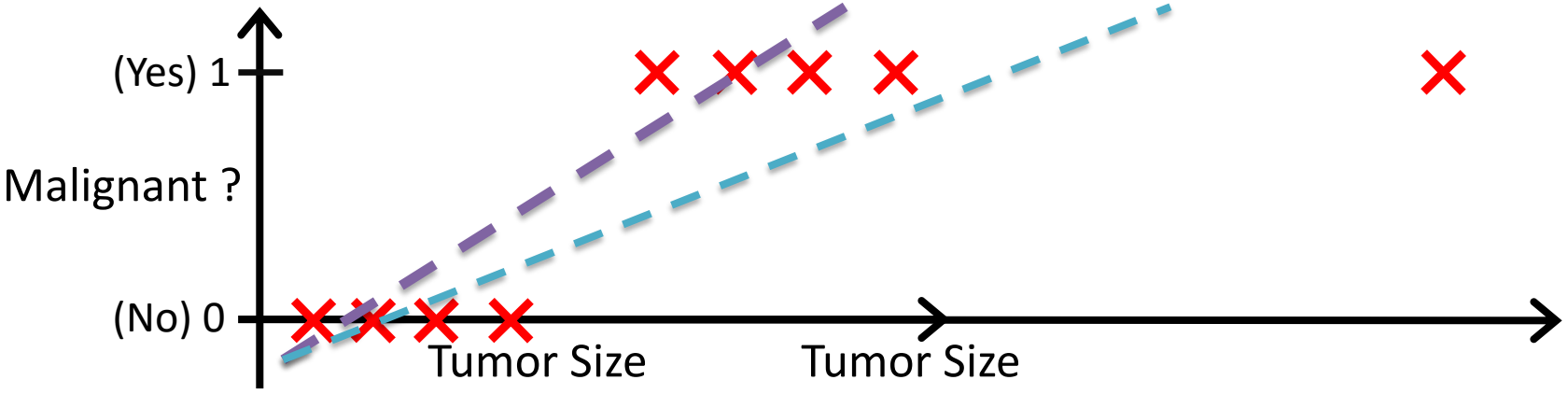
Two advantages:

- fights over-fitting
- guarantees matrix of full rank, and thus invertible

How about classification?

Cast binary classification problem as a regression problem?

- Negative examples get $y = 0$
- Positive examples get $y = 1$
- Predict positive when $h_\theta(x) > 0.5$



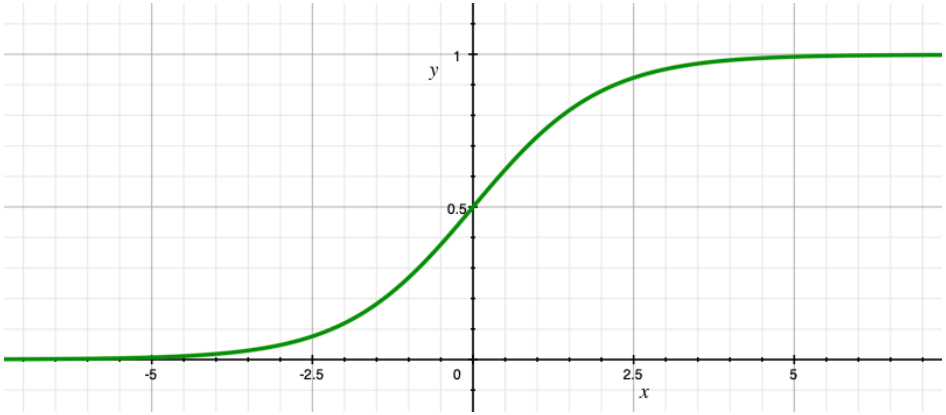
Logistic regression model

$h_\theta(x)$ should only predict values from [0;1]

Probability that y = 1

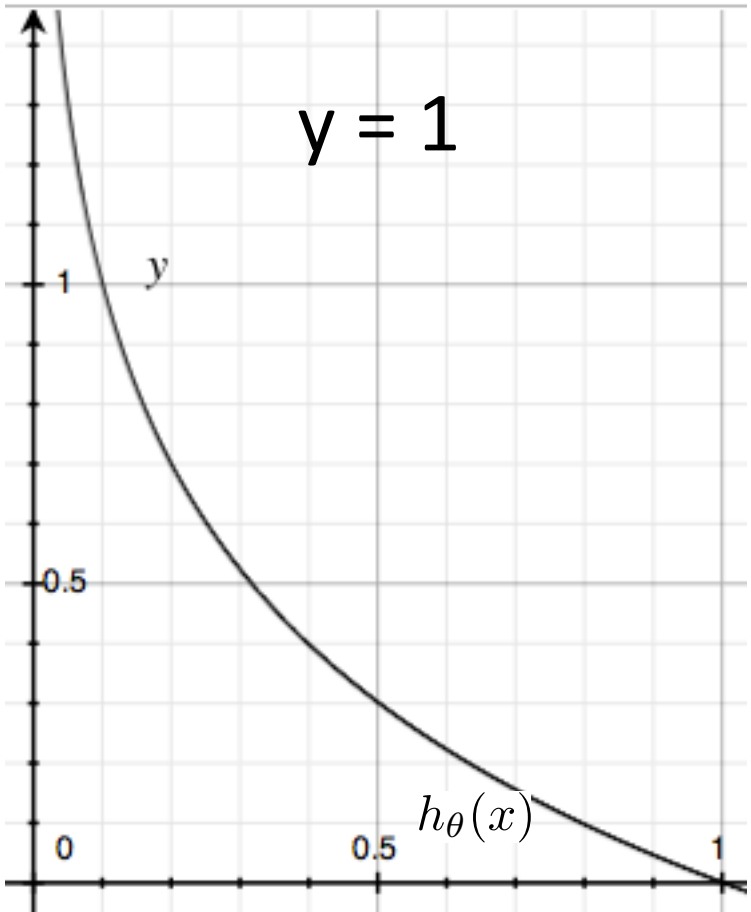
$$h_\theta(\mathbf{x}) = g(\theta^T \mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}} = p(y = 1 | \mathbf{x}; \theta)$$

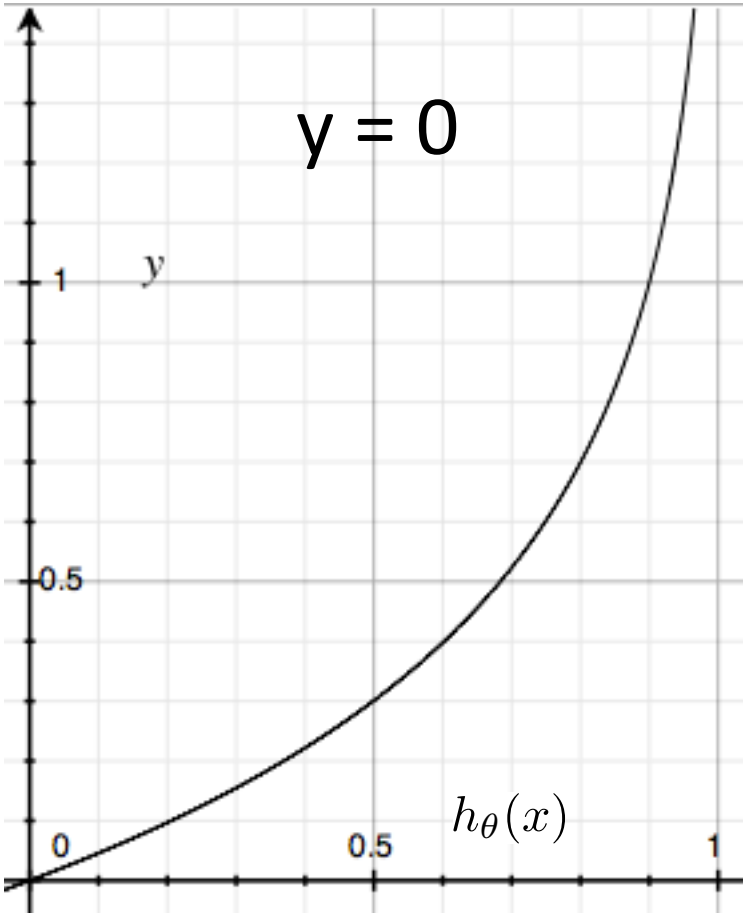
Sigmoid: Logistic function $g(z) = \frac{1}{1 + e^{-z}}$



