stanford university machine learning

what is ⚙ machine learning?

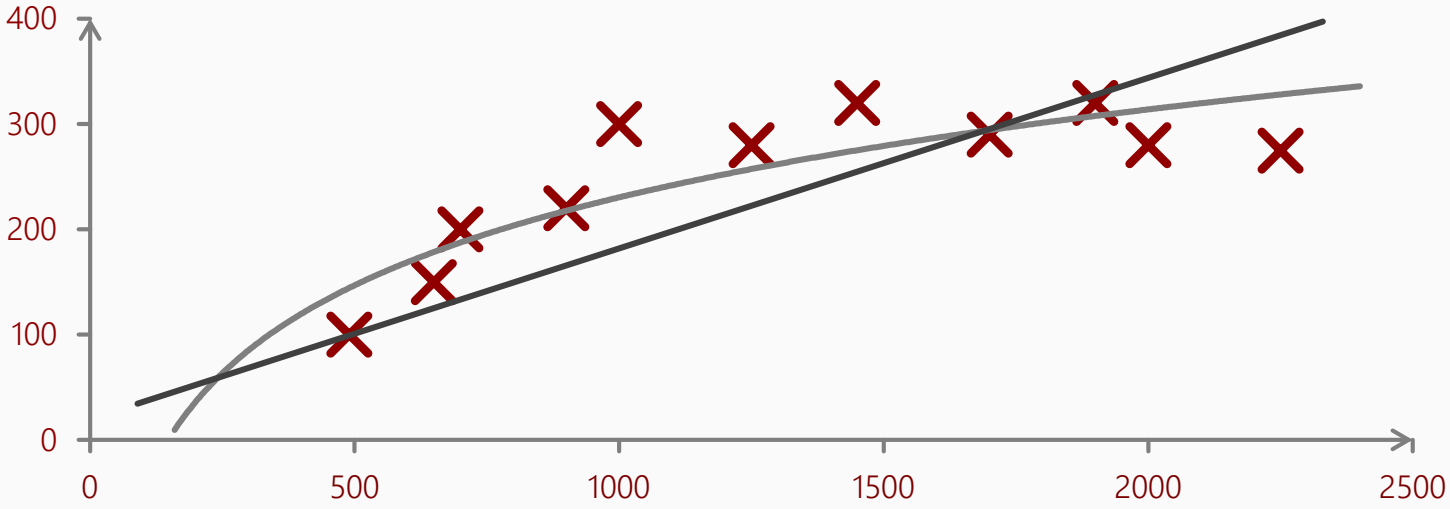
machine learning defined

Arthur Samuel (1959): Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell (1998): Well- posed learning problem is defined as a computer program set to learn from experience E with respect to some task T and some performance measure P, it its performance on T, as measured by P, improves with experience.

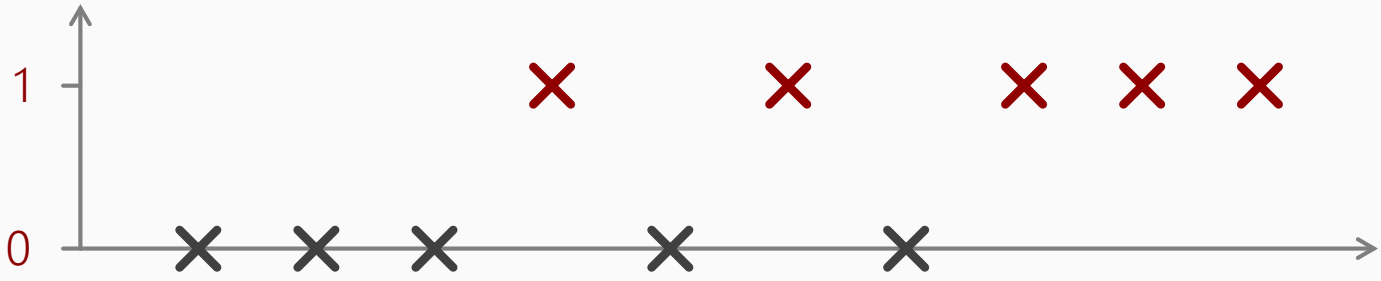
supervised learning

supervised learning gives a learning an algorithm the correct predicted values of the target feature. The following illustration is a regression example of the latter:



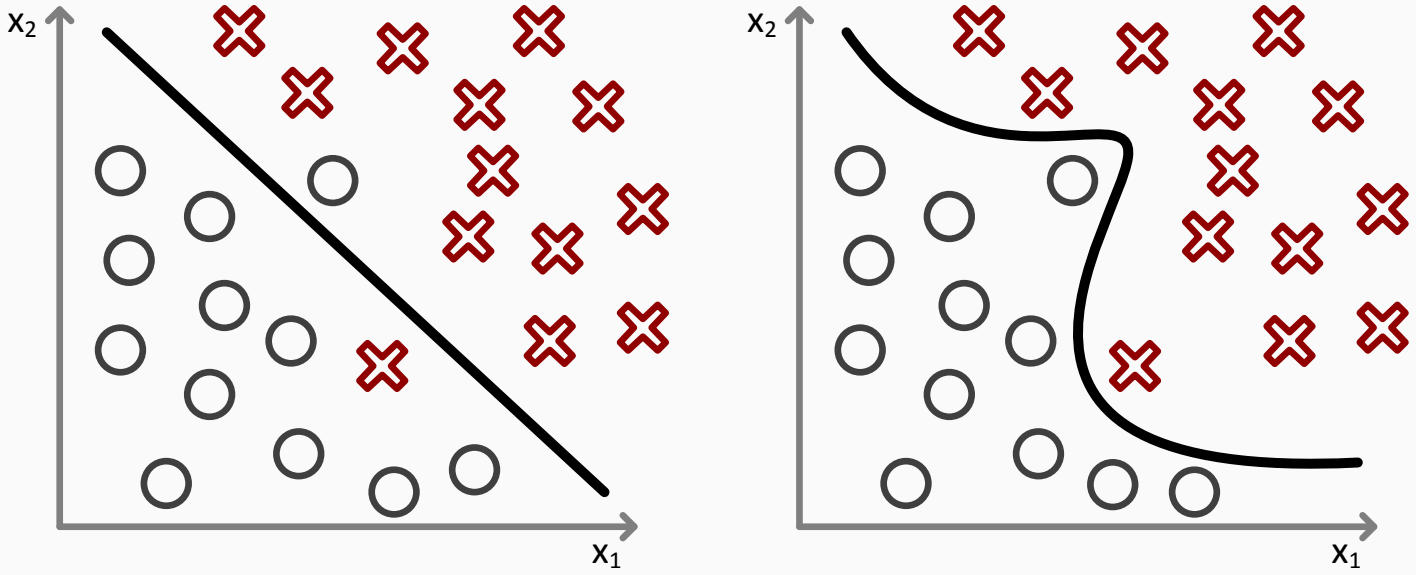
The illustration includes both a linear regressive model fitting the data and a quadratic equation fitting the data above. The model predicts the output of continuous variables

Another supervised learning example is displayed as a classification problem below:



Discrete values plotting the training examples seen above are plotted differently below:

The data is alternatively plotted on a single lined with their values denoted in shapes Other machine learning algorithms will often processes multiple features seen below:



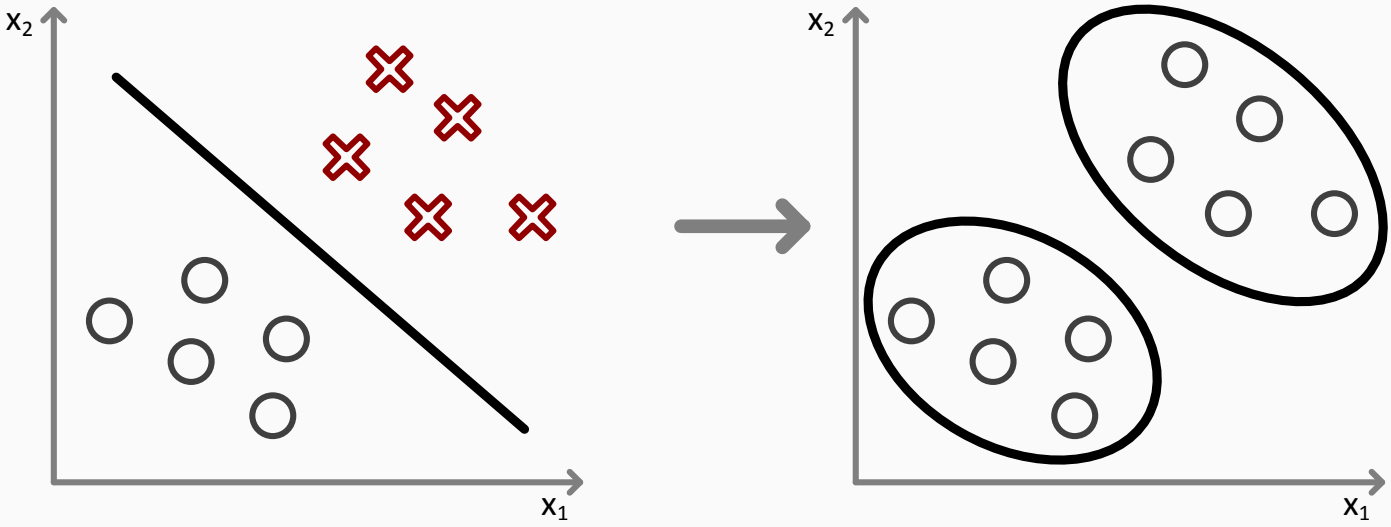
The illustration displays a multivariate logistic regression equation with both linear decision boundaries and more complex decision boundaries commonly used

Complex decision boundaries in learning algorithms are more accurate in classifying examples that contain more extreme features prone to linear misclassification

unsupervised learning

Unsupervised learning applies learning algorithms to unlabeled datasets in order to discover patterns amongst the unlabeled examples within the dataset

The illustration below visually compares the difference between supervised and unsupervised learning situations:



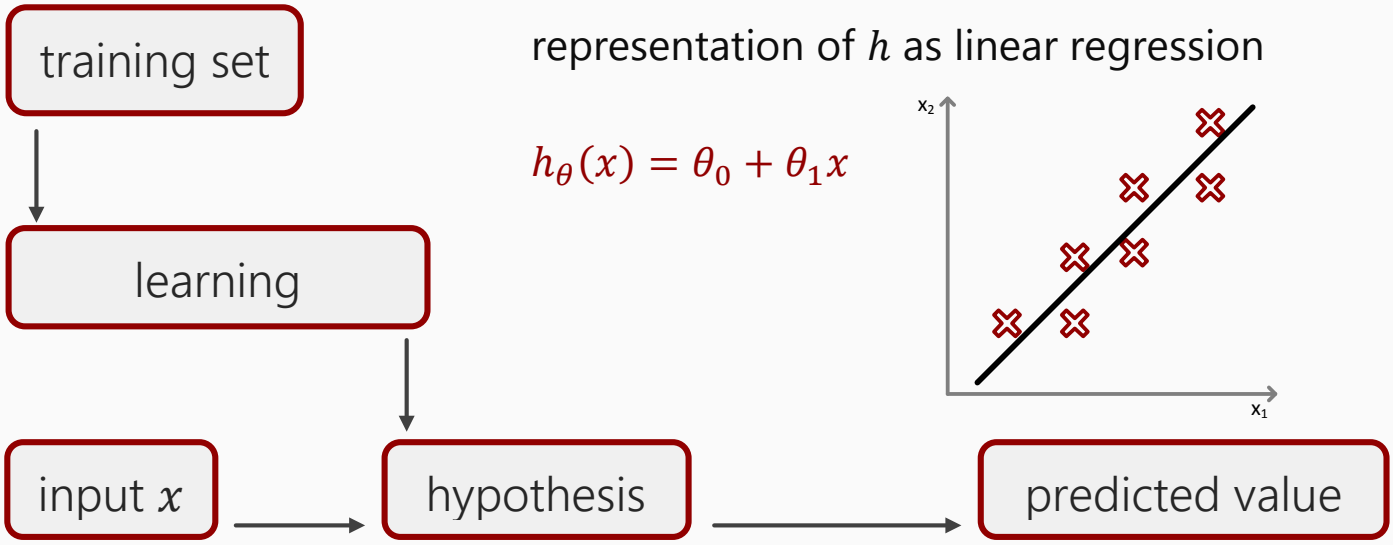
linear regression basics with one variable

model and cost function

model representation

training set definitive notation:

$m =$ number of training examples
$x =$ 'input' variables/features
$y =$ 'output' variables/features
$(x, y) =$ single training example
$(x^{(i)}, y^{(i)}) = i^{th}$ training example

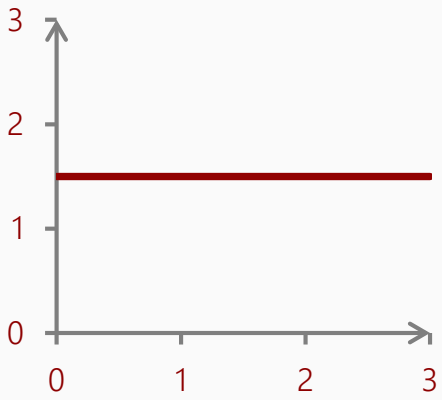


representation of h as linear regression

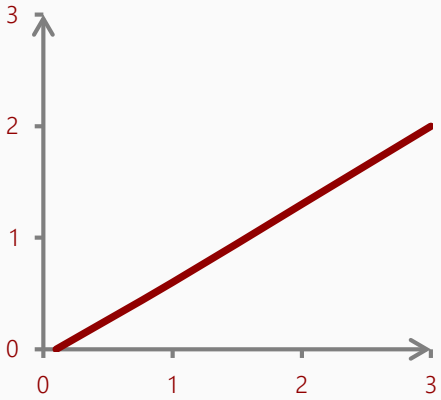
$h_\theta(x) = \theta_0 + \theta_1 x$

shorthand: $h_\theta(x)$**cost function**

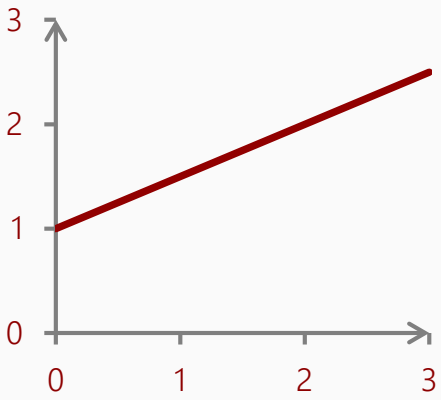
determination of the parameters in $h_\theta(x) = \theta_0 + \theta_1 x$; θ_0 and θ_1:



$\theta_1 = 1.5$
$\theta_0 = 0$



$\theta_1 = 0$
$\theta_0 = 0.5$



$\theta_1 = 1$
$\theta_0 = 0.5$

the motivation is to choose θ_1, θ_0 so that $h_\theta(x)$ is close to y for the training examples $(x^{(i)}, y^{(i)})$

the formal expression takes the form of a minimization objective function of the average of the sum of least squared errors (linear regression function):

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

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

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

$$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1) \rightarrow \textbf{cost function}$$

cost function – intuition i

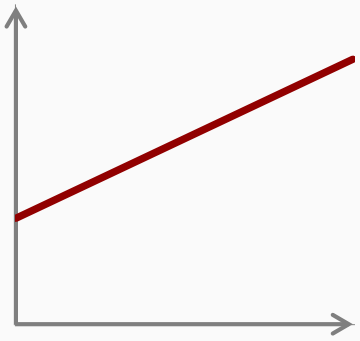
hypothesis:

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

parameters:

$$\theta_0, \theta_1$$

cost function:



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

goal:

$$\underset{\theta_0,\theta_1}{\text{minimize}}\, J(\theta_0, \theta_1)$$

simplified:

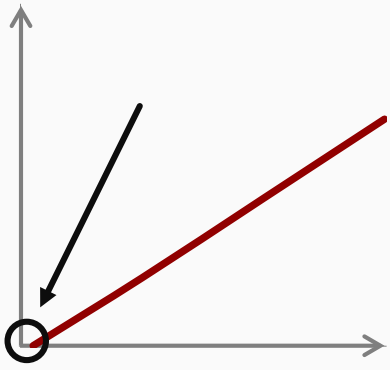
hypothesis:

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

parameters:

$$\theta_1$$

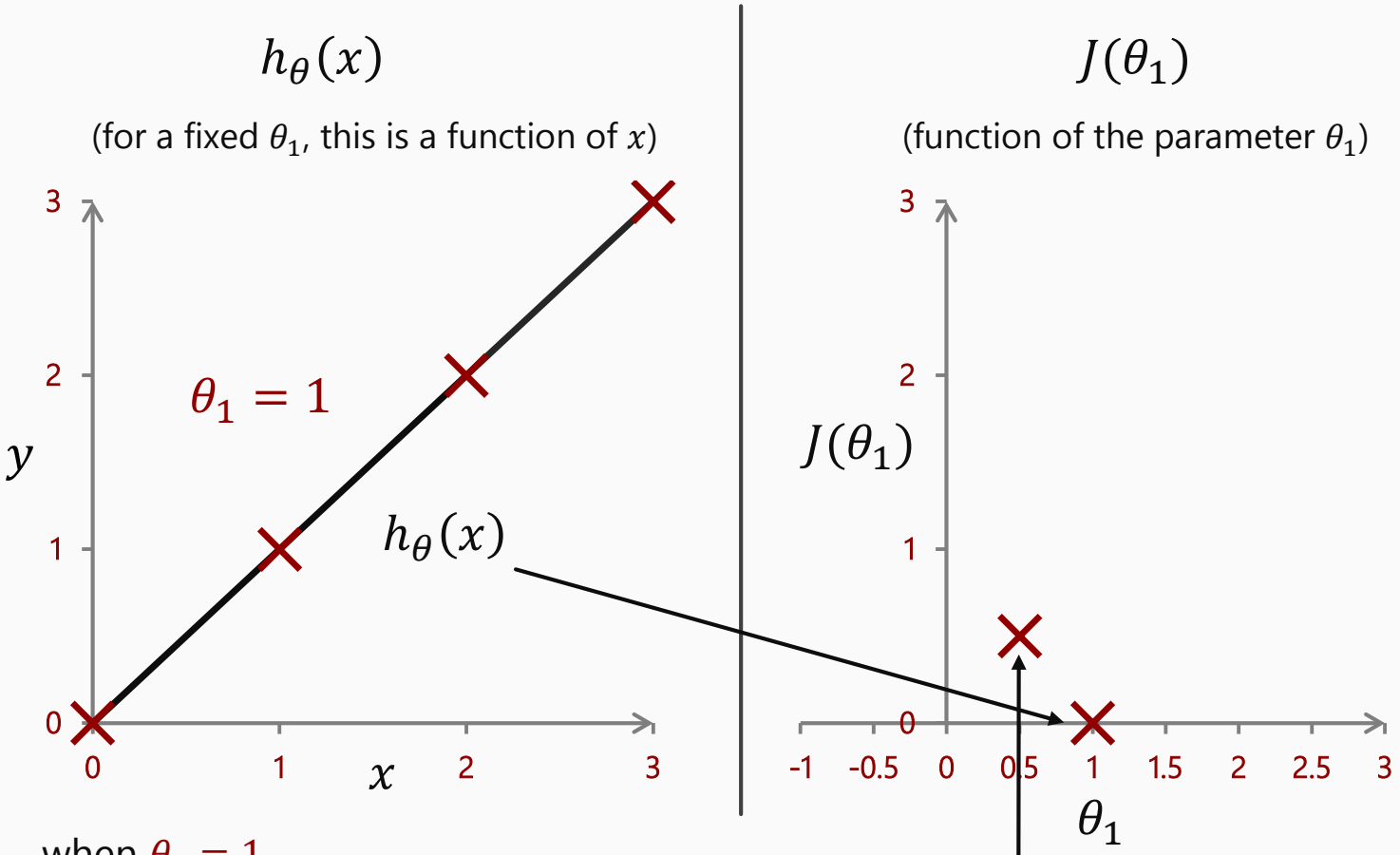
cost function:



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

goal:

$$\underset{\theta_1}{\text{minimize}}\, J(\theta_1)$$



when $\theta_1 = 1$

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

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

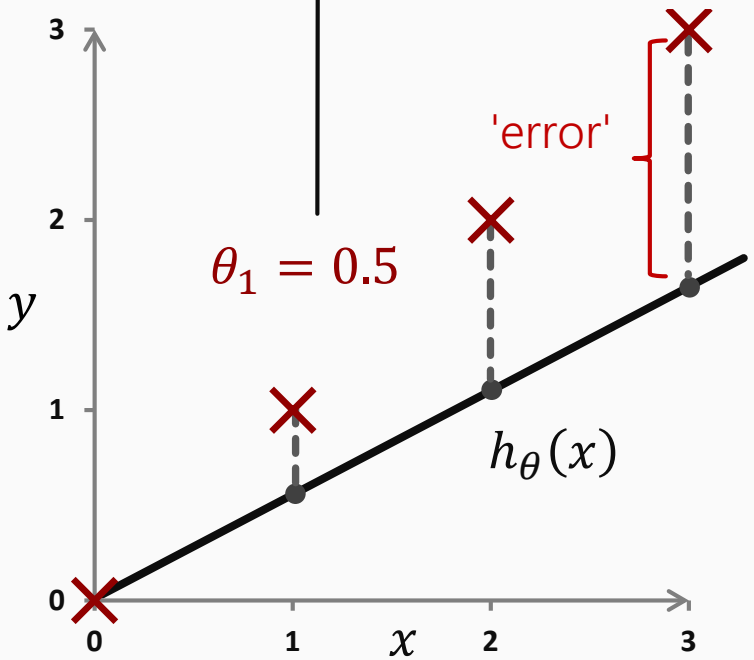
$$= \frac{1}{2m}(0^2, 0^2, 0^2) = 0$$

$$J(1) = 0$$

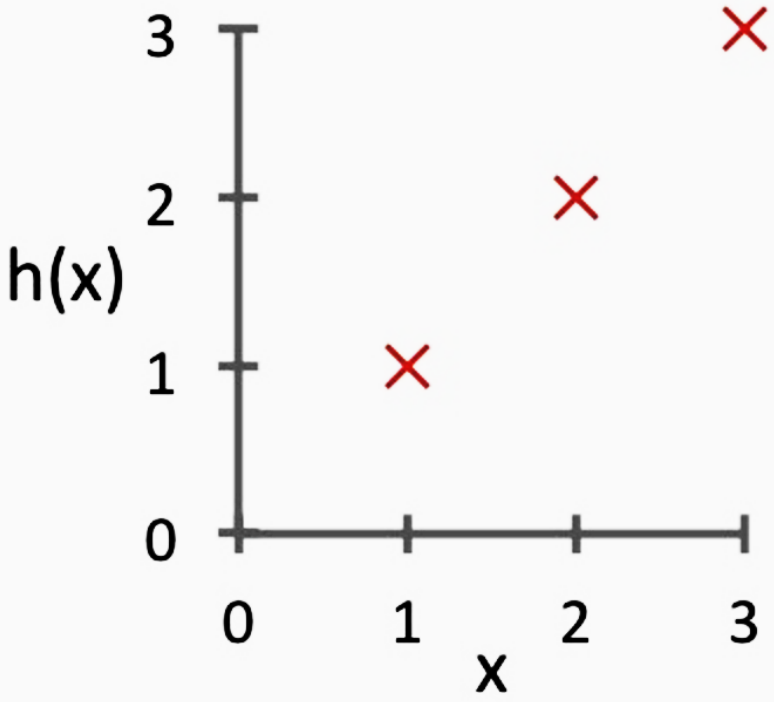
when $\theta_1 = 0.5 \rightarrow J(0.5) \approx 0.58$

$$J(0.5) = \frac{1}{2m}\left[\begin{matrix}(0.5-1)^2 + (1-2)^2 \\ + (1.5-3)^2\end{matrix}\right]$$

$$= \frac{1}{2\times 3}(3 \times 5) = \frac{3.5}{6} \approx 0.58$$



Suppose we have a training set with m=3 examples, plotted below. Our hypothesis representation is $h_\theta(x) = \theta_1 x$, with parameter θ_1. The cost function $J(\theta_1)$ is $J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$. What is $J(0)$?



$$J(0) = \frac{14}{6}$$

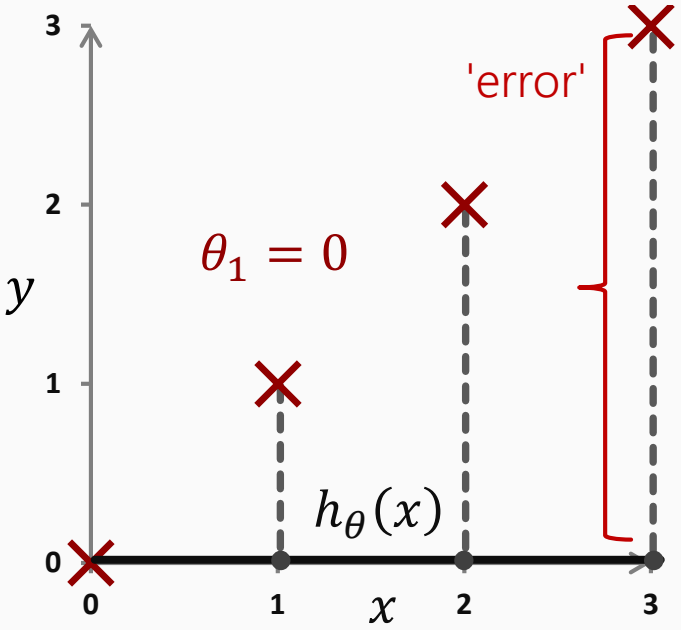
computing all functions to plot the parameters $J(\theta_i)$ ultimately plots an expected polynomial function representative of the cost function

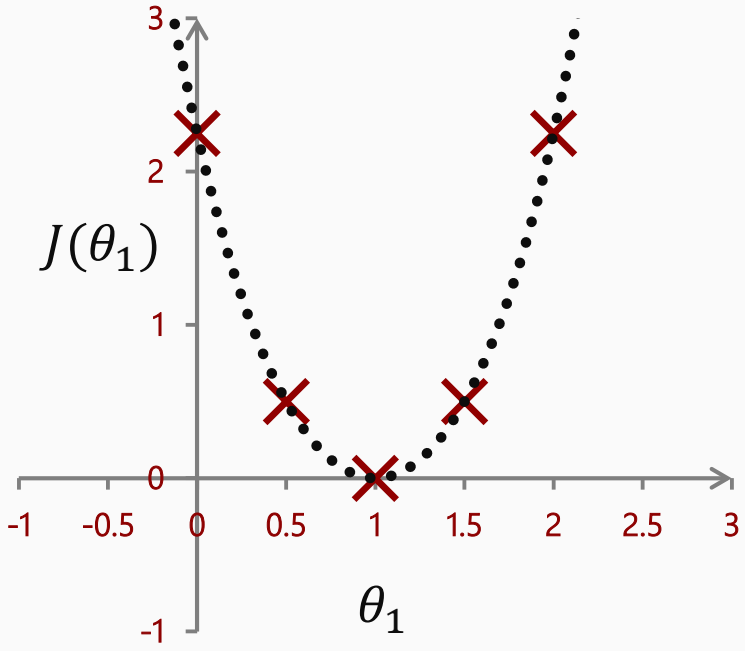
returning to the objective function $\underset{\theta_1}{\text{minimize}}\, J(\theta_1)$: the polynomial function proves the function is minimized when $\theta_1 = 1$. this equally can be seen as fitting the data most accurately in the first illustration

when $\theta_1 = 0 \rightarrow J(0) \approx 2.3$

$$J(0.5) = \frac{1}{2m}[1^2 + 2^2 + 3^2]$$

$$= \frac{1}{6}(14) = \frac{14}{6} \approx 2.3$$

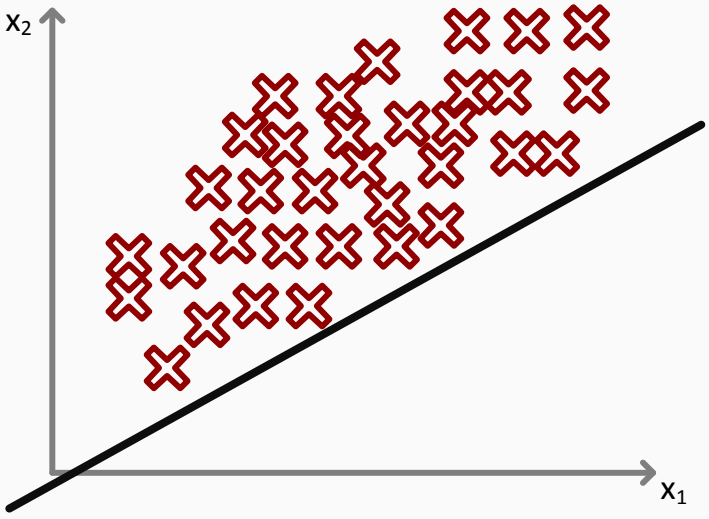




cost function – intuition ii

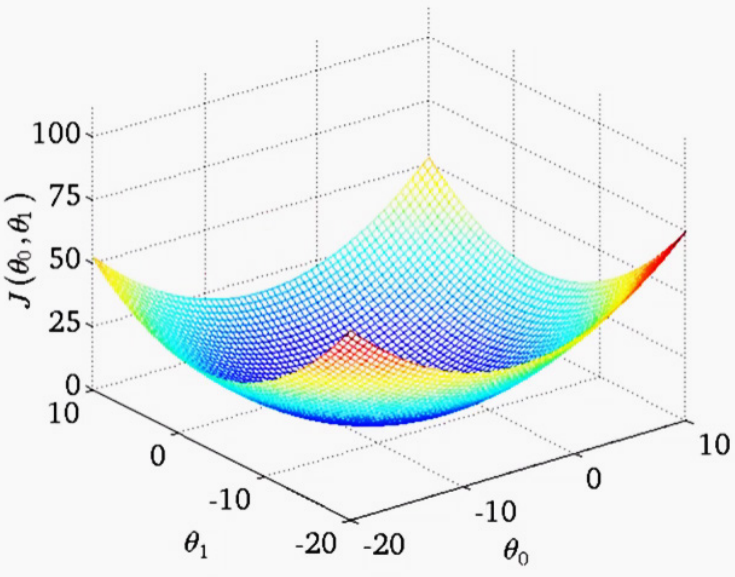
$h_\theta(x)$

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



$J(\theta_1, \theta_0)$

(function of the parameters θ_0, θ_1)



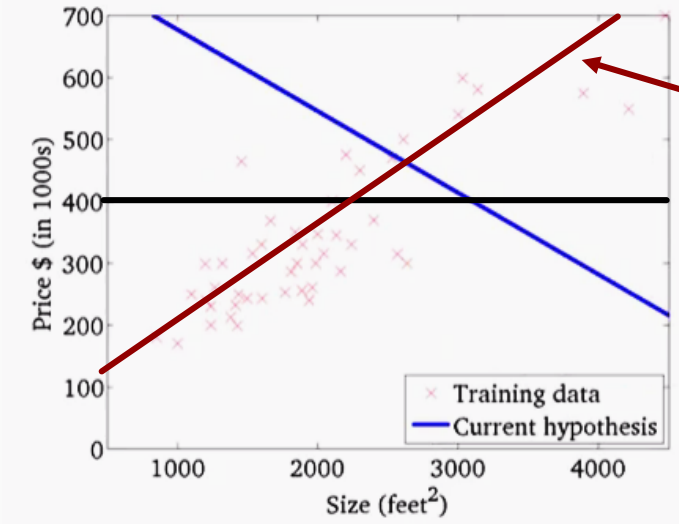
removing the simplification ($J(\theta_1)$) exercised previously, the function becomes slightly more complicated. the polynomial plotted function illustrated in the last example becomes a 3D contour plot through the inclusion of θ_0 back into the function $J(\theta_1, \theta_0)$

the height of the distance from θ_1, θ_0 are representative of $J(\theta_1, \theta_0)$

the above can be illustrated in 2 dimensions as a contour plot or figure

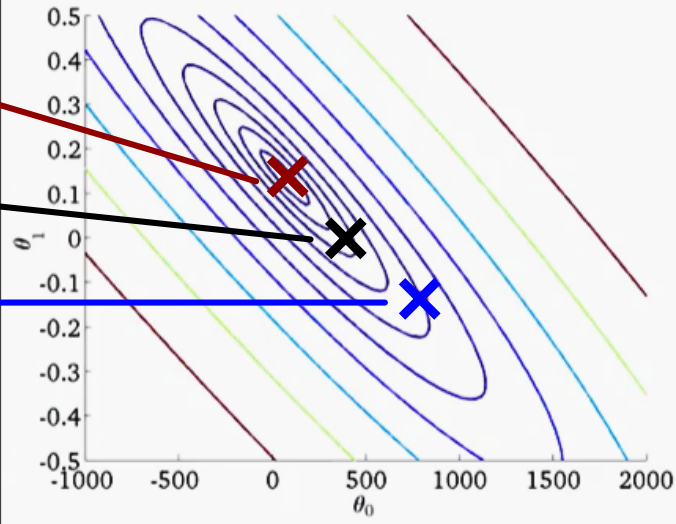
$h_\theta(x)$

(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)



parameter learning $\boldsymbol{\theta}$ gradient descent basics

gradient descent

the use of gradient descent as a mechanism for minimizing the function of $J(\theta_1, \theta_0)$

note: for brevity, the functions $J(\theta_1, \theta_0)$ are shortened throughout the document

$$J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) \rightarrow J(\theta_1, \theta_0)$$

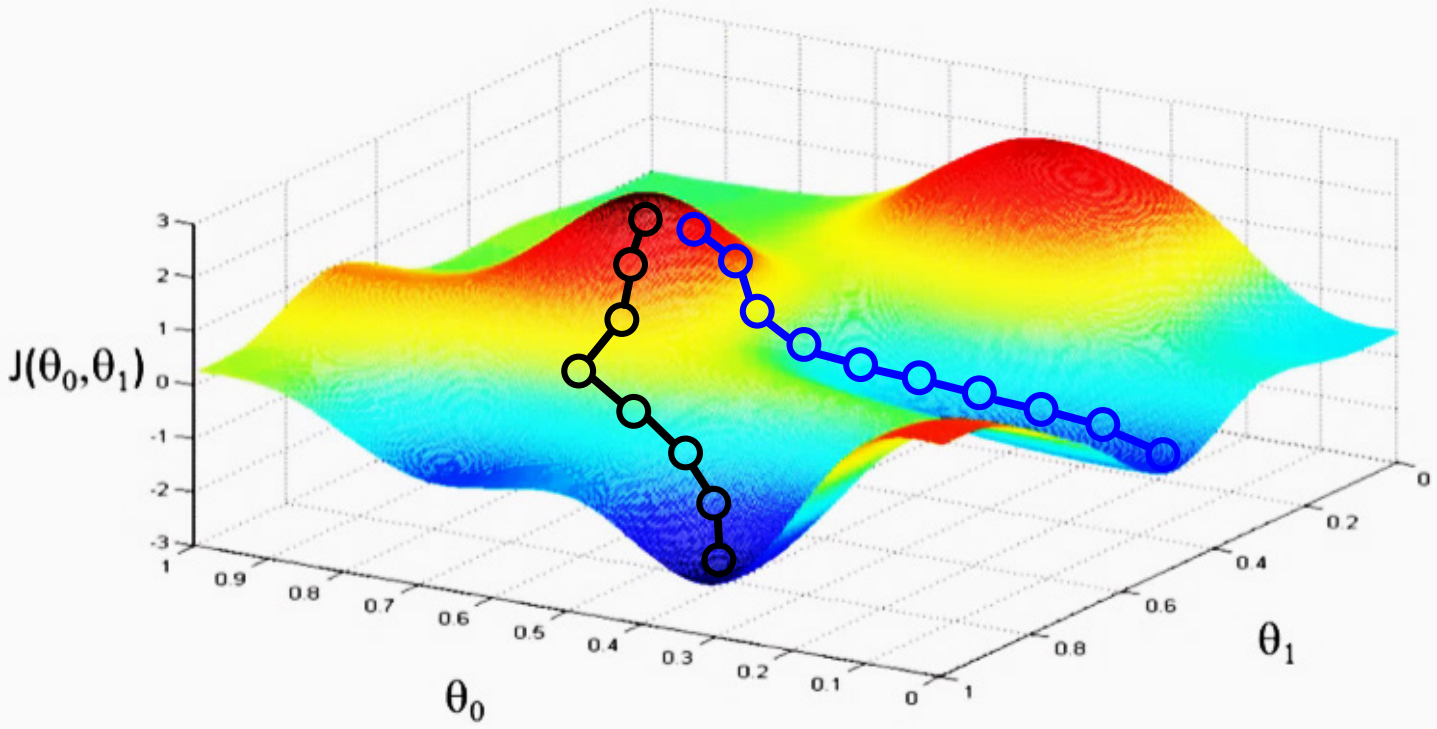
$$\min_{\theta_0, \theta_1, \theta_2, \dots, \theta_n} J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) \rightarrow \min_{\theta_0, \theta_1} J(\theta_1, \theta_0)$$

the process begins by initializing some value of parameters θ_0, θ_1 (e.g. $\theta_0 = 0, \theta_1 = 0$)

the values θ_0, θ_1 will then change through reiterations to reduce the value of $J(\theta_0, \theta_1)$

the reiterations will run continuously with the intended arrival at a minimum

illustrated below, the initialized values will be plotted against the function:



in narrative explanation, gradient descent begins at the points initialized and scans the contour in a 360°-like fashion and chose a direction the descends the counter in the most rapid route. the initialized values will update to the new point on the contour and repeat the process until the functions determines it has reached a **local minimum**

a unique property of gradient descent occurs dependent upon the initialized values of parameters . different initialization values can result in a disjoint **local minimum**

the formal gradient descent algorithm is as follows:

repeat until convergence {

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

}

the a term determines the magnitude of each step in gradient descent

the term $\frac{\partial}{\partial\theta_j}$ is the partial derivative active in gradient descent

the gradient descent algorithm functions as a simultaneous update of θ_0, θ_1:

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 simultaneous update:

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

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

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

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

the incorrect example of simultaneous update displays the use of the updated value of θ_0 in the computation of updating θ_1

Suppose $\theta_0 = 1, \theta_1 = 2$, and we simultaneously update θ_0 and θ_1 using the rule: $\theta_j := \theta_j + \sqrt{\theta_0\theta_1}$ (for j = 0 and j=1) What are the resulting values of θ_0 and θ_1?

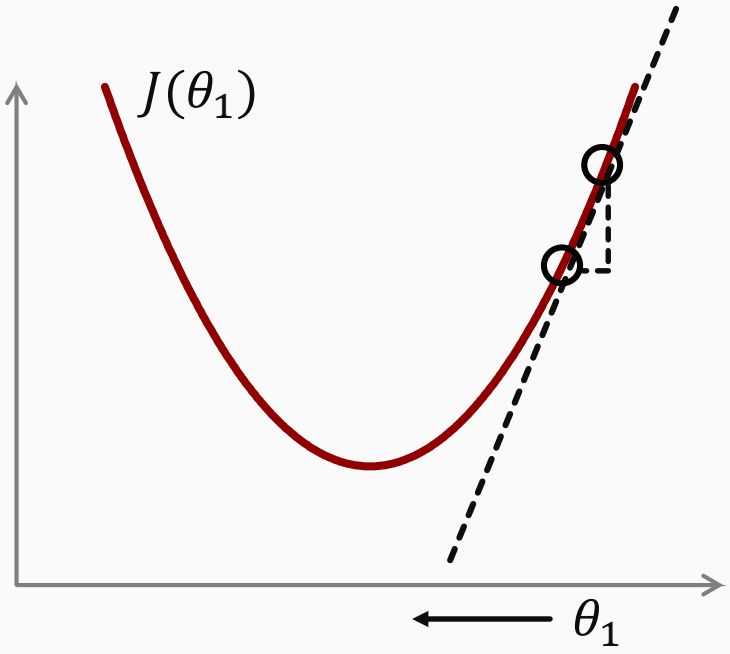
- $\theta_0 = 1, \theta_1 = 2$
- $\theta_0 = 1 + \sqrt{2}, \theta_1 = 2 + \sqrt{2}$

Correct Response

- $\theta_0 = 2 + \sqrt{2}, \theta_1 = 1 + \sqrt{2}$
- $\theta_0 = 1 + \sqrt{2}, \theta_1 = 2 + \sqrt{(1+\sqrt{2})\cdot 2}$

gradient descent intuition

the function of the derivative in the gradient descent function takes the tangent slope of the line plotted against θ_1 and multiplies it against learning rate alpha α and cost function $J(\theta_1)$; the functions then apples the direction and magnitude into a step

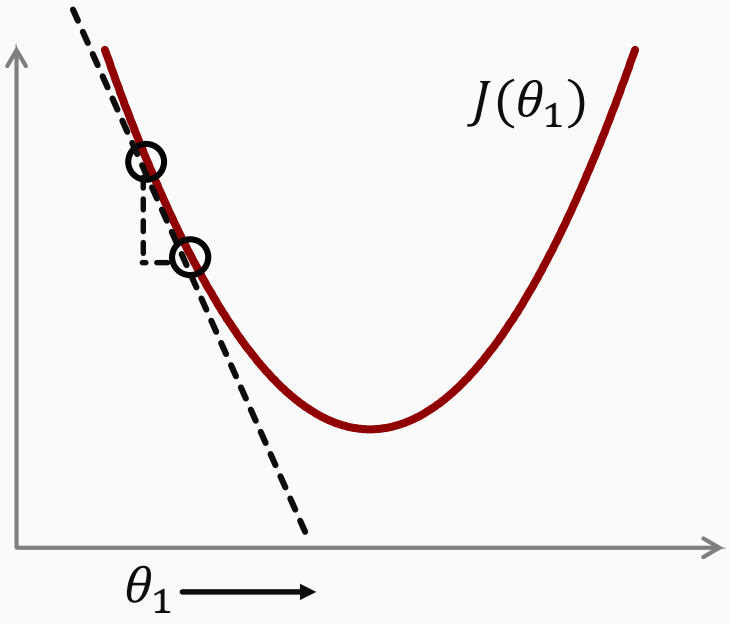


$$(\theta_1 \in \mathbb{R})$$

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

$$\geq 0$$

$$\theta_1 := \theta_1 - \alpha(\textbf{positive number})$$



$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

$$\leq 0$$

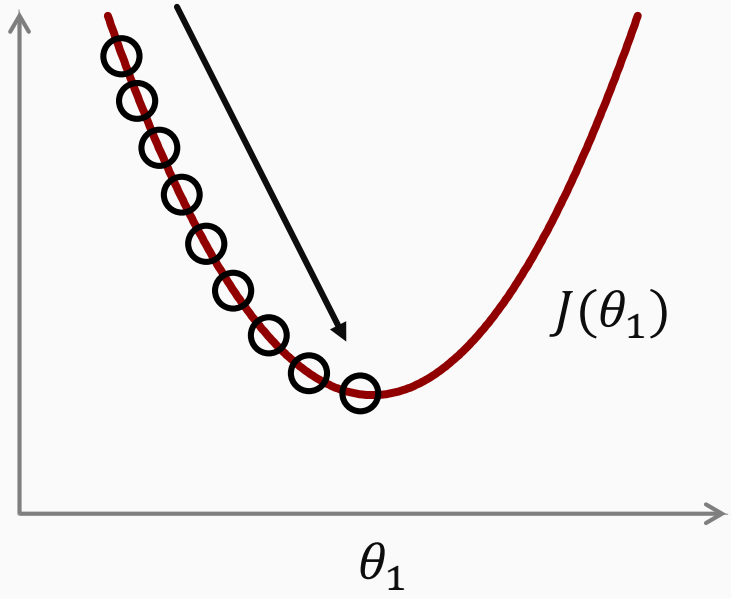
$$\theta_1 := \theta_1 - \alpha(\textbf{negative number})$$

the initialized values in the gradient descent function will determine whether the tangent slope of the derivative will require a **positive** or **negative** number to make a step towards finding a local minimum

with reference to the behavior of the assigned learning rate α:

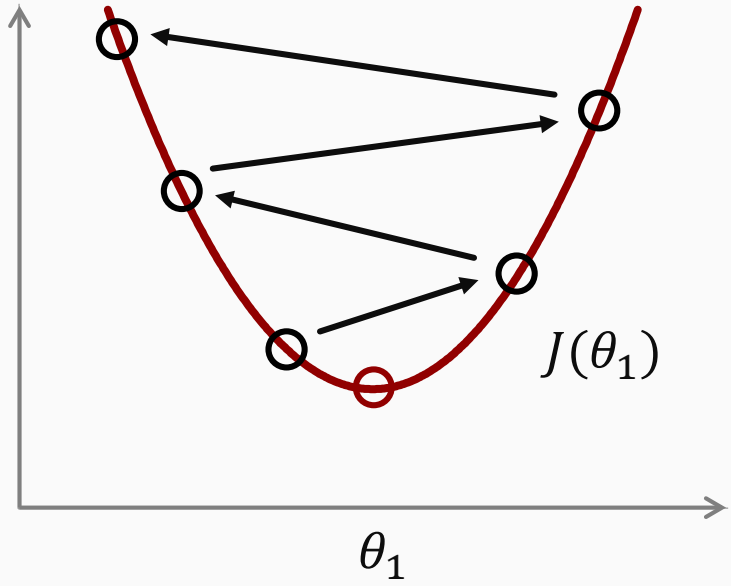
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

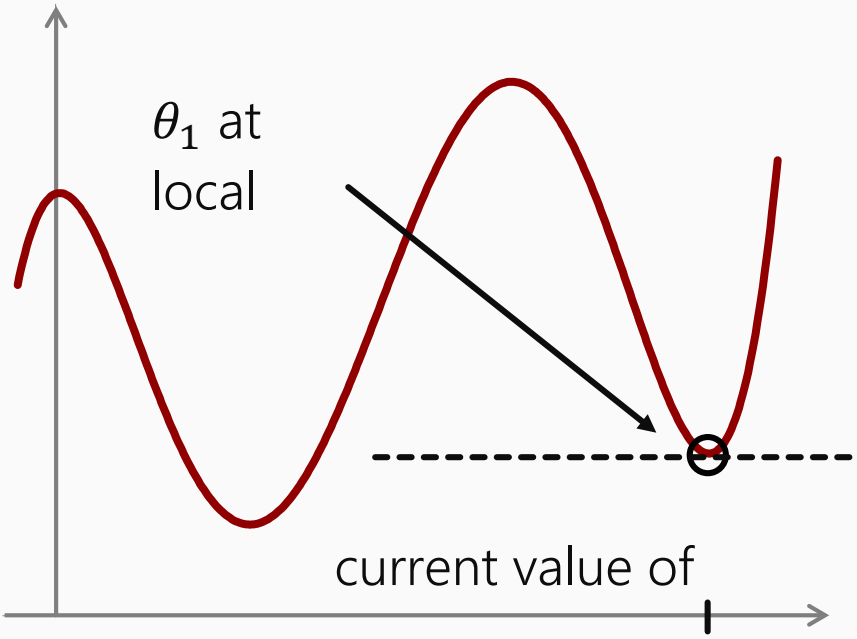
if the learning rate α is too **small**, gradient descent can processes slowly and take substantial time and resources converging to a local optima (minimum)



$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

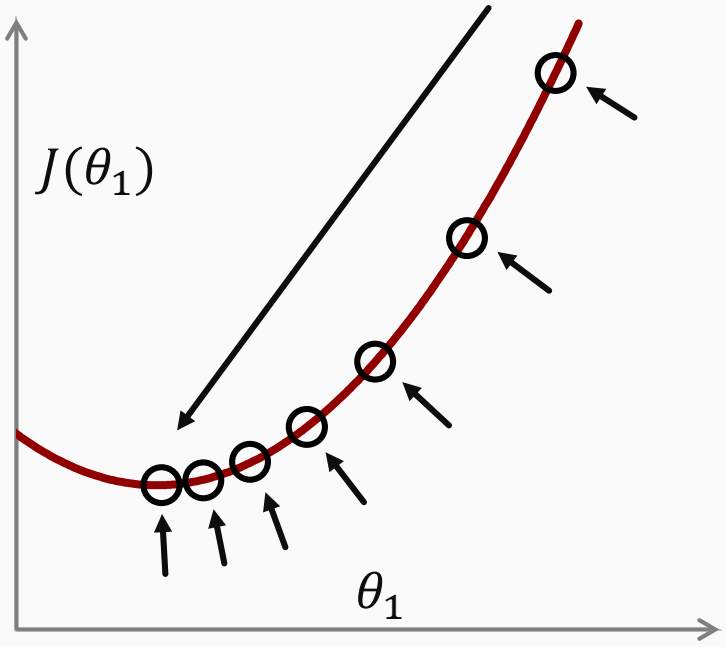
if the learning rate α is too **large**, gradient descent can overshoot the local optima (minimum); it might equally fail to converge, or diverge entirely from obtaining a local optima (minimum)





if θ_1 is already at a local optimum of $J(\theta_1)$, a single additional step gradient descent $\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$ will leave θ_1 unchanged

at local minimum, the tangent slope of the derivative = 0 and thus will not affect θ_1



if the learning rate alpha α is fixed, gradient descent can still converge to a local minimum

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{d}{d\theta_1} J(\theta_1)}$$

after each simultaneous update of gradient descent, the algorithm autonomously takes smaller steps due to a decreasing derivative computation with each iteration

gradient descent for linear regression

applying gradient descent to minimize the linear regression cost function:

gradient descent algorithm

repeat until convergence {

$\theta_j := \theta_j - a \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

(for $j = 0$ and $j = 1$)

}

linear regression model

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

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

$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

determining the function of the derivative term $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$:

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

$$= \frac{\partial}{\partial \theta_j} \times \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

determining the partial derivatives in both cases of θ_0, θ_1

the partial derivatives are simply the slopes of the cost function $J(\theta_0, \theta_1)$

$\theta_0 \rightarrow j = 0$: $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

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

gradient descent algorithm for linear regression:

repeat until convergence {

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

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

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

update θ_0 and θ_1 simultaneously

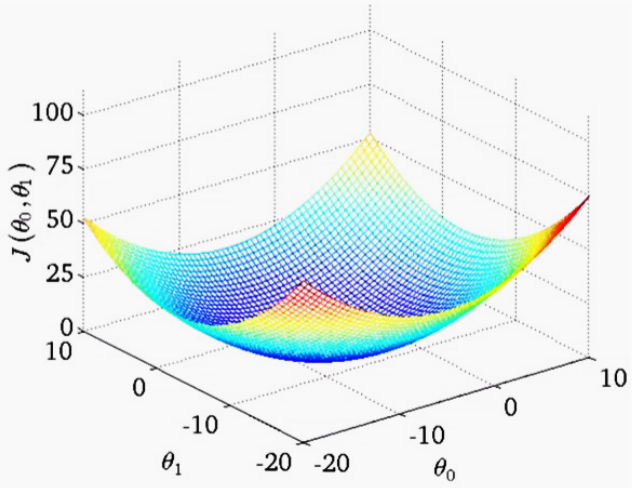
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

}

the cost function for linear regression will always be represented as a **convex function**

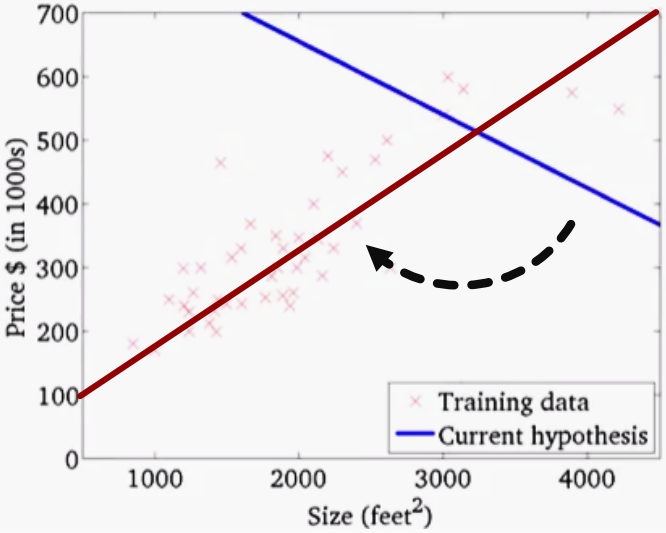
"**batch**" gradient descent is a common term in machine learning to refer to each step of gradient descent using all of the training examples in the dataset

convex functions do not consist of any local optima, there is now only an available **global optima** (global minimum) of the convex function



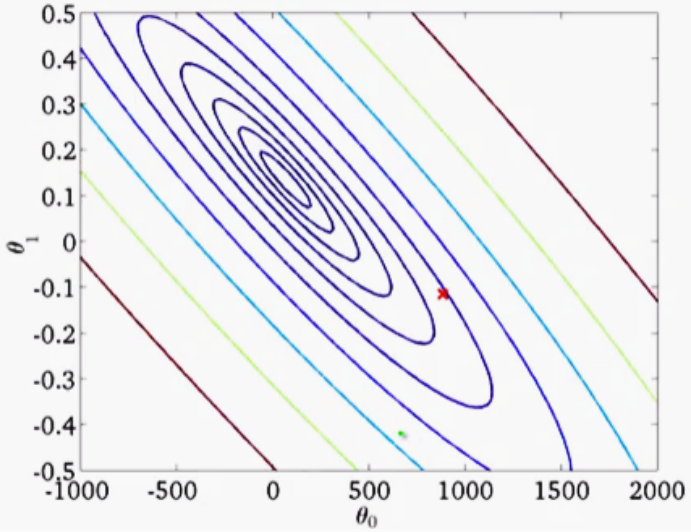
$h_\theta(x)$

(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)



linear algebra basics for machine learning

matrices and vectors

a matrix is a rectangular array of numbers:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

4 x 2 matrix $\mathbb{R}^{2\times 4}$ **2 x 4 matrix** $\mathbb{R}^{2\times 4}$

the dimensions of a matrix are donated as **rows^n x columns^n**

Which of the following statements are true? Check all that apply.