Cost function

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





Cost function (2)

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

since y is always =0 or =1

How to find the right θ? $\min_\theta J(\theta)$

Gradient descent:

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

Gradient descent for logistic regression

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

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

Looks identical to linear regression!!

but with $h_\theta(x) = \dfrac{1}{1 + e^{-\theta^T x}}$

… with regularization

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

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

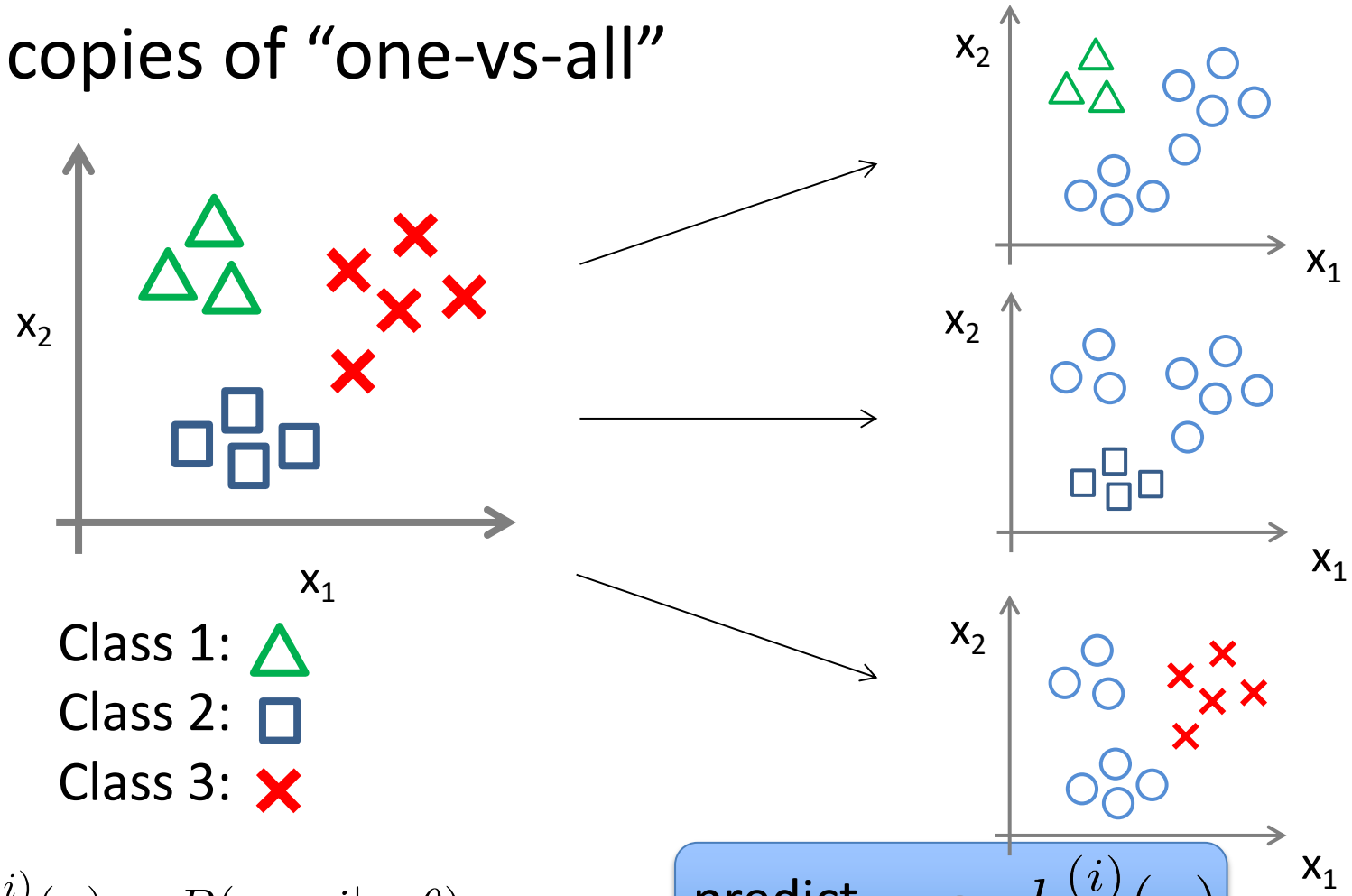
$$\frac{\partial}{\partial \theta_1} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} + \frac{\lambda}{m} \theta_1$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)} + \frac{\lambda}{m} \theta_2$$

…

How about multi-class problems?

k copies of “one-vs-all”



$h_\theta^{(i)}(x) = P(y = i|x; \theta)$ → predict $\max_i h_\theta^{(i)}(x)$

In one go: Softmax Regression

Turn $y^{(i)}$ into

$$\mathbf{y}^{(i)} = \begin{bmatrix} p(y^{(i)} = 1 | x^{(i)}, \theta) \\ p(y^{(i)} = 2 | x^{(i)}, \theta) \\ \vdots \\ p(y^{(i)} = k | x^{(i)}, \theta) \end{bmatrix} \qquad \theta = \begin{bmatrix} — \theta_1^T — \\ — \theta_2^T — \\ \vdots \\ — \theta_k^T — \end{bmatrix}$$

and $h(x^{(i)})$ into

$$\mathbf{h}_\theta(x^{(i)}) = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

Cost Function

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^{(i)} = j\} log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right]$$

Indicator function: 1 if {true}, 0 if {false}

Similar to logistic regression cost function.

No analytical solution ...

Gradients

Vector

For each θ_j:

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = j\} - p(y^{(i)} = j | x^{(i)}; \theta) \right) \right]$$

Vector

However ...

Over-parameterized

Imagine subtracting a constant ψ from each θ_j

$$p(y^{(i)}|x^{(i)},\theta) = \frac{e^{(\theta_j-\psi)^T x^{(i)}}}{\sum_{l=1}^{k} e^{(\theta_l-\psi)^T x^{(i)}}}$$

$$= \frac{e^{\theta_j^T x^{(i)}} e^{-\psi^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}} e^{-\psi^T x^{(i)}}}$$

$$= \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}$$

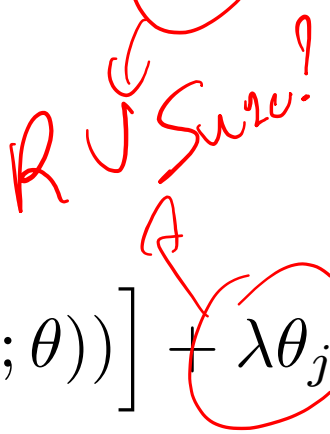
Weight Decay – Just like regularisation

New cost-function:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^{(i)} = j\} log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2$$

New gradient:

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[x^{(i)}(1\{y^{(i)} = j\} - p(y^{(i)} = j|x^{(i)};\theta))\right] + \lambda\theta_j$$