- $\begin{bmatrix} 1 & 2 \\ 4 & 0 \\ 0 & 1 \end{bmatrix}$ is a 3×2 matrix.

  Correct Response

- $\begin{bmatrix} 0 & 1 & 4 & 2 \\ 3 & 4 & 0 & 9 \end{bmatrix}$ is a 4×2 matrix.

  Correct Response

- $\begin{bmatrix} 0 & 4 & 2 \\ 3 & 4 & 9 \\ 5 & -1 & 0 \end{bmatrix}$ is a 3×3 matrix.

  Correct Response

- $\begin{bmatrix} 1 & 2 \end{bmatrix}$ is a 1×2 matrix.

  Correct Response

matrix elements (entries of a matrix):

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \quad \rightarrow \quad A_{ij} = i,j \text{ entry in the } i^{th} \text{ row}, j^{th}, \text{column}$$

for example: $A_{11} = 1,\ A_{12} = 2,\ A_{32} = 6,$ and $A_{41} = 7;\ A_{43} = \text{undefined (error)}$

a vector is an **n x 1** matrix:

$$y = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad \rightarrow \quad 4 - \text{dimensional vector;} \quad \mathbb{R}^4$$

$y_i = i^{th}$ element $\rightarrow\ y_1 = 1,\ y_2 = 2,\ y_3 = 3$

1-indexed vs 0-indexed

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

addition and scalar multiplication

matrix addition only applies when matrices have the same dimensions:

$$\begin{bmatrix}1 & 0\\2 & 5\\3 & 1\end{bmatrix} + \begin{bmatrix}4 & 0.5\\2 & 5\\0 & 1\end{bmatrix} = \begin{bmatrix}5 & 0.5\\2 & 10\\3 & 2\end{bmatrix}$$

$\mathbb{R}^{3\times2}$ $\mathbb{R}^{3\times2}$ $\mathbb{R}^{3\times2}$

$$\begin{bmatrix}1 & 0\\2 & 5\\3 & 1\end{bmatrix} + \begin{bmatrix}4 & 0.5\\2 & 5\end{bmatrix} = \text{error}$$

$\mathbb{R}^{3\times2}$ $\mathbb{R}^{2\times2}$

scalar (real number) matrix multiplication

$$3 \times \begin{bmatrix}1 & 0\\2 & 5\\3 & 1\end{bmatrix} = \begin{bmatrix}3 & 0\\6 & 15\\9 & 3\end{bmatrix}$$

$\mathbb{R}^{3\times2}$ $\mathbb{R}^{3\times2}$

$$\begin{bmatrix}4 & 0\\6 & 3\end{bmatrix}/4 = \frac{1}{4} \times \begin{bmatrix}4 & 0\\6 & 3\end{bmatrix} = \begin{bmatrix}0 & 0\\\frac{3}{2} & \frac{3}{4}\end{bmatrix}$$

$\mathbb{R}^{3\times2}$ $\mathbb{R}^{3\times2}$ $\mathbb{R}^{3\times2}$

combination of operands:

$$3 \times \begin{bmatrix}1\\4\\2\end{bmatrix} + \begin{bmatrix}0\\0\\5\end{bmatrix} - \begin{bmatrix}3\\0\\2\end{bmatrix}/3 = \begin{bmatrix}3\\12\\6\end{bmatrix} + \begin{bmatrix}0\\0\\5\end{bmatrix} + \begin{bmatrix}1\\0\\\frac{2}{3}\end{bmatrix} = \begin{bmatrix}2\\12\\10\frac{1}{3}\end{bmatrix}$$

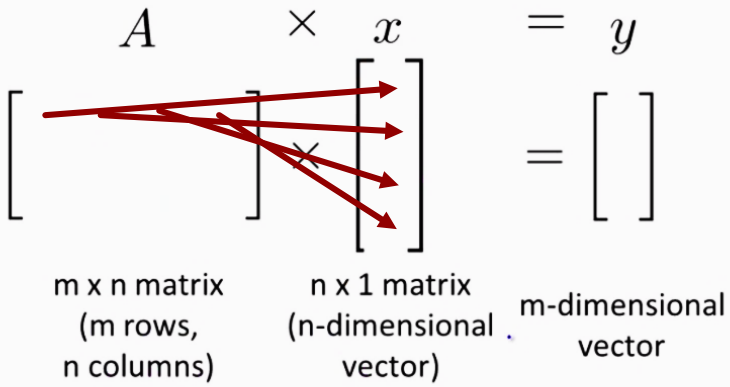
$$\begin{bmatrix}4\\6\\7\end{bmatrix}/2 \; - 3\begin{bmatrix}2\\1\\0\end{bmatrix} = \begin{bmatrix}2\\3\\3.5\end{bmatrix} - \begin{bmatrix}6\\3\\0\end{bmatrix} = \begin{bmatrix}-4\\0\\3.5\end{bmatrix}$$

matrix vector multiplication

$$\begin{bmatrix}1 & 3\\4 & 0\\2 & 1\end{bmatrix} \times \begin{bmatrix}1\\5\end{bmatrix} = \begin{bmatrix}1\times1+3\times5=16\\4\times1+0\times5=4\\2\times1+1\times5=7\end{bmatrix} = \begin{bmatrix}16\\4\\7\end{bmatrix}$$

$\mathbb{R}^{3\times2}$ $\mathbb{R}^{2\times1}$ → $\mathbb{R}^{3\times1}$

Details:



To get y_i, multiply A's i^{th} row with elements of vector x, and add them up.

$$\begin{bmatrix} 1 & 2 & 1 & 5 \\ 0 & 3 & 0 & 4 \\ -1 & -2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \times 1 + 2 \times 3 + 1 \times 2 + 5 \times 1 = 14 \\ 0 \times 1 + 3 \times 3 + 0 \times 2 + 4 \times 1 = 13 \\ -1 \times 1 + (-2) \times 3 + 0 \times 2 + 0 \times 1 = -7 \end{bmatrix} = \begin{bmatrix} 14 \\ 13 \\ -7 \end{bmatrix}$$

$\mathbb{R}^{3\times4}$ $\mathbb{R}^{4\times1}$ → $\mathbb{R}^{3\times1}$

$$\begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 5 \\ 3 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 6 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \times 1 + 0 \times 6 + 3 \times 2 = 7 \\ 2 \times 1 + 1 \times 6 + 5 \times 2 = 18 \\ 3 \times 1 + 1 \times 6 + 2 \times 2 = 13 \end{bmatrix} = \begin{bmatrix} 7 \\ 18 \\ 13 \end{bmatrix}$$

$\mathbb{R}^{3\times3}$ $\mathbb{R}^{3\times1}$ → $\mathbb{R}^{3\times1}$

applying **matrix vector** multiplication to a function $h_\theta(x)$:

examples:

2104
1416
1534
852

hypothesis:

$$h_\theta(x) = -40 + 0.25x$$

matrix x vector = predicted values $h_\theta(x)$ = $h_\theta(x)$

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times \begin{bmatrix} -40 \\ 0.25 \end{bmatrix} = \begin{bmatrix} -40 \times 1 + 0.25 \times 2104 = 486 \\ -40 \times 1 + 0.25 \times 1416 = 314 \\ -40 \times 1 + 0.25 \times 1534 = 343.5 \\ -40 \times 1 + 0.25 \times 852 = 173 \end{bmatrix} = \begin{bmatrix} 486 \\ 314 \\ 343.5 \\ 173 \end{bmatrix}$$

$\mathbb{R}^{4\times2}$ $\mathbb{R}^{2\times1}$ → $\mathbb{R}^{4\times1}$

matrix matrix multiplication

$$\begin{bmatrix} \mathbf{1} & \mathbf{3} & \mathbf{2} \\ \mathbf{4} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{3} \\ \mathbf{0} & \mathbf{1} \\ \mathbf{5} & \mathbf{2} \end{bmatrix} = \begin{bmatrix} 11 & \mathbf{10} \\ 9 & \mathbf{14} \end{bmatrix}$$

$\mathbb{R}^{2\times3}$ $\mathbb{R}^{2\times3}$ → $\mathbb{R}^{2\times2}$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \times 1 + 3 \times 0 + 2 \times 5 = 11 \\ 4 \times 1 + 0 \times 0 + 1 \times 5 = 9 \end{bmatrix} = \begin{bmatrix} \mathbf{11} \\ \mathbf{9} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \times 3 + 3 \times 1 + 2 \times 2 = 10 \\ 4 \times 3 + 0 \times 3 + 1 \times 2 = 14 \end{bmatrix} = \begin{bmatrix} \mathbf{10} \\ \mathbf{14} \end{bmatrix}$$

Details:

$$A \times B = C$$

$$\begin{bmatrix} \quad \end{bmatrix} \times \begin{bmatrix} \quad \end{bmatrix} = \begin{bmatrix} \quad \end{bmatrix}$$

m x n matrix (m rows, n columns)

n x o matrix (n rows, o columns)

m x o matrix

The i^{th} column of the matrix C is obtained by multiplying A with the i^{th} column of B. (for i = 1,2,...,o)

$$\begin{bmatrix} \mathbf{1} & \mathbf{3} \\ \mathbf{2} & \mathbf{5} \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{3} & \mathbf{2} \end{bmatrix} = \begin{bmatrix} 9 & \mathbf{4} \\ 15 & \mathbf{12} \end{bmatrix}$$

$\mathbb{R}^{2\times 2} \quad \mathbb{R}^{3\times 2} \rightarrow \mathbb{R}^{2\times 2}$

$$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 1\times 0 + 3\times 3 = 9 \\ 2\times 0 + 5\times 3 = 15 \end{bmatrix} = \begin{bmatrix} \mathbf{9} \\ \mathbf{15} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1\times 1 + 3\times 2 = 7 \\ 2\times 1 + 5\times 2 = 12 \end{bmatrix} = \begin{bmatrix} \mathbf{7} \\ \mathbf{12} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{1} & \mathbf{3} \\ \mathbf{2} & \mathbf{4} \\ \mathbf{0} & \mathbf{5} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{2} & \mathbf{3} \end{bmatrix} = \begin{bmatrix} 7 & \mathbf{9} \\ 10 & \mathbf{12} \\ 10 & \mathbf{15} \end{bmatrix}$$

$\mathbb{R}^{3\times 2} \quad \mathbb{R}^{2\times 2} \rightarrow \mathbb{R}^{2\times 2}$

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1\times 1 + 3\times 2 = 7 \\ 2\times 1 + 4\times 2 = 10 \\ 0\times 1 + 5\times 2 = 10 \end{bmatrix} = \begin{bmatrix} \mathbf{7} \\ \mathbf{10} \\ \mathbf{10} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 1\times 0 + 3\times 3 = 9 \\ 2\times 0 + 4\times 3 = 12 \\ 0\times 0 + 5\times 3 = 15 \end{bmatrix} = \begin{bmatrix} \mathbf{9} \\ \mathbf{12} \\ \mathbf{15} \end{bmatrix}$$

applying **matrix matrix** multiplication to a function $h_\theta(x)$:

examples:

$$\begin{matrix}2104\\1416\\1534\\852\end{matrix}$$

hypothesis:

$$h_\theta(x) = -40 + 0.25x$$
$$h_\theta(x) = 200 + 0.10x$$
$$h_\theta(x) = -150 + 0.40x$$

matrix x matrix = predicted values $h_\theta(x)$

$$\begin{bmatrix}\mathbf{1} & \mathbf{2104}\\ \mathbf{1} & \mathbf{1416}\\ \mathbf{1} & \mathbf{1534}\\ \mathbf{1} & \mathbf{852}\end{bmatrix} \times \begin{bmatrix}\mathbf{-40} & \mathbf{200} & \mathbf{-150}\\ \mathbf{0.25} & \mathbf{0.10} & \mathbf{0.40}\end{bmatrix} = \begin{bmatrix}\mathbf{486} & \mathbf{410.4} & \mathbf{692.6}\\ \mathbf{314} & \mathbf{341.6} & \mathbf{416.4}\\ \mathbf{343.5} & \mathbf{353.4} & \mathbf{464.6}\\ \mathbf{173} & \mathbf{285.2} & \mathbf{191.8}\end{bmatrix}$$

$$\mathbb{R}^{4\times 2} \qquad \mathbb{R}^{2\times 3} \longrightarrow \mathbb{R}^{4\times 3}$$

$$\begin{bmatrix}1 & 2104\\ 1 & 1416\\ 1 & 1534\\ 1 & 852\end{bmatrix} \times \begin{bmatrix}-40\\ 0.25\end{bmatrix} = \begin{bmatrix}1\times -40 + 2104\times 0.25 = 486\\ 1\times -40 + 1416\times 0.25 = 314\\ 1\times -40 + 1534\times 0.25 = 343.5\\ 1\times -40 + 852\times 0.25 = 173\end{bmatrix} = \begin{bmatrix}\mathbf{486}\\ \mathbf{314}\\ \mathbf{343.5}\\ \mathbf{173}\end{bmatrix}$$

$$\begin{bmatrix}1 & 2104\\ 1 & 1416\\ 1 & 1534\\ 1 & 852\end{bmatrix} \times \begin{bmatrix}200\\ 0.10\end{bmatrix} = \begin{bmatrix}1\times 200 + 2104\times 0.10 =\\ 1\times 200 + 1416\times 0.10 =\\ 1\times 200 + 1534\times 0.10 =\\ 1\times 200 + 852\times 0.10 =\end{bmatrix} = \begin{bmatrix}\mathbf{410.4}\\ \mathbf{341.6}\\ \mathbf{353.4}\\ \mathbf{285.2}\end{bmatrix}$$

$$\begin{bmatrix}1 & 2104\\ 1 & 1416\\ 1 & 1534\\ 1 & 852\end{bmatrix} \times \begin{bmatrix}-150\\ 0.40\end{bmatrix} = \begin{bmatrix}1\times -150 + 2104\times 0.40 =\\ 1\times -150 + 1416\times 0.40 =\\ 1\times -150 + 1534\times 0.40 =\\ 1\times -150 + 852\times 0.40 =\end{bmatrix} = \begin{bmatrix}\mathbf{691.6}\\ \mathbf{416.4}\\ \mathbf{463.6}\\ \mathbf{190.8}\end{bmatrix}$$

matrix multiplication properties

multiplication is **not commutative** when working with the multiplication of matrices and scalars:

in multiplication of real numbers 3 x 5 is the same as multiplying 5 x 3; this is the property of being mathematically commutative

for matrices A and B; in general $A \times B \neq B \times A$ (not commutative):

$$\begin{bmatrix}\begin{bmatrix}1 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}0 & 0\\ 2 & 0\end{bmatrix} = \begin{bmatrix}\mathbf{2} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\end{bmatrix} \neq \begin{bmatrix}\begin{bmatrix}0 & 0\\ 2 & 0\end{bmatrix}\begin{bmatrix}1 & 1\\ 0 & 0\end{bmatrix} = \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{2} & \mathbf{2}\end{bmatrix}\end{bmatrix}$$

if A is a $m \times n$ and B is a $n \times m$ matrix; $A \times B$ is a $m \times m$ and $B \times A$ is an $n \times n$ matrix

multiplication of real numbers have an **associative** property and shares this property with the multiplication of matrices:

multiplying real number $3 \times (5 \times 2) = 3 \times 10 = (3 \times 5) \times 2 = 15 \times 2$ equally

for matrices A, B and C; the computation $A \times (B \times C) = (A \times B) \times C$ equally:

let $D = B \times C$; compute $A \times D \rightarrow A \times (B \times C)$

let $E = A \times B$; compute $E \times C \rightarrow (A \times B) \times C$] the results are **associative**

identity matrix

a matrix with the property of 1's along the diagonals and 0's everywhere else:

denoted I or $I_{n \times n}$ with examples as follows:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$I_{2\times2}$ $I_{3\times3}$ $I_{4\times4}$

for any matrix A, $A \times I = I \times A = A$

$m \times n$ $n \times n$ $m \times m$ $m \times n$ $m \times n$

although matrix matrix multiplication is generally **not commutative**, matrix matrix multiplication is **commutative** when the product involves an identity matrix: $AB \neq BA$ in general, however $AI = IA$ as described in the latter

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \times 1 + 0 \times 3 + 0 \times 2 = 1 \\ 0 \times 1 + 1 \times 3 + 0 \times 2 = 3 \\ 0 \times 1 + 0 \times 3 + 1 \times 2 = 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

inverse and transpose

$1 =$ "identity" $\qquad 3(3^{-1}) = \frac{1}{3} = 1 \qquad 12 \times (12^{-1}) = 1$

not all numbers have an inverse: $0(0^{-1}) =$ undefined

Matrix inverse:

If A is an m x m matrix, and if it has an inverse,

$$AA^{-1} = A^{-1}A = I.$$

an $m \times m$ matrix is also a square matrix (number of rows = number of columns)

$$\begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix} \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_{2\times2}$$

A A^{-1} $A^{-1}A$; computation of A^{-1} through `pinv(A)` in octave

matrices that do not have an inverse are "**singular**" or "**degenerate**" matrices

Matrix Transpose

Example: $A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}$ $A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$

Let A be an m x n matrix, and let $B = A^T$.
Then B is an n x m matrix, and

$$B_{ij} = A_{ji}.$$

for example: $B_{12} = A_{21} = 2$ and $B_{32} = A_{23} = 9$ e.g. $\begin{bmatrix} 0 & 3 \\ 1 & 4 \end{bmatrix}^T = \begin{bmatrix} 0 & 1 \\ 3 & 4 \end{bmatrix}$

linear regression with multiple variables

multivariate linear regression

multiple features

multiple variable definitive notation:

$m =$ number of training examples

$n =$ number of features

$x^{(i)} =$ 'input' features of i^{th} training example

$x_j^{(i)} =$ value of feature j in i^{th} training example

Size (feet)^2	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

In the training set above, what is $x_1^{(4)}$?

- The size (in feet^2) of the 1st home in the training set
- The age (in years) of the 1st home in the training set
- The size (in feet^2) of the 4th home in the training set

Correct Response

- The age (in years) of the 4th home in the training set

previously the hypothesis for singlevariate linear regression was denoted as:

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

the hypothesis for **multivariate linear regression** is denoted as:

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

for convenience of notation, define $x_0 = 1$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\begin{matrix} [\theta_0 & \theta_1 & \theta_2 & \cdots & \theta_n] \\ (n+1) \times 1 \text{ matrix} \end{matrix} \qquad \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

θ^T x

$h_\theta(x) = \theta_0 \boldsymbol{x_0} + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ which can in turn be simplified as

$= \theta^T x$

gradient descent for multiple variables

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

parameters: $\theta_0, \theta_1, \ldots, \theta_n$ → denoted simplay as θ; an $n+1$ dimensional vector

cost function:

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

gradient descent:

repeat {

$$\theta_j := \theta_j - a\frac{\partial}{\partial \theta_j}\left(\cancel{J(\theta_0, \theta_1, \ldots, \theta_n)} \quad J(\theta)\right)$$

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

When there are n features, we define the cost function as

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

For linear regression, which of the following are also equivalent and correct definitions of $J(\theta)$?

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

  Correct Response

- [x] $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\sum_{j=0}^{n}\theta_j x_j^{(i)}\right) - y^{(i)}\right)^2$ (Inner sum starts at 0)

  Correct Response

- [ ] $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\sum_{j=1}^{n}\theta_j x_j^{(i)}\right) - y^{(i)}\right)^2$ (Inner sum starts at 1)

  Correct Response

- [ ] $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\sum_{j=0}^{n}\theta_j x_j^{(i)}\right) - \left(\sum_{j=0}^{n} y_j^{(i)}\right)\right)^2$

  Correct Response

previously ($n = 1$):

repeat until convergence {

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

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

(update θ_0 and θ_1 simultaneously)

}

$$\frac{\partial}{\partial \theta_1}J(\theta_0, \theta_1)$$

new algorithm ($n \geq 1$):

repeat until convergence {

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

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

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

}

$$\frac{\partial}{\partial \theta_1}J(\theta)$$

gradient descent in practice i – feature scaling

it is important to ensure features used as an input with varying algorithms share similar scales. features with scales differing substantially can skew the model and cause gradient descent to run inefficient; causing many iteration to arrive at the global minimum through continuous iterations to converge

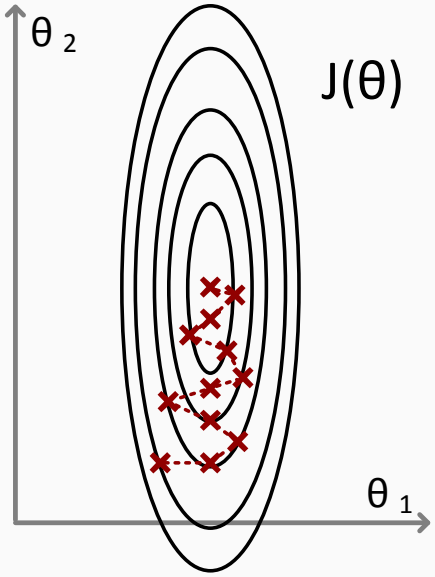
for example:

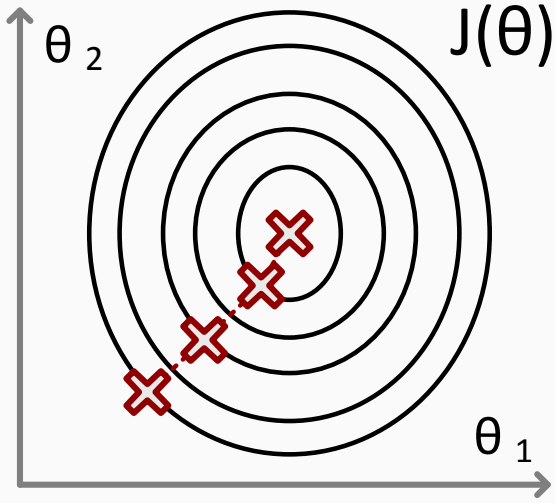
assume x_1 =variable in 10,000s

x_2 =variable in 10s

assume scaled to $x_1 = \frac{x_1}{10{,}000}$

$x_2 = \frac{x_2}{10}$





the general idea is to get each feature into approximately $-1 \leq x_i \leq 1$ range. the rule is subjective and interpretable depending upon the user and data. some data scientists find acceptable ranges as $-3 \leq x_i \leq 3$ or $-\frac{1}{3} \leq x_i \leq \frac{1}{3}$ for larger and smaller feature values, respectively

mean normalization

a prominent method of feature scaling is seen through mean normalization:

replace x_i with $x_i - \mu_i$ to make the features have an approximately zero mean (not applied to $x_0 = 1$):

$$x_i \leftarrow \frac{x_i - \mu_i}{\sigma_i}$$

μ_i = average value of x_i in the training set

σ_i = the standard deviation

Suppose you are using a learning algorithm to estimate the price of houses in a city. You want one of your features x_i to capture the age of the house. In your training set, all of your houses have an age between 30 and 50 years, with an average age of 38 years. Which of the following would you use as features, assuming you use feature scaling and mean normalization?

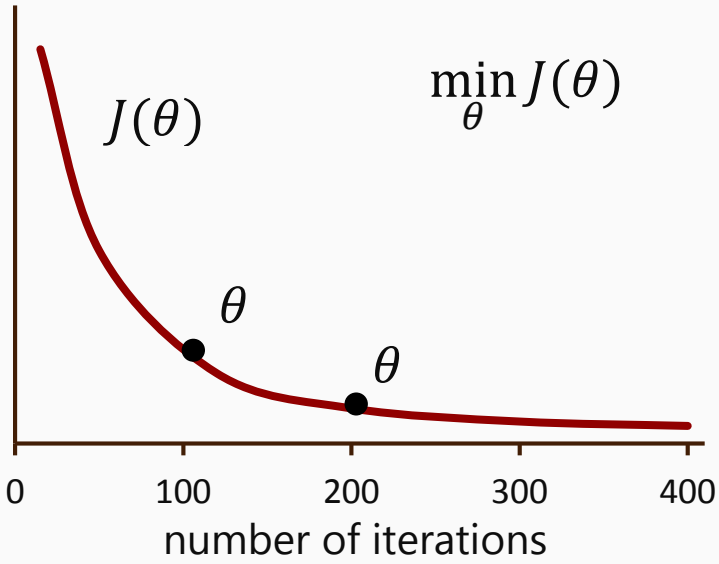
- $x_i = \text{age of house}$
- $x_i = \frac{\text{age of house}}{50}$
- $x_i = \frac{\text{age of house} - 38}{50}$
- $x_i = \frac{\text{age of house} - 38}{20}$

Correct Response

gradient descent in practice ii – learning rate

understanding the learning rate α and how it interacts with the gradient descent update rule $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ in addition to "debugging" gradient descent for proper function and choosing the value of learning rate α

to determine if gradient descent is working as expected:



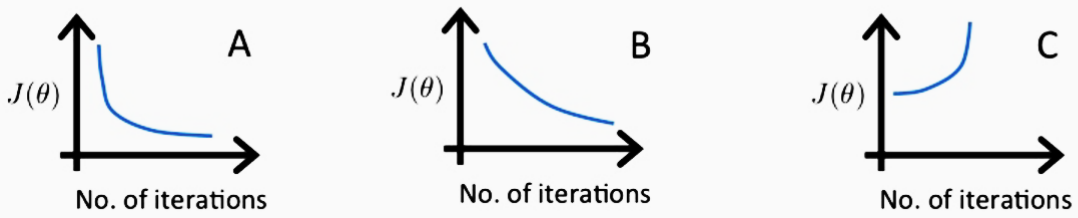
in plotting $J(\theta)$ against the number of gradient descent iterations, the expected behavior is for $J(\theta)$ to effectively decrease after each iteration until convergence

an additional option is to automatically test for convergence through declaring a logical rule; this is difficult (e.g. if $J(\theta)$ decreases by less than 10^{-3} in a single iteration)

gradient descent that experiences a positive slope when plotted against the number of iterations is indicative of divergence likely caused by the learning rate α being too large. a learning rate α too small can cause slow convergence with gradient descent

it is typically to choose α in increments (e.g. , … , 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, …)

Suppose a friend ran gradient descent three times, with $\alpha = 0.01$, $\alpha = 0.1$, and $\alpha = 1$, and got the following three plots (labeled A, B, and C):



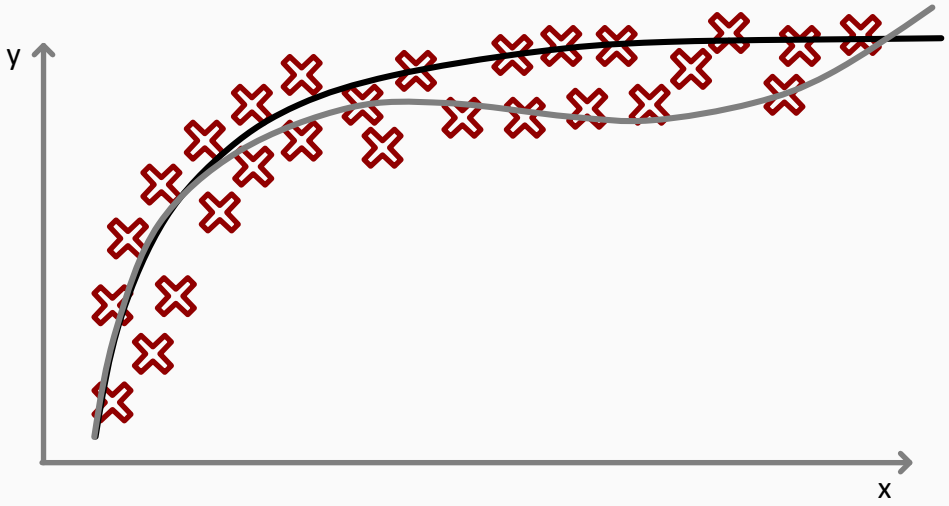
Which plots corresponds to which values of α?

- A is $\alpha = 0.01$, B is $\alpha = 0.1$, C is $\alpha = 1$.
- A is $\alpha = 0.1$, B is $\alpha = 0.01$, C is $\alpha = 1$.

Correct Response
In graph C, the cost function is increasing, so the learning rate is set too high. Both graphs A and B converge to an optimum of the cost function, but graph B does so very slowly, so its learning rate is set too low. Graph A lies between the two.

- A is $\alpha = 1$, B is $\alpha = 0.01$, C is $\alpha = 0.1$.
- A is $\alpha = 1$, B is $\alpha = 0.1$, C is $\alpha = 0.01$.

features and polynomial regression



$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

the use of polynomial expressions require a greater attention to feature scaling for the effects quadratic equations will have on feature values

Suppose you want to predict a house's price as a function of its size. Your model is

$h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2\sqrt{(\text{size})}$

Suppose size ranges from 1 to 1000 (feet^2). You will implement this by fitting a model

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

Finally, suppose you want to use feature scaling (without mean normalization).

Which of the following choices for x_1 and x_2 should you use? (Note: $\sqrt{1000} \approx 32$.)

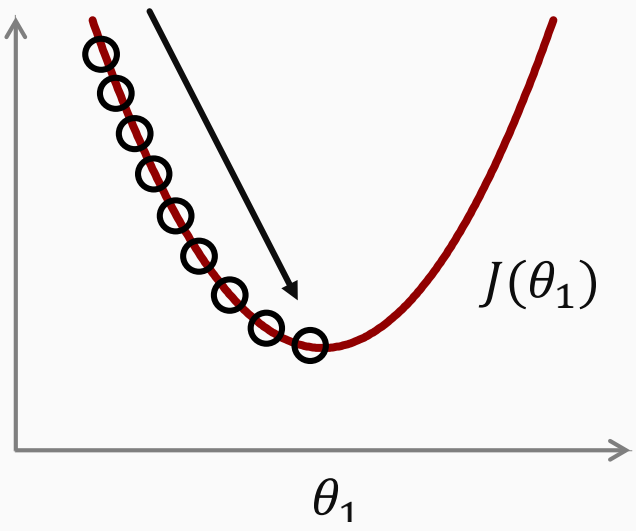
- $x_1 = \text{size},\ x_2 = 32\sqrt{(\text{size})}$
- $x_1 = 32(\text{size}),\ x_2 = \sqrt{(\text{size})}$
- $x_1 = \frac{\text{size}}{1000},\ x_2 = \frac{\sqrt{(\text{size})}}{32}$

Correct Response

computing parameters ⁘ analytically

normal equation

previously, gradient descent was applied in a fashion to take iterative steps in the direction of the global minimum of a training data set by minimizing cost function $J(\theta)$ (illustrated to the right)



alternatively, the normal equation offers a measure to solve for θ analytically without taking steps through a reiterative process algorithm

intuition if 1D ($\theta \in \mathbb{R}$):

$J(\theta) = a\theta^2 + b\theta + c$

they way to minimize cost functions in calculus is to take derivatives and set to 0

$\frac{\partial}{\partial\theta_1}J(\theta) = \cdots \text{set} = 0$ and solve for θ in the case where θ is a real number $\theta \in \mathbb{R}$

in the case where theta θ is a vector $n + 1$;

$\theta \in \mathbb{R}^{n+1}$ $\quad J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

$\frac{\partial}{\partial\theta_1}J(\theta) = \cdots \text{set} = 0$ $\quad$ (for every j) $\quad \rightarrow \quad$ solve for $\theta_0, \theta_1, \dots, \theta_n$

a running example for implementation guidance of the normal equation

training examples: $m = 4$

x_0	size (feet^2) x_1	number bedrooms x_2	number floors x_3	home age (years) x_4	price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

the initial data will all be converted into a matrix with the predicted values as a vector; then setting theta θ to X transposed X inverse times X transposed y will compute the value of theta θ that minimizes the cost function

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \qquad \theta = (X^T X)^{-1} X^T y$$

$$\mathbb{R}^{m \times (n+1)} \qquad \mathbb{R}^m$$

in more general terms:

with $\boldsymbol{m}$ **examples** $(x^1, y^1), \ldots, (x^m, y^m)$; and $\boldsymbol{n}$ **features**

each training example x^i will represent a 1-dimensional vector:

$x^i = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$ with matrix X designed as: $X = \begin{bmatrix} - - & \left(x^{(1)}\right)^T & - - \\ - - & \left(x^{(2)}\right)^T & - - \\ - - & \left(x^{(3)}\right)^T & - - \\ - - & \vdots & - - \\ - - & \left(x^{(m)}\right)^T & - - \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$

e.g. if $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix} \rightarrow X = \begin{bmatrix} 1 & x_1^{(i)} \\ 1 & x_2^{(i)} \\ 1 & \vdots \\ 1 & x_m^{(i)} \end{bmatrix}$ and $y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$; then solve $\theta = (X^T X)^{-1} X^T y$

Suppose you have the training in the table below:

age (x_1)	height in cm (x_2)	weight in kg (y)
4	89	16
9	124	28
5	103	20

You would like to predict a child's weight as a function of his age and height with the model

$\text{weight} = \theta_0 + \theta_1 \text{age} + \theta_2 \text{height}$

What are X and y?

- $X = \begin{bmatrix} 4 & 89 \\ 9 & 124 \\ 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$
- $X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 1 & 16 \\ 1 & 28 \\ 1 & 20 \end{bmatrix}$
- $X = \begin{bmatrix} 4 & 89 & 1 \\ 9 & 124 & 1 \\ 5 & 103 & 1 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$
- $X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$

Correct Response

to reiterate the intuition behind the normal equation: $\theta = (X^T X)^{-1} X^T y$

$(X^T X)^{-1}$ is simply the inverse of matrix $X^T X$

set $A = X^T X$ then $A^{-1} = (X^T X)^{-1}$

in **octave** programming language, the above is denoted as: `pinv(X'*X)*X'*y`

feature scaling is **not** necessary if the normal equation method is used. however, gradient descent still has value in feature scaling.

differentiation between gradient descent and the normal equations when a dataset is with $\boldsymbol{m}$ **examples** $(x^1, y^1), \ldots, (x^m, y^m)$; and $\boldsymbol{n}$ **features**:

gradient descent	**normal equation**
need to choose learning rate α	no need to choose learning rate α
requires multiple iterations	requires no multiple iterations
	need to compute $(X^T X)^{-1}$
works well when n is large	works slowly when n is very large (>~10,000)

in summary, so long as the number of features $= < 1{,}000$ or so, the normal equation method is appropriate to use as opposed to a full scale gradient descent.

normal equation noninvertibility

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

what if $X^T X$ is non-invertible? (singular/degenerate)

this is a rare occurrence, however, `pinv(X'*X)*X'*y` in octave will still solve for θ

causes of $X^T X$ being non-invertible are often:

redundant features (linearly dependent)

e.g. $x_1 = \text{size in feet}^2$

$x_2 = \text{size in meters}^2$

too many features (e.g. $m \leq n$)

either delete insignificant features or use regularization

matlab octave : tutorial

basic operations

```
%% Change Octave prompt
PS1('>> ');
%% Change working directory in windows example:
cd 'c:/path/to/desired/directory name'
%% Note that it uses normal slashes and does not use escape
characters for the empty spaces.

%% elementary operations
5+6
3-2
5*8
1/2
2^6
1 == 2  % false
1 ~= 2  % true.  note, not "!="
1 && 0
1 || 0
xor(1,0)


%% variable assignment
a = 3; % semicolon suppresses output
b = 'hi';
c = 3>=1;

% Displaying them:
a = pi
disp(a)
disp(sprintf('2 decimals: %0.2f', a))
disp(sprintf('6 decimals: %0.6f', a))
format long
a
format short
a


%%  vectors and matrices
A = [1 2; 3 4; 5 6]

v = [1 2 3]
```

```
v = [1; 2; 3]
v = [1:0.1:2]  % from 1 to 2, with stepsize of 0.1. Useful for
plot axes
v = 1:6        % from 1 to 6, assumes stepsize of 1 (row vector)

C = 2*ones(2,3)  % same as C = [2 2 2; 2 2 2]
w = ones(1,3)    % 1x3 vector of ones
w = zeros(1,3)
w = rand(1,3)  % drawn from a uniform distribution
w = randn(1,3) % drawn from a normal distribution (mean=0,
var=1)
w = -6 + sqrt(10)*(randn(1,10000));  % (mean = -6, var = 10) -
note: add the semicolon
hist(w)     % plot histogram using 10 bins (default)
hist(w,50)  % plot histogram using 50 bins
% note: if hist() crashes, try "graphics_toolkit('gnu_plot')"

I = eye(4)    % 4x4 identity matrix

% help function
help eye
help rand
help help
```

moving data around

```
%% dimensions
sz = size(A) % 1x2 matrix: [(number of rows) (number of
columns)]
size(A,1)  % number of rows
size(A,2)  % number of cols
length(v)  % size of longest dimension


%% loading data
pwd    % show current directory (current path)
cd 'C:\Users\ang\Octave files'   % change directory
ls     % list files in current directory
load q1y.dat    % alternatively, load('q1y.dat')
load q1x.dat
who    % list variables in workspace
whos   % list variables in workspace (detailed view)
clear q1y       % clear command without any args clears all vars
v = q1x(1:10);  % first 10 elements of q1x (counts down the
columns)
save hello.mat v;   % save variable v into file hello.mat
save hello.txt v -ascii; % save as ascii
% fopen, fread, fprintf, fscanf also work  [[not needed in
class]]

%% indexing
A(3,2)  % indexing is (row,col)
A(2,:)  % get the 2nd row.
        % ":" means every element along that dimension
A(:,2)  % get the 2nd col
A([1 3],:) % print all  the elements of rows 1 and 3

A(:,2) = [10; 11; 12]     % change second column
A = [A, [100; 101; 102]]; % append column vec
A(:) % Select all elements as a column vector.

% Putting data together
A = [1 2; 3 4; 5 6]
B = [11 12; 13 14; 15 16] % same dims as A
C = [A B]  % concatenating A and B matrices side by side
C = [A, B] % concatenating A and B matrices side by side
C = [A; B] % Concatenating A and B top and bottom
```

computing on data

```
%% initialize variables
A = [1 2;3 4;5 6]
B = [11 12;13 14;15 16]
C = [1 1;2 2]
v = [1;2;3]
%% matrix operations
A * C  % matrix multiplication
A .* B % element-wise multiplication
% A .* C  or A * B gives error - wrong dimensions
A .^ 2 % element-wise square of each element in A
1./v   % element-wise reciprocal
log(v)  % functions like this operate element-wise on vecs or
matrices
exp(v)
abs(v)

-v  % -1*v

v + ones(length(v), 1)
% v + 1  % same

A'  % matrix transpose

%% misc useful functions
% max  (or min)
a = [1 15 2 0.5]
val = max(a)
[val,ind] = max(a) % val -  maximum element of the vector a and
index - index value where maximum occur
val = max(A) % if A is matrix, returns max from each column

% compare values in a matrix & find
a < 3 % checks which values in a are less than 3
find(a < 3) % gives location of elements less than 3
A = magic(3) % generates a magic matrix - not much used in ML
algorithms
[r,c] = find(A>=7)  % row, column indices for values matching
comparison
% sum, prod
sum(a)
prod(a)
floor(a) % or ceil(a)
```

```
max(rand(3),rand(3))
max(A,[],1) -  maximum along columns(defaults to columns -
max(A,[]))
max(A,[],2) - maximum along rows
A = magic(9)
sum(A,1)
sum(A,2)
sum(sum( A .* eye(9) ))
sum(sum( A .* flipud(eye(9)) ))
% Matrix inverse (pseudo-inverse)
pinv(A)         % inv(A'*A)*A'
```

plotting data

```
%% plotting
t = [0:0.01:0.98];
y1 = sin(2*pi*4*t);
plot(t,y1);
y2 = cos(2*pi*4*t);
hold on;  % "hold off" to turn off
plot(t,y2,'r');
xlabel('time');
ylabel('value');
legend('sin','cos');
title('my plot');
print -dpng 'myPlot.png'
close;            % or,  "close all" to close all figs
figure(1); plot(t, y1);
figure(2); plot(t, y2);
figure(2), clf;  % can specify the figure number
subplot(1,2,1);  % Divide plot into 1x2 grid, access 1st element
plot(t,y1);
subplot(1,2,2);  % Divide plot into 1x2 grid, access 2nd element
plot(t,y2);
axis([0.5 1 -1 1]);  % change axis scale
%% display a matrix (or image)
figure;
imagesc(magic(15)), colorbar, colormap gray;
% comma-chaining function calls.
a=1,b=2,c=3
a=1;b=2;c=3;
```

control statements: for, while, if statement

```
v = zeros(10,1);
for i=1:10,
    v(i) = 2^i;
end;

% Can also use "break" and "continue" inside for and while loops
to control execution.

i = 1;
while i <= 5,
  v(i) = 100;
  i = i+1;
end

i = 1;
while true,
  v(i) = 999;
  i = i+1;
  if i == 6,
    break;
  end;
end

if v(1)==1,
  disp('The value is one!');
elseif v(1)==2,
  disp('The value is two!');
else
  disp('The value is not one or two!');
end
```

functions

to create a function, type the function code in a text editor (e.g. gedit or notepad), and save the file as "functionname.m"

example function:

```
function y = squareThisNumber(x)

y = x^2;
```

to call the function in octave, either:

1) navigate to the directory of the functionname.m file and call the function:

```
    % Navigate to directory:
    cd /path/to/function

    % Call the function:
    functionName(args)
```

2) add the directory of the function to the load path and save it:
should not use addpath/savepath for any of the assignments in this course. instead use 'cd' to change the current working directory.

```
    % To add the path for the current session of Octave:
    addpath('/path/to/function/')

    % To remember the path for future sessions of Octave, after
executing addpath above, also do:
    savepath
```

octave's functions can return more than one value:

```
    function [y1, y2] = squareandCubeThisNo(x)
    y1 = x^2
    y2 = x^3
```

call the above function this way:

```
   [a,b] = squareandCubeThisNo(x)
```

vectorization

vectorization is the process of taking code that relies on **loops** and converting it into **matrix operations**. it is more efficient, more elegant, and more concise.

as an example, compute the prediction from a hypothesis. theta is the vector of fields for the hypothesis and x is a vector of variables.

with loops:

```
prediction = 0.0;
for j = 1:n+1,
  prediction += theta(j) * x(j);
end;
```

with vectorization:

```
prediction = theta' * x;
```

recalling the definition multiplying vectors, this one operation does the element-wise multiplication and overall sum in a concise notation.

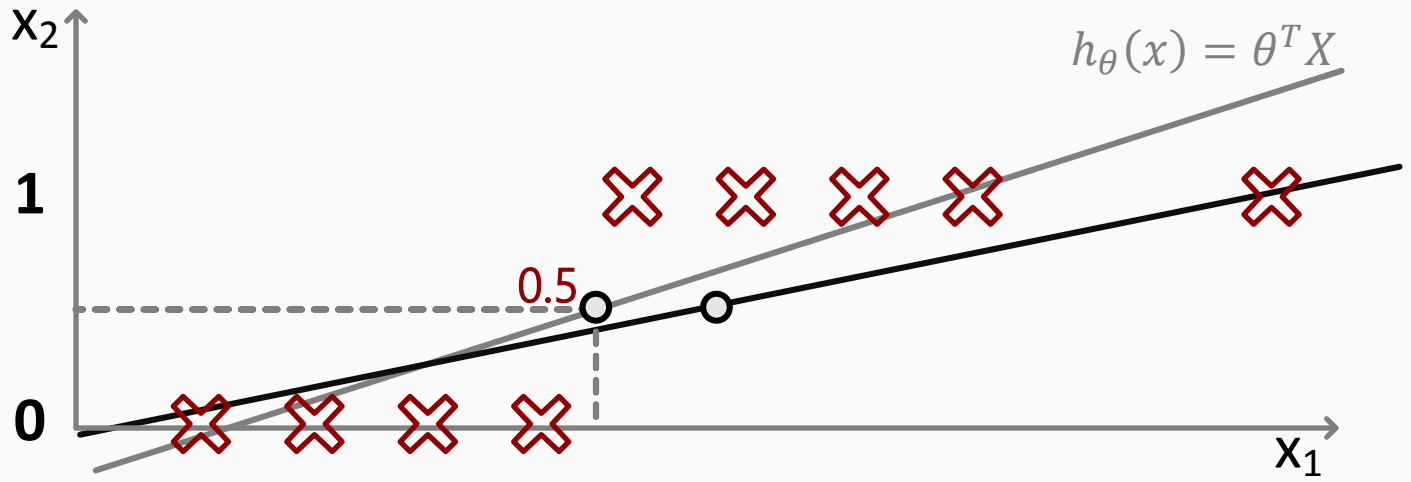
logistic regression basics

classification and representation

classification

the assignments in a single class classification problem are typically as follows:

$$y \in \{0,1\} \quad \rightarrow \quad \begin{matrix} 0: \text{"negative class"} \\ 1: \text{"positive class"} \end{matrix}$$



although circumstances could allow linear regression to predict a proper output for the predicted values, Applying linear regression to classification problems is generally not effective; it does not fit outliers and will result in false negatives and positives:

threshold classifier output $h_\theta(x)$ at 0.5:

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

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

Which of the following statements is true?

- If linear regression doesn't work on a classification task as in the previous example shown in the video, applying feature scaling may help.
- If the training set satisfies $0 \leq y^{(i)} \leq 1$ for every training example $(x^{(i)}, y^{(i)})$, then linear regression's prediction will also satisfy $0 \leq h_\theta(x) \leq 1$ for all values of x.
- If there is a feature x that perfectly predicts y, i.e. if $y = 1$ when $x \geq c$ and $y = 0$ whenever $x < c$ (for some constant c), then linear regression will obtain zero classification error.
- None of the above statements are true.

Correct Response

additionally, linear regression often experiences values where $h_\theta(x)$ can predict values > 1 or < 0; the latter is outside of the classifications of $y = 0$ or $y = 0$

hypothesis representation

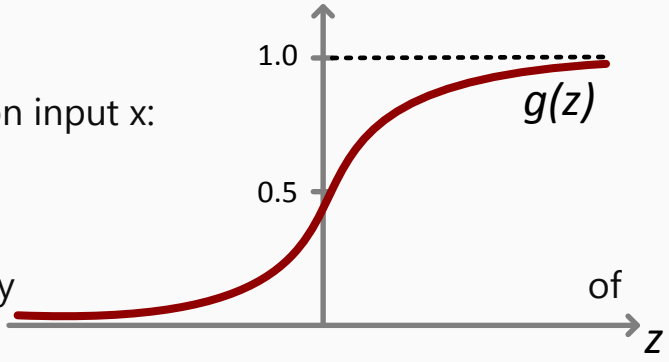
the logistic regression models requires outputs within the range $0 \leq h_\theta(x) \leq 1$

therefore, the linear regression function is altered: $h_\theta(x) = \theta^T X \rightarrow g(\theta^T X)$

intuition expansion: $h_\theta(x) = g(\theta^T X) \qquad g(z) = \frac{1}{1+e^{-z}}$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T X}} \rightarrow \text{sigmoid/logistic function}$$

interpretation of the hypothesis' output



$h_\theta(x) =$ the estimated probability that $y = 1$ on input x:

if $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{some measured value} \end{bmatrix}$

and $h_\theta(x) = 0.7$ then there is a 70% probability of the measured value is the **positive class** indicated in the problem set

$h_\theta(x) = P(y = 1|x; \theta)$; "probability that $y = 1$ given x, parameterized by θ"

concretely, because logistic regression can only return outputs of $y = 0$ and $y = 1$:

$$P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$$

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

Suppose we want to predict, from data x about a tumor, whether it is malignant ($y = 1$) or benign ($y = 0$). Our logistic regression classifier outputs, for a specific tumor, $h_\theta(x) = P(y = 1|x; \theta) = 0.7$, so we estimate that there is a 70% chance of this tumor being malignant. What should be our estimate for $P(y = 0|x; \theta)$, the probability the tumor is benign?

- $P(y = 0|x; \theta) = 0.3$

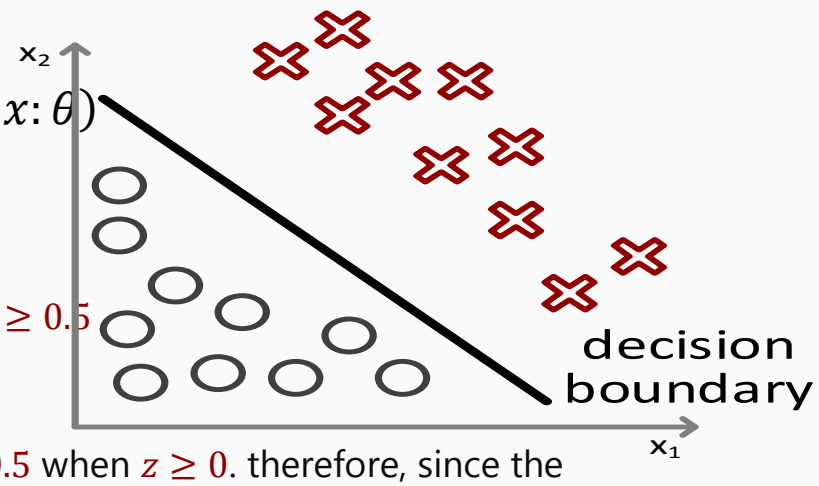
  Correct Response

- $P(y = 0|x; \theta) = 0.7$
- $P(y = 0|x; \theta) = 0.7^2$
- $P(y = 0|x; \theta) = 0.3 \times 0.7$

decision boundary

$$h_\theta(x) = g(\theta^T X) = P(y = 1|x; \theta)$$

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



assuming a prediction of "$y = 1$" if $h_\theta(x) \geq 0.5$

and a prediction of "$y = 0$" if $h_\theta(x) < 0.5$

looking at the sigmoid function; $g(z) \geq 0.5$ when $z \geq 0$. therefore, since the hypothesis for logistic regression is $h_\theta(x) = g(\theta^T X)$, then $h_\theta(x) = g(\theta^T X) \geq 0.5$, whenever $\theta^T X \geq 0$ because $\theta^T X$ effectively taken on the value of z in the sigmoid function $g(z)$

conversely, when $h_\theta(x) < 0.5$ then $g(z) \leq 0.5$ considering that $h_\theta(x) = g(\theta^T X)$ illustrated above. Therefore when $h_\theta(x) = g(\theta^T X) < 0.5$, whenever $\theta^T X < 0$ because $\theta^T X$ effectively taken on the value of z in the sigmoid function $g(z)$

determination of the decision boundary example

given the function $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ with the following parameters:

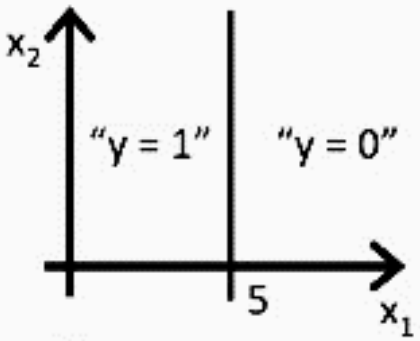
$\theta_0 = -3, \theta_1 = 1, \theta_2 = 1$ produces the parameter vector $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$

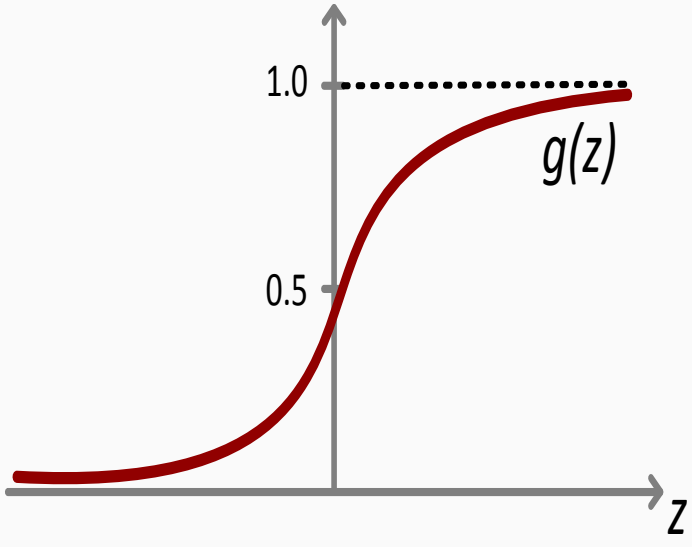
referring to the above formulas, "$y = 1$" will be predicted if $\theta^T X = -3 + x_1 + x_2 \geq 0$

any example of (x_a, x_2) that satisfies the equation $-3 + x_1 + x_2 \geq 0$ will predict "$y = 1$"

additional rule notation is as follows: $(-3 + x_1 + x_2 \geq 0) = (x_1 + x_2 \geq 3)$

Consider logistic regression with two features x_1 and x_2. Suppose $\theta_0 = 5, \theta_1 = -1, \theta_2 = 0$, so t hat $h_\theta(x) = g(5 - x_1)$. Which of these shows the decision boundary of $h_\theta(x)$?





nonlinear decision boundaries

adding additional higher order polynomial terms can adapt to fitting nonlinear datasets

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

determination of the decision boundary example

given the function $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$; with the parameters:

$\theta_0 = -1, \theta_1 = 0, \theta_2 = 0, \theta_3 = 1, \theta_4 = 1$ produces the parameter vector $\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

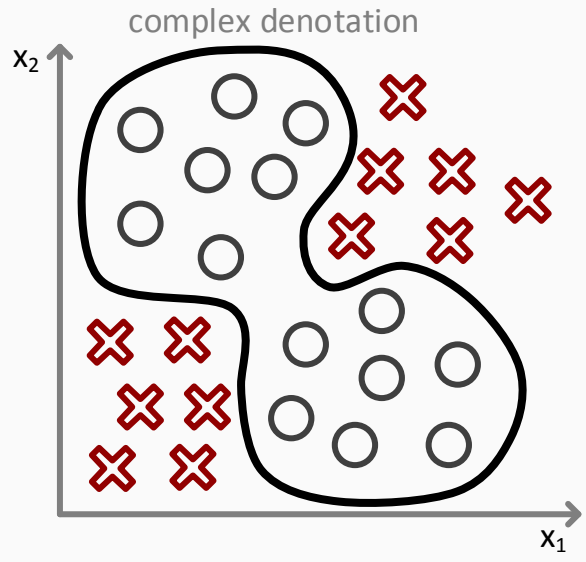
with the above formulas, "$y = 1$" will be predicted if $\theta^T X = -1 + x_1^2 + x_2^2 \geq 0$

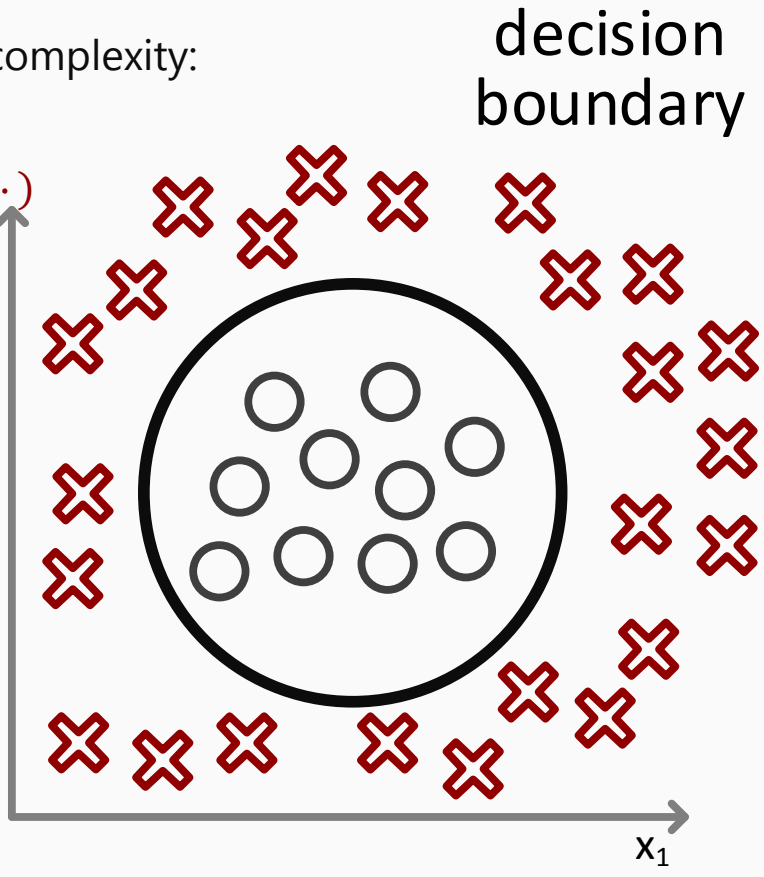
simplified rule notation is as follows: $(-1 + x_1^2 + x_2^2 \geq 0) = (x_1^2 + x_2^2 \geq 1)$

adding more complex polynomial features allows the algorithm to fit more complex decision boundaries. an important principal of logistic regression is that the **decision boundary** is a property **not** of the **training set**, but of the **hypothesis** under the parameters. therefore, as long as parameter vector θ is known, it will define the decision boundary. the training set does not define the decision boundary but can be used to fit the parameters θ

finally, the function determines the complexity:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \cdots)$$





logistic regression model

cost function

given a training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

with m examples denoted as $x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$; where $x_0 = 1, y \in \{0,1\}$

and a hypothesis: $h_\theta(x) = \frac{1}{1+e^{-\theta^T X}}$

the parameters θ can be determined as follows:

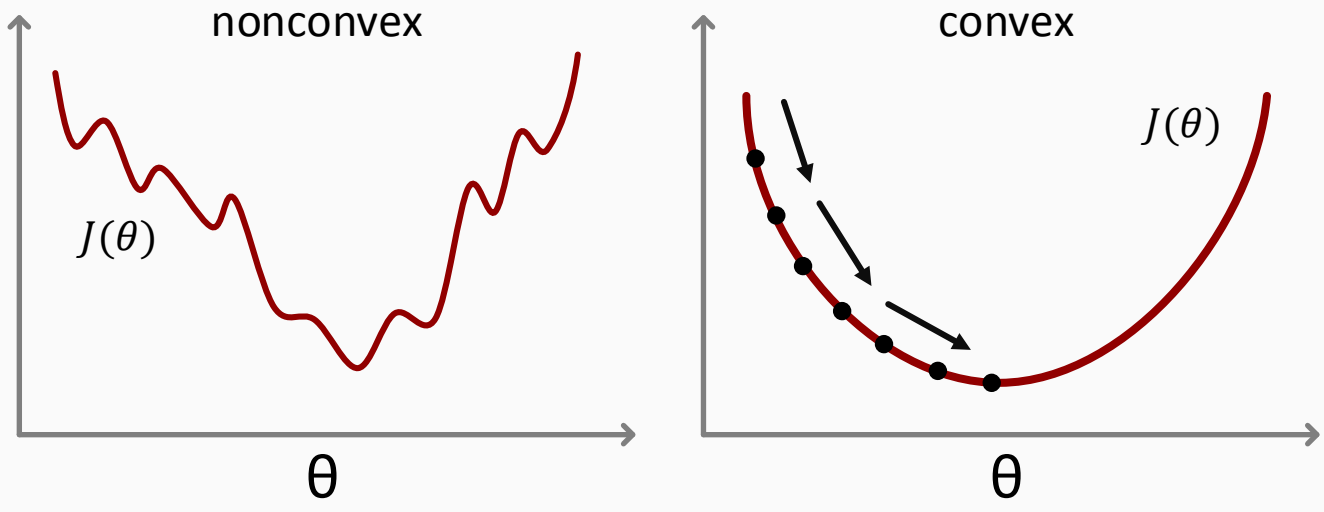
linear regression:

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

$$\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

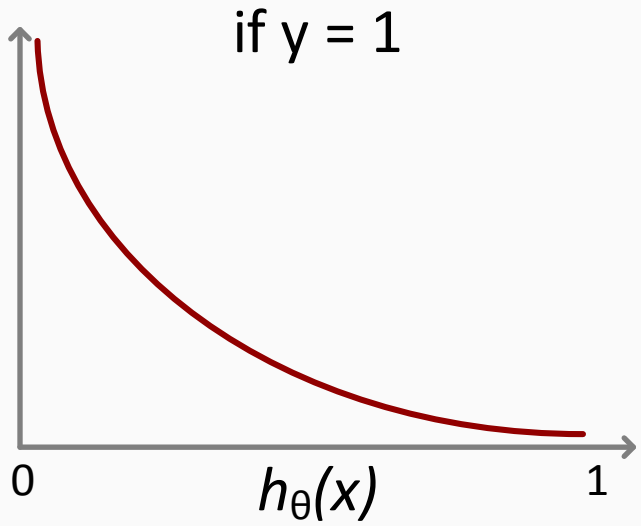
the simplified cost functions serves purpose as to measure the cost the training set will have to pay when $h_\theta\left(x^{(i)}\right)$ is predicted in terms of actual $y^{(i)}$: being $\frac{1}{2}$ the squared error

the Linear Regression Cost Function cannot be applied to logistic regression as it would not create a Convex Function; Gradient Descent would fail to find the global minimum:



instead, logistic regression: $\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(h_\theta\left(x^{(i)}\right)\right) & \text{if } y = 1 \\ -\log\left(1 - h_\theta\left(x^{(i)}\right)\right) & \text{if } y = 0 \end{cases}$

the cost function is illustrated in each class as follows:

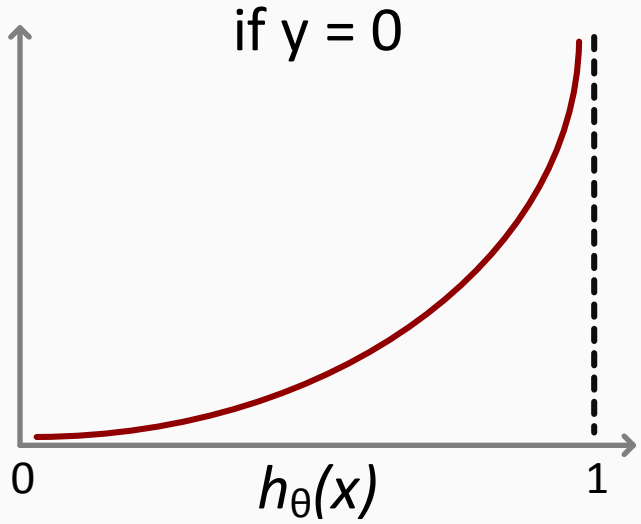


cost $= 0$ if $y = 1, h_\theta(x) = 1$

but as $h_\theta(x) \to 0$

cost $\to \infty$

the illustration (left) captures the intuition if $h_\theta(x^{(i)}) = 0$, (predict $P(y = 1|x;\theta) = 0$), but $y = 1$, the learning algorithm will be heavily penalized with a significant measured cost



cost $= 0$ if $y = 0, h_\theta(x) = 0$

but as $h_\theta(x) \to 1$

cost $\to \infty$

the illustration (left) captures the intuition if $h_\theta(x^{(i)}) = 1$, (predict $P(y = 0|x;\theta) = 0$), but $y = 0$, the learning algorithm will be heavily penalized with a significant measured cost

In logistic regression, the cost function for our hypothesis outputting (predicting) $h_\theta(x)$ on a training example that has label $y \in \{0,1\}$ is:

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

Which of the following are true? Check all that apply.

- [x] If $h_\theta(x) = y$, then $\text{cost}(h_\theta(x), y) = 0$ (for $y = 0$ and $y = 1$).

Correct Response

- [x] If $y = 0$, then $\text{cost}(h_\theta(x), y) \to \infty$ as $h_\theta(x) \to 1$.

Correct Response

- [ ] If $y = 0$, then $\text{cost}(h_\theta(x), y) \to \infty$ as $h_\theta(x) \to 0$.

Correct Response

- [x] Regardless of whether $y = 0$ or $y = 1$, if $h_\theta(x) = 0.5$, then $\text{cost}(h_\theta(x), y) > 0$.

Correct Response

simplified cost function and gradient descent

logistic regression:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

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

note: $y = 0$ or $y = 1$ always

explanation of simplifying the cost function $\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$ above:

$$\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -y\log\left(h_\theta(x)\right) - (1-y)\log\left(1 - h_\theta(x)\right)$$

if **y=1**: $\text{cost}(h_\theta(x), y) = -\log\left(h_\theta(x)\right)$

if **y=0**: $\text{cost}(h_\theta(x), y) = -\log\left(1 - h_\theta(x)\right)$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

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

principle of likelihood maximization derived function (statistical rational behind cost function used)

to fit parameters θ:

the process of minimizing $J(\theta)$; $\min_\theta J(\theta)$ will provide the parameters θ

to make a prediction given a new x training example:

output $h_\theta(x) = \frac{1}{1+e^{-\theta^T X}}$; $P(y = 1 | x; \theta)$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

updating all values of parameter θ is best performed as a vectorized

in order to minimize $J(\theta)$; $\min_\theta J(\theta)$ (gradient descent)

repeat until convergence {

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

$$\rightarrow \left(\text{simultaneously update all } \theta_j\right)\}$$

note the identical cosmetic similarity between the logistic and linear regression update rule is attributed to the altered definition of $h_\theta(x)$ from $h_\theta(x) = \theta^T X$ to $h_\theta(x) = \frac{1}{1+e^{-\theta^T X}}$

Suppose you are running gradient descent to fit a logistic regression model with parameter $\theta \in \mathbb{R}^{n+1}$. Which of the following is a reasonable way to make sure the learning rate α is set properly and that gradient descent is running correctly?

- Plot $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ as a function of the number of iterations (i.e. the horizontal axis is the iteration number) and make sure $J(\theta)$ is decreasing on every iteration.
- Plot $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$ as a function of the number of iterations and make sure $J(\theta)$ is decreasing on every iteration.

Correct Response

- Plot $J(\theta)$ as a function of θ and make sure it is decreasing on every iteration.
- Plot $J(\theta)$ as a function of θ and make sure it is convex.

One iteration of gradient descent simultaneously performs these updates:

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

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

$\vdots$

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

We would like a vectorized implementation of the form $\theta := \theta - \alpha\delta$(for some vector $\delta \in \mathbb{R}^{n+1}$).

What should the vectorized implementation be?

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

Correct Response

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

advanced optimization

optimization algorithm

cost function $J(\theta)$ wanting to $\min_\theta J(\theta)$

given θ, code is written to compute

$J(\theta)$ and $a\frac{\partial}{\partial\theta_j}J(\theta)$

(for $j = 0, 1, \dots, n$)

gradient descent:

repeat {

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

}

optimization algorithms options:

- gradient descent
- conjugate gradient
- bfgs
- l-bfgs

advantages:

- unnecessary to select α manually
- often faster than gradient descent

disadvantages:

- generally more complex

optimization through example:

goal to $\min_{\theta} J(\theta)$ assuming $\theta_1 = 5,\ \theta_2 = 5$:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = 2(\theta_1 - 5)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = 2(\theta_2 - 5)$$

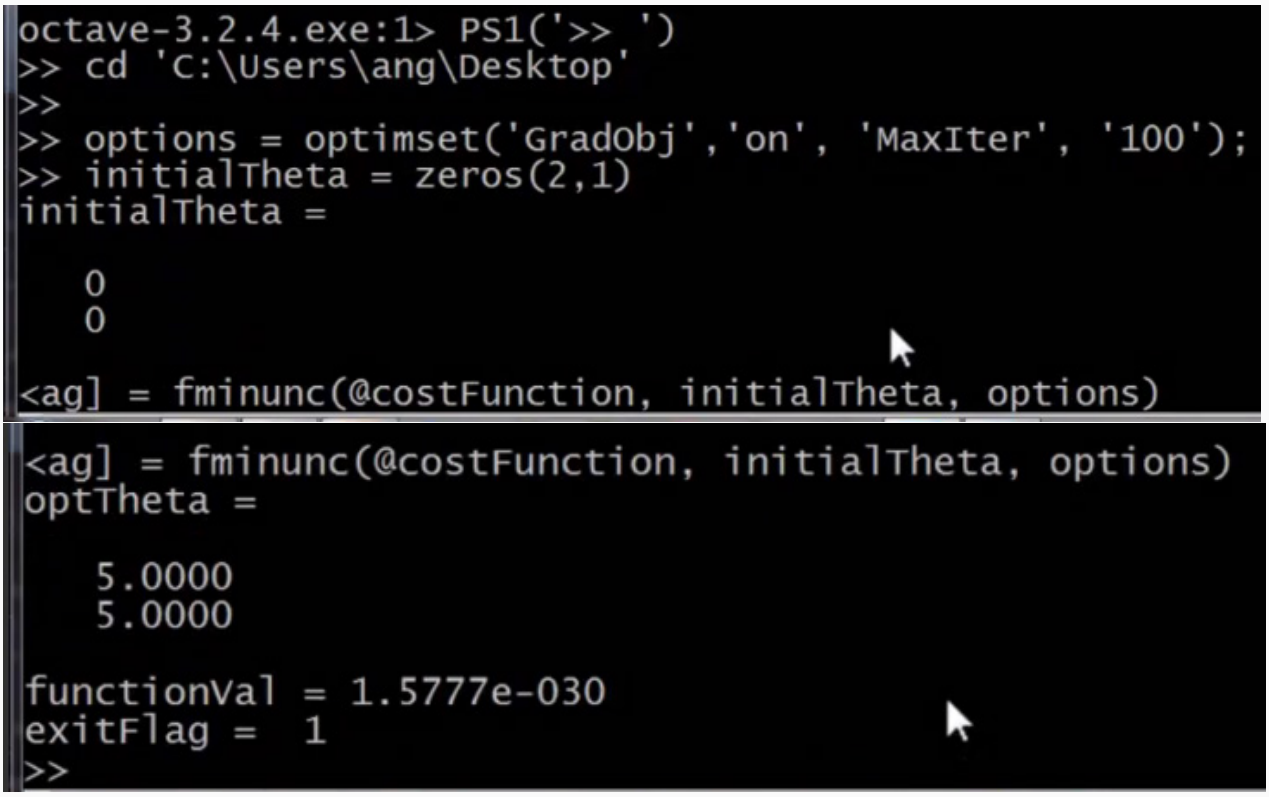
optimization in octave:

```
function [jVal, gradient]
          = costFunction(theta)
  jVal = (theta(1)-5)^2 + ...
        (theta(2)-5)^2
  gradient = zeros(2,1)
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5);
```

after implementing the cost function, call the advanced optimization function in octave:

```
options = optimset('GrabObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag]...
       = fminunc(@costFunction, initialTheta, options);
```

`'GrabObj'` in reference to gradient objective set to `'on'` in reference to the fact that a gradient is going to be provided to the algorithm. `initialTheta` initializes the parameters $\boldsymbol{\theta}$ are at **0**. the advanced function `fminunc` is called to compute `optTheta`, the learning rate $\boldsymbol{\alpha}$ autonomously:



application of algorithm optimization to logistic regression

$$\texttt{theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \begin{matrix} \leftarrow \texttt{theta(1)} \\ \leftarrow \texttt{theta(2)} \\ \vdots \\ \leftarrow \texttt{theta(n)} \end{matrix}$$

note the definition of theta being indexed at 1 in octave as opposed to 0 in the

```
function [jVal, gradient] = costFunction(theta)
    jVal =          [code to compute J(θ)];
    gradient(1) =   [code to compute ∂/∂θ_0 J(θ)];
    gradient(2) =   [code to compute ∂/∂θ_1 J(θ)];
    ⋮
    gradient(n+1) = [code to compute ∂/∂θ_n J(θ)];
```

the above code requires a user to input code to compute both a cost function and relative gradients

Suppose you want to use an advanced optimization algorithm to minimize the cost function for logistic regression with parameters θ_0 and θ_1. You write the following code:

```
function [jVal, gradient] = costFunction(theta)
  jVal = % code to compute J(theta)
  gradient(1) = CODE#1 % derivative for theta_0
  gradient(2) = CODE#2 % derivative for theta_1
```

What should CODE#1 and CODE#2 above compute?

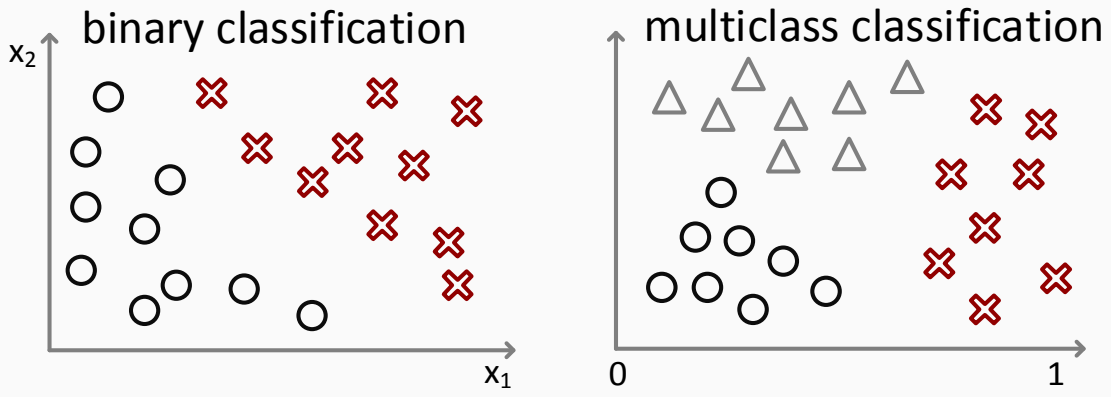
- CODE#1 and CODE#2 should compute $J(\theta)$.
- CODE#1 should be theta(1) and CODE#2 should be theta(2).
- CODE#1 should compute $\frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}] (= \frac{\partial}{\partial\theta_0} J(\theta))$ and CODE#2 should compute $\frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}] (= \frac{\partial}{\partial\theta_1} J(\theta))$

Correct Response

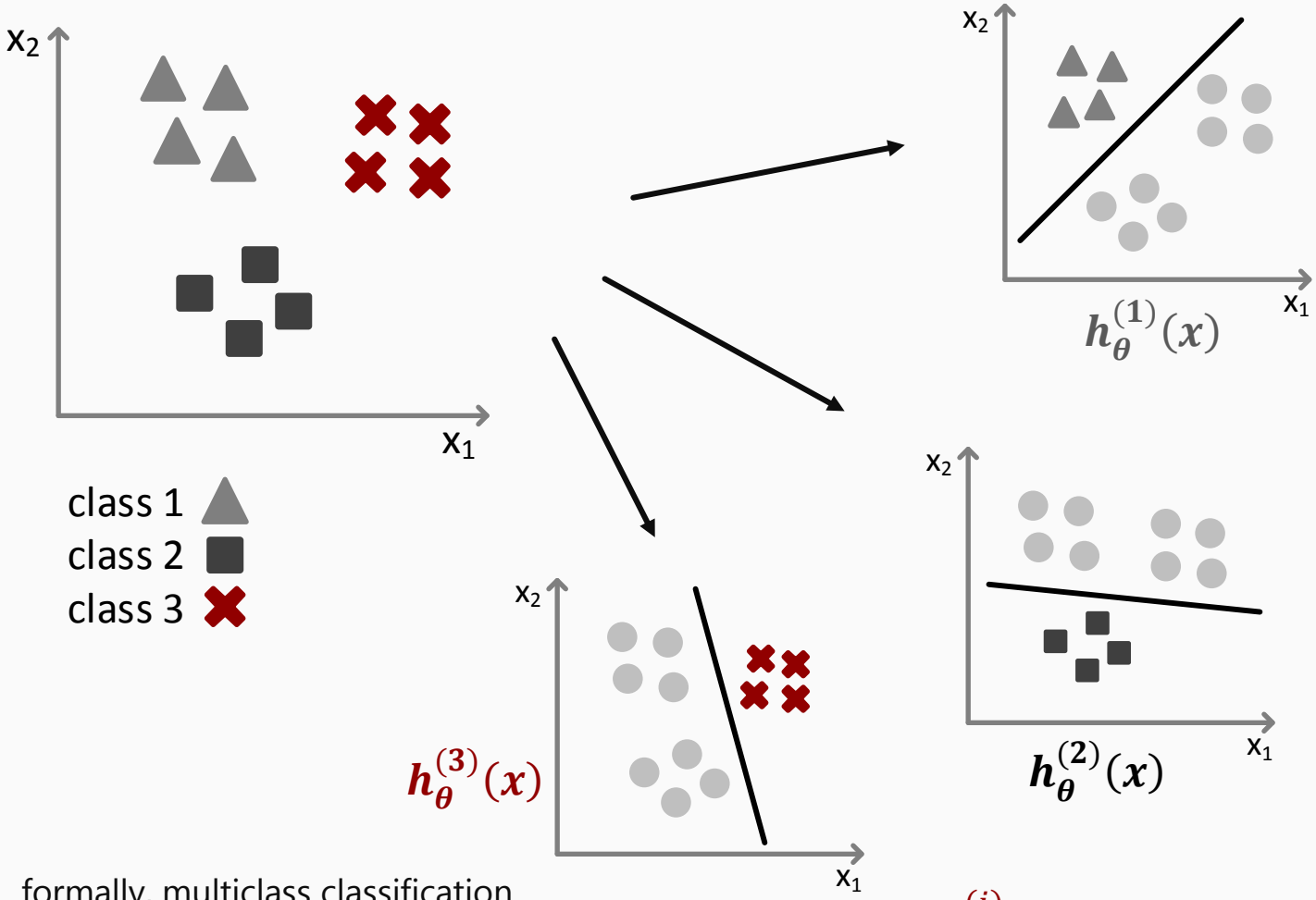
multiclass classification

multiclass classification: one-vs-all

when classification problems have more than a binary classification of 0 or 1



this is possible by separating examples into individual binary classification problems



formally, multiclass classification trains a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i that maximizes $\max_i h_\theta^{(i)}(x)$

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

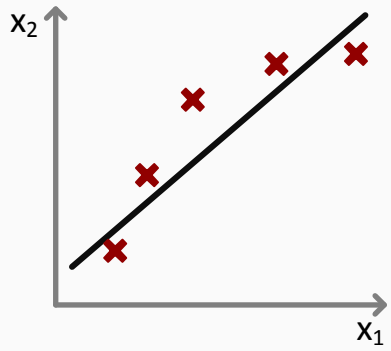
where $(i = 1,2,3)$

regularization λ optimization

solving the problem of overfitting

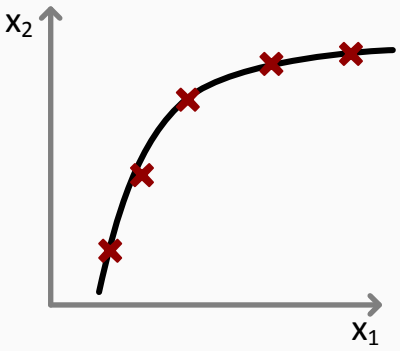
the problem of overfitting

linear regression:



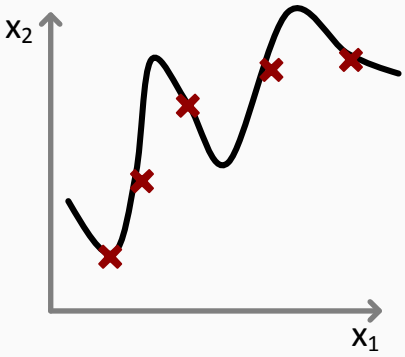
$\theta_0 + \theta_1 x$

"underfitted" or "highly biased"



$\theta_0 + \theta_1 x + \theta_2 x^2$

"ideal"



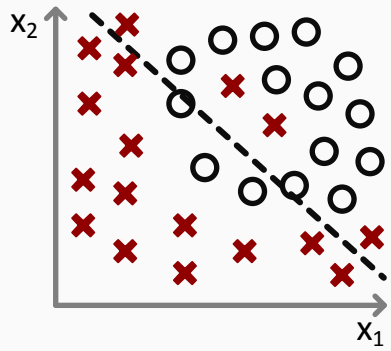
$\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots$
$\theta_3 x^3 + \theta_4 x^4$

"overfitted" or "high variance"

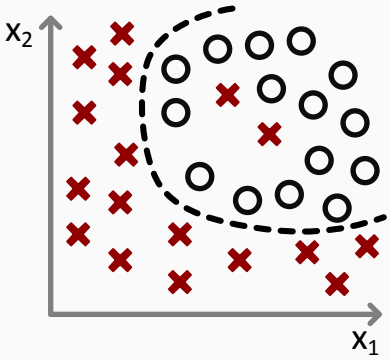
overfitting:

too many features might cause the learned hypothesis to fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \approx 0$), but fail to generalize new examples (prediction)

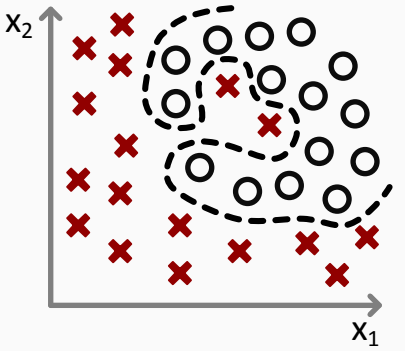
logistic regression:



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



$h_\theta(x) = g\begin{pmatrix}\theta_0 + \theta_1 x_1 + \cdots \\ \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2\end{pmatrix}$



$h_\theta(x) = g\begin{pmatrix}\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots \\ \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_4 x_2^2 + \cdots\end{pmatrix}$

(g = sigmoid function)

"underfitted" or "highly biased"

"ideal"

"overfitted" or "high variance"

measures to reduce overfitting:

- reduce the number of features
  - manually select features to retain
  - model selection algorithm (principal component analysis)
- regularization
  - retain all features, but reduce the magnitude/ values of parameters θ_j
  - works well when there are many features with predictive value of y

Consider the medical diagnosis problem of classifying tumors as malignant or benign. If a hypothesis $h_\theta(x)$ has overfit the training set, it means that:

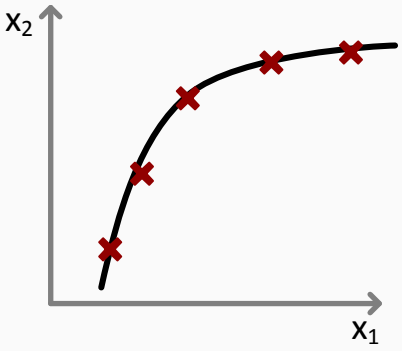
- It makes accurate predictions for examples in the training set and generalizes well to make accurate predictions on new, previously unseen examples.
- It does not make accurate predictions for examples in the training set, but it does generalize well to make accurate predictions on new, previously unseen examples.
- It makes accurate predictions for examples in the training set, but it does not generalize well to make accurate predictions on new, previously unseen examples.

Correct Response

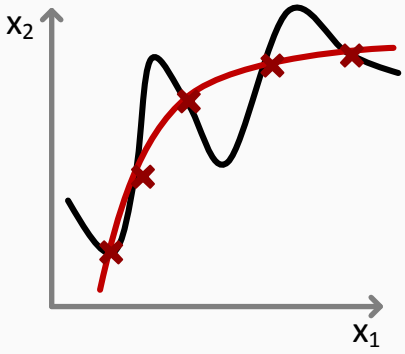
- It does not make accurate predictions for examples in the training set and does not generalize well to make accurate predictions on new, previously unseen examples.

cost function

intuition of the cost function:



$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$

previous examples illustrated how an over parameterized function is prone to overfitting the data as seen in the illustrations above. the quadratic function fits the data ideally; while the high order polynomial fits the training set well but will fail to generalize to new examples.

consider penalizing the parameters θ_3, θ_4 by making their weights insignificant:

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

assigning large values to θ_3, θ_4 in the cost function will force the values of $\theta_3, \theta_4 \approx 0$

this essentially removes the effect of higher order polynomials from above.

regularization

the general idea behind regularization is if there are small values for the parameters $\theta_1, \theta_2, \dots, \theta_n$ will lead to a model with:

a more simple hypothesis with smoother functions
a model that is less prone to overfitting

a example with housing price predictions:

features: $x_1, x_2, \dots, x_{100}$

parameters: $\theta_0, \theta_1, \theta_2, \dots, \theta_{100}$

it is not directly known which parameters will have an overbearing effect on the cost functions in a negative facet. Therefore, the cost function will be extended with a regularization parameter to compensate:

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

regularization parameter
regularization term

the additional regularization term will penalize all parameters equally; by convention, the indexing begins at $i = 1$ because θ_0 will not be penalized for being large.

lambda λ controls a cost function trade off of two goals as follows:

1. fit the training data well → captured by the least squares term
2. maintain small parameters → captured by the regularization term

In regularized linear regression, we choose θ to minimize:

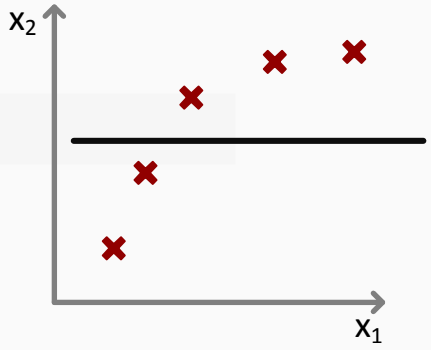
$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

What if λ is set to an extremely large value (perhaps too large for our problem, say $\lambda = 10^{10}$)?

- Algorithm works fine; setting λ to be very large can't hurt it.
- Algorithm fails to eliminate overfitting.
- Algorithm results in underfitting (fails to fit even the training set).

Correct Response

- Gradient descent will fail to converge.



regularized linear regression

the optimization objective for regularized linear regression:

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

$$\min_\theta J(\theta)$$

gradient descent

repeat until convergence {

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

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

(update θ_j for $j = 0, 1, 2, 3 \ldots, n$ simultaneously)

}

alternative notation for θ_j update: $\theta_j := \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

Suppose you are doing gradient descent on a training set of $m > 0$ examples, using a fairly small learning rate $\alpha > 0$ and some regularization parameter $\lambda > 0$. Consider the update rule:

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

Which of the following statements about the term $\left(1 - \alpha\frac{\lambda}{m}\right)$ must be true?

- $1 - \alpha\frac{\lambda}{m} > 1$
- $1 - \alpha\frac{\lambda}{m} = 1$
- $1 - \alpha\frac{\lambda}{m} < 1$

Correct Response

- None of the above.

normal equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \in \mathbb{R}^{m \times (n+1)} \qquad y = \begin{bmatrix} y^1 \\ \vdots \\ y^m \end{bmatrix} \in \mathbb{R}^m$$

$$\min_\theta J(\theta)$$

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

where the identity matrix $\in \mathbb{R}^{(n+1)\times(n+1)}$

noninvertibility

suppose $m \leq n$ (examples less than or equal to the features)

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

the matrix will not be invertible or degenerate

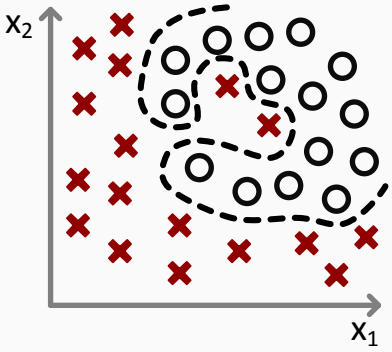
regularization does not have the same issue

if $\lambda > 0$,

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

The matrix is invertible

regularized logistic regression

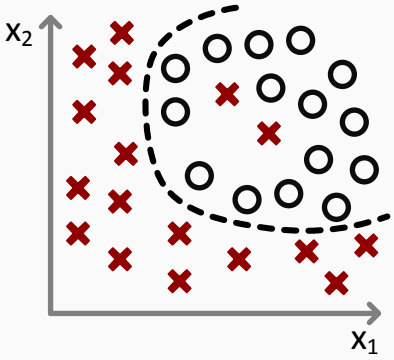


over parameterizing the sigmoid function can lead to equally overfitted models that do not generalize well:

$$h_\theta(x) = g\begin{pmatrix}\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots \\ \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_4 x_2^2 + \cdots\end{pmatrix}$$

logistic regression cost function:

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



by adding a regularization term, the model can penalize all parameters $\theta_1, \theta_2, \ldots, \theta_n$

$$+\frac{\lambda}{2m}\sum_{i=1}^{u}\theta_j^2$$

the regularized implementation is as follows:

gradient descent

repeat until convergence {

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

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

(update θ_j for $j = 1, 2, 3 \ldots, n$ simultaneously)

}

although the implementation for logistic regression is identical in appearance to linear regression; note that the hypothesis $h_\theta(x)$ is defined differently as follows:

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

When using regularized logistic regression, which of these is the best way to monitor whether gradient descent is working correctly?

- Plot $-[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$ as a function of the number of iterations and make sure it's decreasing.
- Plot $-[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1 - h_\theta(x^{(i)}))] - \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$ as a function of the number of iterations and make sure it's decreasing.
- Plot $-[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$ as a function of the number of iterations and make sure it's decreasing.

Correct Response

advanced optimization

$$\texttt{theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \begin{matrix} \leftarrow \texttt{theta(1)} \\ \leftarrow \texttt{theta(2)} \\ \vdots \\ \leftarrow \texttt{theta(n)} \end{matrix}$$

note the definition of theta being indexed at 1 in octave as opposed to 0 in the

```
function [jVal, gradient] = costFunction(theta)
```

`jVal =` [code to compute $J(\theta)$];

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

`gradient(1) =` [code to compute $\frac{\partial}{\partial \theta_0} J(\theta)$];

$$\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_0^{(i)}$$

`gradient(2) =` [code to compute $\frac{\partial}{\partial \theta_1} J(\theta)$];

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

`gradient(2) =` [code to compute $\frac{\partial}{\partial \theta_1} J(\theta)$];

$$\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_1^{(i)} + \frac{\lambda}{m}\theta_2$$

⋮

`gradient(n+1) =` [code to compute $\frac{\partial}{\partial \theta_n} J(\theta)$];

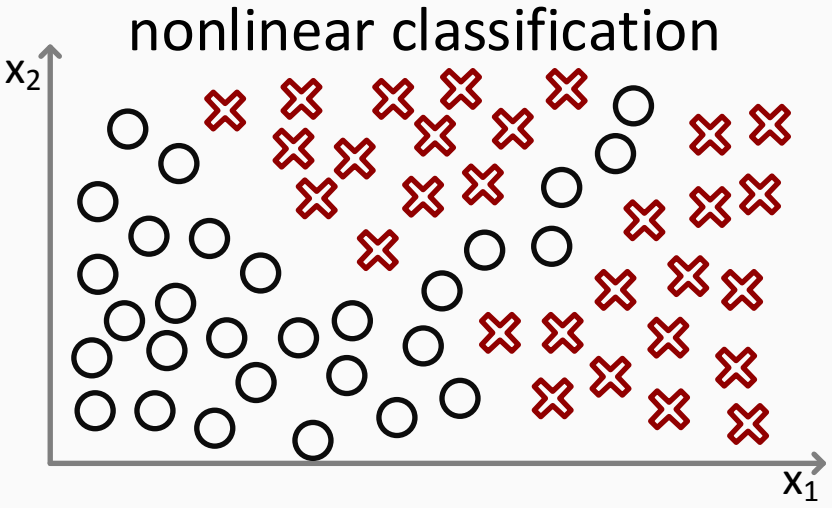
note the lack of a regularization term in the code required to compute `gradient(1)`; the parameter θ_0 is not regularized as a form of convention. The parameter θ_0 is in reference to variable `gradient(1)` in octave because indexing in octave begins at 0.

artificial neural networks representation

motivations

nonlinear hypothesis

with only two features as illustrated below, a traditional logistic regression classification method would suffice to fit the data accurately:



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

assuming multiple features like housing data below, if $n = 100$, there would be ≈ 5000 features; this number grows exponentially with each additional feature.

the quadratic function above becomes computational features:

$x_1 =$ size
$x_2 =$ # bedrooms
$x_3 =$ # floors
$x_4 =$ age
$\dots$
x_{100}

an option would be to take a subset of the features, but this would not be capable of predicting more intricate decision boundaries outside of basic ellipses etc. an additional option would be to use multi polynomial features but this would also result in very large amounts of required feature space.

image recognition purposes are a function of this algorithm

this method uses pixel intensities as a function of n. (50 x 50 pixel images → 2500 pixels n = 2500) (n = 7500 if using rgb)

with quadratic features ($x_i * x_j$): $\approx$ 3 million features

Suppose you are learning to recognize cars from 100×100 pixel images (grayscale, not RGB). Let the features be pixel intensity values. If you train logistic regression including all the quadratic terms ($x_i x_j$) as features, about how many features will you have?