Summary

- Linear and Logistic regression
  - gradient descent
  - normal equations
  - regularization

- Softmax regression vs k-Binary classifiers

Recommender Systems – Collaborative Filtering

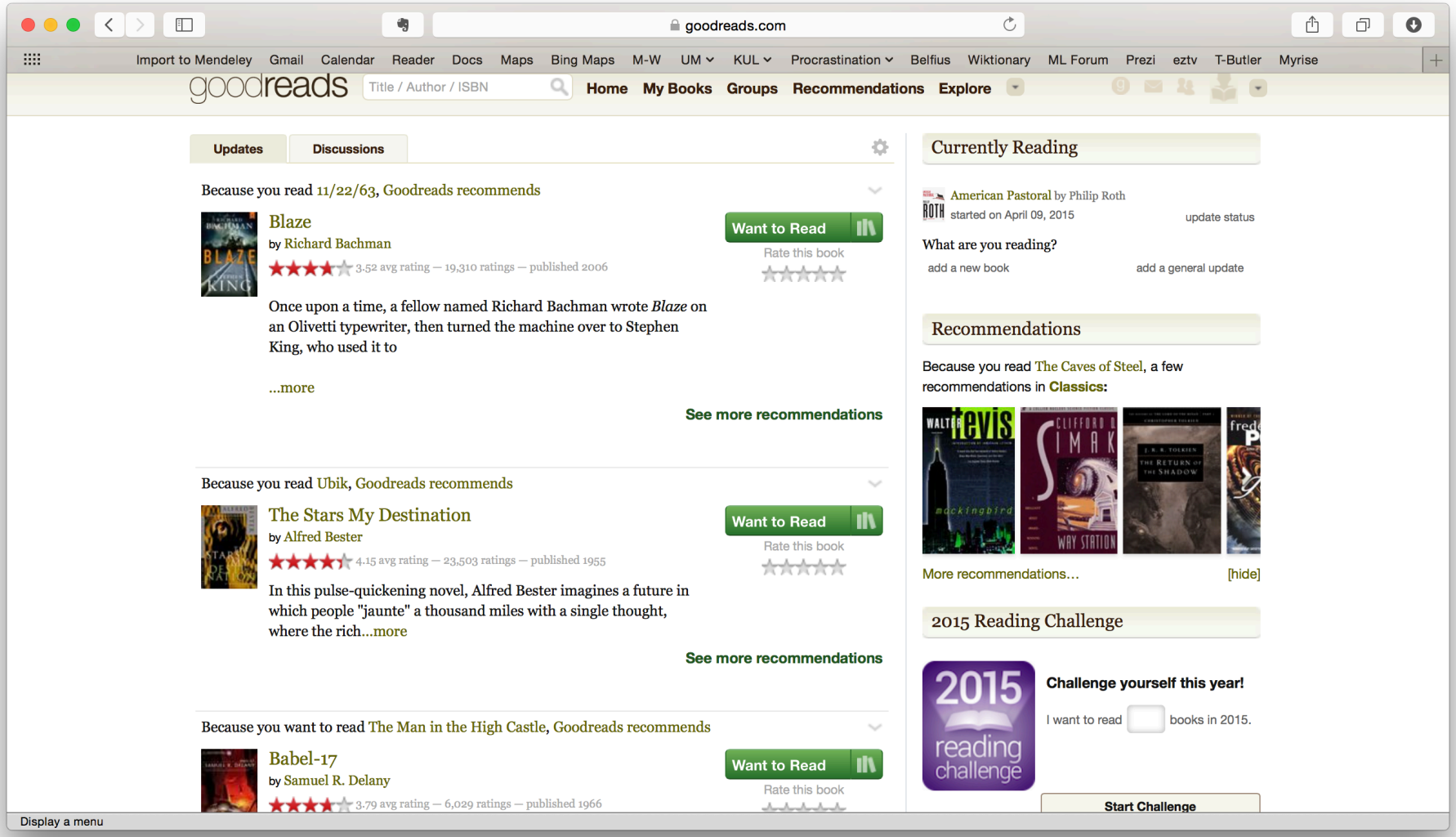
Kurt Driessens

with slides from Andrew Ng, Hendrik Blockeel, Haitham Bou Ammar

Overview

- Recommender Systems
  - ~~Using nearest neighbor~~
  - Using collaborative filtering

Recommender Systems



Recommender systems

Problem:

Predict whether someone will like a Web page, posting, movie, book, CD, etc.

- Old approach:
  - Look at content, topic, tags, …
- Collaborative filtering:
  - Look at what similar users bought/liked
  - Similar users = similar purchases/likes

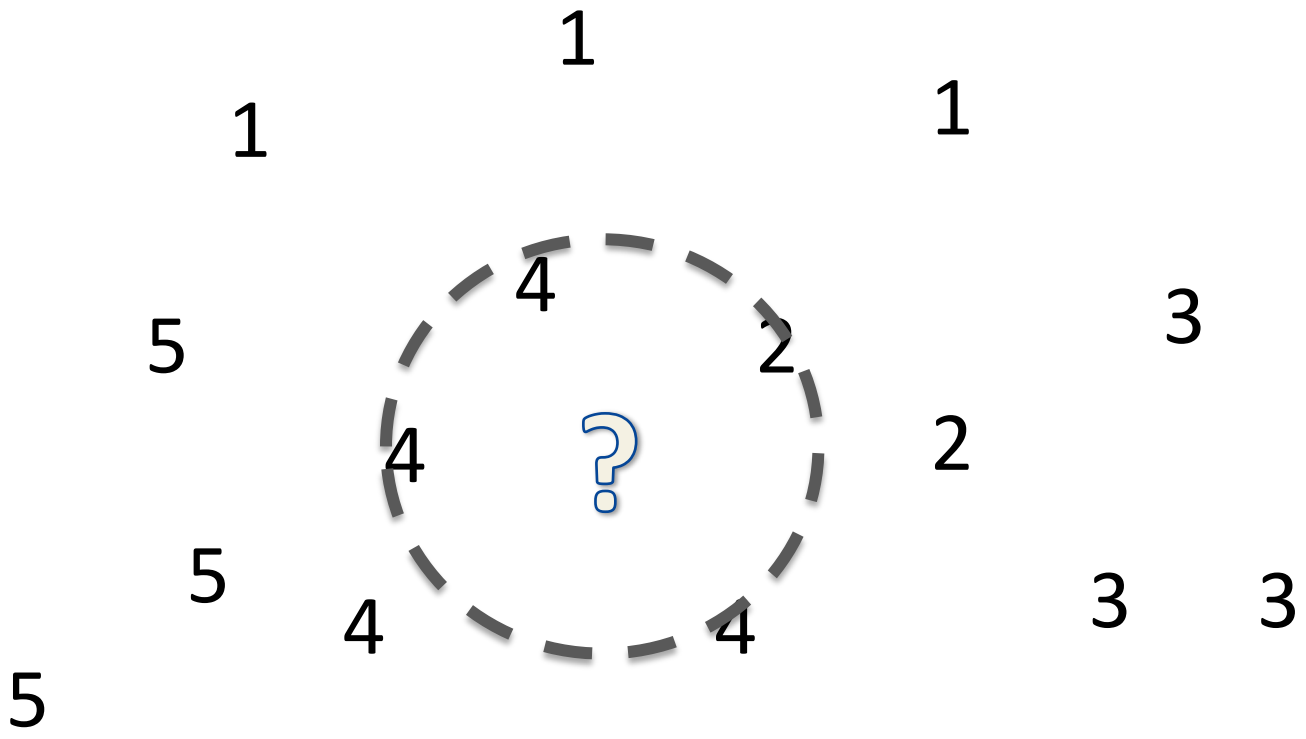
The setting

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

Predict ratings for movies users have not yet seen (or rated).