50 million (5×10^7)

neurons and the brain

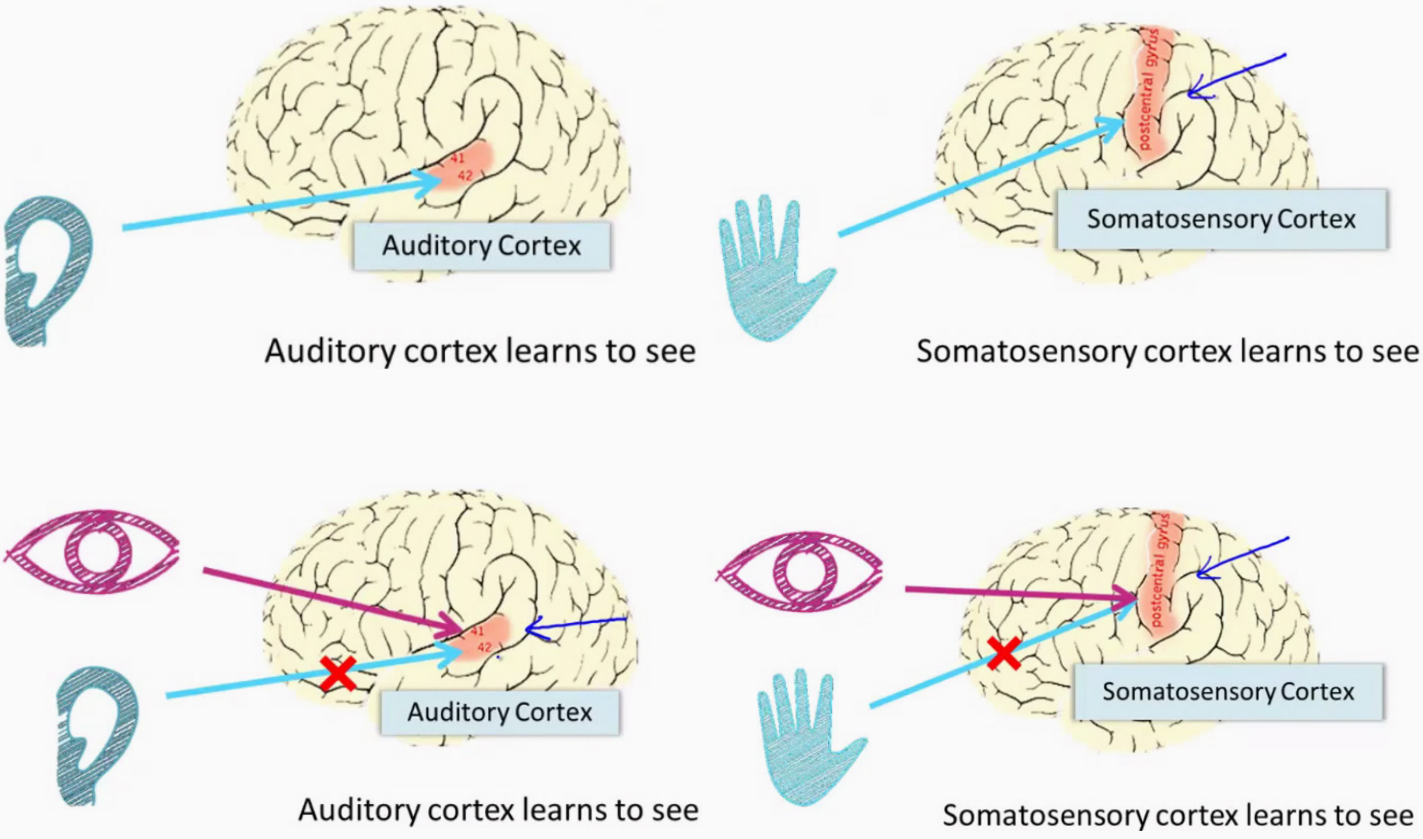
neural networks originated from algorithms that attempt to emulate the human brain. neural networks were used widely in the 80s and 90s; popularity diminished in the late 1990s.

a recent resurgence into artificial intelligence has curated state-of-the-art techniques for many applications with massive success.

the theory of the brain operating of neuron activity claims the lack of many learning algorithms to perform complete function.

the "one learning algorithm hypothesis"

the same brain tissues can process any senses necessary; areas of the brain can repurpose themselves.

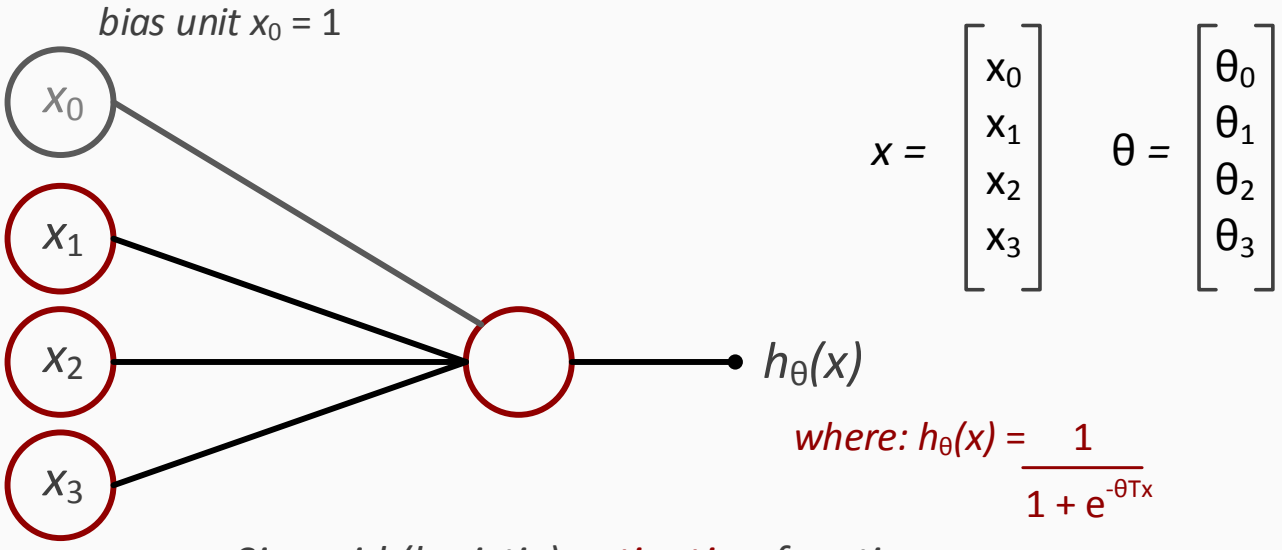


neural networks

model representations i

In artificial neural network, neurons in the brain are modeled as simple logistic unit:

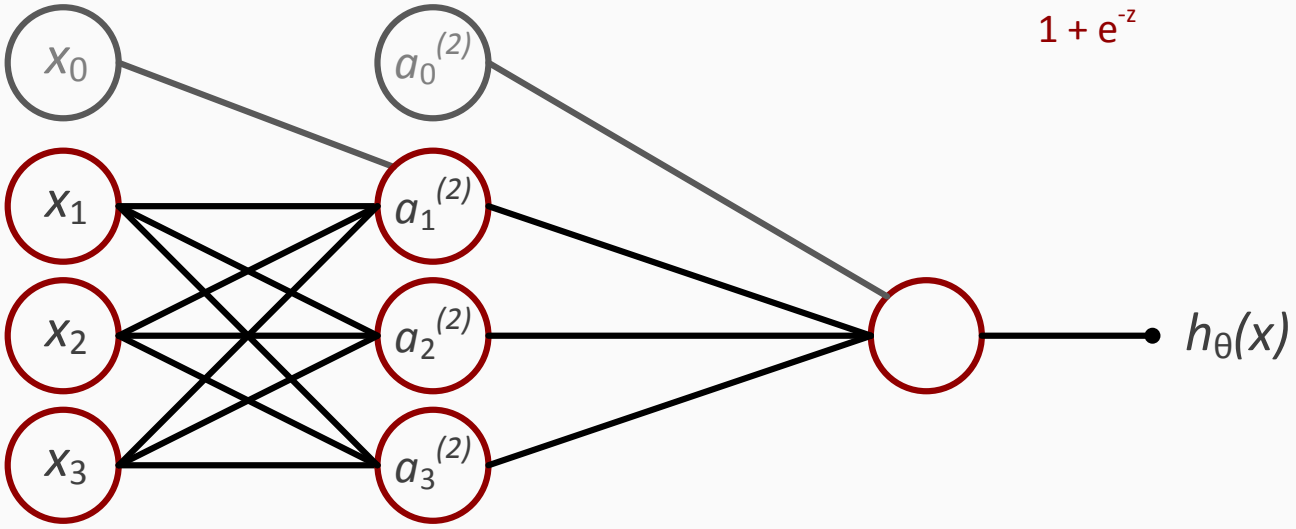
single neuron model: logistic unit



Sigmoid (logistic) activation function

neural network

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



layer 1
input layer

layer 2
hidden layer

layer 3
output layer

computational method for neural network

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

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

$$a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right)$$

$$a_3^{(2)} = g\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right)$$

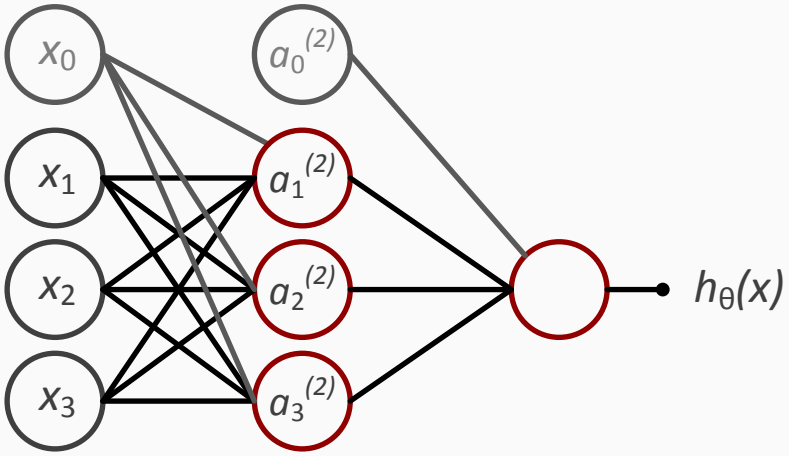
$$h_\theta(x) = g\left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right)$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\theta^{(j)}$ will be a matrix with a dimension of $s_{i+1} \times (s_i + 1)$

model representations ii

The superscripts of the variables represent the values associated to the neural network layer.

Forward propagation: Vectorized implementation



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = g\left(\theta^{(1)}a^{(1)}\right)$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

To compensate for the bias unit:

Add $a_0^{(2)} = 1$

$$z^{(3)} = \theta^{(2)}a^{(2)}$$

$$h_\theta(x) = a^{(3)} = g\left(z^{(3)}\right)$$

$$a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) \rightarrow a_1^{(2)} = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right) \rightarrow a_2^{(2)} = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right) \rightarrow a_3^{(2)} = g\left(z_2^{(2)}\right)$$

$$h_\theta(x) = g\left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right)$$

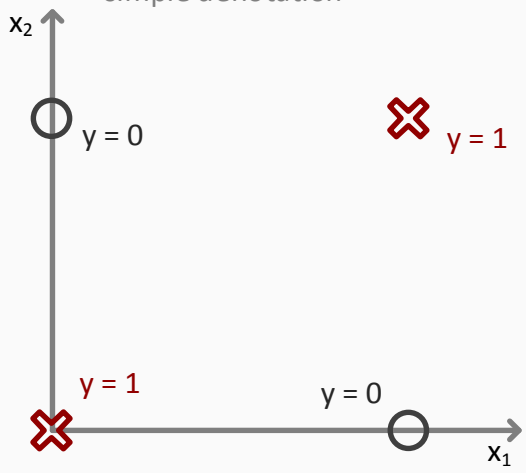
application

examples and intuitions i

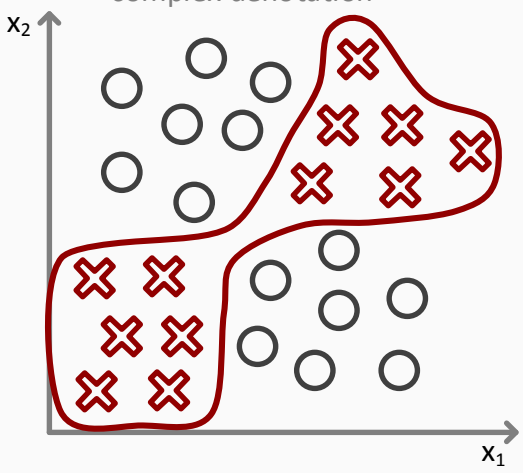
nonlinear classification: XOR/XNOR

x_1, x_2 are binary (0 or 1)

simple denotation



complex denotation

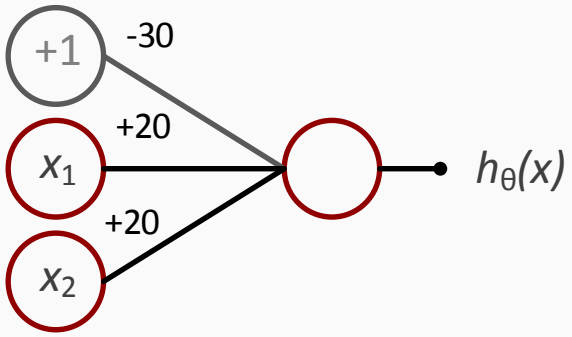


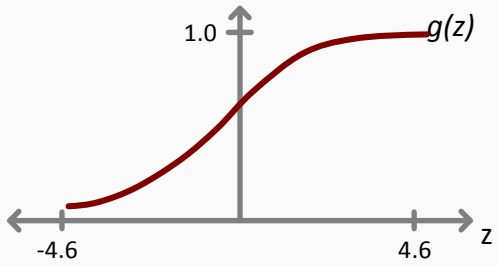
$y = x_1$ XOR x_2 *(true only if exactly 1 of x_1 **or** x_2 = 1)*

x_1 XNOR x_2] same

NOT (x_1 XOR x_2) *(true if x_1 **nor** x_2 = 1)*

example: AND





x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

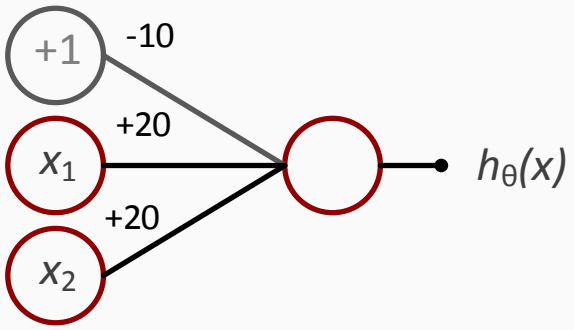
$$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$$

The sigmoid function graphed above denotes *g(z)*

The example neural network computation with the **AND** logical above and the corresponding calculations in the table prove the logical **AND** function. The function is true

when $h_\theta(x) \approx x_1$ AND x_2 . This can be seen in the results above when the computations of $h_\theta(x) = g(z)$ output a true value when both x_1 and x_2 return the values of true.

example: OR



$h_\theta(x) = g(-10 + 20x_1 + 20x_2)$

x_1	x_2	$h_\theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(10) \approx 1$

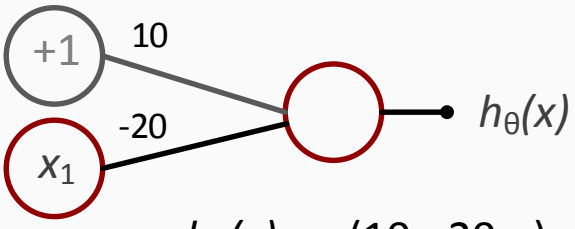
The example neural network computation with the **OR** logical above and the corresponding calculations in the table prove the logical **OR** function. The function is true when $h_\theta(x) \approx x_1$ OR x_2 . This can be seen in the results above when the computations of $h_\theta(x) = g(z)$ output a true value when either x_1 or x_2 return the values of true.

examples and intuitions ii

Previous examples explored the ways to compute nonlinear hypothesis with neural nets works to compute the **AND** and **OR** function. Additionally, **NOT** can be computed equally:

x_1 AND x_2 $\qquad$ x_1 OR x_2

negation: $\qquad$ NOT x_1



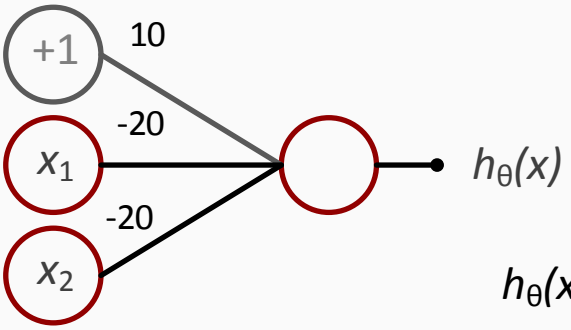
x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

$h_\theta(x) = g(10 - 20x_1)$

The general idea of negation (NOT) is when a large negative weights (-20 in this example) are attached to the variables (x_1 in this example); the result is negation as illustrated. Additionally, in computing the function of (NOT x_1) AND (NOT x_2), negative weights will be attached to both x_1 *and* x_2.

(NOT x_1) AND (NOT x_2) = 1 ; if and only if $x_1 = x_2 = 0$

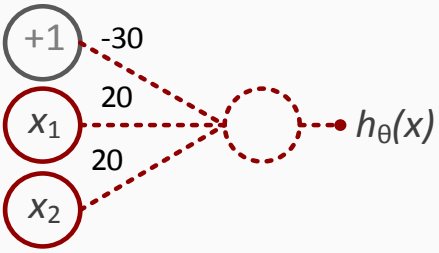
example: (NOT x_1) AND (NOT x_2)



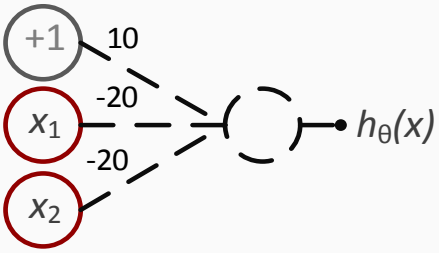
x_1	x_2	$h_\theta(x)$
0	0	$g(10) \approx 1$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(-30) \approx 0$

$$h_\theta(x) = g(10 - 20x_1 - 20x_2)$$

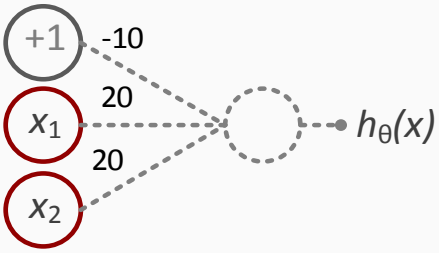
combining the networks: x_1 XNOR x_2



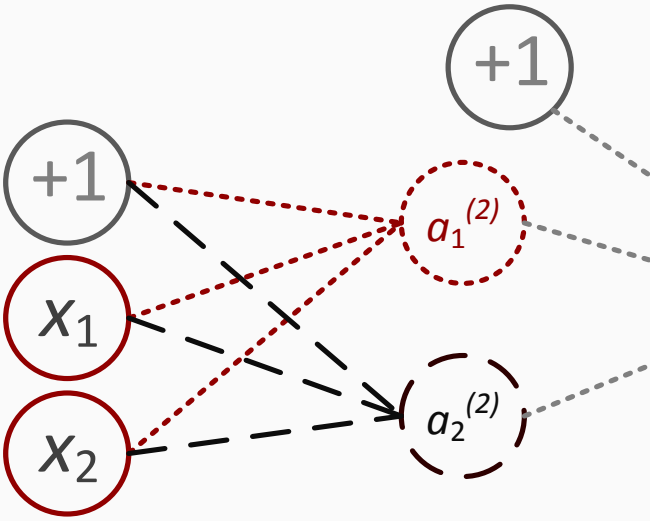
x_1 AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2



$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$

$h_\theta(x) = g(10 - 20x_1 - 20x_2)$

$h_\theta(x) = g(-10 + 20x_1 + 20x_2)$

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

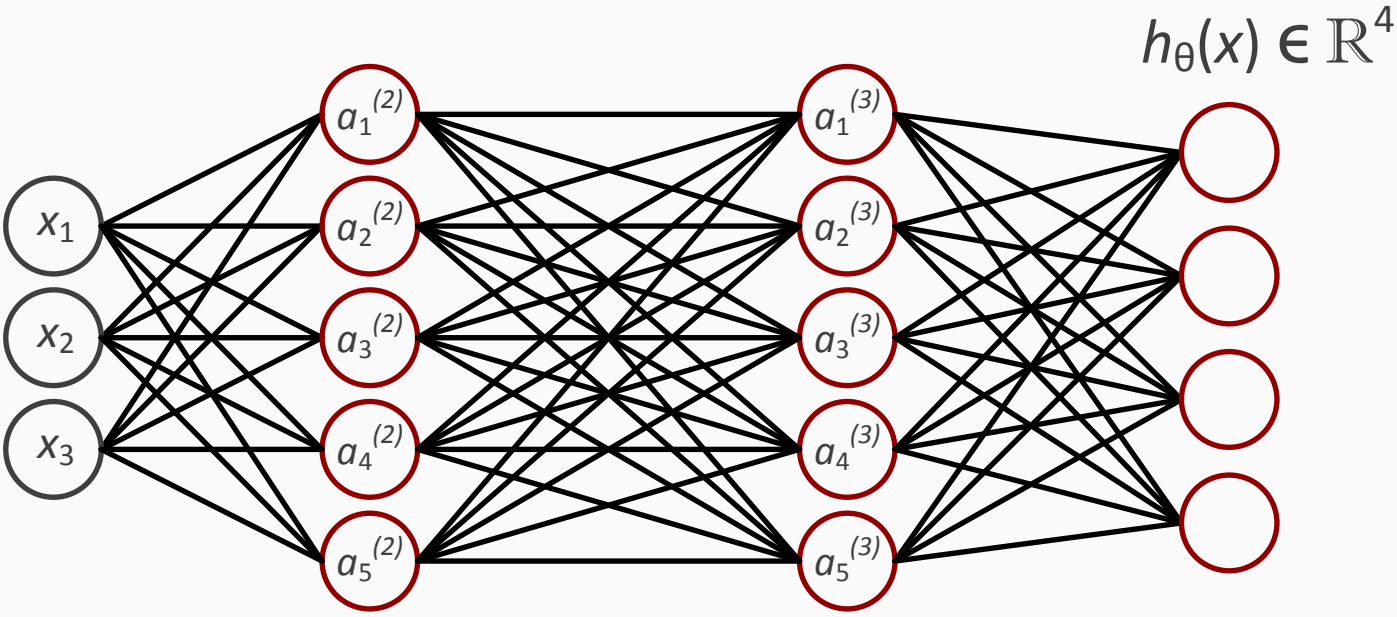
Thus, $h_\theta x = 1$ when either both x_1 and x_2 are 0; or when x_1 and x_2 are both 1.

In other words, $h_\theta x$ outputs 1 at the latter two locations and 0 otherwise.

multiclass classification

Multiclass classification is essentially an extension of the one-vs-all method as illustrated. The output is now a vector of 4 numbers.

multiple output units: one-vs-all



Want $h_\theta x \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\theta x \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\theta x \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $h_\theta x \approx \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$, etc.

In the training set: $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$,..., $(x^{(m)}, y^{(m)})$

$y^{(i)}$ will be represented as one of the following:

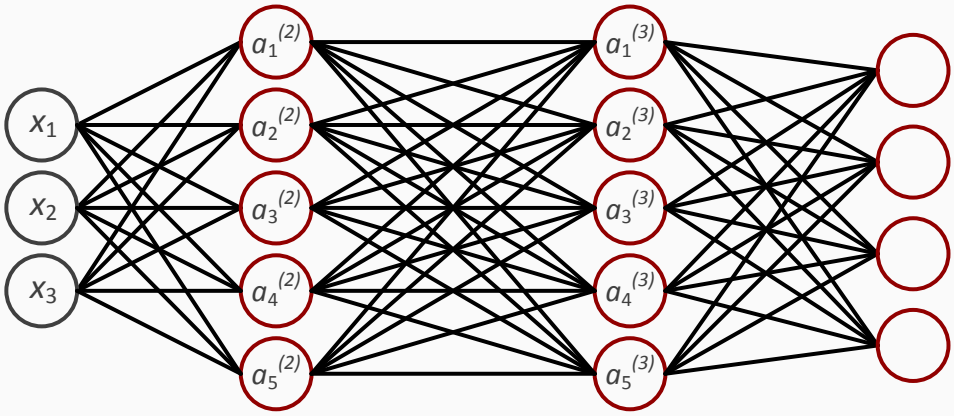
$Y^{(i)}$ = one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$

artificial neural networks ⇄ learning

cost function and backpropagation

cost function

neural network (classification)



L = total number of layers in network

s_l = number of units (excluding bias unit) in layer l

$L = 4$ $s_1 = 3, s_2 = 5, s_3 = 4, s_4 = s_L = 4$

binary classification	multiclass classification (K classes)
y = 0 or 1	y = a vector of binary classification
1 output unit	K output units
$h_\theta(x) \in \mathbb{R}$	$h_\theta(x) \in \mathbb{R}^K$
$s_L = 1$ $K = 1$	$s_L = K$ K is typically ≥ 3

cost function

logistic regression

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

For logistic regression, the goal is to minimize the above cost function. For a neural network, there will be not just a single logistic regression output unit, there will be K.

neural network

$h_\theta(x) \in \mathbb{R}^K$ $\left(h_\theta(x)\right)_i = i^{\text{th}}$ output

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{i=1}^{K} y_k^{(i)} \log\left(h_\theta(x^{(i)})\right)_k + \left(1 - y_k^{(i)}\right) \log\left(1 - \left(h_\theta(x^{(i)})\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_1}\sum_{j=1}^{s_1+1}\left(\theta_{ji}^{(l)}\right)^2$$

example

to minimize $J(\theta)$ as a function of θ, when programming code into an algorithm, the necessary code to supply is for $J(\theta)$ and the (partial) derivative terms $\frac{\partial}{\partial \theta_{ij}^{(l)}}$ for every *i, j, l*.

backpropagation algorithm

gradient computation

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{i=1}^{K} y_k^{(i)} \log\left(h_\theta(x^{(i)})\right)_k + \left(1 - y_k^{(i)}\right) \log\left(1 - \left(h_\theta(x^{(i)})\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_1}\sum_{j=1}^{s_1+1}\left(\theta_{ji}^{(l)}\right)^2$$

with the goal to $\min_\theta J(\theta)$, it is necessary to compute $-J(\theta)$ and $-\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$

given **one** training example (x, y):

first apply forward propagation in order to compute the hypothesis outputs

$a^{(1)} = x$ *input layer*

$z^{(2)} = \theta^{(1)} a^{(1)}$ *1st hidden layer*

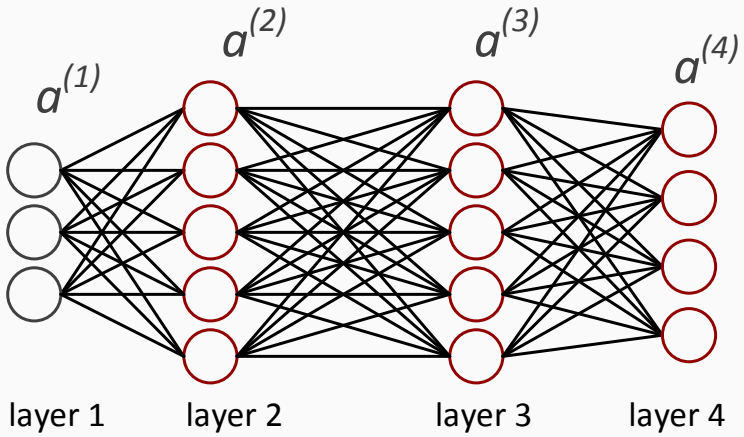
$a^{(2)} = g(z^{(2)})$ add $a_0^{(2)} = 1$

$z^{(3)} = \theta^{(2)} a^{(2)}$ *2nd hidden layer*

$a^{(3)} = g(z^{(3)})$ add $a_0^{(2)} = 1$

$z^{(4)} = \theta^{(3)} a^{(3)}$ *output layer*

$a^{(4)} = h_\theta(x) = g(z^{(4)})$



this is a vectorized implementation of forward propagation allowing to compute the activation values for all neurons in the neural network.

next apply backward propagation in order to compute the partial derivatives

intuition: $\delta_i^{(l)}$ = "error" of node *j* in layer *l*.

recall $a_i^{(l)}$ that does the activation of *j* unit in layer *l*; thus, the $\delta_i^{(l)}$ is going to capture the error in that activation of neural duo.

for each output unit (layer L = 4)

$\delta_i^{(4)} = a_j^{(4)} - y_j$ → vectorized implementation $\delta^{(4)} = a^{(4)} - y$

The error ($\delta_i^{(4)}$) is = to the activation of that unit ($a_j^{(4)} = (h_\theta(x))_j$) minus the actual value of 0 in our training example (y_j)

next compute the delta terms for prior network layers.

each factor in the below function ($(\theta^{(3)})^{T}\delta^{(4)}$ and $g'(z^{(3)})$) are vectors; the .* operator indicates element wise multiplication of the vectors.

$\delta^{(3)} = (\theta^{(3)})^{T}\delta^{(4)} .* g'(z^{(3)})$ → vectorized implementation $a^{(3)} .* (1 - a^{(3)})$

$\delta^{(2)} = (\theta^{(2)})^{T}\delta^{(3)} .* g'(z^{(2)})$ $a^{(2)} .* (1 - a^{(2)})$

(no $\delta^{(1)}$ term), the first layer corresponds to the input layer; being the input features (do not want to change those values); the computation starts at the output later and solves backward

backpropagation algorithm in the context of a larger sample

bring together the theory above to compute derivatives with respect to parameters

in the training set: {($x^{(1)}$, $y^{(1)}$),..., ($x^{(m)}$, $y^{(m)}$)}

set $\Delta_{ji}^{(l)} = 0$ (for all l, i, j) (used to compute the partial derivatives, slowly adding together in the process)

next will loop through the training set ($x^{(i)}$, $y^{(i)}$)

for $i = 1\ to\ m$

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

perform forward propagation to compute $a^{(1)}$ for $l = 2, 3, \ldots, L$

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

compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ as before, no $\delta^{(1)}$ because error is not associated with input layer

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)}\delta_i^{(l+1)}$ → vectorized implementation $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

then outside the for loop

$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\theta_{ij}^{(l)}$ if $j \neq 0$

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

$$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta) = D_{ij}^{(l)}$$

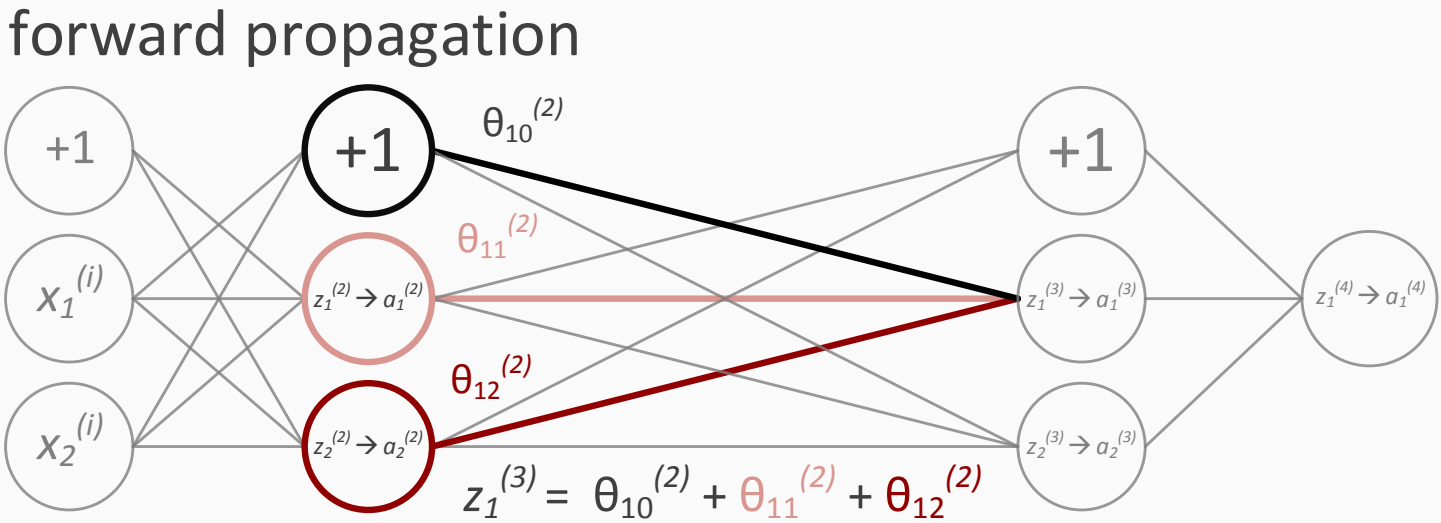
example

with two training examples ($x^{(1)}$, $y^{(1)}$) and ($x^{(2)}$, $y^{(2)}$), the correct sequence of operations for computing the gradient are as follows (FP = forward propagation and BP = backward propagation)

FP using $x^{(1)}$ followed by BP using $y^{(1)}$. Then FP using $x^{(2)}$ following by BP using $y^{(2)}$

backpropagation intuition

stepping through the mechanics in a systematic method: using backpropagation to compute the derivatives of a function



in performing forward propagation, the i inputs will be inserted into the first layer. Then the weighted sum of inputs (z) of the input units are forward propagated to the second layer. The sigmoid logistic function (a) will be applied to the z values, giving the activation values. The step will be repeated in an additional forward propagation to the next layer. The final output value will be given in the output layer.

what is back propagation doing?

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{i=1}^{K} y_k^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right)\log\left(1 - \left(h_\theta\left(x^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_1}\sum_{j=1}^{s_1+1}\left(\theta_{ji}^{(l)}\right)^2$$

focusing on a single example , the case of 1 output unit, and ignoring regularization:

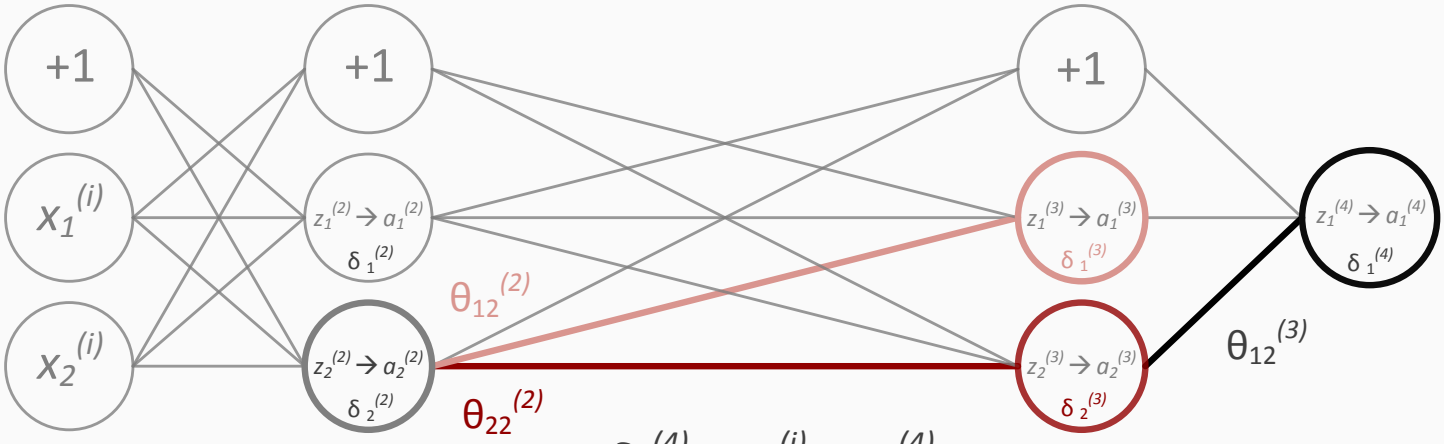
$$\text{cost}(i) \approx\ y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right)\log h_\theta\left(x^{(i)}\right)$$

think of the cost as the sum of least squared errors: $\text{cost}(i) \approx\ \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

the cost(i) is measuring how well the network is performing on example i

backward propagation

$\delta_j^{(l)}$ = the "error" of cost for $a_j^{(l)}$ (unit j in layer l)



$$\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$$

$$\delta_2^{(3)} = \theta_{12}^{(3)}\delta_1^{(4)}$$

$$\delta_2^{(2)} = \theta_{12}^{(2)}\delta_1^{(3)} + \theta_{22}^{(2)}\delta_2^{(3)}$$

backpropagation in practice

implementation note: unrolling parameters

the process of unrolling parameters form matrices into vectors in order to use advanced optimization:

function taken to calculate cost function (`jVal`) and derivatives (`gradient`). then pass to advanced optimization algorithm (`fminunc`), taking the input, pointing to the cost function and an initial value of theta:

```
function [ jVal, gradient ] = costFunction (theta)
    ...
optTheta = fminunc (@costFunction, initialTheta, options)
```

the gradient, theta, and initialTheta terms are vectors $\mathbb{R}^{n+1}$ which work in terms of logistic regression.

in neural networks, the parameters are matrices:

Neural Network (L=4):

$\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$ – matrices (`Theta1, Theta2, Theta3`)

$D^{(1)}, D^{(2)}, D^{(3)}$ – matrices (`D1, D2, D3`)

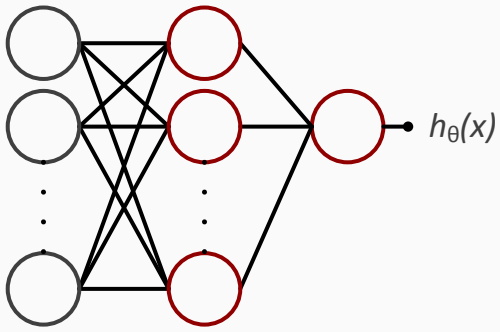
"unroll" into vectors

example

$s^{(1)} = 10, s^{(2)} = 10, s^{(3)} = 1$

$\theta^{(1)} \epsilon\ \mathbb{R}^{10\times11}, \theta^{(2)} \epsilon\ \mathbb{R}^{10\times11}, \theta^{(3)} \epsilon\ \mathbb{R}^{1\times11}$

$D^{(1)} \epsilon\ \mathbb{R}^{10\times11}, D^{(2)} \epsilon\ \mathbb{R}^{10\times11}, D^{(3)} \epsilon\ \mathbb{R}^{1\times11}$



code to unroll the matrices into a vector:

```
thetaVec = [ Theta1(:); Theta2(:); Theta3(:) ];
DVec = [ D1(:); D2(:); D3(:) ];
```

code to roll the vector back into the original matrices:

```
Theta1 = reshape(thetaVec(1:110), 10, 11) ;
Theta2 = reshape(thetaVec(111:120), 10, 11) ;
Theta3 = reshape(thetaVec(221:231), 1, 11) ;
```

utilizing unrolling to implement the learning algorithm

have initial parameters $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$

unroll matrices into a long vector to get `initialTheta` to pass to `fminunc (@costFunction, initialTheta, options)`

then implement the cost function:

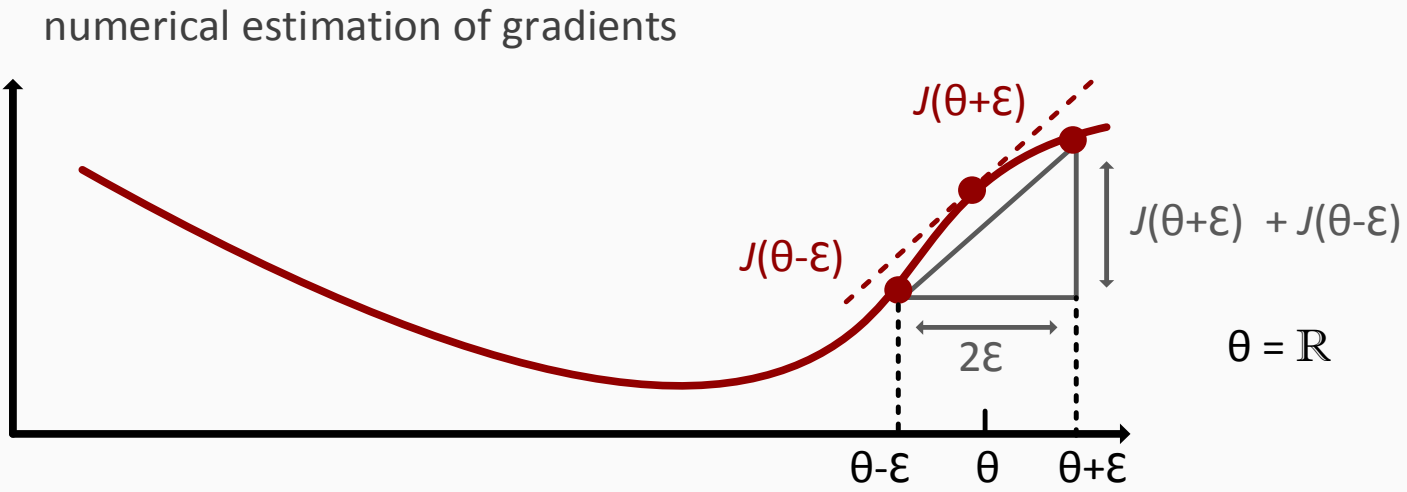
```
function [ jVal, gradient ] = costFunction (thetaVec)
```

from `thetaVec`, get $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$

use forward/backward propagation to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\theta)$ unroll to get `gradientVec`

gradient checking

backpropagation is prone to errors in practice which can appear to decrease for every iteration in gradient descent. gradient checking is a way to ensure there are no unforeseen bugs in implementation and the function is correctly computing the derivatives of cost function $J(\theta)$



$\frac{\partial}{\partial\theta}J(\theta) \approx \frac{J(\theta+\varepsilon)-(\theta-\varepsilon)}{2\varepsilon}$ typically Ɛ will be labeled as a small value ($Ɛ=10^{-4}$), while avoiding an overly small value, because the right hand term mathematically becomes to the slope of the function at the given point.

implementation in octave:

```
gradApprox = (J(theta+EPSILON) - J(theta-EPSILON))/(2*EPSILON)
```

example

$J(\theta) = \theta^3, \theta = 1, \varepsilon = 0.01$

using the formula $\frac{J(\theta+\varepsilon)-(\theta-\varepsilon)}{2\varepsilon}$ the derivative approximates a value of 3.0001

(when $\theta = 1$, the true derivative is $\frac{\partial}{\partial\theta}J(\theta) = 3$)

a more general case when θ ≠ $\mathbb{R}$

parameter vector θ

$\theta \in \mathbb{R}^n$ (e.g. θ is unrolled version of $\theta^{(1)}$, $\theta^{(2)}$, $\theta^{(3)}$)

$\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \varepsilon, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1 - \varepsilon, \theta_2, \theta_3, \ldots, \theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \varepsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2 - \varepsilon, \theta_3, \ldots, \theta_n)}{2\varepsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n + \varepsilon) - J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n - \varepsilon)}{2\varepsilon}$$

The above functions provide a numerical computation of the partial derivative of J with respect to any one of the parameters θ_i

implementation in octave:

```
for I = 1:n,
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*EPSILON);
end;
```

check that `gradApprox` ≈ `DVec` (`DVec` is derivative from backpropagation)

the above computes the derivative of the cost function with respect to every parameter in the neural network, then taking the gradient computed from backpropagation

implementation note:

implement backpropagation to compute `DVec` (unrolled $D^{(1)}$, $D^{(2)}$, $D^{(3)}$)
implement numerical gradient check to compute `gradApprox`
ensure the above results in similar values
turn off gradient checking. using backpropagation code for learning

important:

be sure to disable gradient checking code before training the classifier. If numerical gradient computation is run on every iteration of gradient descent (or in the inner loop of `costFunction(...)`) the code will be sufficiently slow.

The main reason for using backpropagation algorithm rather than numerical gradient descent computation method during **learning**:

The numerical gradient descent algorithm is very slow to execute

random initialization

initial value of θ

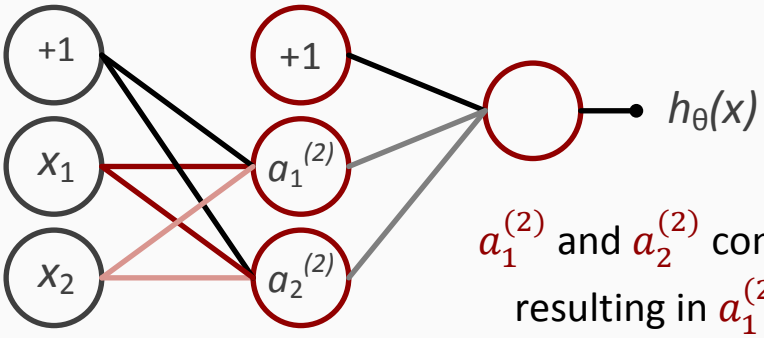
for gradient descent and advanced optimization method, initial θ value is needed

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

consider gradient descent

set `initialTheta = zeros(n,1)` → setting θ to 0 initially works in logistic regression but does not work in neural networks

zero initialization



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

all weights on initialization will be the same and result in both hidden units $a_1^{(2)}$ and $a_2^{(2)}$ computing the same function of the inputs, resulting in $a_1^{(2)} = a_2^{(2)}$. after each update, parameters corresponding to the inputs going into each of the two hidden units are identical

random initialization: symmetry breaking

initialize each $\theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$

(i.e. $-\epsilon \leq \theta_{ij}^{(l)} \leq \epsilon$)

implementation in octave:

```
Theta1 = rand(10,11)*(2*INIT_EPSILON)-INIT_EPSILON;
Theta2 = rand(1,11)*(2*INIT_EPSILON)-INIT_EPSILON;
```

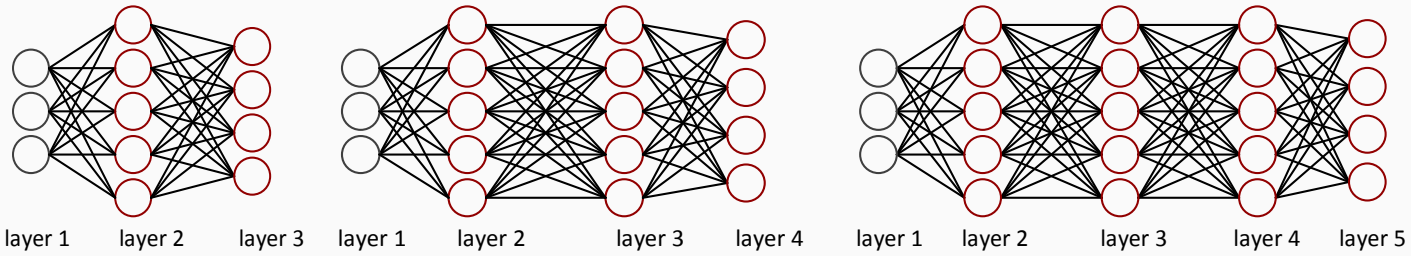
putting it together

fitting the constructs of neural networks together and developing the process of implementing a neural network learning algorithm

training a neural network

network architecture

determining a network architecture depends on the number of input units (dimensions of features $x^{(i)}$ and the number of output units (number of classes; single or multiclass classification)



in multiclass classification where the output units $y \in \{1, 2, 3, \ldots, 10\}$, the network will not be shown the number of outputs as a notation of y for each possible classes (i.e. y = 5) but as vectors:

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \; or \; y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ etc.

common practice is typical of a single hidden layer. if hidden layers >1, it is best practice to have the same number of hidden units for each hidden layer (i.e. if a^1 has 5 units, then a^2 has 5 units). lastly having a number of features in the hidden layers is typically comparable to the number of input units.

implementation process to train a neural network

set up inputs and randomly initialize values of the weights (typically values near zero)

implement forward propagation to compute $h_\theta(x)$, the output vector of the *y* values

compute the cost function of $J(\theta)$

implement backpropagation to compute partial derivative terms $\frac{\partial}{\partial \theta_1} J(\theta)$ with respect to the parameters. usually achieved with a for loop over the training examples:

for `i = 1:m`

perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

the loop iterated FP and BP over the first example $(x^{(1)}, y^{(1)})$, second example $(x^{(2)}, y^{(2)})$, and finally the last example , … , $(x^{(m)}, y^{(m)})$

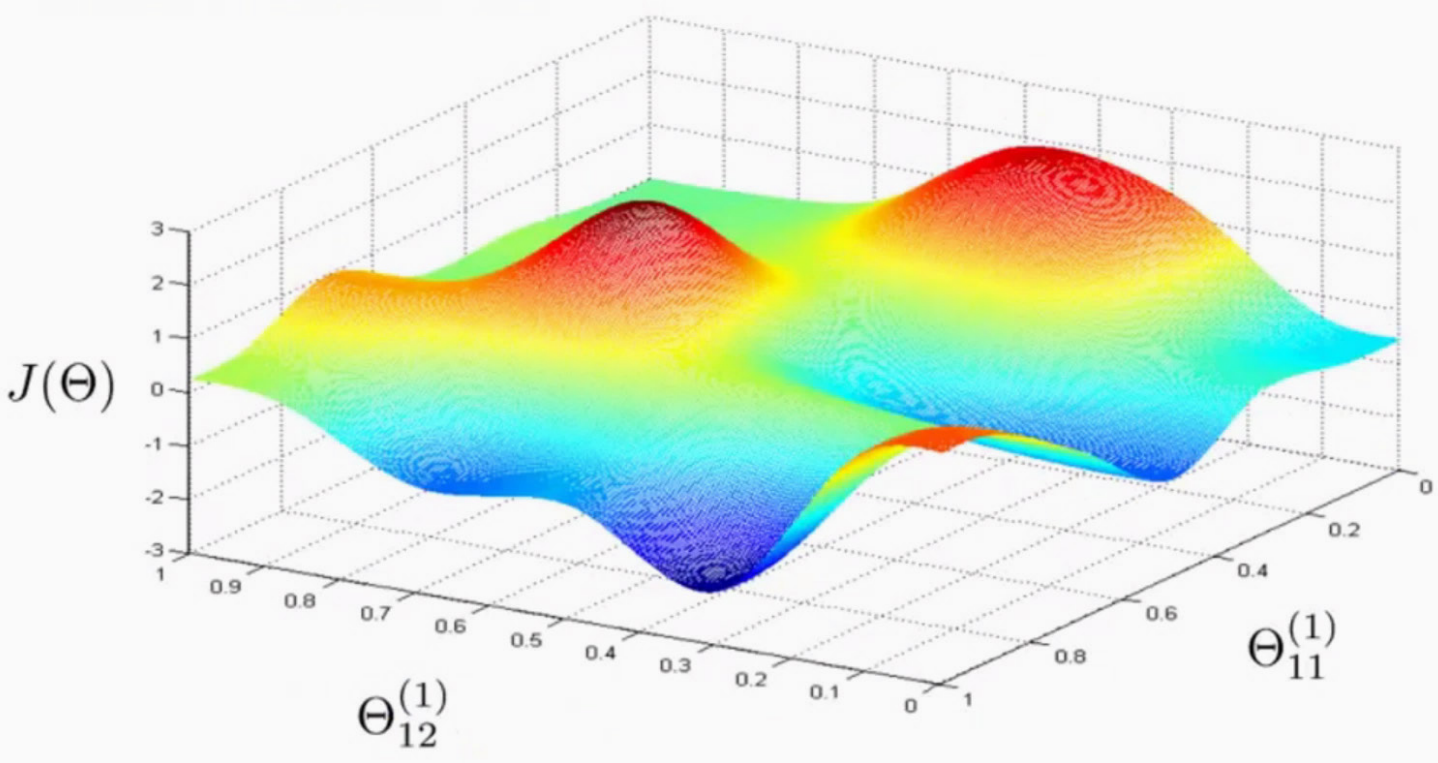
(get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \dots, L$)

use gradient checking to compare $\frac{\partial}{\partial \theta_1} J(\theta)$, computed using backpropagation versus using numerical estimate gradient of $J(\theta)$ to ensure similar values

subsequently disable the gradient checking code

use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\theta)$ as a function of parameters θ

in neural networks, cost function $J(\theta)$ is non-convex and susceptible to local optima



example

in order to verify a learning algorithm is running correctly when using gradient descent with backpropagation to minimize $J(\theta)$ as a function of θ, plot $J(\theta)$ as a function of the number of iterations and ensure it is decreasing (or at least not increasing) with every iteration

1. while training a three layer neural network to use backpropagation to compute the gradient of the cost function. one of the steps is to update
$\Delta_{ij}^{(2)} := \Delta_{ij}^{(2)} + \delta_i^{(3)} \times \left(a^{(2)}\right)_j$ in the backpropagation algorithm

    for every i, j, $\Delta^{(2)} := \Delta^{(2)} + \delta^{(3)} \times \left(a^{(2)}\right)^T$ is a vectorization of this step

2. `Theta1` is a 5x3 matrix, and `Theta2` is a 4x6 matrix.