k-Nearest-Neighbor Regression

Use a distance-weighted mean



kNN Regression

function kNN(k, T, $\mathbf{x}$) **returns** real:

let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_k, y_k)\}$ be the set of the k nearest neighbors of $\mathbf{x}$ in T

$$\hat{y} := \frac{1}{k} \sum_{(\mathbf{x}_i, y_i) \in S} y_i \quad \textbf{or} \quad \hat{y} := \frac{1}{\sum_{j=1}^{k} \frac{1}{dist(\mathbf{x}, \mathbf{x}_j)}} \sum_{(\mathbf{x}_i, y_i) \in S} \frac{1}{dist(\mathbf{x}, \mathbf{x}_i)} y_i$$

if target attribute is discrete **then** round $\hat{y}$ to nearest target value

return $\hat{y}$

Recommender Systems through Weighted Nearest Neighbor

Predict through instance based regression:

$$\hat{R}_{ik} = \bar{R}_i + \alpha \sum_{X_j \in N_i} W_{ij}(R_{jk} - \bar{R}_j)$$

Avg. rating of user i

rating by user j of entry k

can be all entries or kNN

$$W_{ij} = \frac{\sum_k (R_{ik} - \bar{R}_i)(R_{jk} - \bar{R}_j)}{\sqrt{\sum_k (R_{ik} - \bar{R}_i)^2} \sqrt{\sum_k (R_{jk} - \bar{R}_j)^2}}$$

Pearson coefficient

$$\alpha = (\sum |W_{ij}|)^{-1}$$

Content based approach

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

$r(i,j) = 1$ if user j has rated movie i (0 otherwise)
$y^{(i,j)} =$ rating by user j on movie i (if defined)

$\theta^{(j)}$ = parameter vector for user j
$x^{(i)}$ = feature vector for movie i
For user j , movie i , predicted rating: $(\theta^{(j)})^T(x^{(i)})$

$m^{(j)}$ = no. of movies rated by user j

The optimization criterion

To learn $\theta^{(j)}$ (parameter for user j):

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

To learn $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)},\ldots,\theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

Movie features

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

Assume the users indicate their preferences:

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$$

Learn movie-feature values

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(1)}, \ldots, x^{(n_m)}$:

$$\min_{x^{(1)}, \ldots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

So ...

Given the movie ratings

and $x^{(1)}, \ldots, x^{(n_m)}$, we can learn $\theta^{(1)}, \ldots, \theta^{(n_u)}$

and $\theta^{(1)}, \ldots, \theta^{(n_u)}$, we can learn $x^{(1)}, \ldots, x^{(n_m)}$

Start with a guess:

$$\theta \rightarrow x \rightarrow \theta \rightarrow x \rightarrow \theta \rightarrow \ldots$$

Collaborative Filtering

Given $x^{(1)}, \ldots, x^{(n_m)}$, estimate $\theta^{(1)}, \ldots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)},\ldots,\theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, estimate $x^{(1)}, \ldots, x^{(n_m)}$:

$$\min_{x^{(1)},\ldots,x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Estimating $x^{(1)}, \ldots, x^{(n_m)}$ and $\theta^{(1)}, \ldots, \theta^{(n_u)}$ simultaneously:

$$J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$$\min_{\substack{x^{(1)},\ldots,x^{(n_m)} \\ \theta^{(1)},\ldots,\theta^{(n_u)}}} J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$$

Collaborative Filtering Algorithm

1. Initialize $x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ to small random values.
2. Minimize $J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$ using gradient descent (or another optimization algorithm).
   E.g. for every $j = 1, \ldots, n_u, i = 1, \ldots, n_m$:

$$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

3. For a user with (learned) parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

Brand new users

Users who have not rated any movies

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

$$\min_{\substack{x^{(1)},\ldots,x^{(n_m)} \\ \theta^{(1)},\ldots,\theta^{(n_u)}}} \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad (\theta^{(5)})^T (x^{(i)}) = 0$$

Mean normalization

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \qquad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

Now learn θ and x

For user j, on movie i predict:

$$(\theta^{(j)})^T(x^{(i)}) + \mu_i$$

User 5 (Eve):

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$(\theta^{(5)})^T(x^{(1)}) + \mu_1 = 2.5$$

Learning a representation

Making predictions is cool, but …

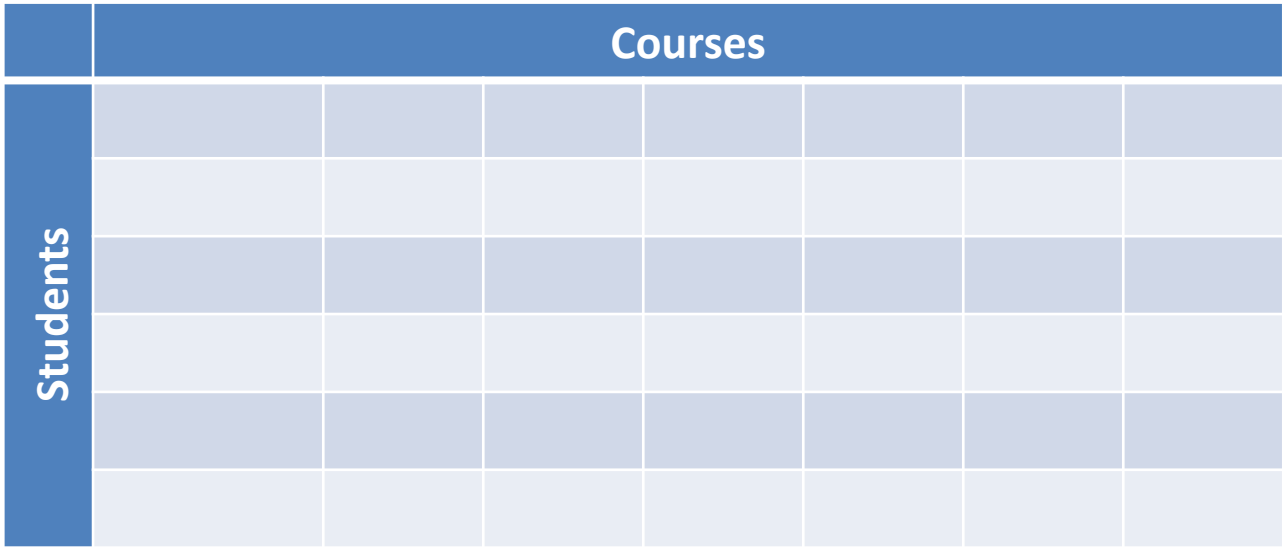
Learning a representation $\boldsymbol{x}$ for movies/books/… and a representation $\boldsymbol{\theta}$ for users

… from only preferences is much much cooler!

Example application

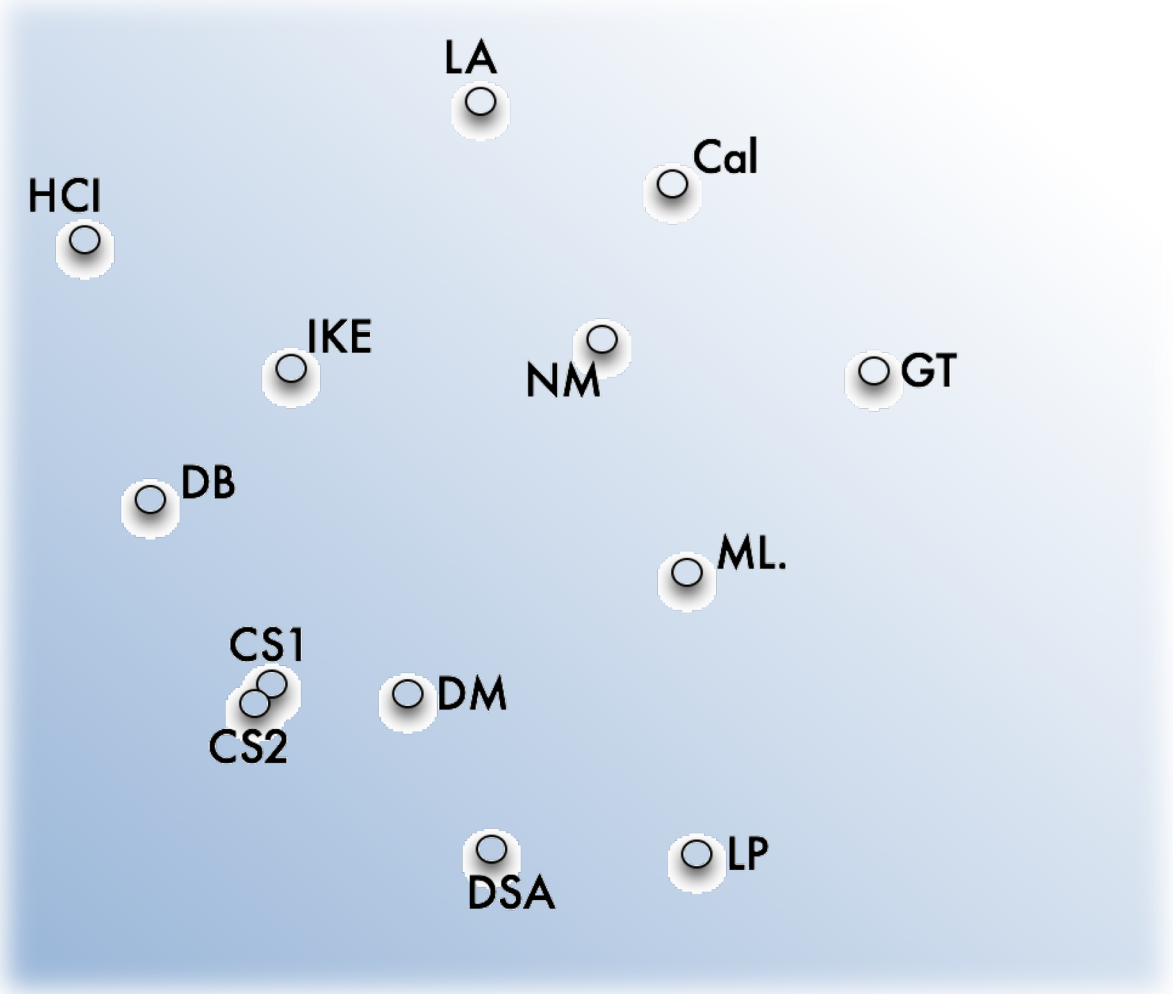
Master project:

From:



Learn: a representation for courses and students

Compare courses based on discovered representation



Summary

- Recommender Systems
  - nearest neighbor approach
  - as application/extention of linear regression

Artificial Neural Networks

Kurt Driessens

with slides from Andrew Ng, Hendrik Blockeel, Haitham Bou Ammar

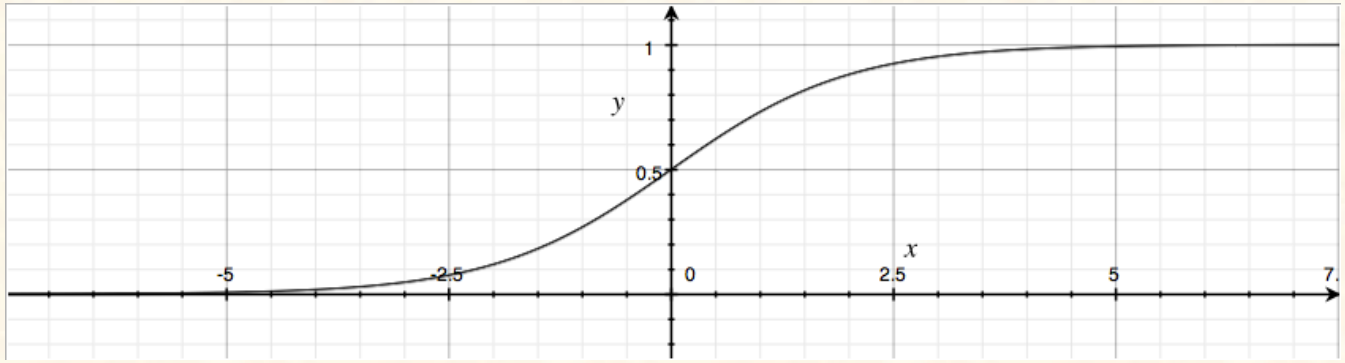
Logistic regression model

$h_\theta(x)$ should only predict values from [0;1]

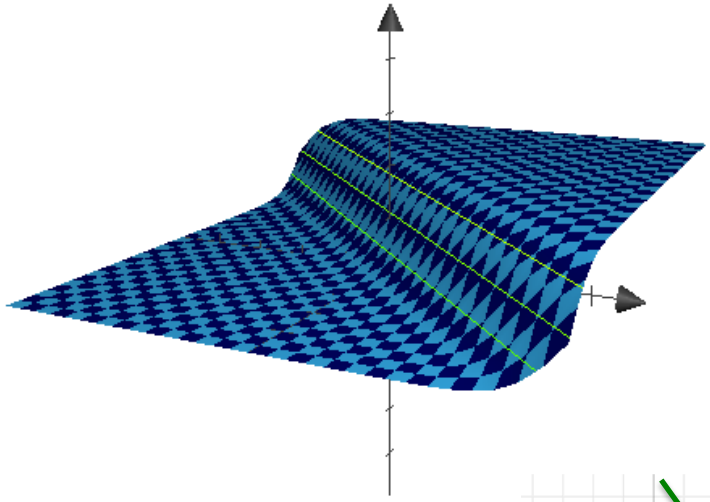
Probability that y = 1

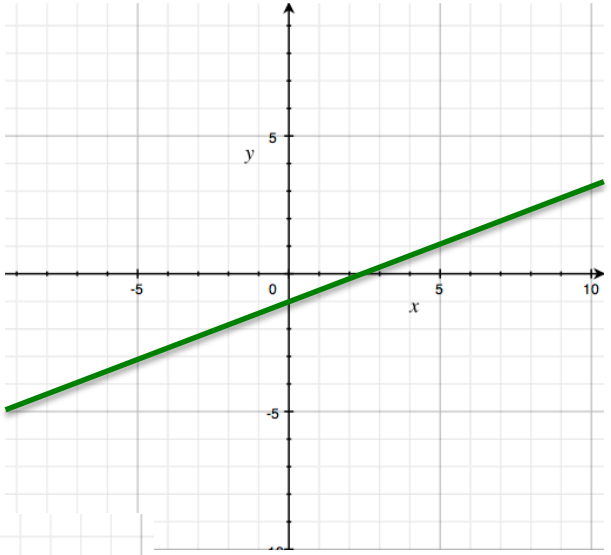
$$h_\theta(\mathbf{x}) = g(\theta^T \mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}} = p(y = 1 | \mathbf{x}; \theta)$$