`thetaVec` is set to `thetaVec=[Theta1(:);Theta2(:)]`.

the following correctly recovers `Theta2` → `reshape(thetaVec(16:39),4,6)`

3. let $J(\theta)=2\theta^4+2$. Let $\theta=1$, and $\epsilon=0.01$. Use the formula $J(\theta+\epsilon)-J(\theta-\epsilon)$ / 2ϵ to numerically compute an approximation to the derivative at $\theta=1$.

(when θ=1, the true/exact derivative is dJ(θ)dθ=8.) the computed value is 8.0008

4. the following statements are true to neural networks

    using gradient checking can help verify if one's implementation of backpropagation is bug-free.

    if our neural network overfits the training set, one reasonable step to take is to increase the regularization parameter λ.

5. the following statements are true to neural networks

    if we are training a neural network using gradient descent, one reasonable "debugging" step to make sure it is working is to plot j(θ) as a function of the number of iterations, and make sure it is decreasing (or at least non-increasing) after each iteration.

    suppose you are training a neural network using gradient descent. depending on your random initialization, your algorithm may converge to different local optima (i.e., if you run the algorithm twice with different random initializations, gradient descent may converge to two different solutions).

machine learning hypothesis evaluation

evaluation a learning algorithm

after implementing regularized linear regression to predict a target

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

large prediction errors can be addressed with the following methods:
obtain additional training examples
more carefully selected smaller set of features $(x_1, x_2, x_3, \dots, x_{100})$
obtaining additional features
adding polynomial features (x_1^2, x_1^2, x_1x_2, etc.)
decreasing λ
increasing λ

machine learning diagnostics

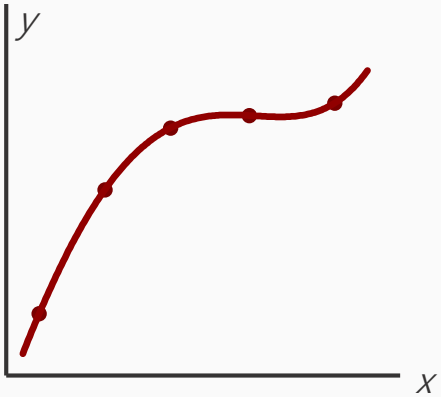
diagnostic: a test run to gain insight on what is and is not working with a learning algorithm; gaining guidance on how to best improve its performance.

machine learning diagnostics can be time consuming to implement, however, doing so can save significant time in design as opposed to applying the above examples blindly

machine learning diagnostics can rule out certain courses of action to change the learning algorithm when evaluated as being unlikely to significantly improve performance

evaluating a hypothesis

evaluating a hypothesis



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

low training error fails to generalize to new examples outside of the training set · offers poor predictive power

randomly partitioning a dataset into a training and test set (~70%/30% split) allows a model to be evaluated for accuracy (ability to generalize)

$$\begin{matrix}\left(x^{(1)}, y^{(1)}\right)\\ \vdots\\ \left(x^{(m)}, y^{(m)}\right)\end{matrix}$$
$$\begin{matrix}\left(x_{test}^{(1)}, y_{test}^{(1)}\right)\\ \vdots\\ \left(x_{test}^{(m)}, y_{test}^{(m)}\right)\end{matrix}$$

example
implementation of linear regression (without regularization) will overfit the training set if the training error $J(\theta)$ is **low** and the test error $J_{test}(\theta)$ is **high**

training/testing procedure for linear regression

learn parameter θ from training data (minimizing training error $J(\theta)$)
compute test error using the squared error metric:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta \left(x_{test}^{(i)} \right) - y_{test}^{(i)} \right)^2$$

training/testing procedure for logistic regression

learn parameter θ from training data (minimizing training error $J(\theta)$)
compute test error using the squared error metric:

$$J_{test}(\theta) = -\frac{1}{m_{test}} \left[\sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta \left(x_{test}^{(i)} \right) + \left(1 - y_{test}^{(i)} \right) \log \left(1 - h_\theta \left(x_{test}^{(i)} \right) \right) \right]$$

alternative test error metric: misclassification error (0/1)

$$\text{err}(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5, \quad y = 0 \\ & \text{or if } h_\theta(x) \leq 0.5, \quad y = 1 \\ 0 & \text{otherwise} \end{cases}$$

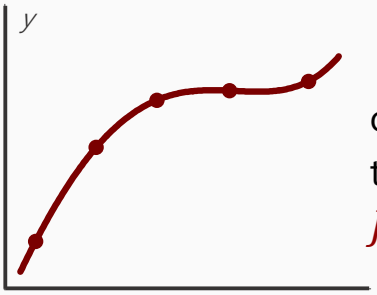
$$\text{test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \text{err} \left(h_\theta \left(x_{test}^{(i)} \right), y_{test}^{(i)} \right)$$

the misclassification error during training is often a standard approach to evaluating the accuracy of a model. a model with excessive false positives/negatives will yield low predictive accuracy. this method is commonly displayed in a classification matrix

$$\begin{bmatrix} \text{true positive} & \text{false positive} \\ \text{true negative} & \text{false negative} \end{bmatrix} \text{ or } \begin{bmatrix} \text{true positive} & \text{false positive} \\ \text{false negative} & \text{true negative} \end{bmatrix} \text{ etc.}$$

model selection and train/validation/test sets

determining what degree of polynomial to fit a dataset, what features to include in an algorithm, and what regularization parameter for a learning algorithm are components to the model selection process. Partitioning data into training, validation, and test sets.



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

once parameters $\theta_0, \theta_1, \dots, \theta_4$ fit to some set of data (training set), the error of parameters measured on that data (the training error $J(\theta)$) is likely to be lower than the actual generalization error

model selection

determining what degree of polynomial (linear, quadratic, cubic, etc.) to fit to the data

a hypothetical additional parameter to (*d*) to represent the best degree of polynomial

first option is to minimize the training error of each model for parameter vector $\theta^{(xi)}$

from the parameter vectors, test set error can be calculated, choosing the lowest error

$$\begin{aligned}
d = 1 &\rightarrow h_\theta(x) = \theta_0 + \theta_1 x &&\rightarrow \theta^{(1)} &&\rightarrow J_{test}(\theta^{(1)}) \\
d = 2 &\rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 &&\rightarrow \theta^{(2)} &&\rightarrow J_{test}(\theta^{(2)}) \\
d = 3 &\rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3 &&\rightarrow \theta^{(3)} &&\rightarrow J_{test}(\theta^{(3)}) \\
&\vdots \quad\quad\quad\quad \vdots \\
d = 10 &\rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10} &&\rightarrow \theta^{(10)} &&\rightarrow J_{test}(\theta^{(10)})
\end{aligned}$$

choose $\theta_0 + \theta_1 x + \cdots + \theta_5 x^5$

how well does the model generalize: report test set error $J_{test}(\theta^{(5)})$

problem: $J_{test}(\theta^{(5)})$ is a likely optimistic estimate of generalization error

(i.e. the parameter (*d* = degree of polynomial is fit to the **test set**)

parameters developed to determine predictive power on the test set is less likely to fit the data to samples never seen before by the model. overfitting the test set is common

in addition to a training and test set, an additional validation set can prevent overfitting

possibly a ~60%/20%/20% split as opposed to the 70%/30% split previously referred

train · validation · test error

training error:

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)^2$$

cross validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right), y_{cv}^{(i)}\right)^2$$

test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}\left(h_\theta\left(x_{test}^{(i)}\right), y_{test}^{(i)}\right)^2$$

opposed to selecting the model from the **test set**, the model will be scored against the **validation set** for a less biased selection

$$\begin{aligned}
&d = 1 \rightarrow h_\theta(x) = \theta_0 + \theta_1 x \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})\\
&d = 2 \rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})\\
&d = 3 \rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3 \rightarrow \theta^{(3)} \rightarrow J_{cv}(\theta^{(3)})\\
&\vdots \qquad\qquad \vdots\\
&d = 10 \rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10} \rightarrow \theta^{(10)} \rightarrow J_{cv}(\theta^{(10)})
\end{aligned}$$

choose $\theta_0 + \theta_1 x + \cdots + \theta_4 x^4$

estimate generalization set error $J_{test}(\theta^{(4)})$

because the parameter *d* is fit to the **validation set** as opposed to the **test set**, the test set can now be used toscore the generaliztion error of the selected model

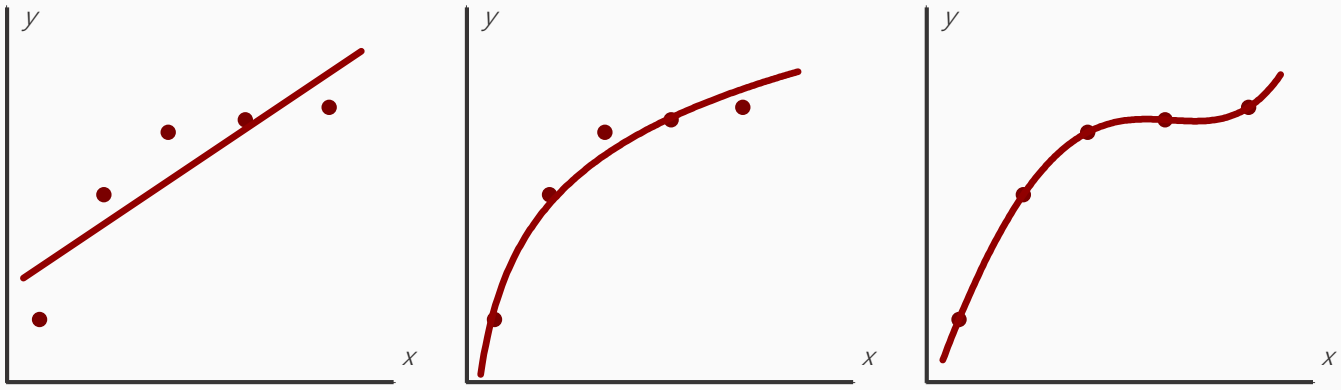
example

a model selection procedure chooses the degree of polynomial using a cross validation set: for the final model (with parameters θ), it would be generally expected for $J_{cv}(\theta)$ to be lower than $J_{test}(\theta)$ because...

an additional parameter (*d* = degree of polynomial) has been fit to the **cross validation set**

diagnosing bias vs variance

diagnosing bias vs variance

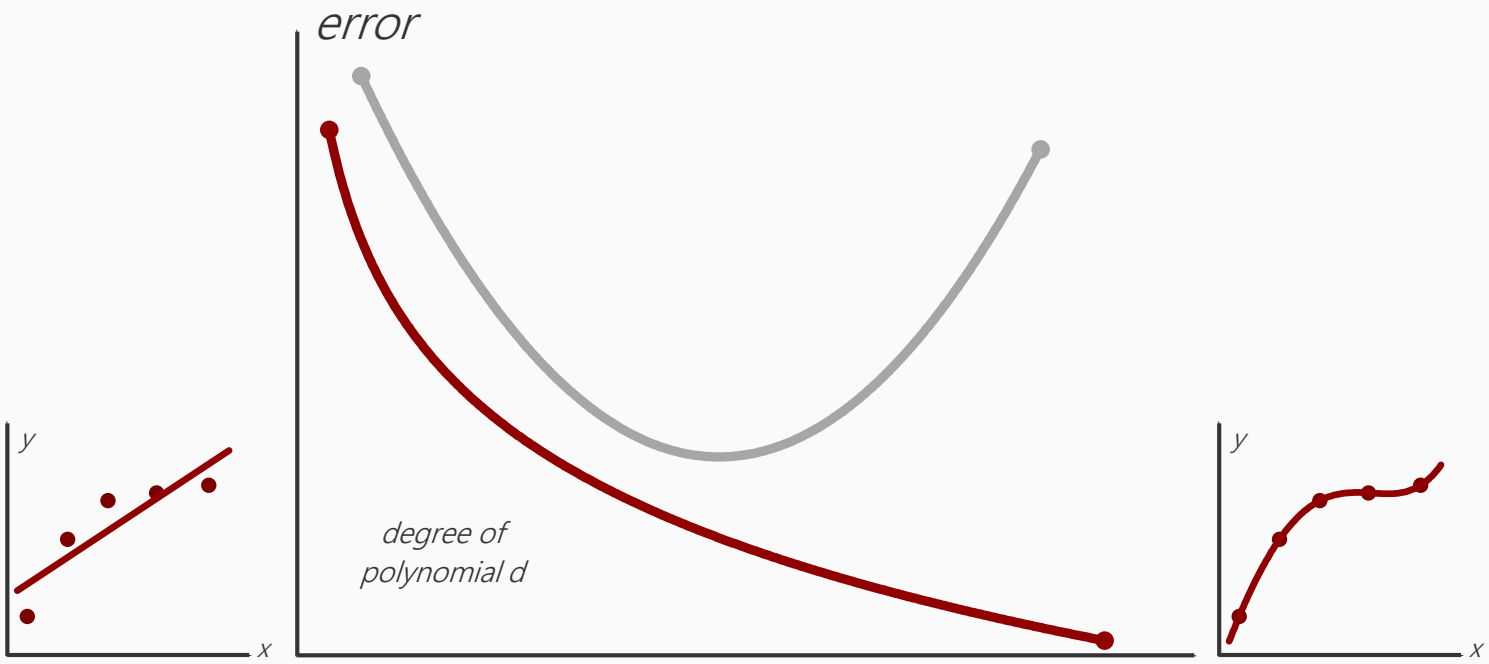


$\theta_0 + \theta_1 x$
high bias
(underfit) d=1

$\theta_0 + \theta_1 x + \theta_2 x^2$
"just right"
(fitted) d=2

$\theta_0, \theta_1 x, \ldots, \theta_4 x^4$
high variance
(overfit) d=4

the relationship between error and the degree of polynomial used on the training and cross validation and test datasets illustrates where the errors depart correlation



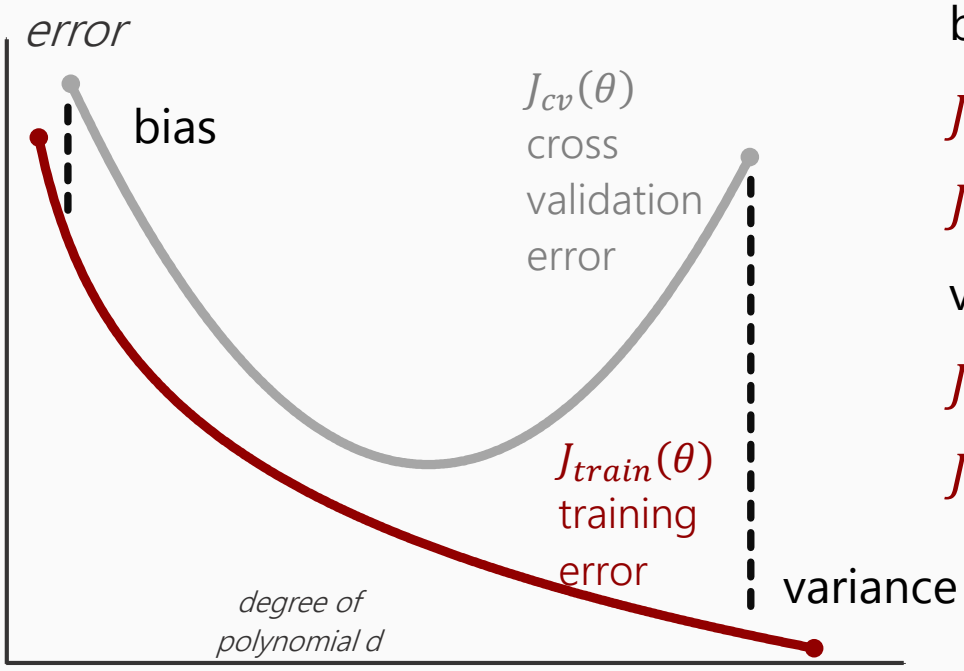
$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right), y_{cv}^{(i)}\right)^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}\left(h_\theta\left(x_{test}^{(i)}\right), y_{test}^{(i)}\right)^2$$

general expectations where bias and variance are experienced

when a learning algorithm is underperforming ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high): the diagnosis could be either a bias or variance problem.



bias (underfitted):

$J_{train}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

variance (overfitted):

$J_{train}(\theta)$ will be low

$J_{cv}(\theta) \gg J_{train}(\theta)$

example

in a classification problem: the (misclassification) error is defined as $\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \text{err}\left(h_\theta\left(x_{test}^{(i)}\right), y_{test}^{(i)}\right)$, and the cross validation (misclassification) error is similarly defined, using cross validation examples $\left(x_{cv}^{(1)}, y_{cv}^{(1)}\right), \ldots, \left(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})}\right)$. the training error is 0.10 and the cross validation error is 0.30. The problem the algorithm is likely suffering from:

high variance (overfitting)

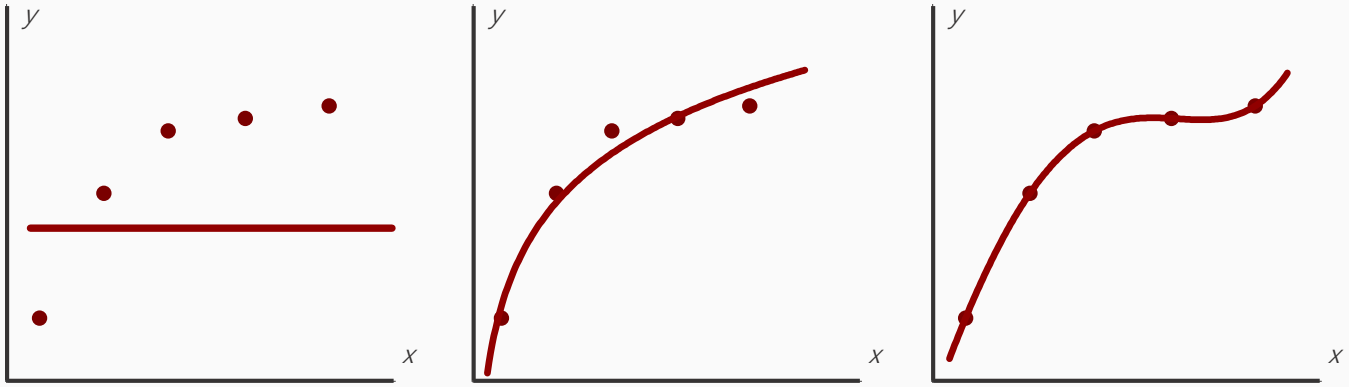
regularization and bias/variance

regularization can prevent overfitting, however, it also can affect the bias and variances of a learning algorithm.

linear regression

model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

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



large λ	intermediate λ	small λ
high bias (underfit)	"just right"	high variance (overfit)
λ = 10000:	λ = 0	
$\theta_1 x \approx 0, \theta_2 \approx 0, \ldots$		
$h_\theta(x) \approx \theta_0$		

choosing the regularization parameter

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right), y_{cv}^{(i)}\right)^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}\left(h_\theta\left(x_{test}^{(i)}\right), y_{test}^{(i)}\right)^2$$

the definition of the parameter $J(\theta)$ in the context *train*, *cv*, *test*:

half the average squared error of without the regularization term

$$
\begin{aligned}
&\text{attempt } \lambda = 0 \quad \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)}) \\
&\text{attempt } \lambda = 0.01 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)}) \\
&\text{attempt } \lambda = 0.02 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(3)} \rightarrow J_{cv}(\theta^{(3)}) \\
&\text{attempt } \lambda = 0.04 \quad . \quad . \quad . \\
&\text{attempt } \lambda = 0.08 \quad . \quad . \quad . \\
&\vdots \quad \vdots \quad \vdots \quad \vdots \\
&\text{attempt } \lambda = 10 \quad \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(10)} \rightarrow J_{cv}(\theta^{(10)})
\end{aligned}
$$

the chosen $\theta^{(m)}$ value will then be applied to the test error: $J_{test}(\theta)$

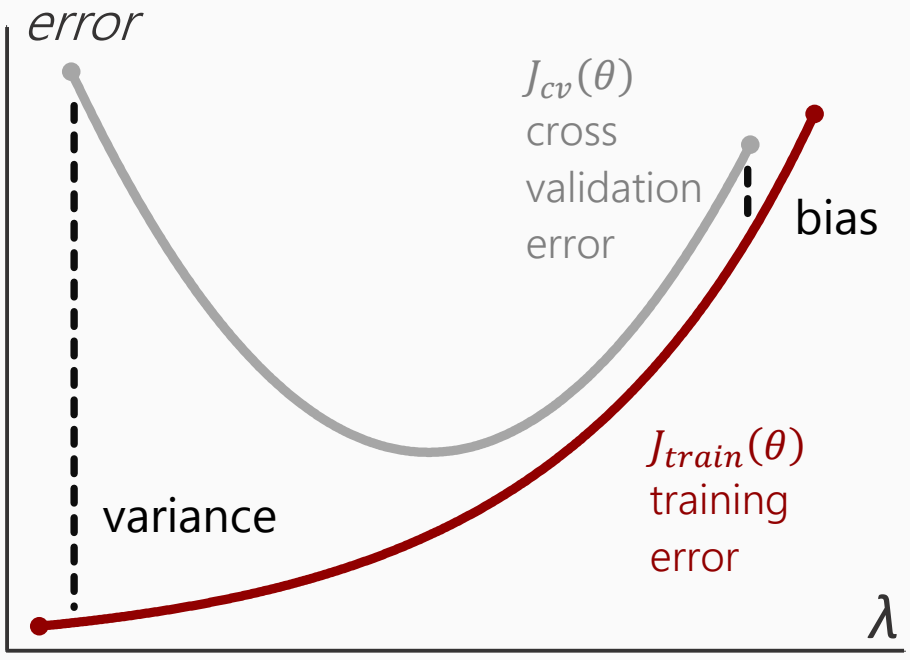
bias and variance as a function of the regularization parameter λ

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

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}), y^{(i)}\right)^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right), y_{cv}^{(i)}\right)^2$$

as lambda increases in regularization, the training error will increase while the validation error decreases. After lambda has increased to an extent, the validation error regress upward



learning curves

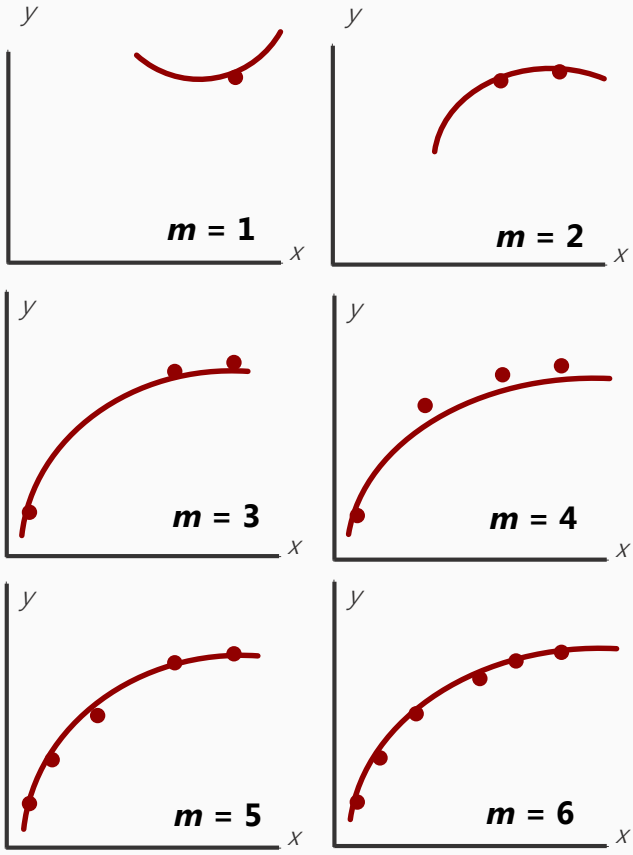
a tool to sanity check if learning algorithms are performing as expected, correctly, or identify areas of improvement from suffering from bias/variance

initially plot:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right), y_{cv}^{(i)}\right)^2$$



as a function of *m*

a single parameter will fit perfectly without regularization

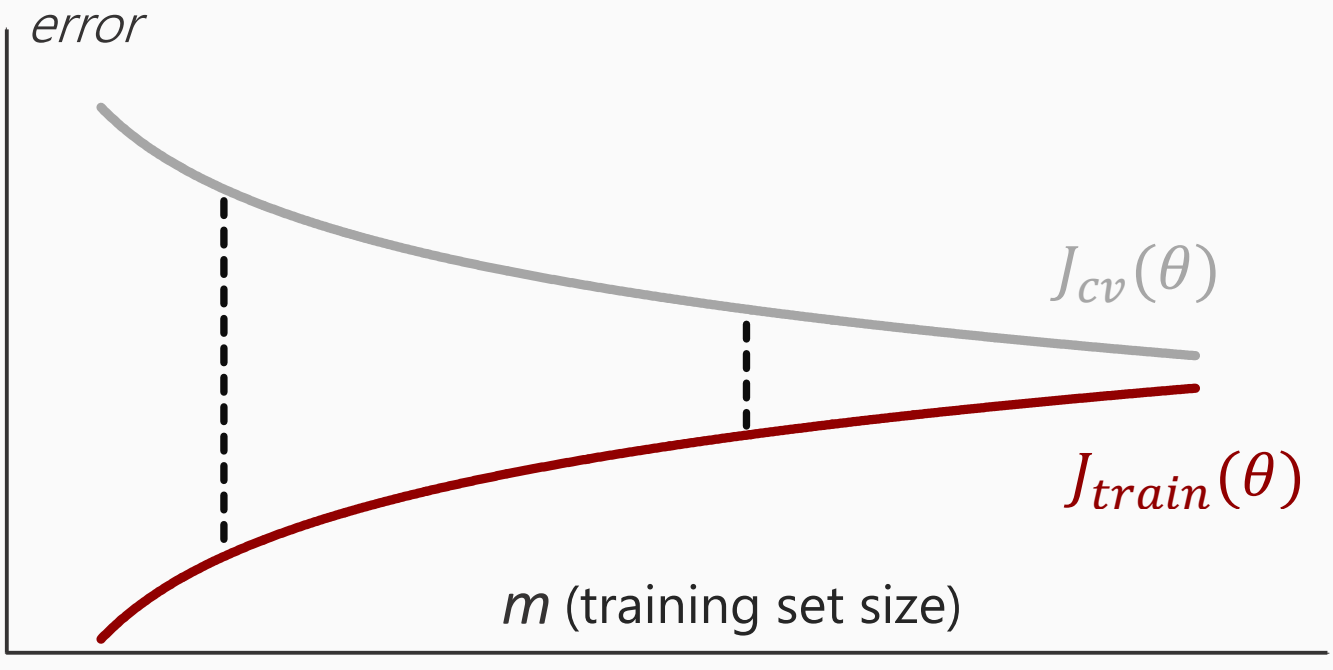
two parameters will fit almost perfectly without regularization and perfectly with regularization

three parameter behave similarly to the latter two. Thus, training error will effectively be 0 without regularization and almost 0 with regularization

four parameters will not plot perfectly and will increase training error

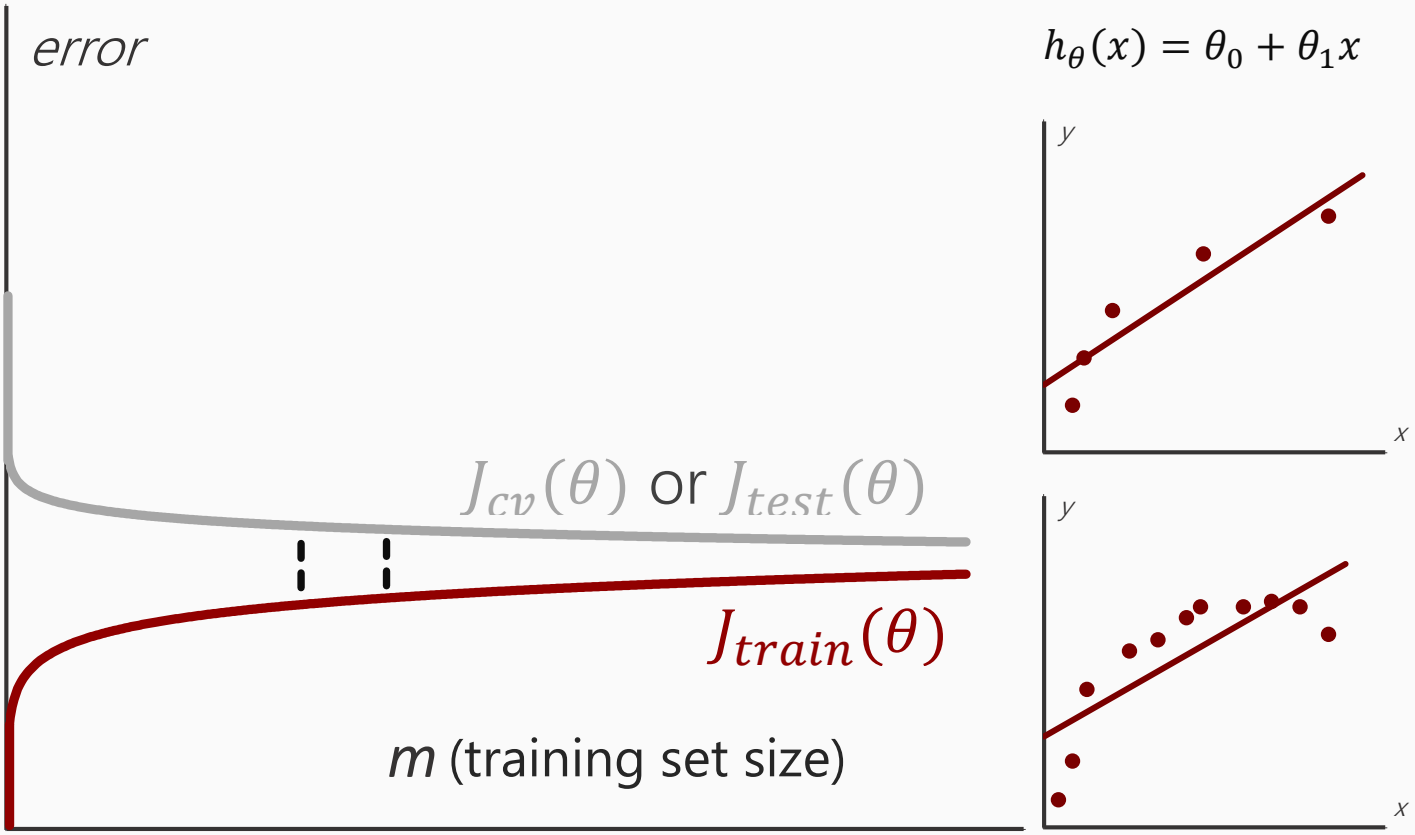
five parameters will continue to increase training

the more data, the better a model will generalize to new examples:



a problem that experiences high bias

the errors of the training and validation data will converge at a faster rate and remain level as the sample continues to grow. The error rate will reach an optimal goodness of fit after so many examples before increases in samples have little effect on the model:

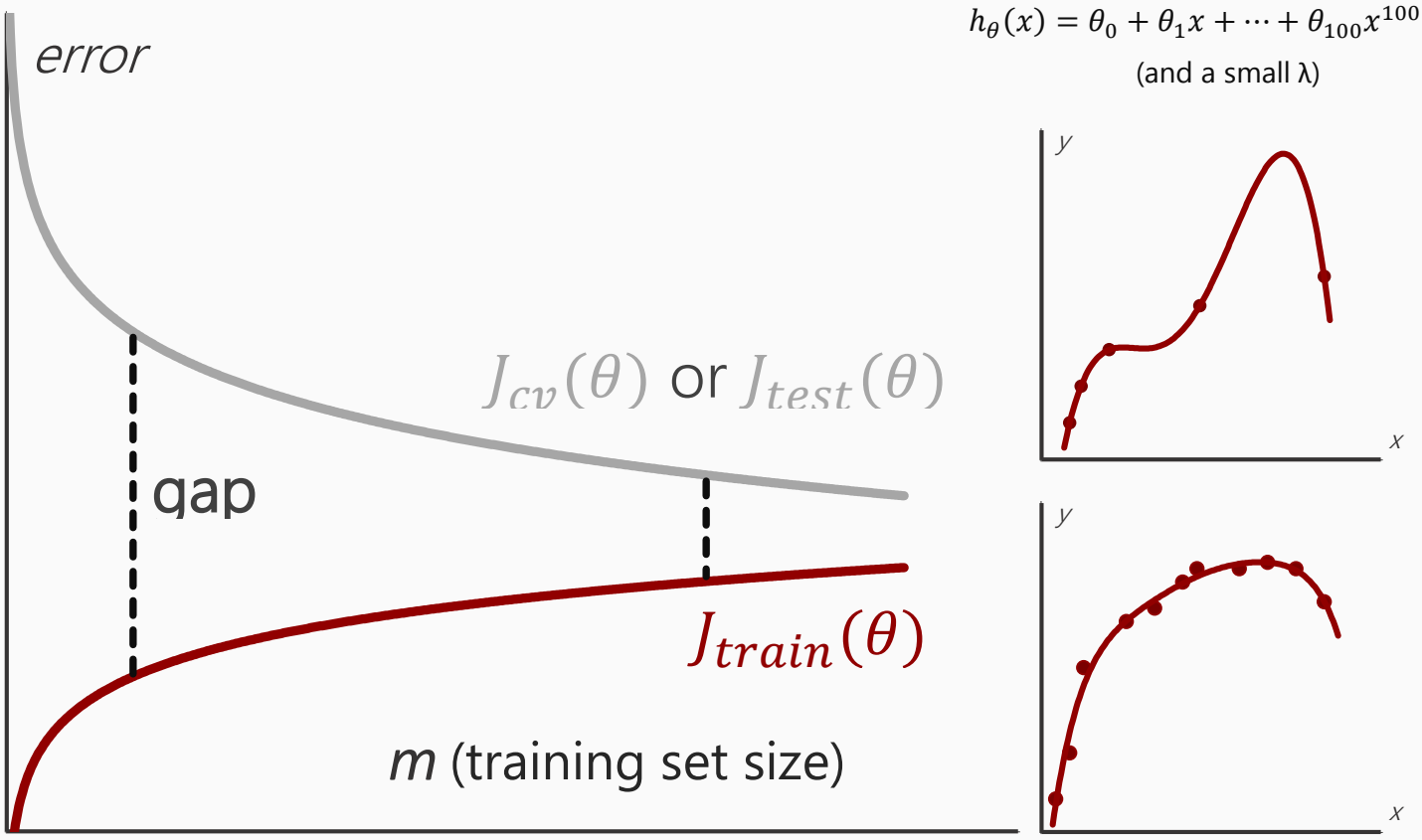


the problem with high bias reflects in the fact that both validation and training error are high, ending up with a relatively high value for both $J_{train}(\theta)$ and $J_{cv}(\theta)$. essentially the error will not increase by a large amount when the sample continues to increase thus the model will remain overfitted to the data if it was overfit in the initial expression

a learning algorithm suffering from high bias **will not (alone)** be substantially effective in reaching a more generalized fit through obtaining more training data

a problem that experiences high variance

with high variance problems, $J_{train}(\theta)$ error will initially be low and remain relatively low. reciprocally, the $J_{cv}(\theta)$ error will initially be high and remain as such. The **gap** between the two errors is indicative of high variance. extrapolating the sample to the right indicates the conditions of obtaining more samples is likely to help with training



a learning algorithm suffering from high variance **will likely** be substantially effective in reaching a more generalized fit through obtaining more training data

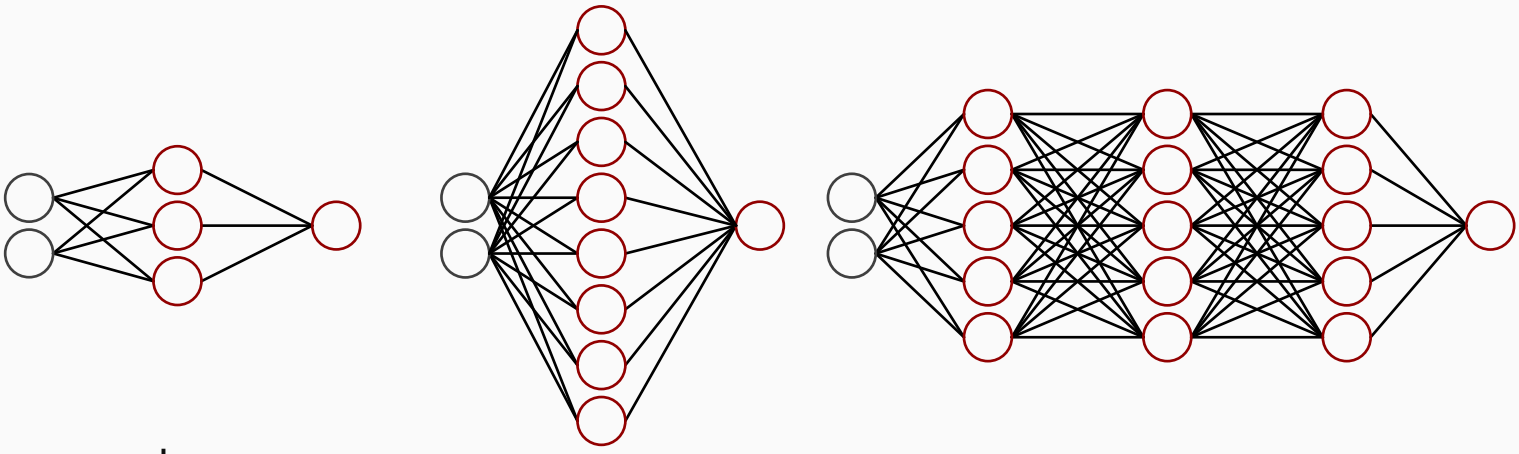
an algorithm suffering from high variance will be synonymous to the $J_{cv}(\theta)$ (cross validation error) being **>>** (much larger) than the $J_{train}(\theta)$ (training error)

debugging a learning algorithm

after implementing a regularized linear regression algorithm to predict some target variable a unacceptably large amount of errors occur in a prediction of *new* data:

obtain more training examples	→ addresses high variance (not bias)
use smaller sets of features	→ addresses high variance (not bias)
obtain additional features	→ addresses high bias
attempt adding polynomial features	→ addresses high bias
attempt decreasing or increasing λ	→ decrease in bias/increase in variance

neural networks and overfitting



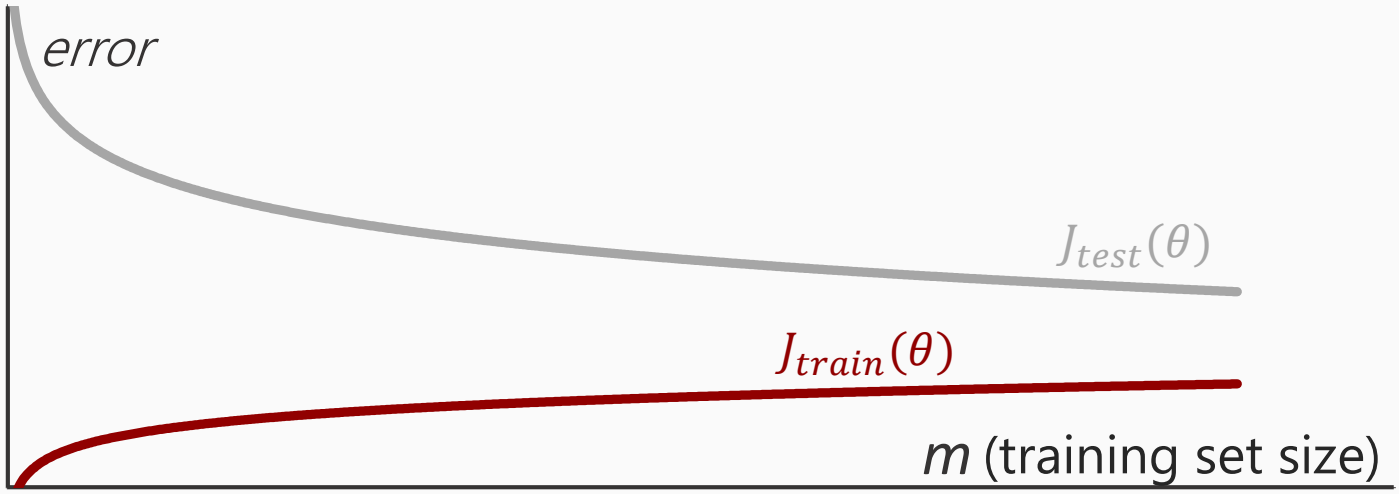
example

when fitting a neural network with one hidden layer to a training set, the cross validation error $J_{cv}(\theta)$ is much larger than the training error $J_{train}(\theta)$. Increasing the number of hidden units is likely to:

not help because the model suffers from high variance. adding hidden units to a model with a high variance problem is not effective.

advice for applying machine learning

while training a learning algorithm, it has unacceptably high error on the test set. After plotting the learning curve, and obtaining the figure below, the algorithm is suffering from high variance



given the following hypotheticals:

after implementing regularized logistic regression to classify what object is in an image (i.e., to do object recognition), the tested hypothesis on a new set of images makes unacceptably large errors with its predictions on the new images. However, the hypothesis performs **well** (has low error) on the training set. methods to address:

obtain more training examples	→ the gap in errors between the training and test suggest high variance problem where the algorithm has overfit the training set. addresses high variance. adding more training data increases the complexity of the training set and help with variance
use smaller sets of features	→ reducing the feature set will ameliorate the overfitting and help with the variance problem
attempt increasing λ	→ increasing the regularization parameter will reduce overfitting in a high variance problem

after implementing regularized logistic regression to predict what items customers will purchase on a web shopping site, the tested hypothesis on a new set of customers makes unacceptably large errors in predictions. Furthermore, the hypothesis performs **poorly** on the training set. the following addresses the issue:

obtain additional features	→ poor performance on both training and test sets indicate a high bias problem. additional features increase complexity of the hypothesis, improving the fit to both the training and test data under high bias
attempt adding polynomial features	→ adding more complex features increases the complexity of the hypothesis and improves the fit to both training and test data under high bias
attempt decreasing λ	→ decreasing the regularization parameter will reduce overfitting in a high bias problem

given the following hypotheticals:

suppose you are training a regularized linear regression model. the recommended way to choose what value of regularization parameter λ to use is to choose the value of λ which gives the lowest **training set** error.

the training error should not be used to choose the regularization parameter. the training error can always improve through using less regularization (smaller value of λ). however, too small of a value will not generalize well on the test dataset.

suppose you are training a regularized linear regression model. the recommended way to choose what value of regularization parameter λ to use is to choose the value of λ which gives the lowest **cross validation** error.

the cross validation allows for finding the “just right” setting of the regularization parameter given the fixed model.

the performance of a learning algorithm on the training set will typically be better than its performance on the test set.

suppose you are training a regularized linear regression model. the recommended way to choose what value of regularization parameter λ to use is to choose the value of λ which gives the lowest **test set** error.

the test set should not be used to choose the regularization parameter, considering the model will have an artificially low value for test error and it will not provide good estimate of generalization error.

when debugging learning algorithms, it is useful to plot a learning curve to understand if there is a high bias or high variance problem. the shape of the learning curve is indicative of bias or variance within the learning algorithm.

a model with more parameters is more prone to overfitting and typically has higher variance. more model parameters increases the model’s complexity, tightly fitting it to the training data and increasing the chances of overfitting.

if a learning algorithm is suffering from high bias, only adding more training examples may **not** improve the test error significantly.

if a neural network has much lower training error than test error, then adding more layers will further increase the variance problem through an increased complexity in the model. the variance problem will become worse than before.

machine learning system design

topics and issues face when designing a machine learning system. strategizing how to put together a complex machine learning system

building a spam classifier

prioritizing what to work on

build a classifier with supervised learning to classify emails as spam (1) or not spam (0)
supervised learning:
x = features of email y = spam (1) or not spam (0)
features x: choose 100 words indicative of spam/not spam

e.g. "deal "buy" "discount" "address" "now" etc...

$$y = \begin{cases} 1 \text{ if work } j \text{ appears in email} \\ 0 \text{ otherwise} \end{cases}$$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \dots \\ 1 \\ \dots \end{bmatrix}$$

address
buy
deal
discount
...
now

$x \in \mathbb{R}^{100}$

```
From: cheapslaes@buystuffromme.com
To: ang@cs.stanford.edu
Subject: Buy Now!

Deal of the week! Buy now!
```

note: in practice, take the most frequently occurring n words (10,000 to 50,000) in the training set, opposed to manually selecting 100 words as above

spending time to lower the training error

- collect large amounts of data ("honeypot projects")
- develop sophisticated features based on email routing information (headers)
- develop sophisticated features from message body (e.g. decision to treat "discount" and "discounts" as the same word; "deal" and "dealer"; punctuation features etc...)
- develop sophisticated algorithm to detect misspellings (e.g. m0rtgage, med1cine, w4tches, etc...)

calculating many different features is possible but it is difficult to determine what features will be most useful in advance
using an analyst' "gut feeling" to choose varying ideas for algorithm development is not a best practice
it is not always advantageous to spend time gathering as much data is possible before trying to initially model the problem

error analysis

the concept of error analysis to systematically improve algorithms

example

m_{cv} = 500 examples in cross validation set

algorithm misclassifies 100 emails

user manually exams the 100 errors and categorizes them based on:

(i) the type of email (pharma, replica, steal passwords, etc…)

(ii) the cues (features) that would have helped the algorithm correctly classify

pharma:	12	→	deliberate misspellings:	5
replica/fake:	4		(m0rtgage, med1cine, etc…)	
steal passwords:	53	→	unusual email routing:	16
other:	31	→	unusual (spamming) punctuation:	32

the improtance of numerical evaluation

should "discount"/"discounts"/"discounted"/"discounting" be treated alike?

will the use of "stemmiung" software (e.g. porter stemmer) benefit?