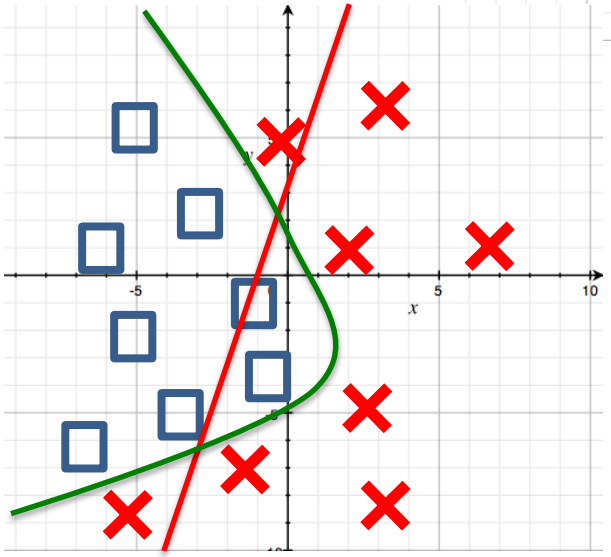
Sigmoid: Logistic function $g(z) = \frac{1}{1 + e^{-z}}$



What about non-linear problems?

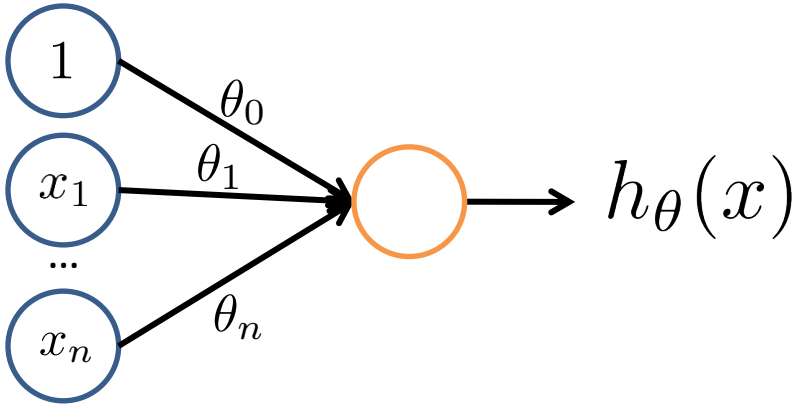




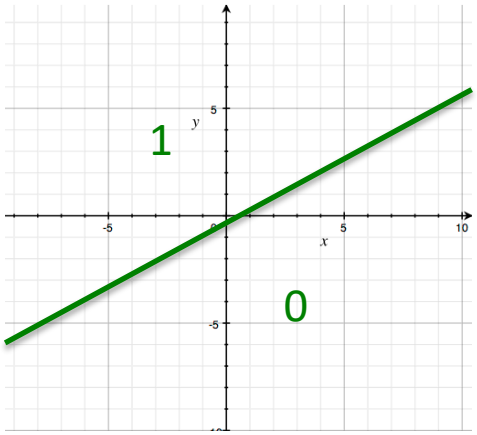


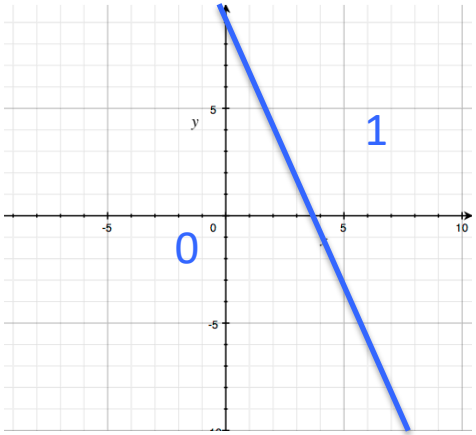
Starting from logistic regression…

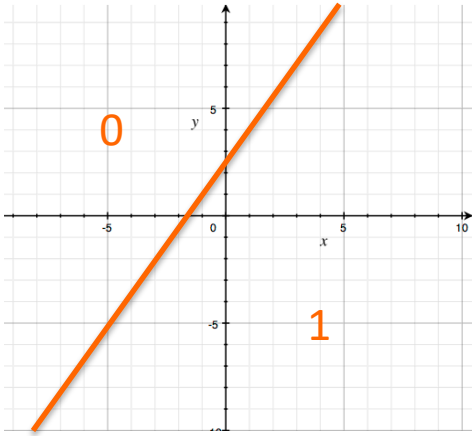
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



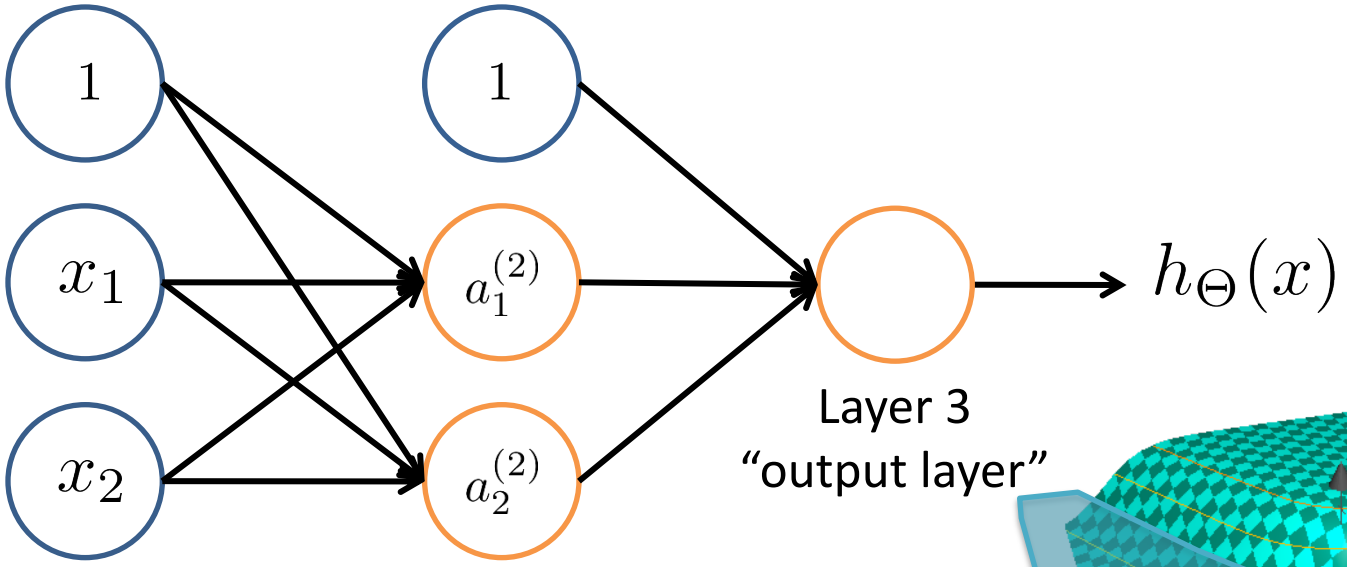
Assigns 1 to half-plane (half-space) and 0 to other