"universe"/"university"

error analysis may not be helpful for deciding if such is likely to improve performance; only method for conclusion is trial and error

numerical evaluation (e.g. cross validation error) is needed for an algorithm's performance with and without stemming

without stemming: 5% error with stemming: 3% error

distinguish upper vs. lower case (Mom/mom): 3.2%

the recommended approach to perform error analysis is by using the cross validation data to compute $J_{cv}(\theta)$ rather than the test data to compute $J_{test}(\theta)$

if new features are developed by examining the test set, chosen features are likely to work well specifically for the test set; $J_{test}(\theta)$ is no longer a good estimate of how well the algorithm will generalize to new examples

build a "quick and dirty" implementation first

error metrics handling skewed data

error metrics for skewed classes

cancer classification example

train a logistic regression model $h_\theta(x)$ $\quad y = \begin{cases} 1 \text{ if cancer} \\ 0 \text{ otherwise} \end{cases}$

the test set results in 1% error (99% correct diagnoses')

only 50% of the patients contract cancer (the classes are skewed)

```
function y = predictCancer(x)
    y = 0; %ignore x!
return
```

→ 0.5% error

→ 99.2% accuracy (0.8% error)

→ 99.5% accuracy (0.5% error)

an alternative evaluation metric for the above problem

precision/recall

y = 1 in presence of rare class that is the target to detect

y = 0
recall = 0

predicted class \ actual class	**1**	**0**
1	true postives (+)	false positives (+)
0	false negatives (-)	true negatives (-)

precision

(of all patients where prediction = 1, what fraction actually has cancer?)

$$\frac{\text{true positives}}{\#\ \textbf{predicted}\ \text{positives}} = \frac{\text{true positives}}{\text{true positives} + \textbf{false positives}}$$

recall

(of all patients that actually have cancer, what fraction was correctly detected as having cancer?)

$$\frac{\text{true positives}}{\#\ \textbf{actual}\ \text{positives}} = \frac{\text{true positives}}{\text{true positives} + \textbf{false negatives}}$$

trading off precision and recall

logistic regression: $0 \leq h_\theta(x) \leq 1$

predict 1 if $h_\theta(x) \geq 0.5$ (default)

predict 0 if $h_\theta(x) < 0.5$ (default)

suppose prediction of $y = 1$ (cancer) only if very confident

predict 1 if $h_\theta(x) \geq 0.7$ or 0.9

predict 0 if $h_\theta(x) < 0.7$ or 0.9

→ the model will produce higher precision but lower recall

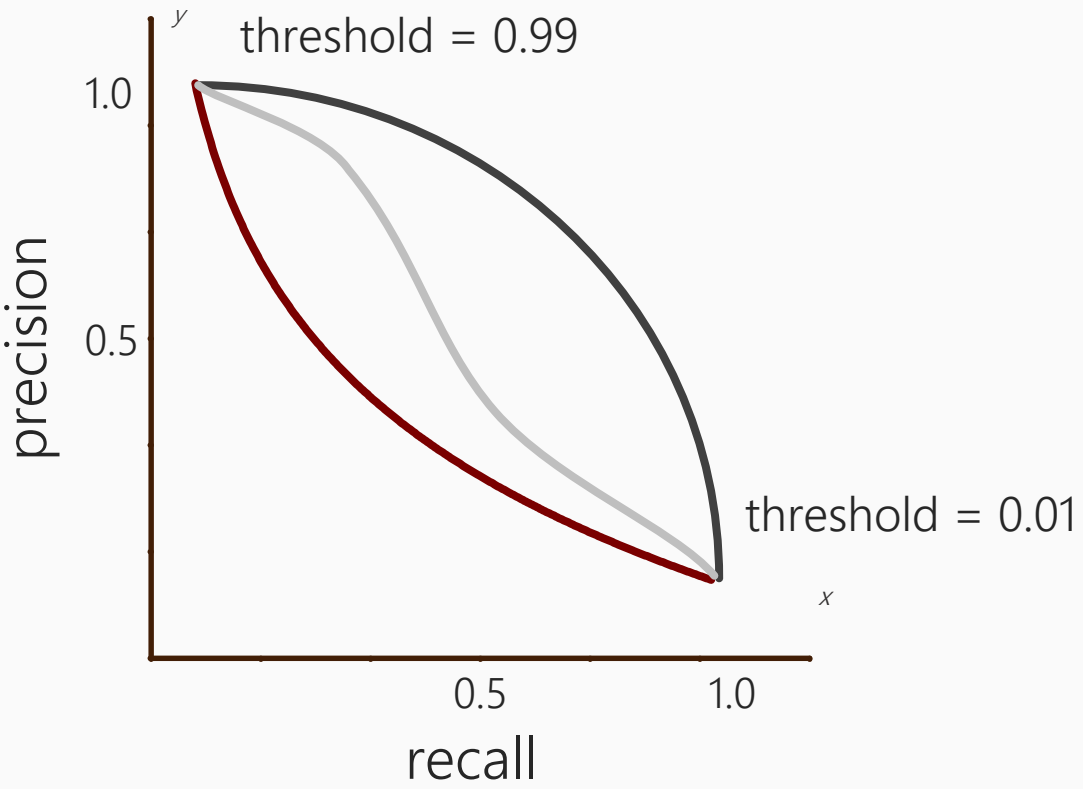
suppose avoidance to missing too many cases of cancer (avoid false negatives)

predict 1 if $h_\theta(x) \geq 0.7$ or 0.9

predict 0 if $h_\theta(x) < 0.7$ or 0.9

→ the model will produce lower precision but higher recall

in general terms: predict 1 if $h_\theta(x) \geq$ **given threshold**



F_1 score

comparing precision/recall metrics

precision and recall illustrate relationships among the two. using these metrics to select algorithms requires performance to be measured as a whole. the table below displays:

	precision (p)	**recall (r)**	~~**average**~~	**F_1 score**
algorithm 1	0.5	0.4	~~0.45~~	0.444
algorithm 2	0.7	0.1	~~0.4~~	0.175
algorithm 3	0.02	1.0	~~0.51~~	0.0392

average = $\frac{p+r}{2}$ F_1 score = $2\frac{pr}{p+r}$

the average of precision and recall dictate that the 3rd algorithm is most attractive. this is misleading because that model predicted y =1 for the entire sample (recall = 100%) and greatly skewed the average performance between precision and recall

the F_1 score applies a weight to the lower value, making it penalize the metric accordingly. the illustration above shows the first algorithm as the most effective and conversely the 3rd algorithm substantially worse than compared to the average metric.

if P = 0 or R = 0 → F_1 score = 0

if P = 1 or R = 1 → F_1 score = 1

example

a logistic c regression classifier has been trained as follows:

predict y = 1 if $h_\theta(x)$ ≥ threshold

predict y = 0 if $h_\theta(x)$ < threshold

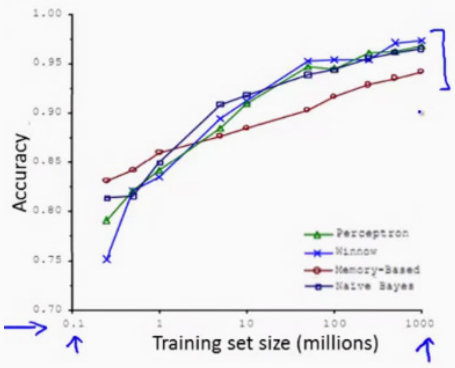
different values for the threshold parameter will yield different values of precision (p) and recall (r). a reasonable way to determine the threshold would be to:

measure precision (p) and recall (r) on the **cross validation set** and choose the value of threshold which maximizes the F_1 score = $2\frac{pr}{p+r}$

using large data sets

data for machine learning

determining how much data to train on



designing a high accuracy learning system

example classify between confusable words

{to, two, too} {then, than} {there, their}

→ "For breakfast I ate _two_ eggs."

the algorithms used to predict the word that belongs in the sentence above:

- perceptron (logistic regression)
- winnow
- memory-based
- naïve bayes

"it is not who has the best algorithm that wins, but who has the most data"

large data rationale

assuming feature $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately

example: "For breakfast I ate _two_ eggs."

counterexample" "Predict housing price from only the size (feet^2) as a feature."

useful test: given the input x, can a human expert confidently predict y?

using a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units).

→ low bias algorithms

$J_{train}(\theta)$ will be small

using a very large training set (unlikely to overfit)

→ low variance algorithms

$J_{train}(\theta) \approx J_{test}(\theta)$

$J_{test}(\theta)$ will be small

large training sets can help improve a learning algorithm's performance, except: when the features x do not contain enough information to predict y, regardless if the chosen algorithm is simple (logistic regression) or complex (neural networks)

Machine Learning System Design

1. You are working on a spam classification system using regularized logistic regression. "Spam" is a positive class (y = 1) and "not spam" is the negative class (y = 0). You have trained your classifier and there are m = 1000 examples in the cross-validation set. The chart of predicted class vs. actual class is:

	Actual Class: 1	Actual Class: 0
Predicted Class: 1	85	890
Predicted Class: 0	15	10

For reference:

- Accuracy = (true positives + true negatives) / (total examples)
- Precision = (true positives) / (true positives + false positives)
- Recall = (true positives) / (true positives + false negatives)
- *F*1 score = (2 * precision * recall) / (precision + recall)

What is the classifier's *F*1 score (as a value from 0 to 1)? 0.16

2. Suppose a massive dataset is available for training a learning algorithm. Training on a lot of data is likely to give good performance when:

The features *x* contain sufficient information to predict *y* accurately. (For example, one way to verify this is if a human expert on the domain can confidently predict *y* when given only *x*).

We train a learning algorithm with a large number of parameters (that is able to learn/represent fairly complex functions).

3. Suppose you have trained a logistic regression classifier which is outputing $h\theta(x)$.

Currently, you predict 1 if $h\theta(x) \geq$ threshold, and predict 0 if $h\theta(x) <$ threshold, where currently the threshold is set to 0.5.

Suppose you **increase** the threshold to 0.9:

The classifier is likely to now have higher precision.

4. Suppose you are working on a spam classifier, where spam emails are positive examples (y=1) and non-spam emails are negative examples (y=0). You have a training set of emails in which 99% of the emails are non-spam and the other 1% is spam:

If you always predict non-spam (output y=0), the classifier will have accuracy of 99%.

If you always predict non-spam (output y=0), your classifier will have 99% accuracy on the training set, and it will likely perform similarly on the cross validation set.

5. Using a **very large** training set makes it unlikely a model to overfit the training data.

On skewed datasets (e.g., when there are more positive examples than negative examples), accuracy is not a good measure of performance and should instead use *F*1 score based on the precision and recall.

support vector machines large margin classification

SVMs are considered to be the most powerful 'black box' learning algorithm, and by posing a cleverly-chosen optimization objective, one of the most widely used learning algorithms today

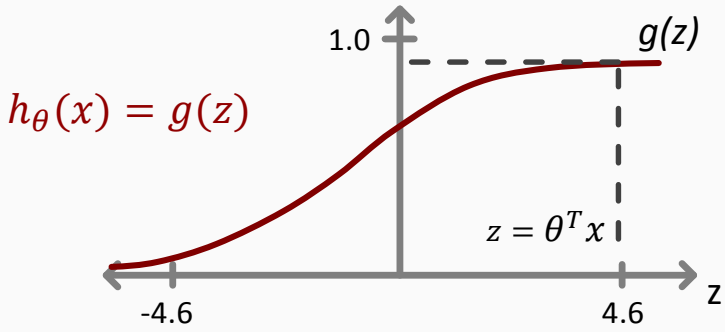
large margin classification

optimization objective

considerations among supervised learning algorithms are those of the amounts of data consumed and method in which they are implemented

compared to both logistic regression and neural networks, SVM can provide a cleaner and more powerful way of learning complex nonlinear functions

alternate view of logistic regression



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

$$z = \theta^T x$$

if $y = 1$, the desired output is $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

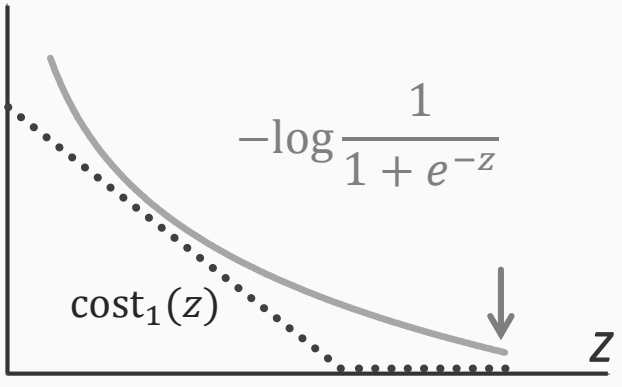
if $y = 0$, the desired output is $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

cost of example with a single observations (x, y):

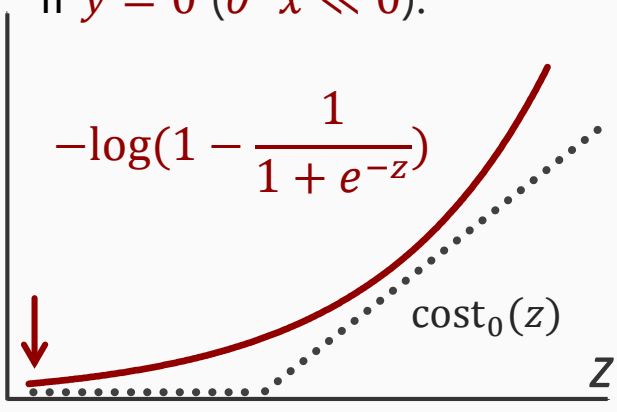
$$J(\theta) = -\left[y \log h_\theta(x) + (1 - y) \log\left(1 - h_\theta(x)\right)\right]$$

$$= -y \log \frac{1}{1 + e^{-\theta^T x}} - (1 - y) \log\left(1 - \frac{1}{1 + e^{-\theta^T x}}\right)$$

if $y = 1$ $(\theta^T x \gg 0)$:

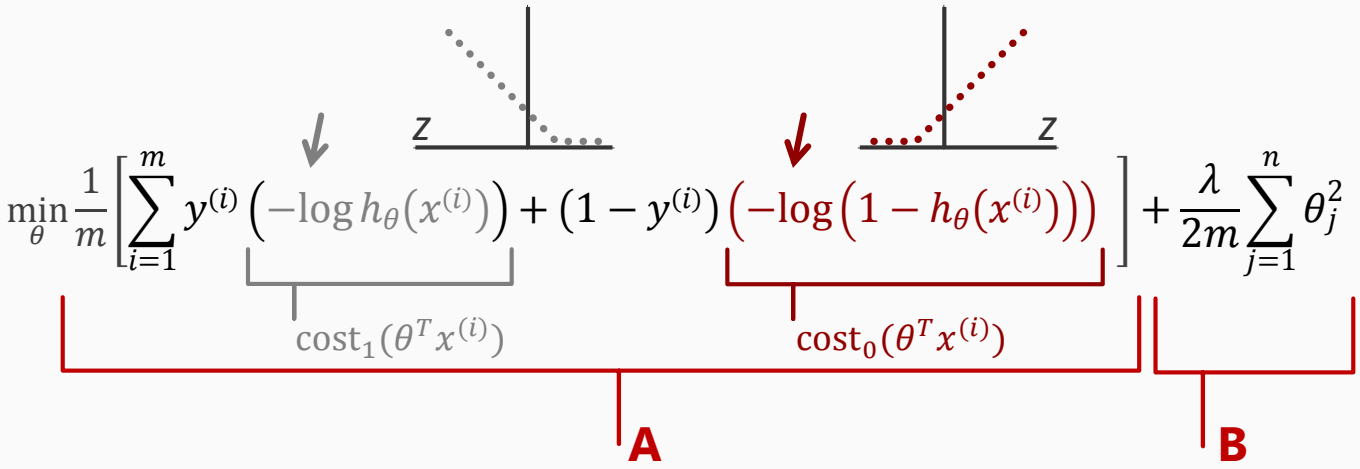


if $y = 0$ $(\theta^T x \ll 0)$:



support vector machines

logistic regression: $J(\theta) =$



support vector machine:

$$\min_\theta \cancel{\frac{1}{m}} \; \mathbf{C} \sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_0\left(\theta^T x^{(i)}\right) + \cancel{\frac{\mathbf{1}}{2m}}\sum_{j=1}^{n}\theta_j^2$$

svm will use a separate convention for applying $\frac{1}{m}$; removing them will result in the same optimal value of θ

example

$\min_4 \quad ((u-5)^2) \times 10 + 1 \quad \rightarrow u = 5$

$\min_4 \; 10(u-5)^2 \; + 10 \qquad \rightarrow u = 5$

trading off weights for svm

logistic regression → **A** + λ**B**

svm → **CA** + **B**

$$C = \frac{\mathbf{1}}{\lambda}$$

overall optimization objective function for support vector machine:

$$\min_\theta \frac{1}{m} C \sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + \left(1-y^{(i)}\right)\text{cost}_0\left(\theta^T x^{(i)}\right) + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

minimizing the above function will result in the parameters learned by the svm; unlike logistic regression, the svm will not output a probability

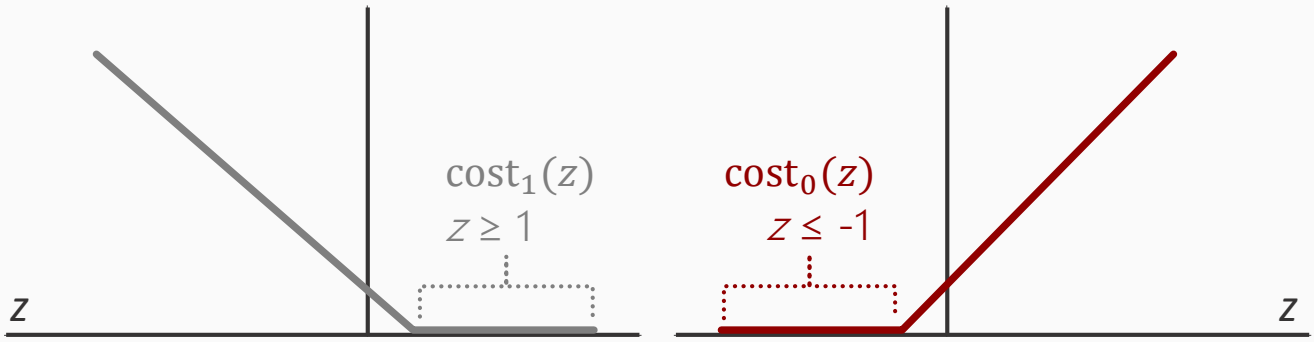
support vector machines output value of 1 or 0 after minimizing the cost function:

$$h_\theta x = \begin{cases} 1 \text{ if } \theta^T x \text{ is } \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

large margin intuition

support vector machines are sometimes referred to as large margin classifiers:

$$\min_\theta \frac{1}{m} C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0\left(\theta^T x^{(i)}\right) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$



if $y = 1$, the desired output is $\theta^T x \geq 1$ (not just ≥ 0) ~~$\theta^T x \geq 0$~~ 1

if $y = 0$, the desired output is $\theta^T x \leq -1$ (not just < 0) ~~$\theta^T x \leq 0$~~ -1

$\theta^T x$ ideally should be considerably larger than or less than 0; consequentially, $\theta^T x$ should be greater than +1 or less than -1: built in safety margin factor

svm decision boundary

$$\min_\theta \frac{1}{m} C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0\left(\theta^T x^{(i)}\right) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

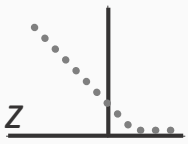
$= 0$

if C is a very large value (i.e. C = 100,000), the optimization objective will be highly motivated to make the first term in the objective = 0 during minimization

whenever $y^{(i)} = 1$:

to ensure $\text{cost}_1(z)$ then: $\theta^T x \geq 1$

~~$\min C \times 0$~~ $+ \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$

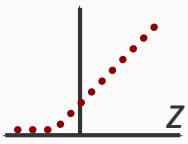


whenever $y^{(i)} = 0$

to ensure $\text{cost}_0(z)$ then: $\theta^T x \leq -1$

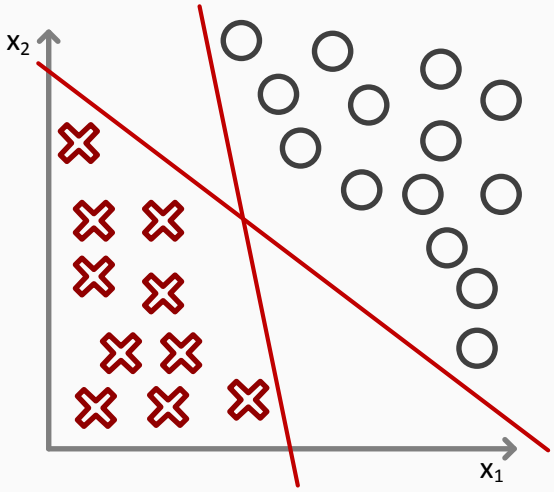
subject to: $\theta^T x \geq 1$ if $y^{(i)} = 1$

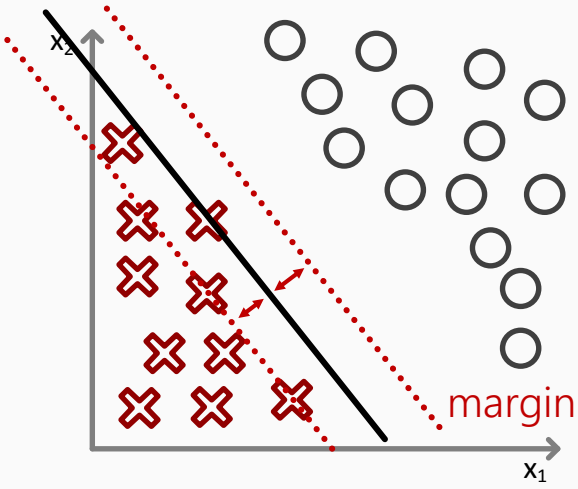
and $\theta^T x \leq -1$ if $y^{(i)} = 0$



svm decision boundary: linearly separable case

linear boundaries can exist as many functions to classify the data as shown below (left):

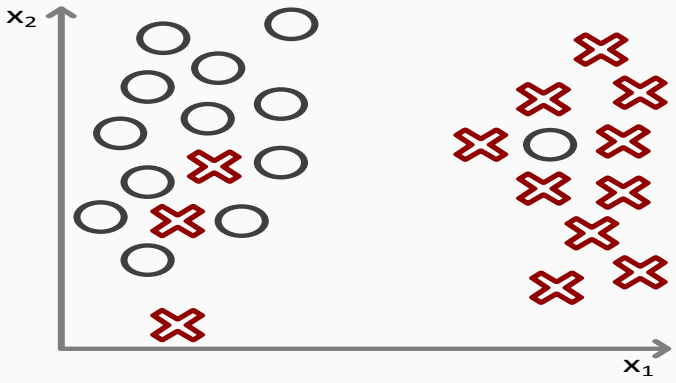




the above examples (left) are not natural boundaries and are typically not good choices for a learning algorithm. support vector machines will instead determine a more robust decision boundary (right). the svm decision boundary performs better at separating the positive(+) and negative(-) examples than the latter. mathematically, the svm has a much larger margin (shown in dotted line) which represents some minimum distance from any of the training examples: thus referred to as a large margin classifier

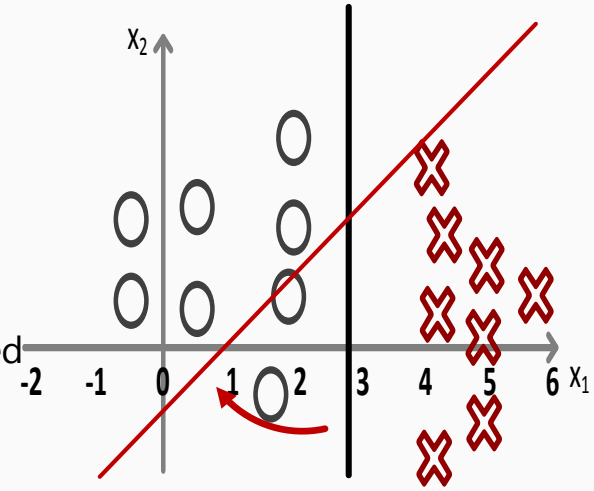
example
in the training set (right), "x" denotes positive examples (y=1) and "o" denotes negative examples (y=0). training an svm to predict 1 when $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$ might return values for θ_0, θ_1, and θ_2 of:

$$\theta_0 = -3, \theta_1 = 1, \text{and } \theta_2 = 0$$



large margin classifier in presence of outliers

if C is very large, the svm will adopt the new decision boundary (red line). a small or nominal C value with maintain the optimal decision boundary (black line). the svm will also choose the optimal decision boundary in the presence of nonlinear data (outliers  within the clustered data). note: C plays a role similar to $\frac{1}{\lambda}$



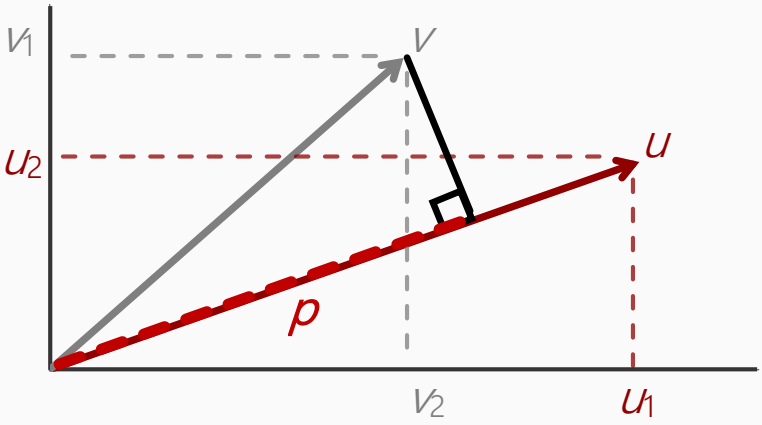
mathematics behind large margin classification

how the optimization problem of support vector machines lead to large margin classifiers

vector inner product

with two vectors u and v:

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$



$u^T v = ?$

$\|u\| = \text{eucledian length of vector } u$

$= \sqrt{u_1^2 + u_2^2} \quad \in \mathbb{R}$

take an orthogonal (90°) projection of *v* down to *u* and measure the resulting length (red dashed line) of ***p***

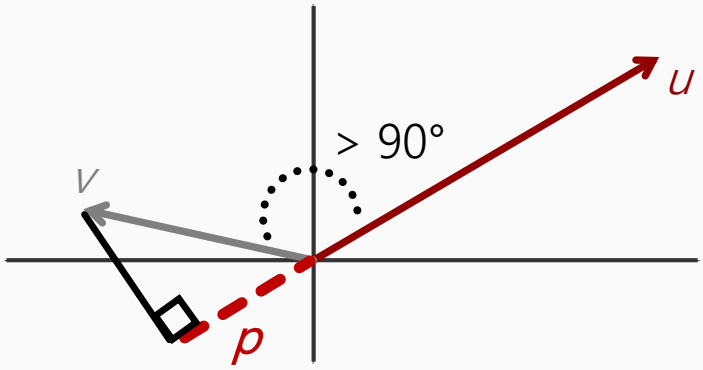
p = length of the projection *v* onto *u*

$u^T v = \boldsymbol{p} \times \|u\| \qquad \boldsymbol{p} \in \mathbb{R}$

$= u_1 v_1 + u_2 v_2$

computing $v^T u$ would yield same result

note that ***p*** can be negative (shown left)



svm decision boundary

$$\min_\theta \frac{1}{2}\sum_{j=1}^{n} \theta_j^2 = \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$

$= \|\theta\|$

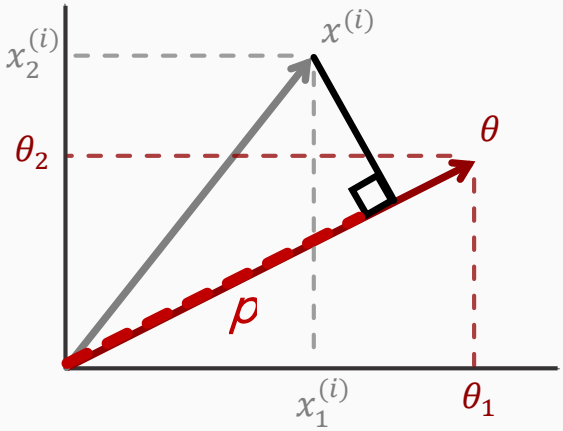
subject to: $\theta^T x \geq 1$ if $y^{(i)} = 1$

and $\theta^T x \leq -1$ if $y^{(i)} = 0$

simplification: $\theta_0 = 0$, $n = 2$

$$\theta^T x^{(i)} = \boldsymbol{p}^{(i)} \times \|\theta\|$$

$$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$$



$u^T v$

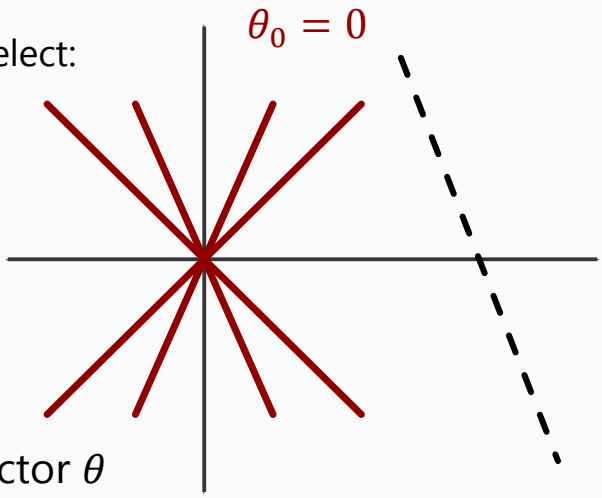
$\theta^T x^{(i)} = ?$

considering the training example below with simplification $\theta_0 = 0$

the simplification only utilizes svm decision boundaries that pass through the origin

determine the decision boundary that the svm will select:

$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n} \theta_j^2 = \frac{1}{2}\|\theta\|^2$$

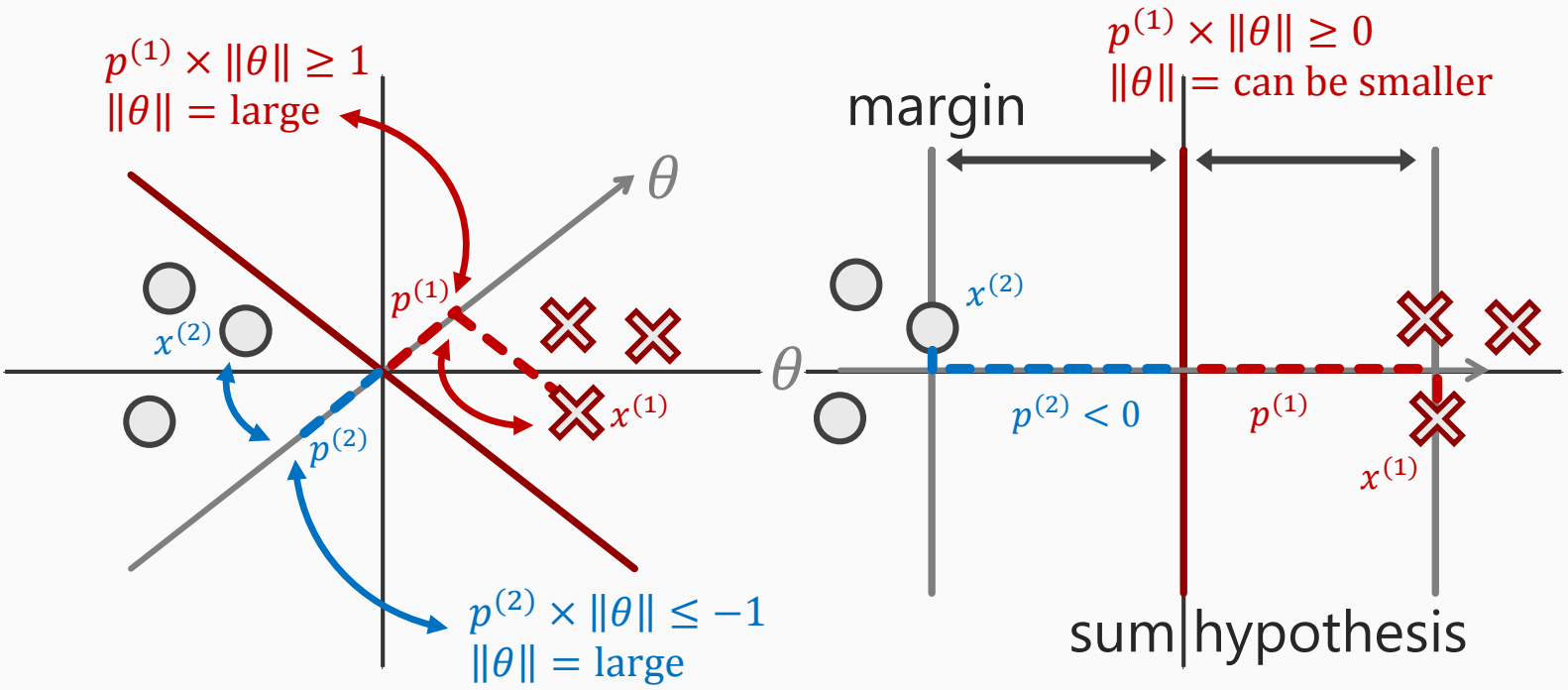


subject to: $p^{(i)} \times \|\theta\| \geq 1$ if $y^{(i)} = 1$

$p^{(i)} \times \|\theta\| \leq 1$ if $y^{(i)} = -1$

where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ

with simplification for illustration $\theta_0 = 0$ (intersects origin as seen above)

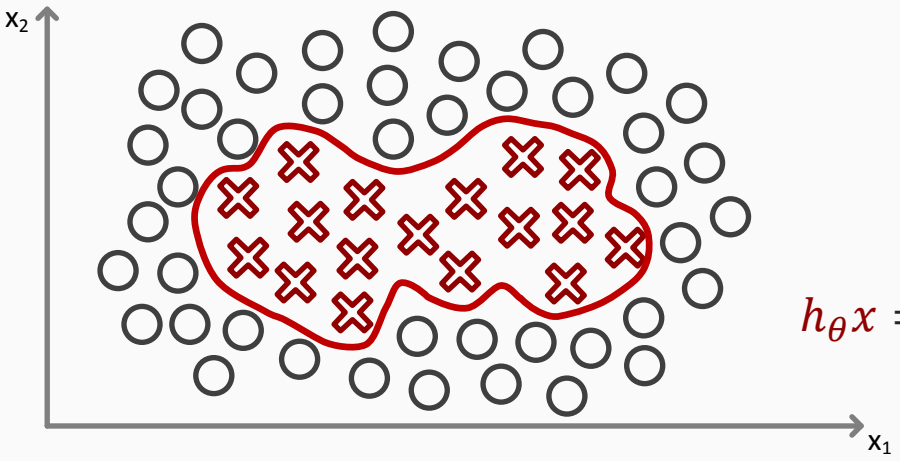


the red line in the left figure represents a poor decision boundary choice that is avoided through the use of svm. the grey line represents the svm tactics through projection from the points $x^{(1)}$ and $x^{(2)}$. in this example, $\|\theta\|$ and $\|\theta\|$ would need to be large considering that $p^{(1)}$ and $p^{(2)}$ are small. the red line in the right figure represents a correct decision boundary choice that is achieved through the use of svm. it is shown the projections from the data $x^{(1)}$ and $x^{(2)}$ create much larger lengths of $p^{(1)}$ and $p^{(2)}$ creating a larger margin which all for a smaller value for $\|\theta\|$ and $\|\theta\|$

support vector machines kernels

kernels i & ii

nonlinear decision boundary



predict $y = 1$ if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

$$h_\theta x = \begin{cases} 1 \text{ if } \theta_0 + \theta_1 x_1 \ldots \text{ is } \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

the technique to adapt support vector machines to complex nonlinear classifiers is achieved through the use of kernels

an alternative method to write the above notation is as follows:

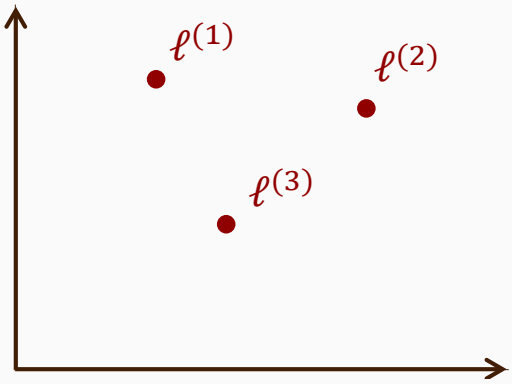
$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \theta_4 f_4 + \theta_5 f_5 + \cdots$$

$$f_1 = x_1,\ f_2 = x_2,\ f_3 = x_1 x_2,\ f_4 = x_1^2,\ f_5 = x_2^2,\ \ldots$$

the question is whether there is a better/alternative choice for features $f_1,\ f_2,\ f_3,\ \ldots$

Kernel example based on three manually chosen points $\ell^{(1)}$, $\ell^{(2)}$, $\ell^{(3)}$

given x, compute new feature depending on proximity to landmarks $\ell^{(1)}$, $\ell^{(2)}$, $\ell^{(3)}$



given x:

$$f_1 = \text{similarity}\left(x, \ell^{(1)}\right) = \exp\left(-\frac{\left\|x - \ell^{(1)}\right\|^2}{2\sigma^2}\right)$$

$$f_2 = \text{similarity}\left(x, \ell^{(1)}\right) = \exp\left(-\frac{\left\|x - \ell^{(1)}\right\|^2}{2\sigma^2}\right)$$

$$f_3 = \text{similarity}\left(x, \ell^{(1)}\right) = \exp(\ldots)$$

the functions above referred to as: **kernels** and **gaussian kernels**, denoted → $\mathrm{k}\left(x, \ell^{(i)}\right)$

kernels and similarity

the similarity between x and $\ell^{(1)}$ is given by the following expression:

$$f_1 = \text{similarity}(x, \ell^{(1)}) = \exp\left(-\frac{\left\|x - \ell^{(1)}\right\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}\left(x_j - \ell_j^{(1)}\right)^2}{2\sigma^2}\right)$$

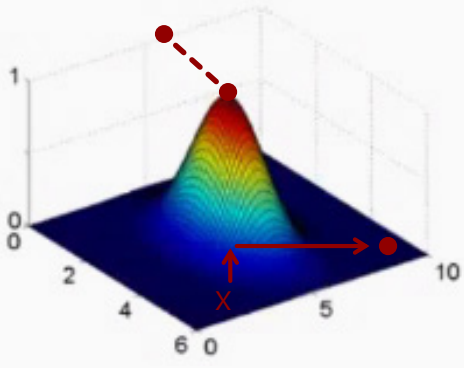
the numerator term is equal to the component wise distance between vector X and I

if $x \approx \ell^{(1)}$: $f_1 = \exp\left(-\frac{(0)^2}{2\sigma^2}\right) \approx 1$

if x is far from $\ell^{(1)}$: $f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0$

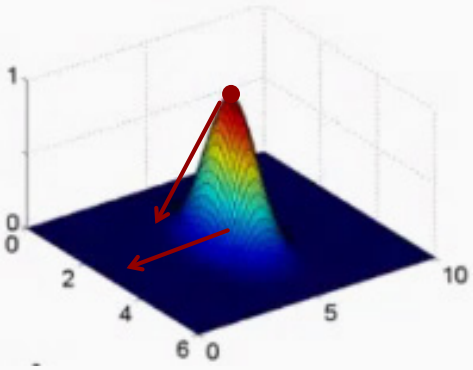
example

$\ell^{(1)} = \begin{bmatrix}3\\5\end{bmatrix}, f_1 = \exp\left(-\frac{\|x-\ell^{(1)}\|^2}{2\sigma^2}\right)$ and setting $\sigma^2 = 1$



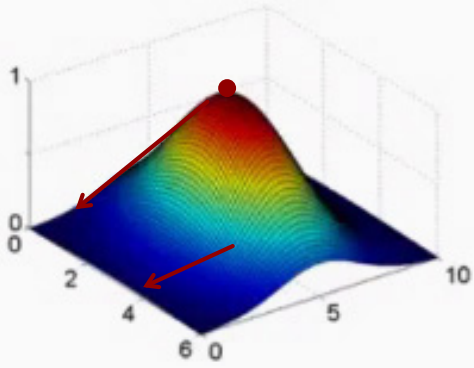
$\sigma^2 = 1$

the of a point x on the contour plot represents the value of f. when x = exactly, f = 1. when x moves farther away from $\ell^{(1)}$, f ≈ 0



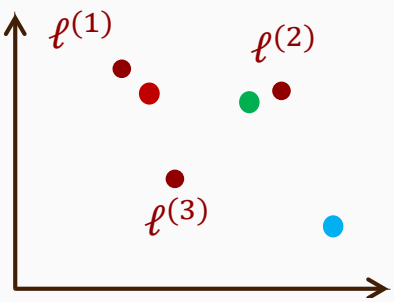
$\sigma^2 = 0.5$

as x moves away from $\ell^{(1)}$, the value of the feature falls away more quickly because the point of the contour is steep



$\sigma^2 = 3$

as x moves away from $\ell^{(1)}$, the value of the feature falls away more slowly because the point of the contour is wider



predict "1" when $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

supposed outputs of θ: $\theta_0 = -0.5,\ \theta_0 = 1,\ \theta_0 = 1,\ \theta_0 = 3$

$f_1 \approx 1 \quad f_2 \approx 0 \quad f_3 \approx 0$

$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$ = -0.5 + 1 → 0.5 ≥ 0

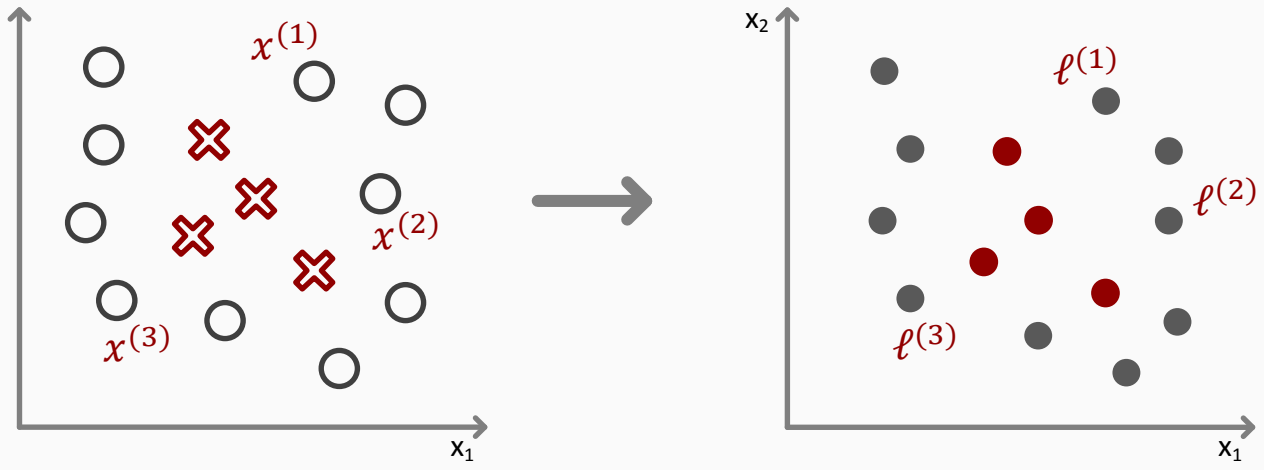
$f_1 \approx 0 \quad f_2 \approx 0 \quad f_3 \approx 0$

$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$ → -0.5 ≤ 0

green point has similar expected output as red training data

choosing the landmarks

initially place landmarks at exactly each point of the training examples in a dataset



support vector machines with kernels

given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

choose $\ell^{(1)} = x^{(1)}, \ell^{(2)} = x^{(2)}, \ldots, \ell^{(m)} = x^{(m)}$

given example x:

$$f_1 = \text{similarity}(x, \ell^{(1)})$$

$$f_1 = \text{similarity}(x, \ell^{(1)})$$

$$\vdots$$

grouped into vector f

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \qquad f_0 = 1$$

for training example $(x^{(i)}, y^{(i)})$:

$$f_1^{(i)} = \text{similarity}(x^{(i)}, \ell^{(1)})$$

$$f_2^{(i)} = \text{similarity}(x^{(i)}, \ell^{(2)})$$

$$\vdots \longrightarrow f_i^{(i)} = \text{similarity}(x^{(i)}, \ell^{(i)}) = \exp\left(-\frac{0}{2\sigma^2}\right) = \mathbf{1}$$

$$\vdots$$

$$f_1^{(m)} = \text{similarity}(x^{(i)}, \ell^{(m)})$$

$$x^{(i)} \in \mathbb{R}^{n+1}$$

$$f^{(i)} = \begin{bmatrix} f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

$$f_0^{(i)} = 1$$

making predications given an obtained set of parameters of θ

hypothesis: given x, compute features $f \in \mathbb{R}^{m+1}$

predict "y=1" if $\theta^T f \geq 0$

in order to get the set of parameters of θ

training:

$$\min_{\theta} \frac{1}{m} C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + \left(1 - y^{(i)}\right) \text{cost}_0\left(\theta^T f^{(i)}\right) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

~~$\theta^T x^{(i)}$~~

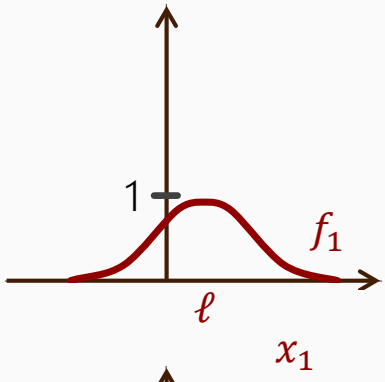
support vector machine parameters

c$\left(=\frac{1}{\lambda}\right)$ large C → **lower bias, higher variance** (small λ)

small C → **higher bias, lower variance** (large λ)

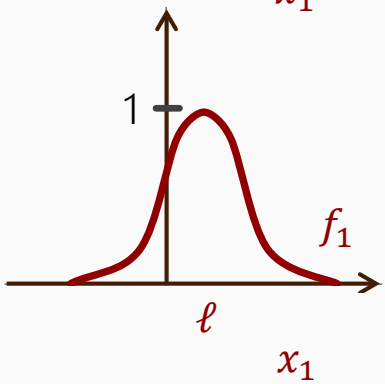
σ^2 large σ^2: features vary more smoothly

→ **higher bias, lower variance**



large σ^2: features vary less smoothly

→ **lower bias, higher variance**



during training an svm and the model is overfitting the training data. a reasonable step would be to either decrease C and/or decrease σ^2

svms in practice

using support vector machines

svm software packages (e.g. liblinear, libsvm, ...) to solve for parameters θ

specifications with a library package:

choice of parameter C

çhoice of kernel (similiarity function):

no kernel ("linear kernel") $\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

predict "y=1" if $\theta^T f \geq 0$ n = **large**, m = **small** $x^{(i)} \in \mathbb{R}^{n+1}$

linear kernels are a good option when the dataset is large but only a small amount of training data. fitting a nonlinear function with n = **large,** m = **small** is likely to overfit

gaussian kernel

$f_i = \exp\left(-\frac{\|x-\ell^{(i)}\|^2}{2\sigma^2}\right)$ where $\ell^{(i)} = x^{(i)}$ $x^{(i)} \in \mathbb{R}^{n+1}$

need to choose σ^2 n = **small** and/or m = **large**

gaussian kernels are a good option when the dataset smaller and/or there is a substantial amount of training data. an svm can better fit a complex nonlinear function

kernel (similarity) functions

```
function f = kernel (x1,x2)
```

$$f_i = \exp\left(-\frac{\|x1 - x2\|^2}{2\sigma^2}\right)$$

```
return
```

it is important to perform feature scaling prior to using a gaussian kernel:

$\|x - \ell^{(1)}\|^2$ $v = x - \ell$ $x \in \mathbb{R}^{n+1}$

$$\|v\|^2 = v_1^2 + v_2^2 + \cdots + v_n^2$$

$$= (x_1 - \ell_1)^2 + (x_2 - \ell_2)^2 + \cdots + (x_n - \ell_n)^2$$

multiple scales → 10s 1,000,000s

features that take on vast ranges of values, the computation will suffer by overweighed functions with higher scales than others. this will ensure the svm applies the proper attention to each feature appropriately

other choices of kernel

note: not all similarity functions $\text{similarity}(x, \ell)$ make valid kernels. (need to satisfy technical condition called "mercer's theorem" to ensure svm packages' optimizations run correctly and do not diverge the theorem ensures that support vector machine software packages can use a large class of optimizations and arrive at the parameter θ quickly and efficiently

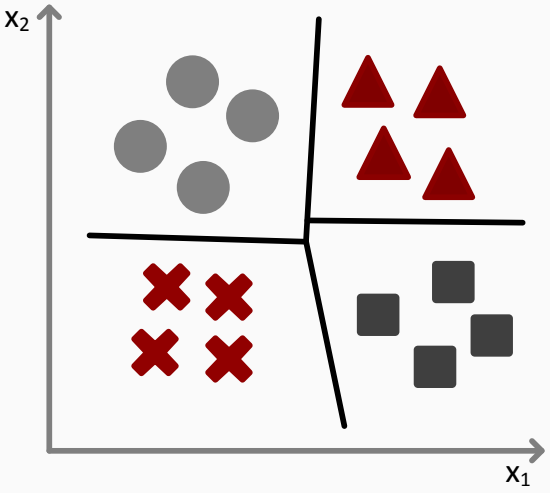
multiple available kernel packages:

polynomial kernel → $k(x, \ell) = (x^T \ell + \mathbf{0})^{\mathbf{2}}$ → $(x^T \ell + \mathbf{constant})^{\text{degree}}$

more esoteric: string kernel, chi-square kernel, histogram, intersection, …

under consideration of a kernel along with parameters such as **C, σ^2**, etc… the kernel choice should be influenced by the best **cross-validation dataset** performance

multiclass classification



$$y \in \{1, 2, 3, \ldots, K\}$$

many svm packages have built-in classification functionality

otherwise, utilize one-vs-all method (train K svms, one to distinguish svms, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^1, \theta^2, \ldots, \theta^K$

pick class i with largest $\left(\theta^{(i)}\right)^T x$

logistic regression versus support vector machines

n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

if n is **large** (relative to m): (e.g. $n \geq m$, n = 10,000, m = 10…1000)
→ logistic regression, or svm without a kernel ("linear kernel")

if n is **small**, m is **intermediate**: (e.g. n= 1-1000, m = 10-10,000)
→ svm with a guassian kernel

if n is **small**, m is **large**: (e.g. n= 1-1000, m = 50,000+)
→ create/add more features, then apply logistics regression or svm without a kernel

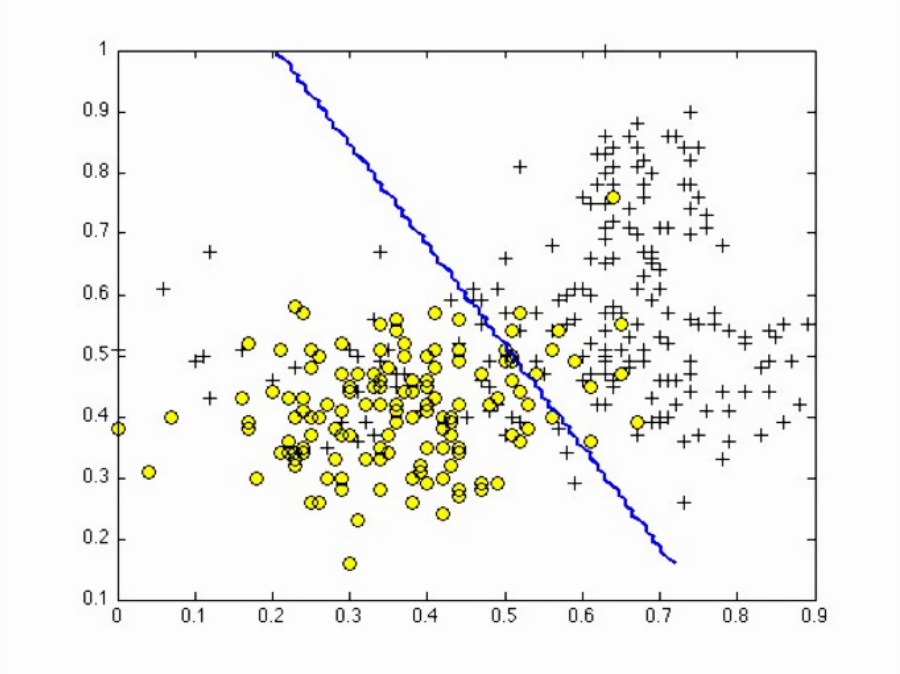
support vector machines and logistic regression are similar in effect but sometimes one will be more efficient or optimized for the data than the other

nueral networks will likely functions with most of the above settings but slower to train

the optimization problem that an svm has is a convex optimization problem; the software packages will consequentially always find the global minimum (or close to)

support vector machines

1. Suppose you have trained an SVM classifier with a Gaussian kernel, and it learned the following decision boundary on the training set:



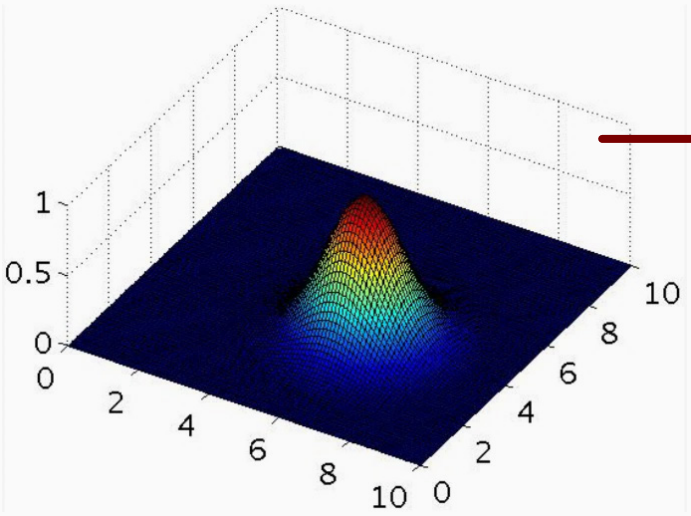
You suspect that the SVM is underfitting your dataset. Should you try increasing or decreasing C? Increasing or decreasing σ^2?