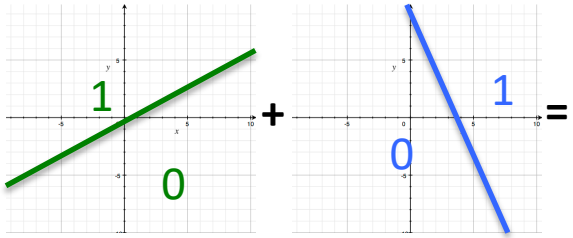
...to Neural Networks

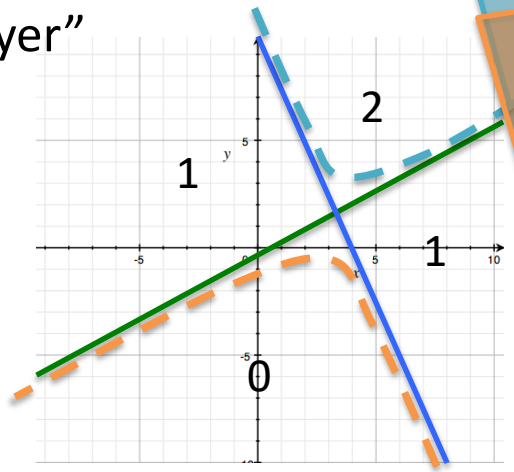


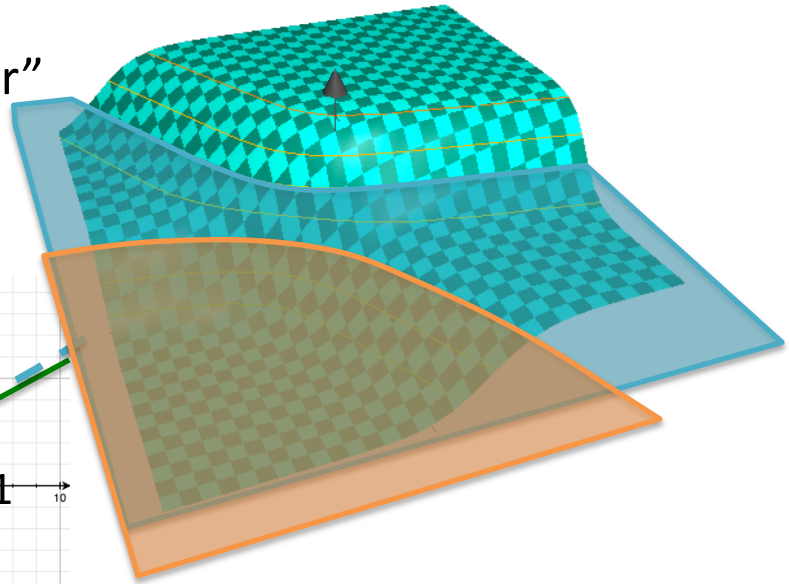
Layer 1
"input layer"

Layer 2
"hidden layer"

Layer 3
"output layer"





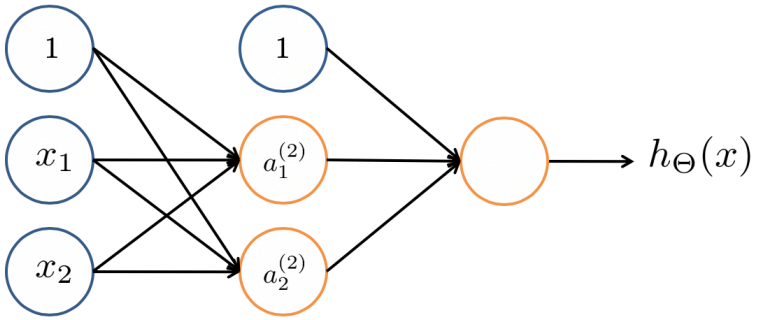


Forward propagation

$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer j+1

- dimension $s_{j+1} \times (s_j + 1)$ where s_j is the number of nodes on layer j



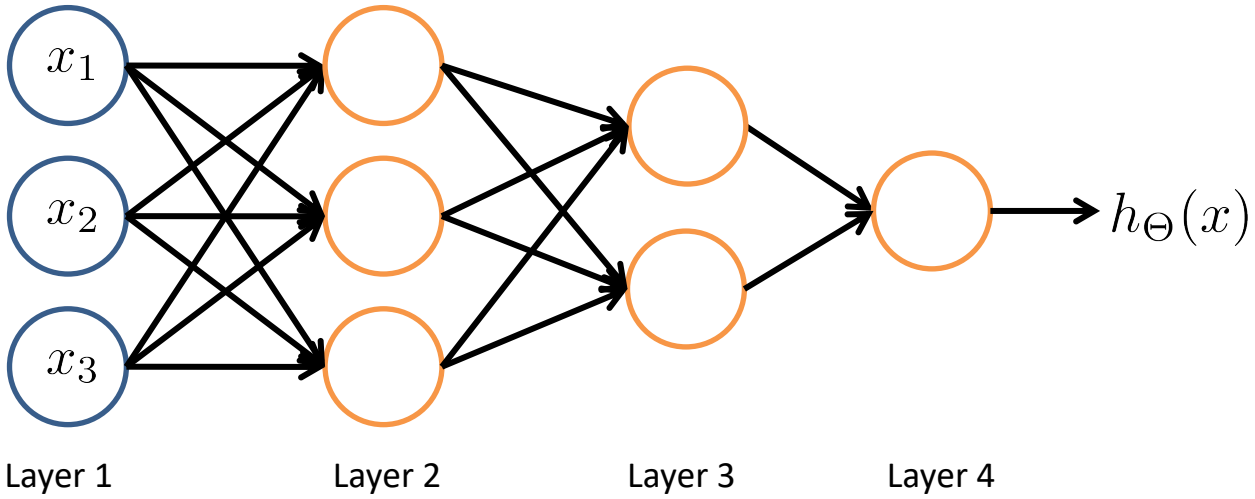
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2) = z_1^{(2)}$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)})$$

Other network architectures

Multiple hidden layers

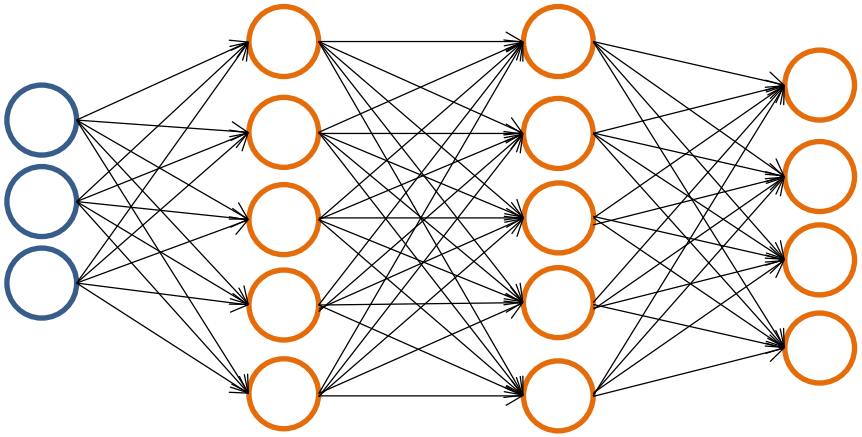


Recurrent networks

– gives memory effect (e.g. counting, adding, …)

Other network architectures (2)

Multiclass problems



$h_\Theta(x) \in \mathbb{R}^4$

E.g. 4 classes: 