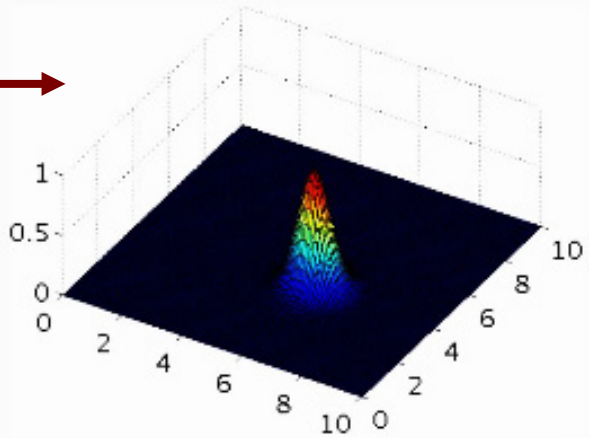
- It would be reasonable to try **increasing** C. It would also be reasonable to try **decreasing** σ^2.

The figure shows a decision boundary that is underfit to the training set, so we'd like to lower the bias / increase the variance of the SVM. We can do so by either increasing the parameter C or decreasing σ^2.

2. The formula for the Gaussian kernel is given by $\text{similarity}(x, l^{(1)}) = \exp\left(-\frac{||x-l^{(1)}||^2}{2\sigma^2}\right)$.

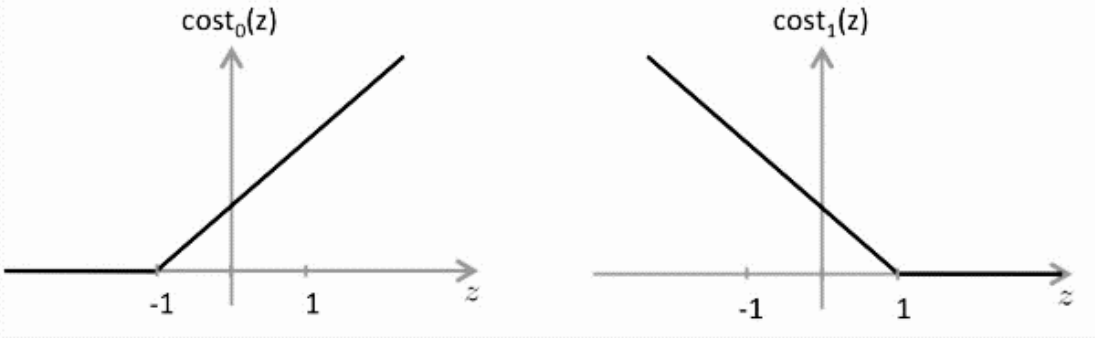
The figure below shows a plot of $f_1 = \text{similarity}(x, l^{(1)})$ when $\sigma^2 = 1$.





This figure shows a "narrower" Gaussian kernel centered at the same location which is the effect of decreasing σ^2.

The SVM solves $\min_\theta \; C\sum_{i=1}^{m} y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)}) + \sum_{j=1}^{n}\theta_j^2$ where the functions $\text{cost}_0(z)$ and $\text{cost}_1(z)$ look like this:



The first term in the objective is: $C\sum_{i=1}^{m} y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)})$.

This first term will be zero if two of the following four conditions hold true. Which are the two conditions that would guarantee that this term equals zero?

3.

for every example where $y^{(i)} = 1$, $\theta^T x^{(i)} \geq 1$
for every example where $y^{(i)} = 0$, $\theta^T x^{(i)} \leq -1$

4. Suppose you have a dataset with n = 10 features and m = 5000 examples.

After training your logistic regression classifier with gradient descent, you find that it has underfit the training set and does not achieve the desired performance on the training or cross validation sets.

try using a neural network with a large number of hidden units
create/add new polynomial features
reduce the number of examples in the training data

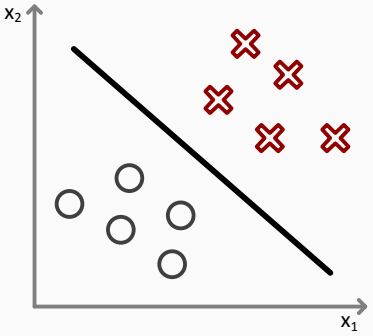
5. Which of the following statements are true? Check all that apply.

it is important to perform feature normalization before using a gaussian kernel
the maximum value of the gaussian kernel ($\text{similarity}(x, \ell^{(1)})$) is 1
2D input examples (i.e. $x^{(i)} \in \mathbb{R}^2$) plots a linear decision boundary as a straight line (linear kernel)

unsupervised learning k-means basics

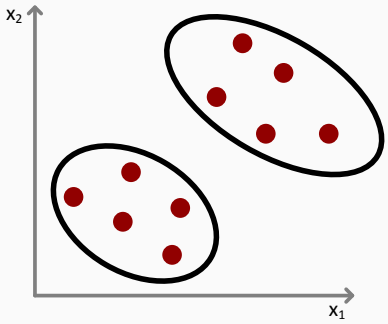
clustering

supervised learning



$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

unsupervised learning

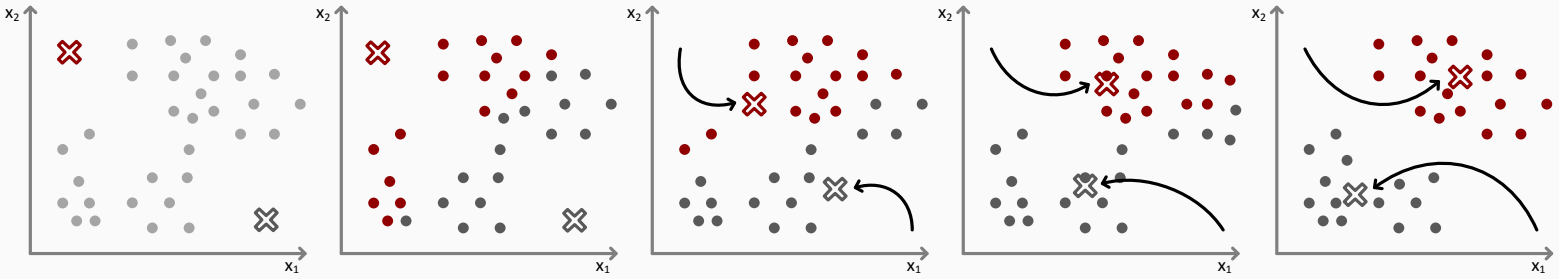


$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

supervised learning algorithms provide training datasets with pairs of known outputs to model off of. unsupervised learning only provides a set of single observations for algorithms like clustering methods are used to find patterns or "structure" in the data

k-means algorithm

iterative algorithm that assigns clusters and adjusts initially randomized cluster centroids to the mean



once the algorithm has converged to the mean, the centroids will no longer adjust or assign color

input:

K (number of clusters)

training set $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention $x^{(i)} \in \mathbb{R}^{n+1}$)

randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

repeat {

cluster assignment

for $i = 1$ to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

$$c^{(i)} = \min_k \|x^{(i)} - u_k\|^2$$

move centroid

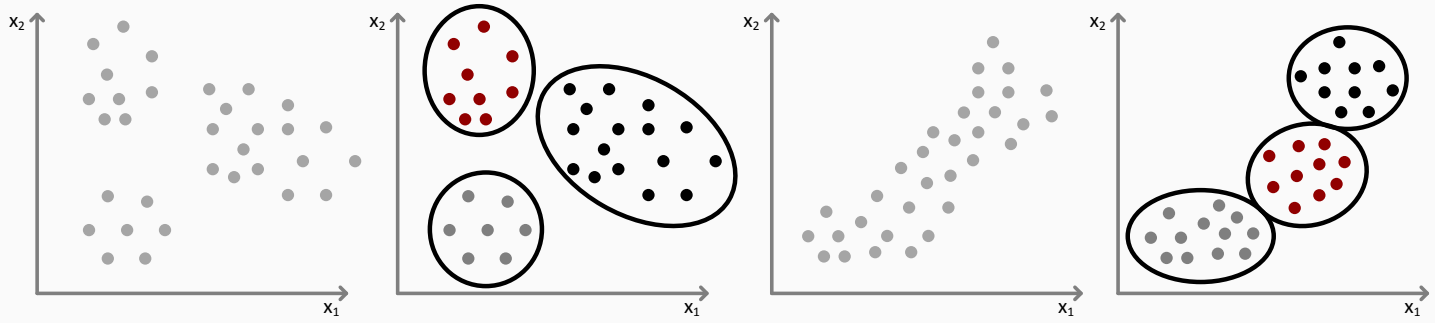
for $k = 1$ to K

μ_k := average (mean) of points assigned to cluster k

}

average distance: $\mu_k = \frac{1}{k}\left[x_1^{(i)} + x_2^{(i)} + x_3^{(i)} + \cdots\right] \in \mathbb{R}^n$

k-means for non-separated clusters



optimization objective of k-means

$c^{(i)}$ = index of cluster (1, 2, …, K) to which example $x^{(i)}$ has been assigned

μ_k = cluster of centroid k $(\mu_K \in \mathbb{R}^n)$

$x^{(i)} \rightarrow 5 \quad c^{(i)} = 5 \quad \mu_c{}^{(i)} = \mu_5$

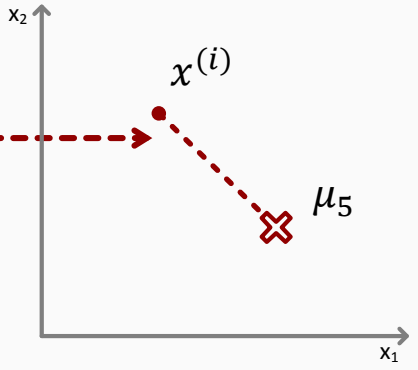
$\mu_c{}^{(i)}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

$$J\left(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - \mu_c{}^{(i)}\right\|^2$$

distance

$$\min_{\substack{c^{(1)},\dots,c^{(m)},\\ \mu_1,\dots,\mu_K}} J\left(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K\right)$$

cost distortion function



randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

repeat **{**

cluster assignment:
for $i = 1$ to m
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

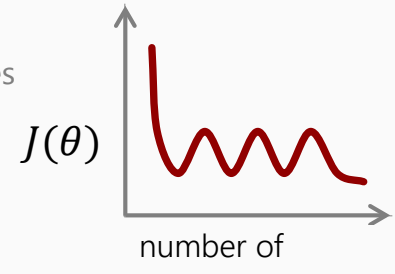
move centroid:
for $k = 1$ to K
μ_k := average (mean) of points assigned to cluster k

}

minimize $J(\dots)$ with respect to the variables $c^{(1)}, c^{(2)}, \dots, c^{(m)}$ while holding $\mu_1, \dots, \mu_K$ fixed

minimize $J(\dots)$ with respect to the variables $\mu_1, \dots, \mu_K$

while running k-means, it is not possible for the cost function to sometimes increase (right illustration); the code should be debugged for errors



random initialization of k-means algorithm

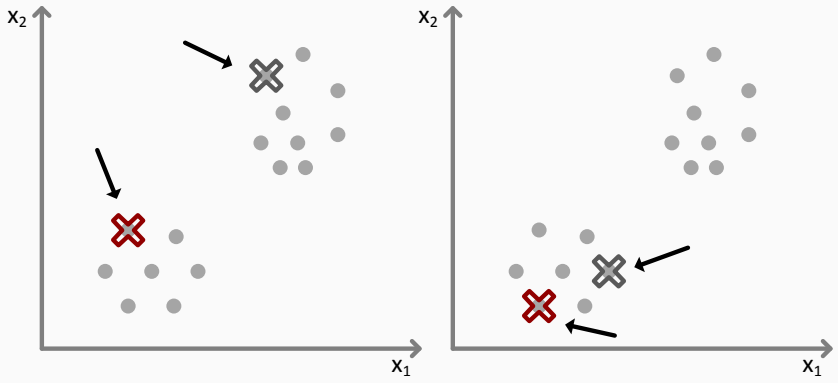
should have $K < m$

$k = 2$

randomly pick K training examples

set $\mu_1, \dots, \mu_K$ equal to the K examples

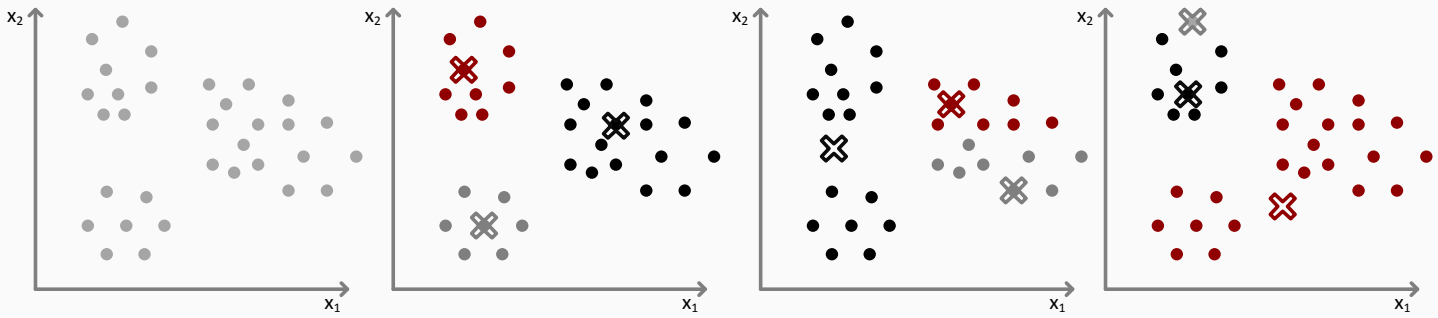
$\mu_1 = x^{(i)}$
$\mu_1 = x^{(i)}$
$\vdots$



local optima

k means risks local optima when minimizing the cost distortion function:
$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$ depending on initial randomization of k

ideal k-means | global optima | local optima | local optima



multiple iterations of random initialization can defend against local optima

for i = 1 to 100 **{**

randomly initialize k-means
run k-means and get $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$
compute cost function (distortion)
$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

}

the resulting process provides 100 different ways of clustering the data. the clustering method that returns lowest cost $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$ will provide the best optima when ***k* is small** (e.g. k = 2-10) then multiple random initializations have a high probability of increasing the effectiveness of clustering optima. it ***k* is large**, there is a better chance the first initial randomization will provide a decent local/global optima.

recommended initialization of k-means:

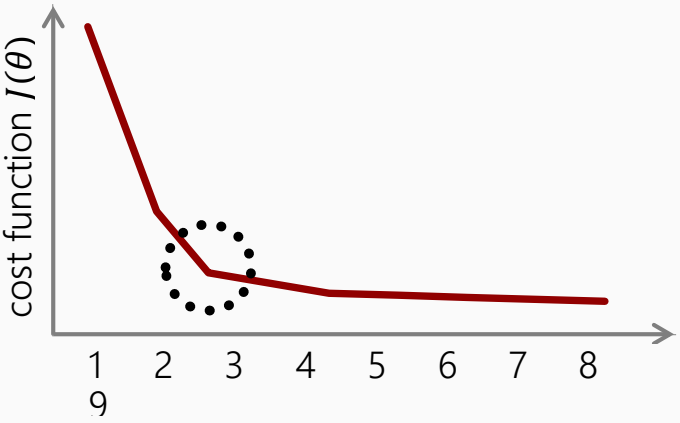
pick k distinct random integers $i_1, \dots, i_k$, from $\{1, \dots, m\}$
set $u_1 = x^{(i_1)}, u_2 = x^{(i_2)}, \dots, u_k = x^{(i_k)}$

choosing the number of clusters

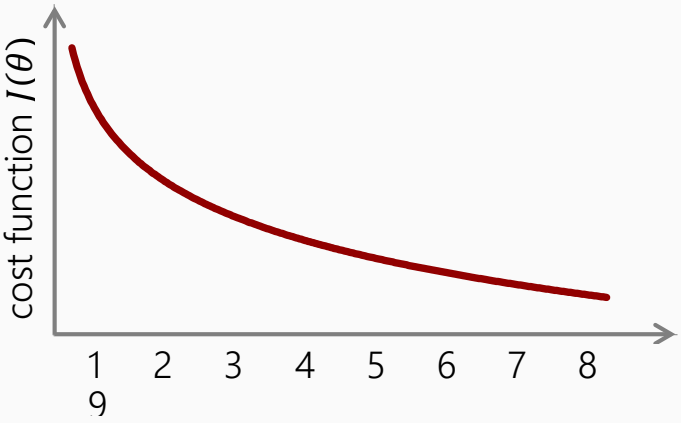
elbow method:

the process of choosing an appropriate amount of k-clusters where the cost function $J(\theta)$ ceases to significantly improve with each additional iteration with new clusters.

in theory



in practice

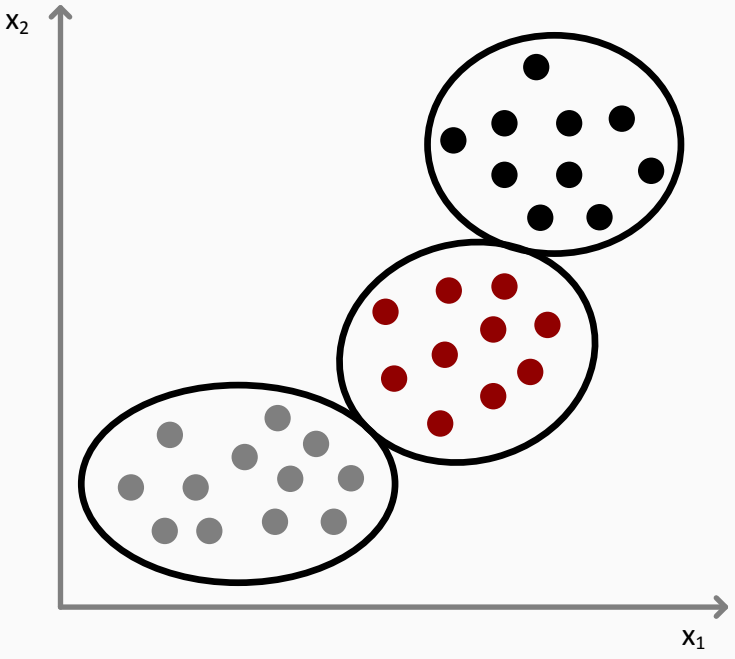


choosing the value of *K*

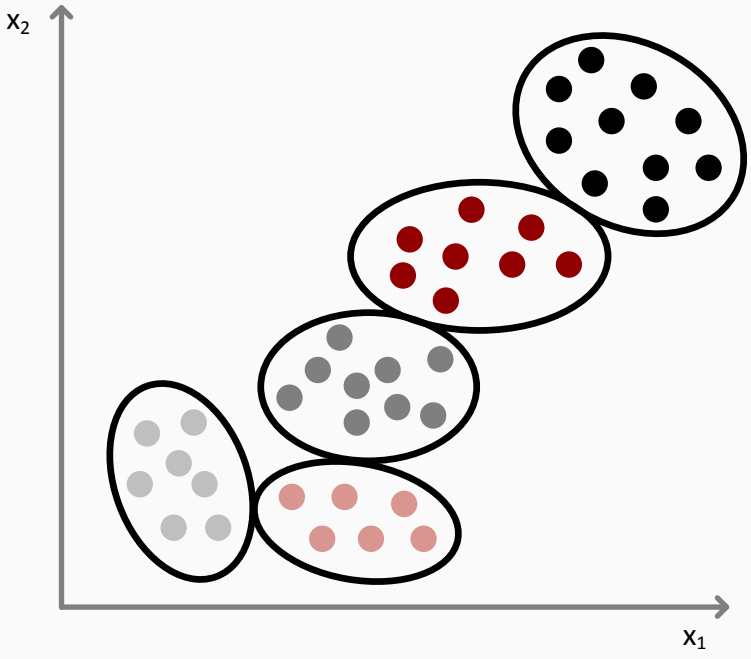
evaluating *K*-means based on a metric for how well it performs on downstream area

using domain knowledge about the purpose in running k-means can provide insight on how many clusters should be appropriately expected

$K = 3$



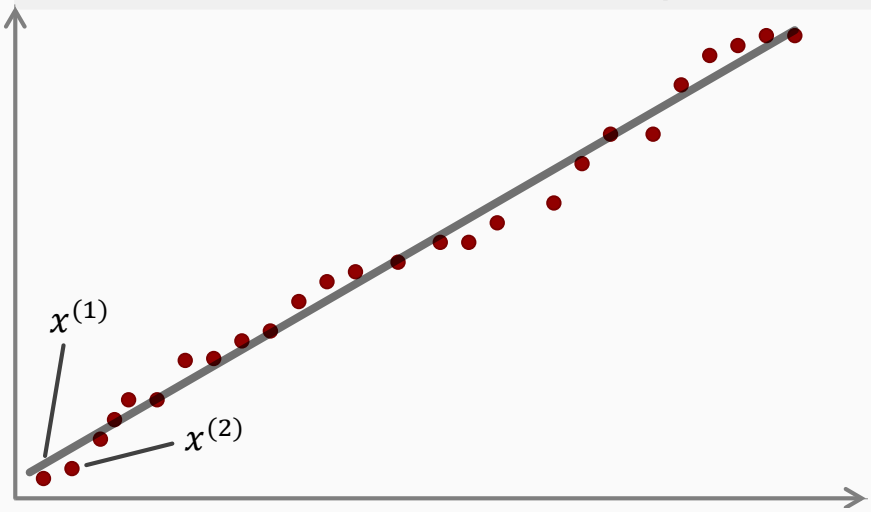
$K = 5$



if data portrays a certain amount of expected known clusters, *K* should reflect such

dimensionality reduction 🗜 principal components

motivation i & ii: data compression and visualization



two features in a dataset that are highly correlated (multicollinearity) create redundancy in the data and do not add predictive power to models considering the correlated variables represent essentially similar data.

(e.g. a unit in $ dollars or € euros)

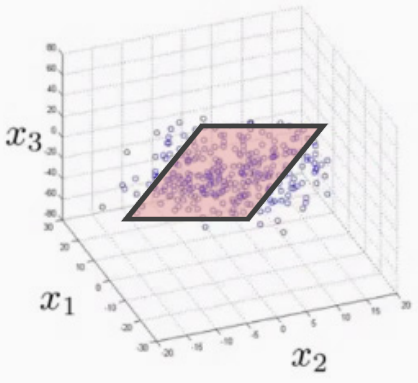
reducing the data from 2D to 1D:

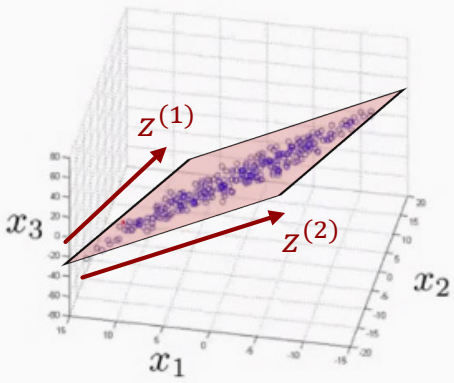
projecting all of the correlated data points on a straight line will provide a basis to measure the distance between each example. a new feature can be created ($z^{(1)}$) that specifies the location of each point on the original linear trend. only one number from each of the plotted examples will need to be retained to meet linear requirements. this will in effect, optimize algorithms for computational resources required.

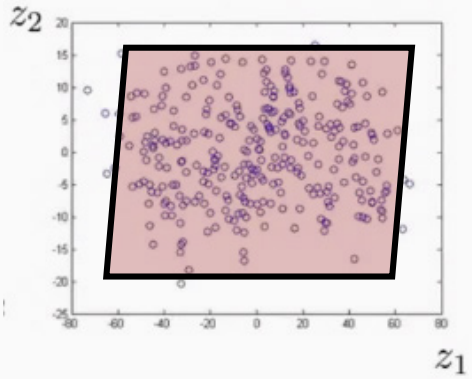
$$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$$
$$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$$
$$\vdots$$
$$x^{(m)} \in \mathbb{R}^2 \rightarrow z^{(m)} \in \mathbb{R}$$

reducing the data from 3D to 2D:

$x^{(i)} \in \mathbb{R}^3$







$z^{(i)} \in \mathbb{R}^2$

applying dimensionality reduction to a dataset of m examples $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, where $x^{(i)} \in \mathbb{R}^n$ will result in a lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ of m examples where $z^{(i)} \in \mathbb{R}^k$ for some value of k and $k \leq n$.

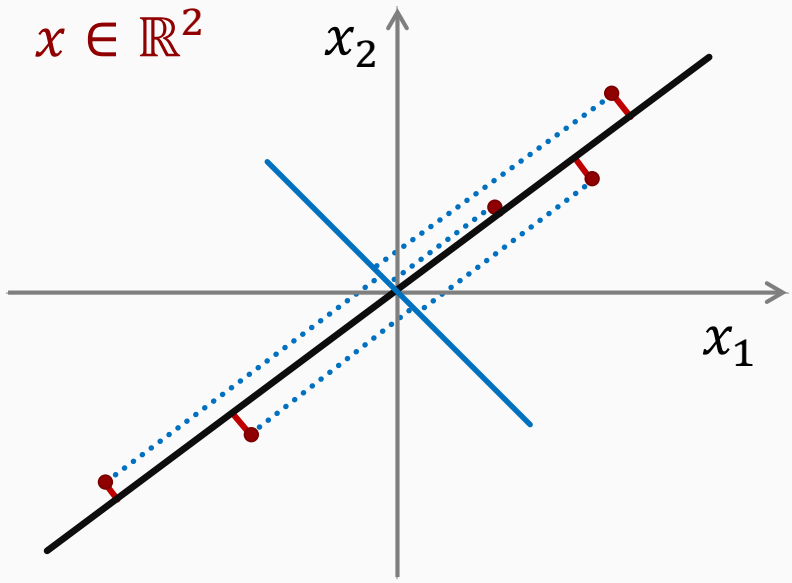
$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

principal components analysis: problem formulation

pca finds a lower dimensional surface (a line in many cases) to project the data so that the sum of squares of the distances between the points and the lines is minimized
this is often referred to as the projection error

before applying pca, it is standard practice to perform mean normalization and feature scaling so that the features x_1 and x_2 have zero mean and comparable value ranges

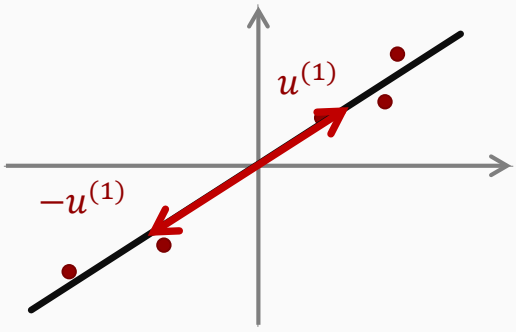
the secondary line in the examples represents a poor choice to project that data onto due to the large distance the that would exist when projecting that data onto the line. principal component analysis will avoid choosing such a line for data projection

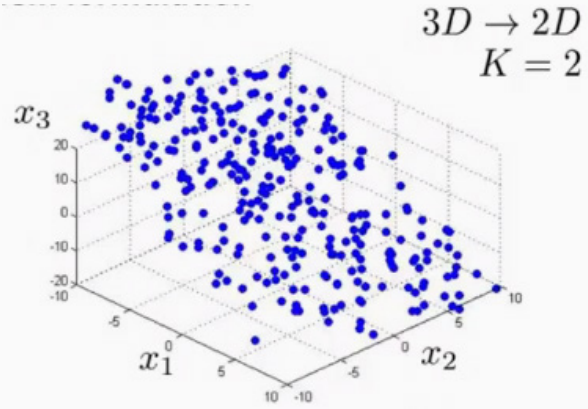


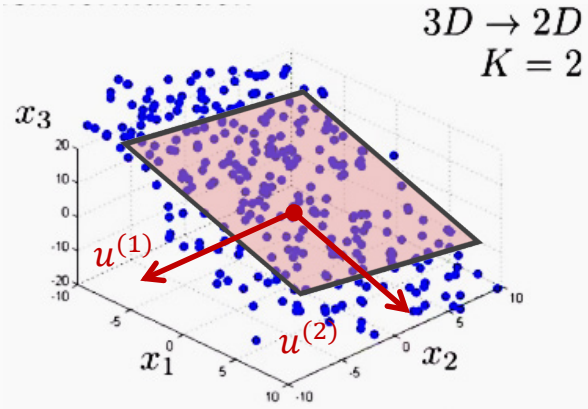
formally stated in the example:
principal component analysis reduces the data from 2-dimensions to 1-dimensions: find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error

It is irrelevant whether $u^{(1)}$ or $-u^{(1)}$ is returned as they both represent the same linear equation onto which to project the data points onto

formally stated in more general terms:
principal component analysis reduces the data from n-dimensions to k-dimensions: find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data so as to minimize the projection error

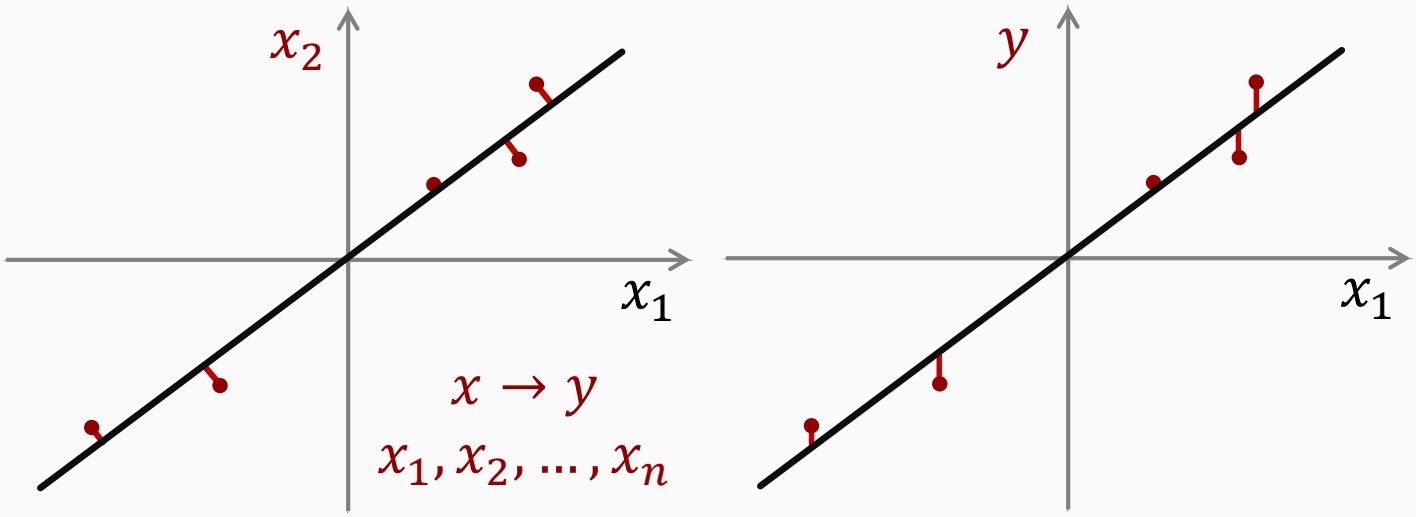


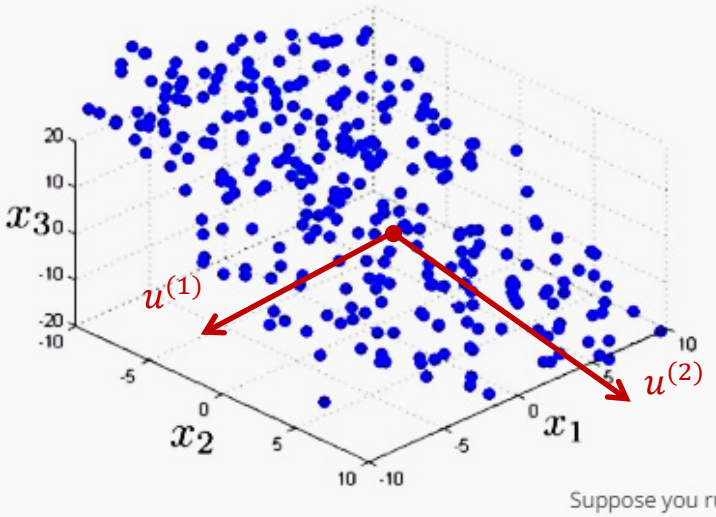




principal component analysis is **not** linear regression

linear regression (left example) projects the data onto a line in reference to its corresponding axis. pca (right example) computes the smallest orthogonal distance between the data points and the line onto which the data is projected

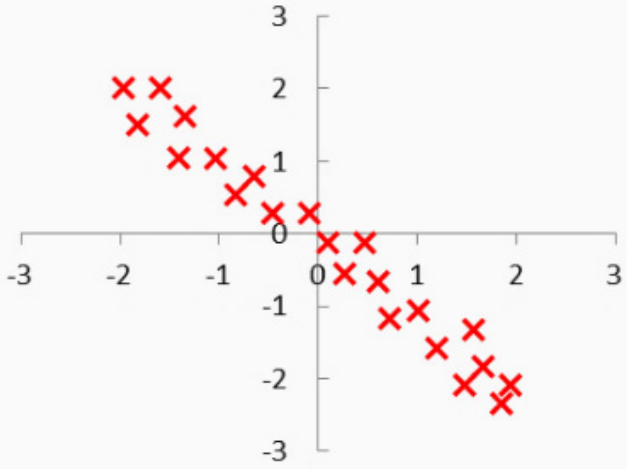




Suppose you run PCA on the dataset below. Which of the following would be a reasonable vector $u^{(1)}$ onto which to project the data? (By convention, we choose $u^{(1)}$ so that $\|u^{(1)}\| = \sqrt{(u_1^{(1)})^2 + (u_2^{(1)})^2}$, the length of the vector $u^{(1)}$, equals 1.)

- $u^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
- $u^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- $u^{(1)} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$
- $u^{(1)} = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

Correct Response



principal components analysis algorithm

data preprocessing

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

preprocessing (feature scaling/mean normalization):

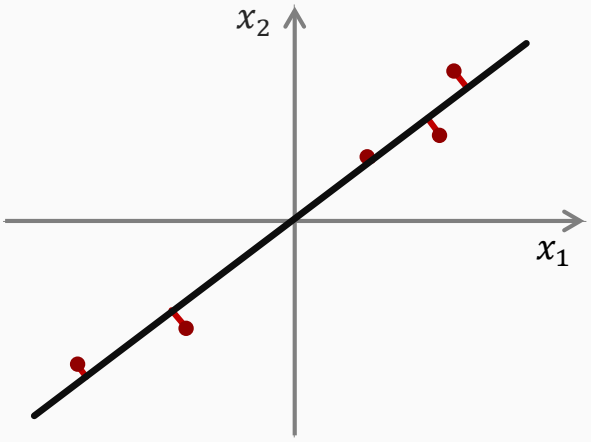
$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

compute mean (μ_j) of each feature and then replace each $x_j^{(i)}$ with $x_j - \mu_j$

scale features to have comparable value ranges :

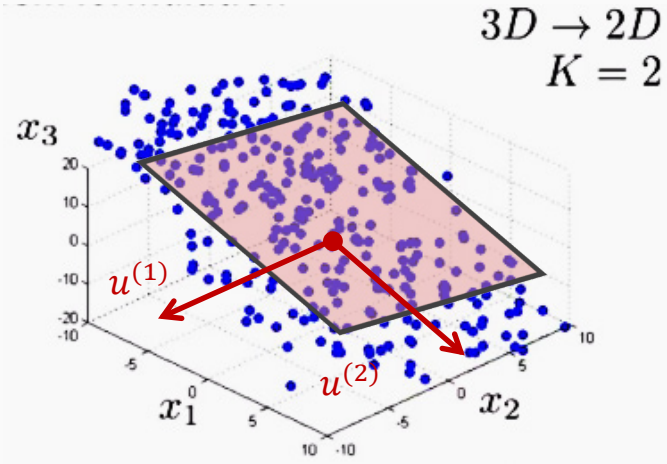
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

principal component analysis (pca) algorithm



reduce data from 2D to 1D

$x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \mathbb{R}$



reduce data from 3D to 2D

$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$

reduce data from n-dimensions to k-dimensions

compute covariance matrix:

$$\sum(\text{sigma}) = \frac{1}{m}\sum_{i=1}^{n}\left(x^{(i)}\right)\left(x^{(i)}\right)^T$$

compute eigenvectors of matrix $\sum(\text{sigma})$:

`[U, S, V] = svd(Sigma);` (single value decomposition, similar to `eig(Sigma)`)

`Sigma` is in n x n matrix and `[U, S, V]` are the matrix outputs

`[U, S, V] = svd(Sigma)` produces:

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \ldots & u^{(m)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

in order to reduce the dimensions of the data from n-dimensions to k-dimensions, take the k vectors $(u^{(1)}, \dots, u^{(k)})$ that will provide the k-direction on which to project the data

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(\boldsymbol{n})} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times \boldsymbol{n}}$$

$$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$$

take the original dataset $x \in \mathbb{R}^n$ and find a lower dimensional representation $z \in \mathbb{R}^n$
this is achieved by taking the first k-columns of the U-matrix: (example of single training example $x^{(i)}$)

$$z^{(i)} = U_{\boldsymbol{reduce}} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(\boldsymbol{k})} \\ | & | & & | \end{bmatrix}^T x^{(i)} \in \mathbb{R}^{n \times \boldsymbol{k}} = \begin{bmatrix} - & \left(u^{(1)}\right)^T & - \\ - & \vdots & - \\ - & \left(u^{(k)}\right)^T & - \end{bmatrix} x^{(i)} \in \mathbb{R}^{\boldsymbol{k} \times 1}$$

after taking the first k-columns of the U-matrix, a reduced version of the original U-matrix's dimensions. z will be computed as $U_{\boldsymbol{reduce}}$ transposed times x. the result are vectors now in rows. the transposed matrix will be a $k \times n$ matrix, x is a $n \times 1$ matrix, so the product of the two is a $k \times 1$ matrix

note: a vectorized implementation of computing the matrix `Sigma` in Octave

if data is provided in the form $\mathrm{X} = \begin{bmatrix} - & \left(x^{(1)}\right)^T & - \\ - & \vdots & - \\ - & \left(x^{(m)}\right)^T & - \end{bmatrix}$ the implantation

`Sigma = (1/m)*X'*X;` creates a vectorized implementation

after Sigma has been computed, compute the matrices:

```
[U, S, V] = svd(Sigma);
Ureduce = U(:,1:k);
z = Ureduce'*x;
```

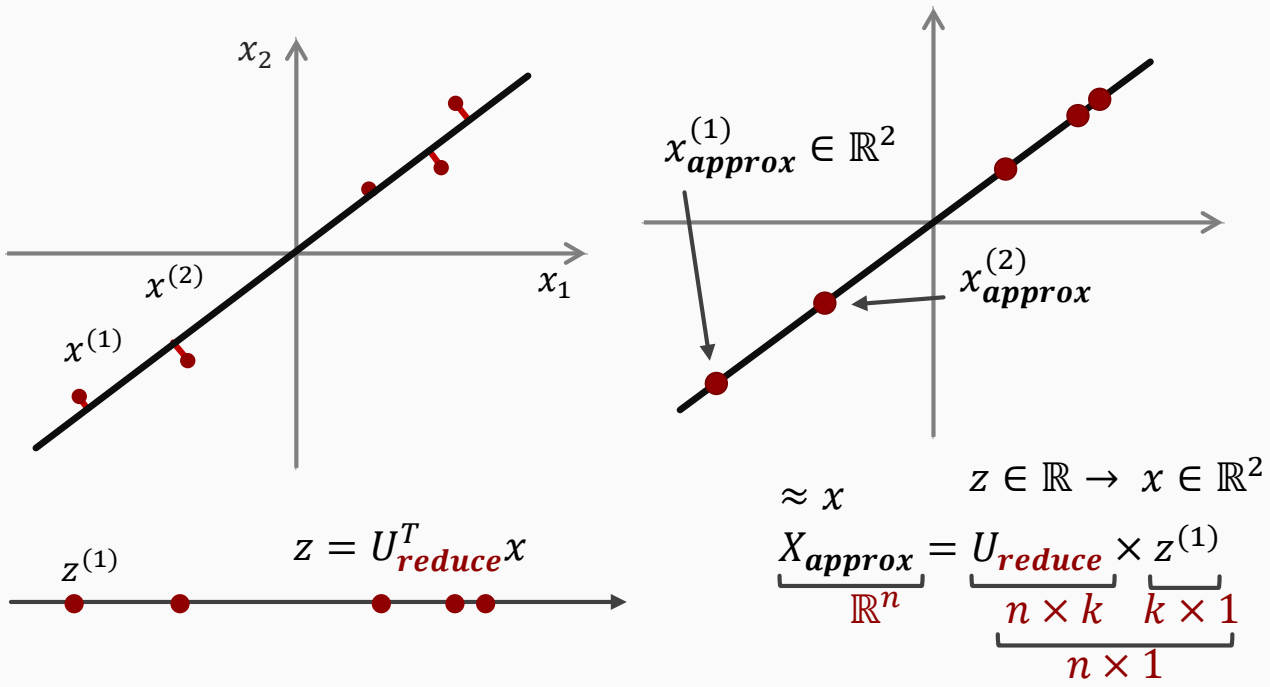
example: in pca, $z \in \mathbb{R}^k$ is obtained from $x \in \mathbb{R}^n$ as follows:

$$z = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(\boldsymbol{k})} \\ | & | & & | \end{bmatrix}^T x = \begin{bmatrix} - & \left(u^{(1)}\right)^T & - \\ - & \vdots & - \\ - & \left(u^{(k)}\right)^T & - \end{bmatrix} x$$

the correct expression of z_j is $z_j = \left(u^{(j)}\right)^T x$

applying pca

reconstruction from compressed representation



after pca has been utilized to project the data onto a line by measuring the minimum orthogonal distance, the data will be decompressed as illustrated above

Suppose we run PCA with k = n, so that the dimension of the data is not reduced at all. (This is not useful in practice but is a good thought exercise.) Recall that the percent / fraction of variance retained is given by: $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$ Which of the following will be true? Check all that apply.

- [x] U_{reduce} will be an $n \times n$ matrix.

  Correct Response

- [x] $x_{\text{approx}} = x$ for every example x.

  Correct Response

- [x] The percentage of variance retained will be 100%.

  Correct Response

- [ ] We have that $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} > 1$.