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

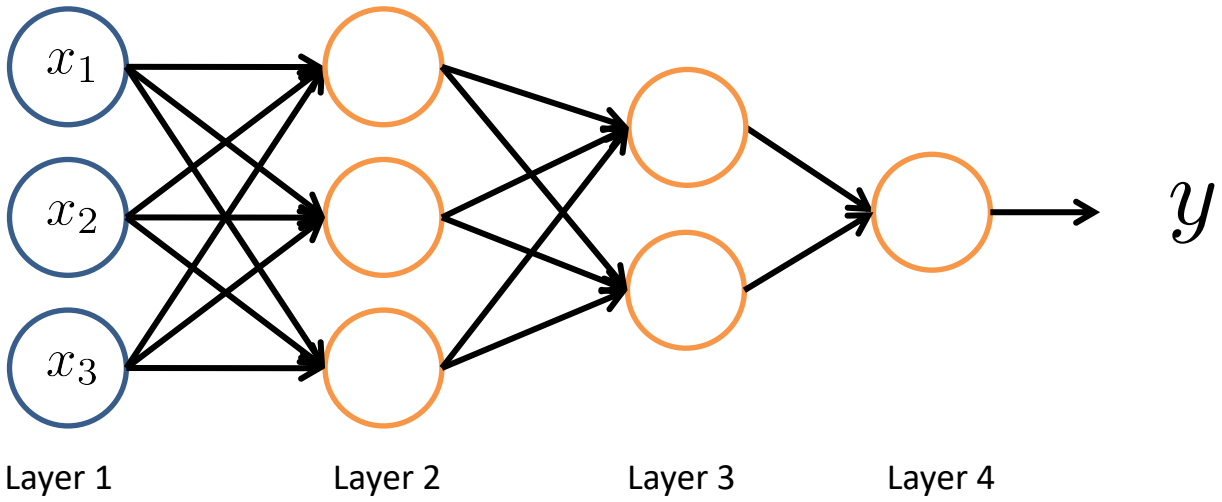
Learning the weights

Backpropagation

- gradient descent similar to lin. & log. regression
- where to get errors for internal nodes?

Given that:

$$\frac{d}{dx}g(x) = g(x)(1 - g(x))$$



Gradient descent for logistic regression

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

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

Looks identical to linear regression!!

but with $h_\theta(x) = \dfrac{1}{1 + e^{-\theta^T x}}$

Learning the weights

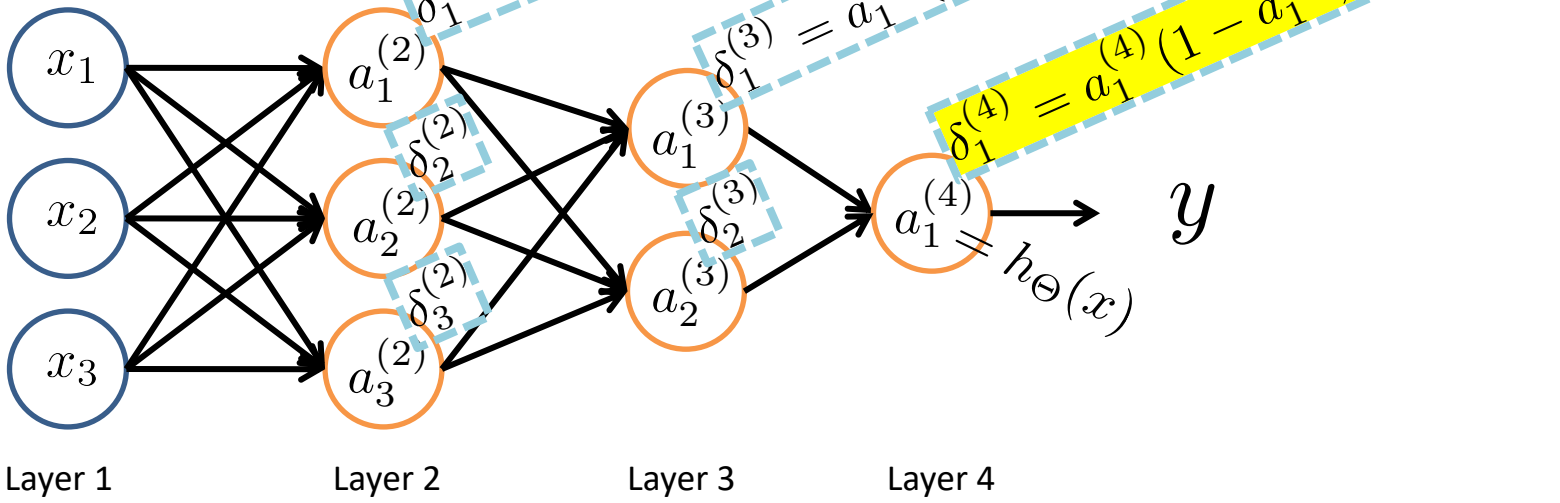
Backpropagation

– gradient descent similar to lin. & log. regression

– where to get errors for internal nodes

Given that:

$$\frac{d}{dx}g(x) = g(x)(1 - g(x))$$



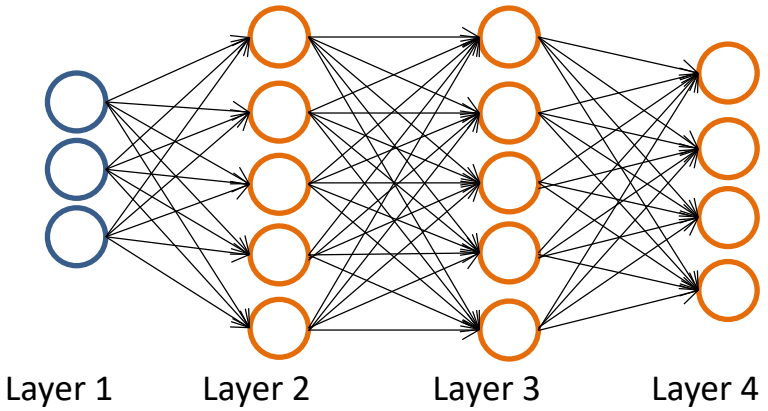
Learning the weights (2)

Computation can be vectorised

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l.

For each output unit (layer L = 4)

$$\delta^{(4)} = (a^{(4)} - y) .* g'(z^{(4)})$$



Vectorised

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

$$= a^{(2)} .* (1 - a^{(2)})$$

Learning the weights (3)

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)}(1 - a^{(L)})(a^{(L)} - y^{(i)})$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$D_{ij}^{(l)} := \frac{1}{m}[\triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}]$ if $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)}$ if $j = 0$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Repeat {

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

(simultaneously update all θ_j)

}

Properties of neural networks

- Useful for modeling complex, non-linear functions of numerical inputs & outputs
  - symbolic inputs/outputs represented using some encoding, cf. true/false = 1/-1
  - 2 or 3 layer networks can approximate a huge class of functions (if enough neurons in hidden layers)
- Robust to noise

  Risk of over-fitting! (because of high expressiveness)

  E.g. when training too long
  - usually handled using e.g. validation sets

- All inputs have some effect
  - cf. decision trees: selection of most important attributes
  - ANN “selects” attributes by giving them higher/lower weights
- Explanatory power of ANNs is limited
  - model represented as weights in network
  - no simple explanation why network makes a certain prediction
    - contrast with e.g. trees: can give a “rule” that was used
  - Networks can not easily be translated into symbolic model (tree, ruleset)

Hence, ANNs are good when

- high-dimensional input and output (numeric or symbolic)
- interpretability of model unimportant

Examples:

- typical: image recognition, speech recognition, …
  - e.g. images: one input per pixel
  - see http://www.cs.cmu.edu/~tom/faces.html for illustration
- less typical: symbolic problems
  - cases where e.g. trees would work too
  - performance of networks and trees then often comparable

Summary

- Neural networks:
  - what they are, how they work
  - backpropagation
  - representation power
  - explanatory power

Next week preparation

1. Homework: implement your own NN

2. Watch a video lecture:

   Andrew Ng on (Sparse) Autoencoders

   https://www.youtube.com/watch?v=wqhZaWR-J94

Come up with some discussion points!

If you like Andrew, this is also interesting (and a lot easier to watch ..)

   https://www.youtube.com/watch?v=n1ViNeWhC24