  Correct Response

choosing the number of principal components

attempt to minimize the average squared projection error: $\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2$

total variation in the data measured as average distance from the origin: $\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2$

typically, k will be chosen to be the smallest value so that the average squared projection error (average distance between x and its projections) divided by the total variation of the data (how much the data varies. this value is ideally ≤ 0.01 (1%):

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2} \leq 0.01 \quad (1\%) \quad (5\%, 10\%, \dots)$$

k is chosen so that 99% of variance is retained (sometimes 95%, 90%, 85% ...)

algorithm:

apply pca with $k = 1$

compute $U_{reduce}, z^{(1)}, \dots, z^{(m)}, x_{approx}^{(1)}, \dots, x_{approx}^{(m)}$

check if $\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2} \leq 0.01$

if $\not\leq 0.01$ with $k = 1$, attempt $k = 2, k = 3, \dots$

```
[U, S, V] = svd(Sigma)
```

$$S = \begin{bmatrix} S_{11} & 0 & 0 & 0 & 0 \\ 0 & S_{22} & 0 & 0 & 0 \\ 0 & 0 & S_{33} & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & S_{nn} \end{bmatrix}$$

pick smallest value of k for which

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$$

(99% of variance retained in dataset)

Previously, we said that PCA chooses a direction $u^{(1)}$ (or k directions $u^{(1)}, \dots, u^{(k)}$) onto which to project the data so as to minimize the (squared) projection error. Another way to say the same is that PCA tries to minimize:

- $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$
- $\frac{1}{m}\sum_{i=1}^{m}\|x_{\text{approx}}^{(i)}\|^2$
- $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}-x_{\text{approx}}^{(i)}\|^2$

Correct Response

- $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}+x_{\text{approx}}^{(i)}\|^2$

application of principal component analysis

pca can be used to increase the running time speed of an algorithm

supervised learning speedup:

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$ $\quad\quad$ $x^{(i)} \in \mathbb{R}^{10,000}$

extract inputs:

unlabeled dataset: $x^{(1)}, x^{(2)}, \dots, x^{(m)} \in \mathbb{R}^{10,000}$

↓ apply principal component analysis

$z^{(1)}, z^{(2)}, \dots, z^{(m)} \in \mathbb{R}^{1,000}$

new training set:

$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \dots, (z^{(m)}, y^{(m)})$

the training examples will now be represented with z and will be applied forward

new x examples will be mapped through the same process and applied forward

note: mapping $x \rightarrow z$ should be defined by running pca only on the training set. this mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets. (e.g. finding U_{reduce} should only be mapped on the training set)

summary of principal component analysis application

compression

- reduce memory/disk needed to store data
- speed up learning algorithm
  - choose k by % of variance retained

visualization

- $k = 2$ or $k = 3$

poor application of principal component analysis

using $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$ with the intention of assuming fewer features will make an algorithm less likely to overfit

regularization should be used instead to reduce overfitting:

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

principal component analysis should not be used when:

design of the machine learning system:

get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

✗ run pca to reduce $x^{(i)}$ in dimension to get $z^{(i)}$

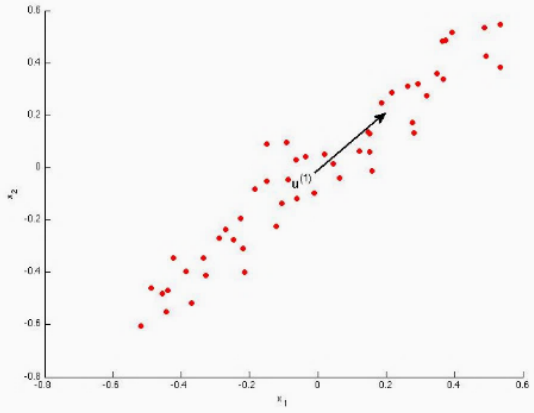
train logistic regression on $\{(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \dots, (z^{(m)}, y^{(m)})\}$

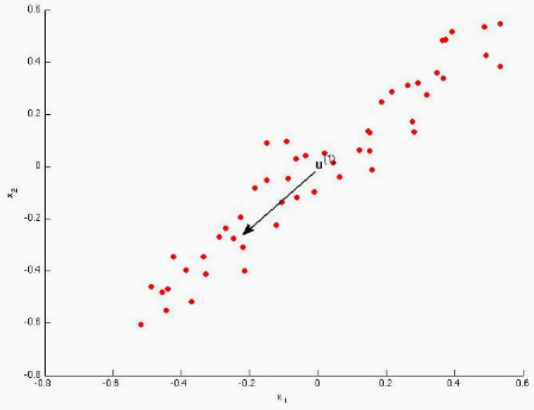
test on test set: map $x_{test}^{(i)}$ to $z_{test}^{(i)}$. run $h_\theta(z)$ on $\left\{\left(z_{test}^{(1)}, z_{test}^{(1)}\right), \dots, \left(z_{test}^{(m)}, z_{test}^{(m)}\right)\right\}$

instead, apply the machine learning process entirely without pca

before impementing pca, attempt running the original/raw data $x^{(i)}$. only if that does not perform as expected, then implement pca and consider using $z^{(i)}$

Which of the following figures correspond to possible values that PCA may return for $u^{(1)}$ (the first eigenvector / first principal component)? Check all that apply (you may have to check more than one figure).

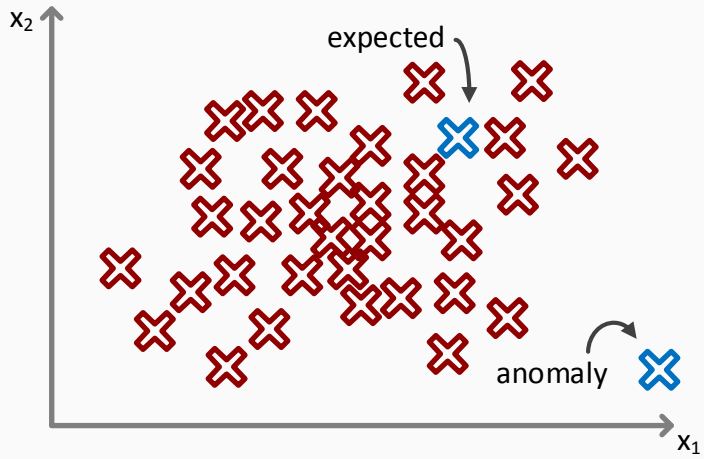




anomaly detection basics

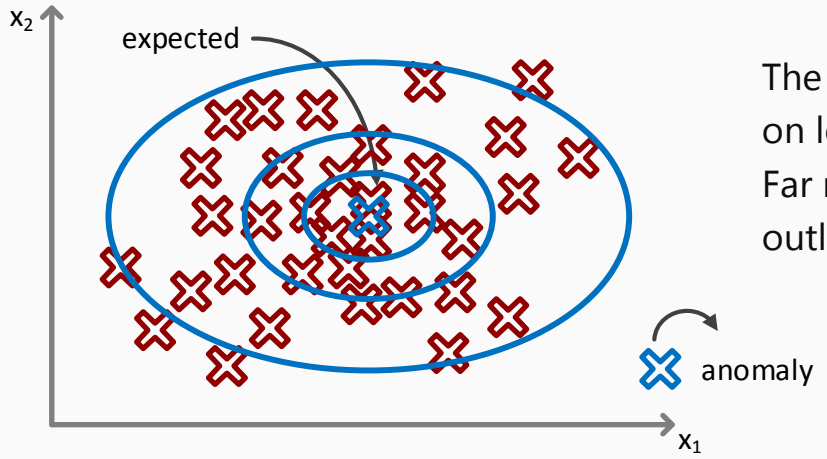
density estimation

problem motivation



when given a dataset of features $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, the points plotted (red) represent the data correlations. When a new dataset is added to the model (x_{test}), points that fall within the majority or expected, but any far reaching points are considered to be an anomaly

formally, a model $p(x)$will determine anomaly detection: $p(x_{test}) < \varepsilon$ anomaly
$(x_{test}) \geq \varepsilon$ expected

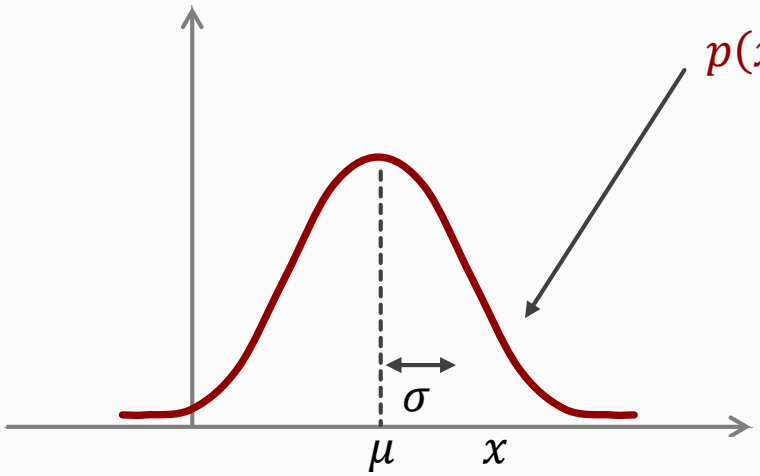


The model p(x) will set thresholds based on logic acceptable to the given data. Far reaching points will be labeled as an outlier or anomaly.

gaussian (normal) distribution

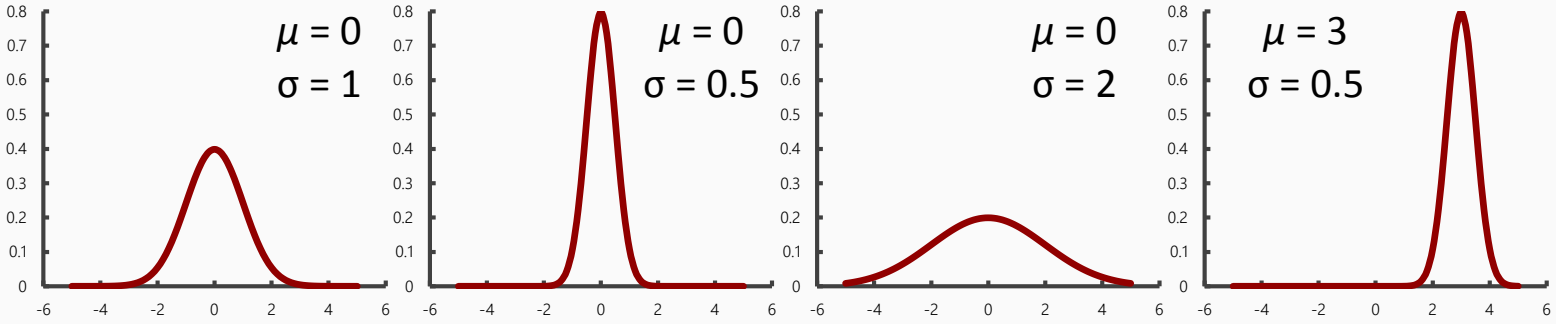
$x \in \mathbb{R}$. if x is a distributed gaussian with mean μ, variance σ^2, denoted as $\boldsymbol{x \sim \mathcal{N}(\mu, \sigma^2)}$

$\boldsymbol{x}$ represents a row values random variable (x is a row number). **~** tilde represents the variable "distributed as". The script $\boldsymbol{\mathcal{N}}$ represents the "normal" gaussian distribution and is parameterized by two parameters $\boldsymbol{\mu}$ mu (mean) and $\boldsymbol{\sigma^2}$ sigma squared (variance)



$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

gaussian distribution examples



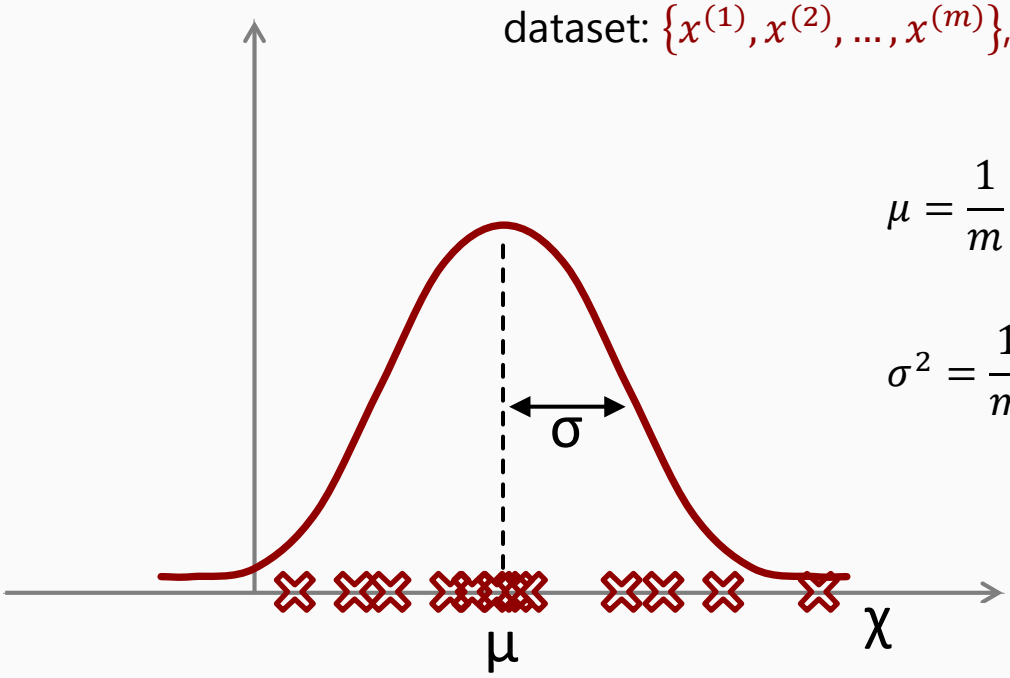
parameter estimation

dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, $x^{(i)} \in \mathbb{R}$

$x \sim \mathcal{N}(\mu, \sigma^2)$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

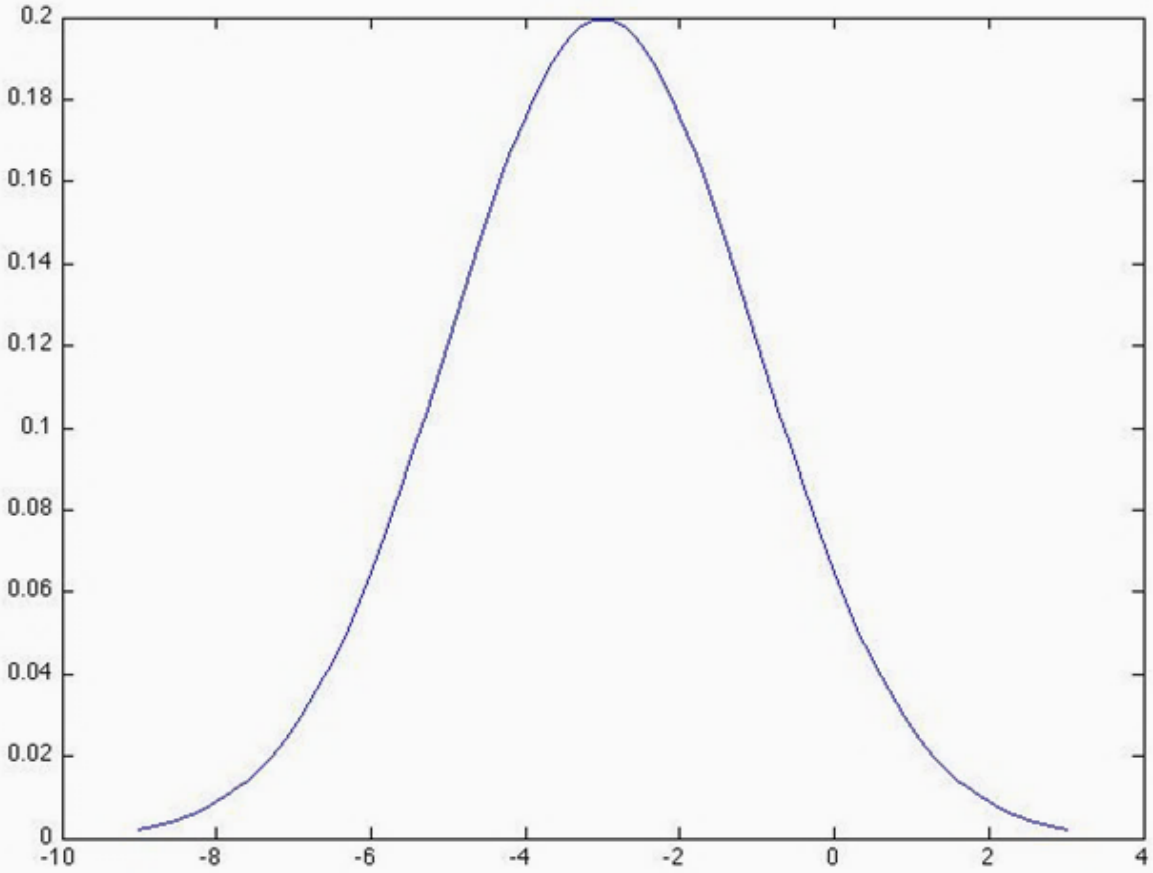
$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} \left(x^{(i)} - \mu\right)^2$$



The formula for the Gaussian density is:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{(x-\mu)^2}{2\sigma^2} \right)$$

Which of the following is the formula for the density to the right?



- $p(x) = \frac{1}{\sqrt{2\pi}\times 2} \exp\left(- \frac{(x-3)^2}{2\times 4} \right)$
- $p(x) = \frac{1}{\sqrt{2\pi}\times 4} \exp\left(- \frac{(x-3)^2}{2\times 2} \right)$
- $p(x) = \frac{1}{\sqrt{2\pi}\times 2} \exp\left(- \frac{(x+3)^2}{2\times 4} \right)$

Correct Response

- $p(x) = \frac{1}{\sqrt{2\pi}\times 4} \exp\left(- \frac{(x+3)^2}{2\times 2} \right)$

anomaly detection algorithm

density estimation

dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$,

each example is $x^{(i)} \in \mathbb{R}^n$

$x \sim \mathcal{N}(\mu_1, \sigma_1^2)$

$x \sim \mathcal{N}(\mu_2, \sigma_2^2)$

$x \sim \mathcal{N}(\mu_3, \sigma_3^2)$

$$p(x) = p(x_1; \mu_1, \sigma_1^2)p(x_2; \mu_2, \sigma_2^2)p(x_3; \mu_3, \sigma_3^2) \dots p(x_n; \mu_n, \sigma_n^2)$$
$$= \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

to describe the product operator in compassion to the summation operator:

$$\sum_{i=1}^{n} 1 + 2 + 3 + \cdots + n \qquad \text{versus} \qquad \prod_{i=1}^{n} 1 \times 2 \times 3 \times \dots \times n$$

estimating the above distribution $p(x)$ is referred to as the density estimation problem

given training set $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$, each would be estimated as (note $\mu_j \in \mathbb{R}, \sigma_j^2 \in \mathbb{R}$):

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \qquad \text{and} \qquad \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m} \left(x_j^{(i)} - \mu_j\right)^2$$

steps in the anomaly detection algorithm

choose features x_i that might be indicative of anomalous examples: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$
$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m} \left(x_j^{(i)} - \mu_j\right)^2$$

vectorized version:

$p(x_j; \mu_j, \sigma_j^2)$ $\quad u_1, \mu_2, \dots \mu_n$

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

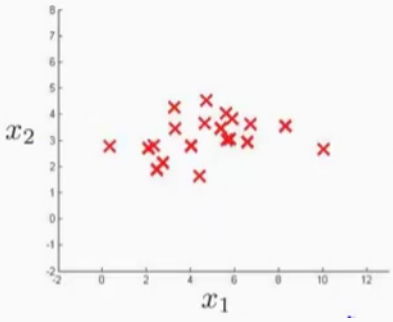
given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma^2} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

anomaly if $p(x) < \varepsilon$

anomaly detection example

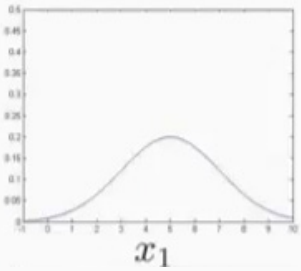
The dataset below has an average measured feature x_1 of ~5 with a standard deviation of ~2 and the feature x_2 has an average value of ~3 with a standard deviation ~1



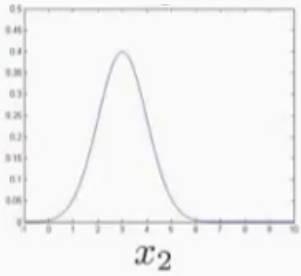
$$\mu_1 = 5, \sigma_1 = 2$$
$$\mu_2 = 3, \sigma_2 = 1$$

Note the use of σ (standard deviation) as opposed to σ^2 (variance) used previously

The features above plotted as in terms of $p(x_i; \mu_i, \sigma_i^2)$ distribution would appear as:

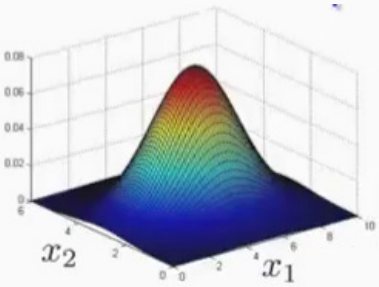


$$p(x_1; \mu_1, \sigma_1^2)$$



$$p(x_2; \mu_2, \sigma_2^2)$$

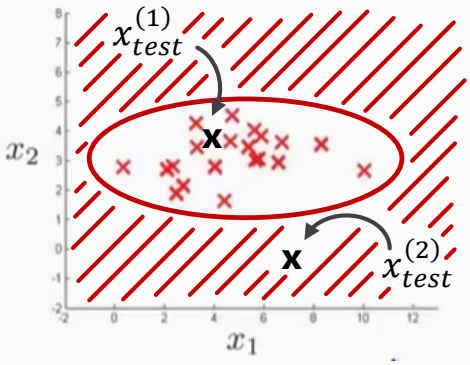
p(x) for both can be taken and plotted in a surface plot as seen below. The height of the surface given a particular values is p(x):



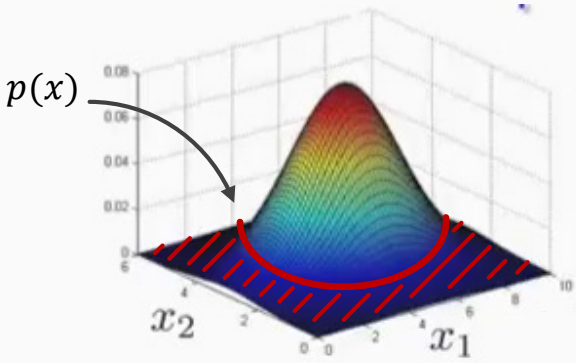
the height of the plot is literally equal to:

$$p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2)$$

if additional data points are added to the dataset:



$$\mu_1 = 5, \sigma_1 = 2$$
$$\mu_2 = 3, \sigma_2 = 1$$



to determine if either is an anomaly: a value is set for epsilon ε, (in this case, ε = 0.02):

$p\left(x_{test}^{(1)}\right) = 0.0426 \geq \varepsilon$ (non-anomaly) and $p\left(x_{test}^{(2)}\right) = 0.0021 < \varepsilon$ (anomaly)

anomaly detection system construction

developing and evaluating an anomaly detection system

the importance of real · number evaluation

when developing learning algorithm (choosing features, etc…), making decisions is easier if a method of algorithm evaluation is available

assuming procession of labeled data, of anomalous and non-anomalous examples ($y = 0$ if normal, $y = 1$ if anomalous)

training set: $\left(x^{(1)}, x^{(2)}, \dots, x^{(m)}\right)$ assuming normal examples/non-anomalous

cross validation set: $\left(x_{cv}^{(1)}, y_{cv}^{(1)}, \dots, x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})}\right)$

test set: $\left(x_{test}^{(1)}, y_{test}^{(1)}, \dots, x_{test}^{(m_{test})}, y_{test}^{(m_{test})}\right)$

motivating example

10,000 good (normal) examples

20 flawed (anomalous) examples

$\mu_1, \sigma_1^2, \dots, \mu_n, \sigma_n^2$

training set: 6000 good examples ($y = 0$) $\quad p(x) = p(x_1; \mu_1, \sigma_1^2), \dots, p(x_n; \mu_n, \sigma_n^2)$

cross validation: 2000 good examples ($y = 0$), 10 anomalous ($y = 1$)

test set: 2000 good examples ($y = 0$), 10 anomalous ($y = 1$)

alternative (**not recommended**, same examples in cv and test):

training set: 6000 good examples

cross validation: 4000 good examples ($y = 0$), 10 anomalous ($y = 1$)

test set: 4000 good examples ($y = 0$), 10 anomalous ($y = 1$)

algorithm evaluation

fit model $p(x)$ on the training set $\{x^{(1)}, \dots, x^{(m)}\}$

predict x on the cross validation/test example

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomoly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

possible evaluation metrics: $\left(x_{test}^{(i)}, y_{test}^{(i)}\right)$

- true positive, false positive, false negative, true negative
- precision/recall
- F_1-score

can also use cross validation set to choose parameter ε

Suppose you have fit a model p(x). When evaluating on the cross validation set or test set, your algorithm predicts:

$$y = \begin{cases} 1 & \text{if } p(x) \leq \epsilon \\ 0 & \text{if } p(x) > \epsilon \end{cases}$$

Is classification accuracy a good way to measure the algorithm's performance?

- Yes, because we have labels in the cross validation / test sets.
- No, because we do not have labels in the cross validation / test sets.
- No, because of skewed classes (so an algorithm that always predicts y = 0 will have high accuracy).

Correct Response

- No for the cross validation set; yes for the test set.

anomaly detection versus supervised learning

when processing labeled data, the question arises as to using a supervised learning method for a portion of the data (logistic regression, etc...) as opposed to an anomaly detection algorithm. the following example compares the advantages of each:

anomaly detection	supervised learning
very small number of positive examples ($y = 1$) (0-20 is common)	large number of positive and negative examples
large number of negative ($y = 0$) examples $p(x)$	
many different "types" of anomalies. difficult for any algorithm to learn from positive examples what the anomalies might appear as;	enough positive examples for an algorithm to obtain a sense of what the positive examples will appear as; future positive examples are likely to be similar to those in the training set
future anomalies may appear nothing like any of the preceding anomalous examples witnessed.	

application examples

anomaly detection: fraud detection, manufacturing, monitoring data centers

supervised learning: email spam classification, weather prediction, cancer research

Which of the following problems would you approach with an anomaly detection algorithm (rather than a supervised learning algorithm)? Check all that apply.

- [x] You run a power utility (supplying electricity to customers) and want to monitor your electric plants to see if any one of them might be behaving strangely.

Correct Response

- [ ] You run a power utility and want to predict tomorrow's expected demand for electricity (so that you can plan to ramp up an appropriate amount of generation capacity).

Correct Response

- [x] A computer vision / security application, where you examine video images to see if anyone in your company's parking lot is acting in an unusual way.

Correct Response

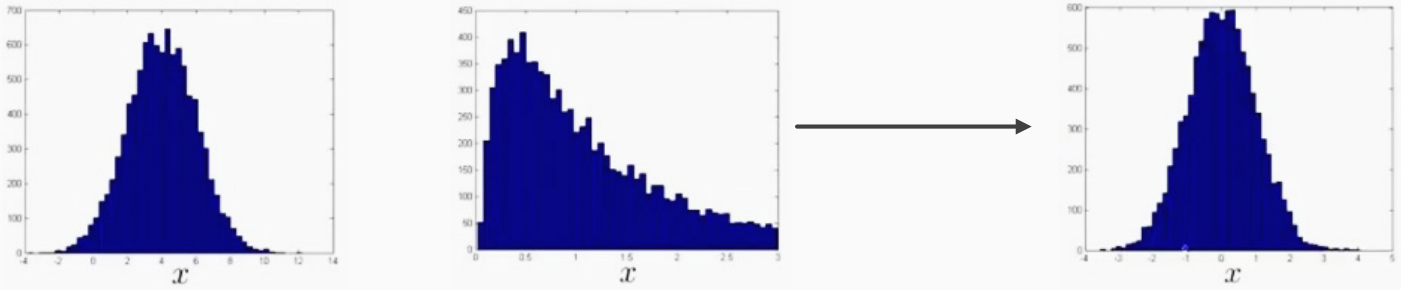
- [ ] A computer vision application, where you examine an image of a person entering your retail store to determine if the person is male or female.

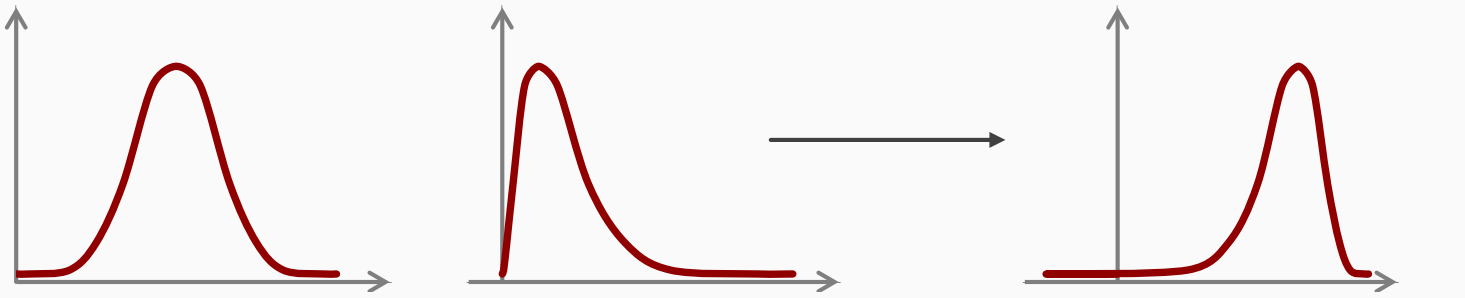
Correct Response

choosing what features to use

transformations of non · gaussian features

$p(x_1; \mu_1, \sigma_1^2)$





normal distribution $x_1 \leftarrow \log(x_1)$ $x_2 \leftarrow \log(x_2 + c)$ $x_3 \leftarrow \sqrt{x_3} = x_3^{\frac{1}{2}}$ $x_4 \leftarrow \sqrt[3]{x_4} = x_4^{\frac{1}{3}}$

transformations to the dataset can be applied with various methods to achieve a gaussian (normal) distribution from that of a skewed dataset

error analysis for anomaly detection

creating features for an anomaly detection problem by observing misclassifications on cross validation dataset

expectation of $p(x)$ as large for normal examples x

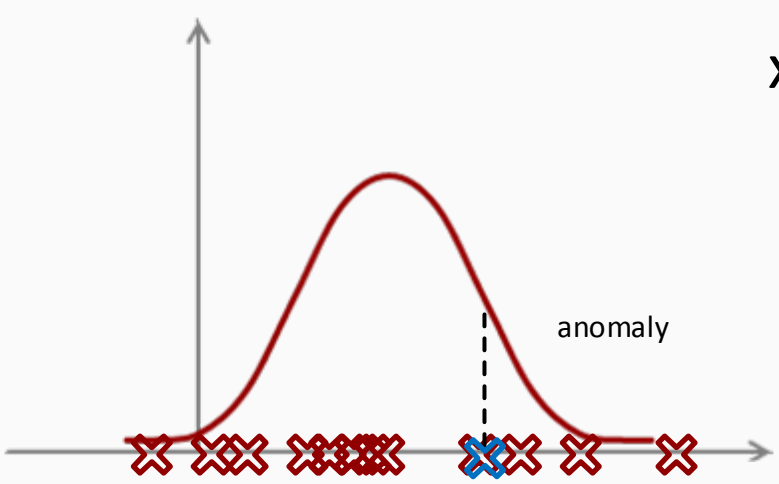
$p(x)$ as small for anomalous examples x

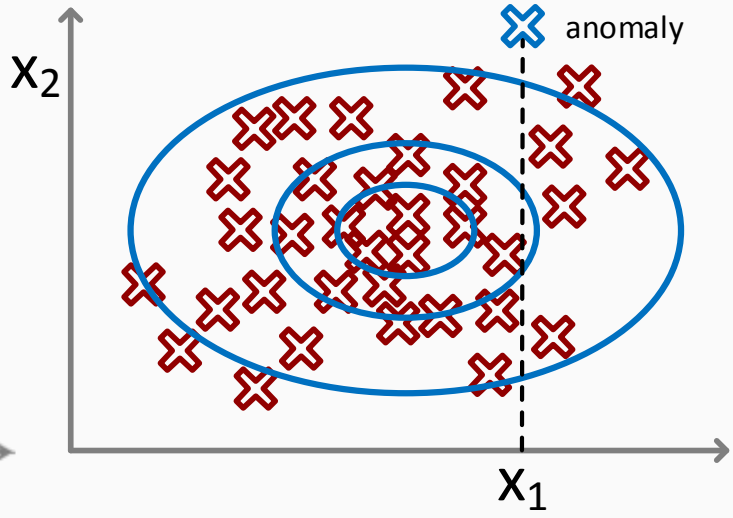
the common problem: $p(x)$ is comparable (e.g., both are large)
for normal and anomalous examples

after fitting a Gaussian distribution to a dataset, a new example known to be anomalous is misclassified as normal according to the current algorithm's performance

a new feature (x_2) is created and takes on an unusual value when plotted against the same new example known to be anomalous

the process involves physical interactions and observation of the data as it is being introduced into the model in order to create new features for anomaly detection





ideology for choosing features for anomaly detection

choose features which potentially take on unusually large or small values in the event of an anomaly

x_1 = measured feature

x_2 = measured feature

x_3 = measured feature

x_4 = measured feature

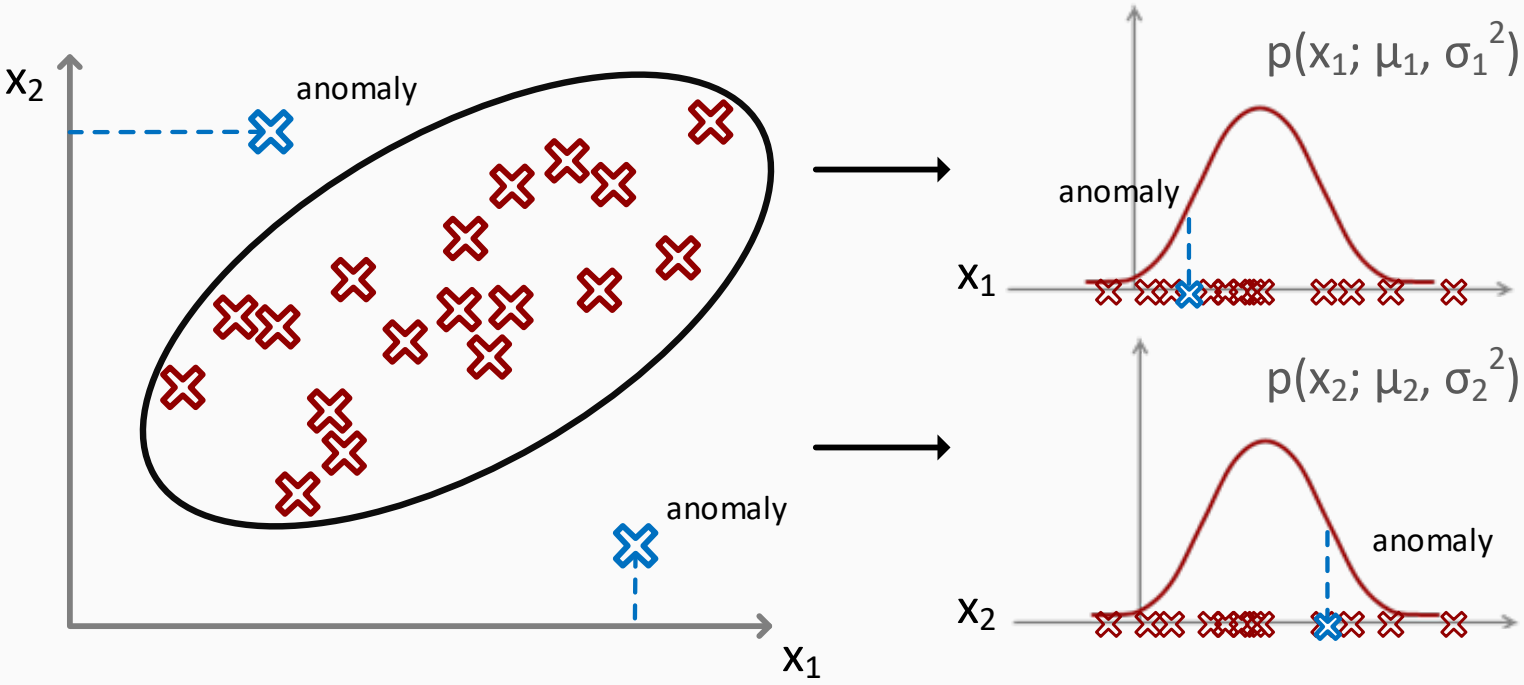
assuming two features have a particular relationship, it might benefit anomaly detection to relate them against a new feature:

$$\frac{x_1}{x_2} = x_5 \text{ or } \frac{(x_4)^2}{x_3} = x_6 \text{ etc...}$$

multivariate gaussian distribution

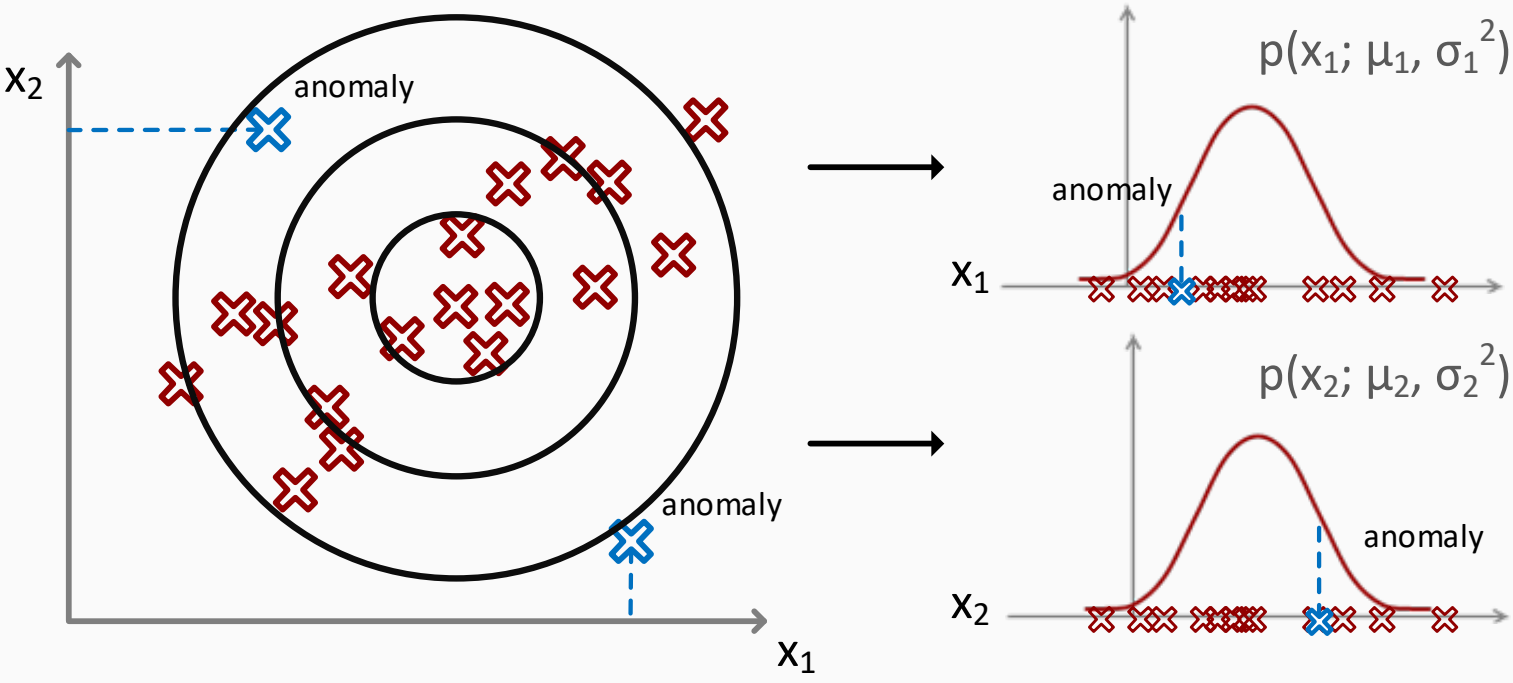
certain instances of single variable gaussian distribution might register examples amongst separtaley evaluated features as normal or non-anomalous. There are instances in which a multivariate gausian distribution would otherwise detect anomaly within a dataset that otherwise would have been misclassified elsewhere

multivariate gaussian model classification (left):



both variables plotted with a single variable gausian distribution algorithm would identify the anomolous probability regions not as the ellispe is shown in the above left illustration but as a serious of circles representative of the distributions' σ

single variable gaussian distribution (left):



$x \in \mathbb{R}^n$, do not model multiple variables $p(x_1, x_2, \dots, x_n)$ separately.

model $p(x)$ together at once.

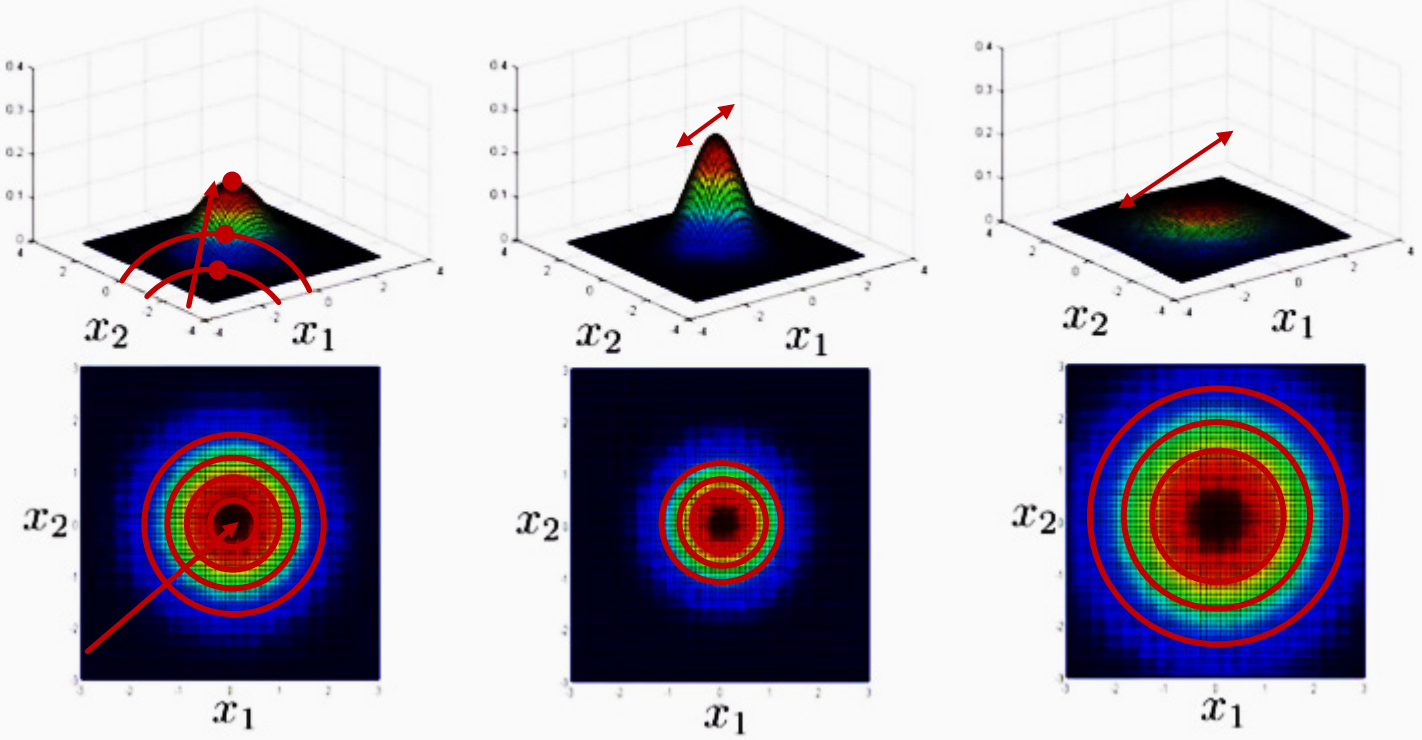
parameters: $\mu \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n \times n}$ covariance matrix

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

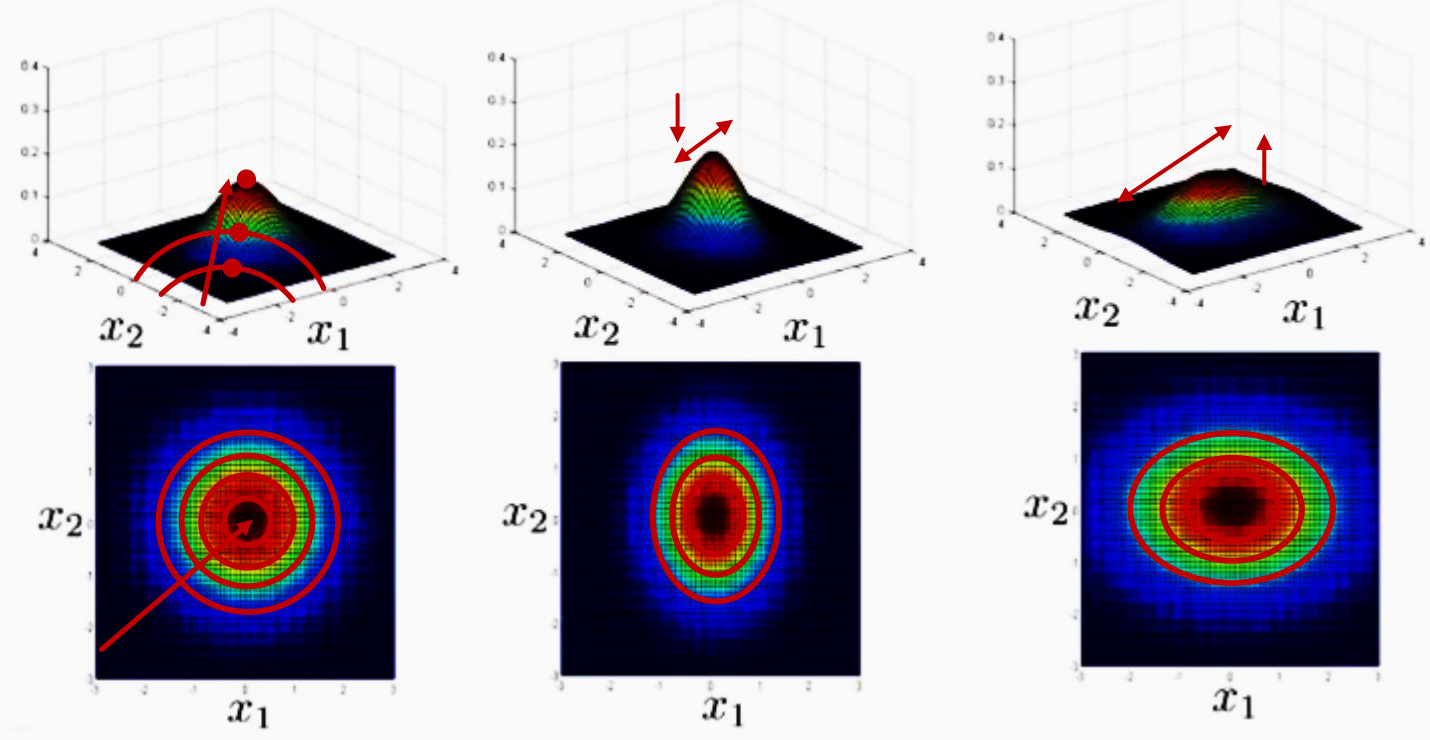
$|\Sigma| =$ determinant of Σ | det(Sigma)

effects of transforming parameters in $p(x; \mu, \Sigma)$

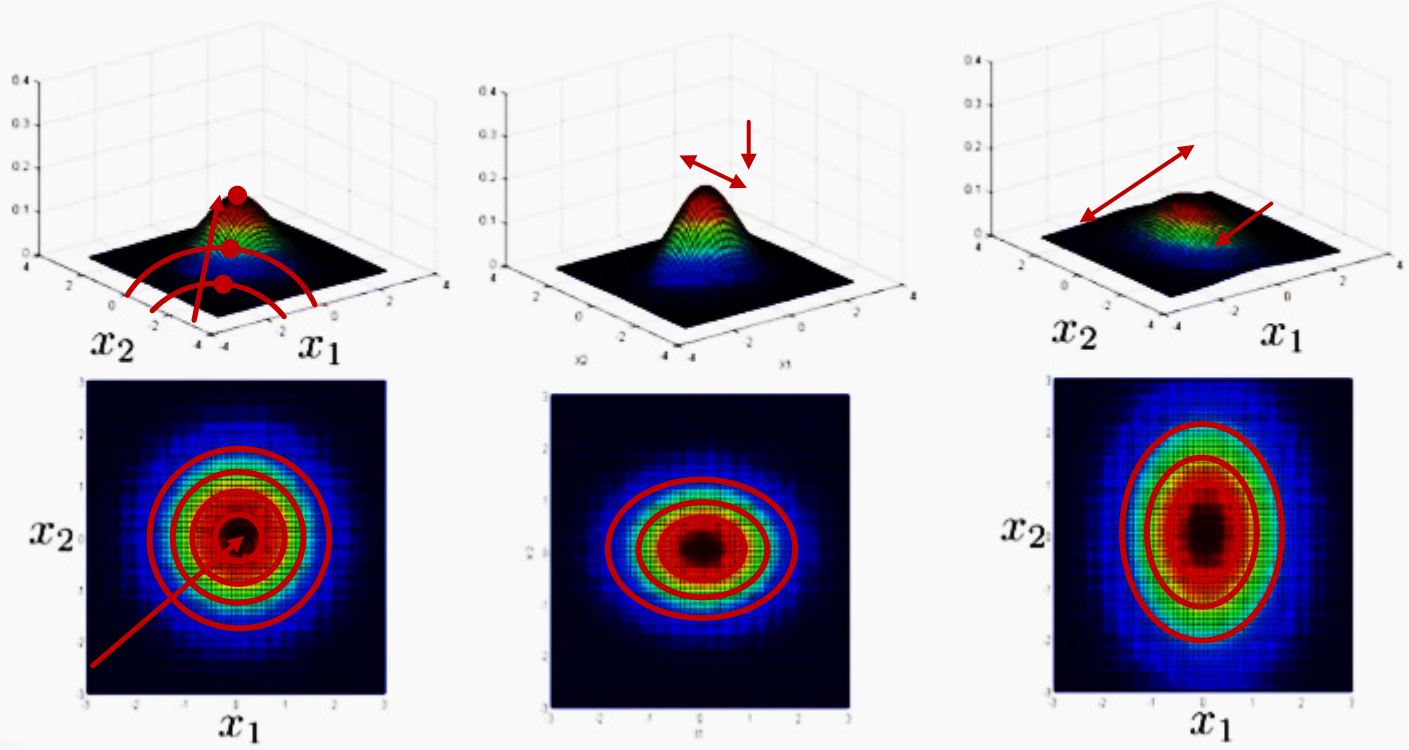
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$

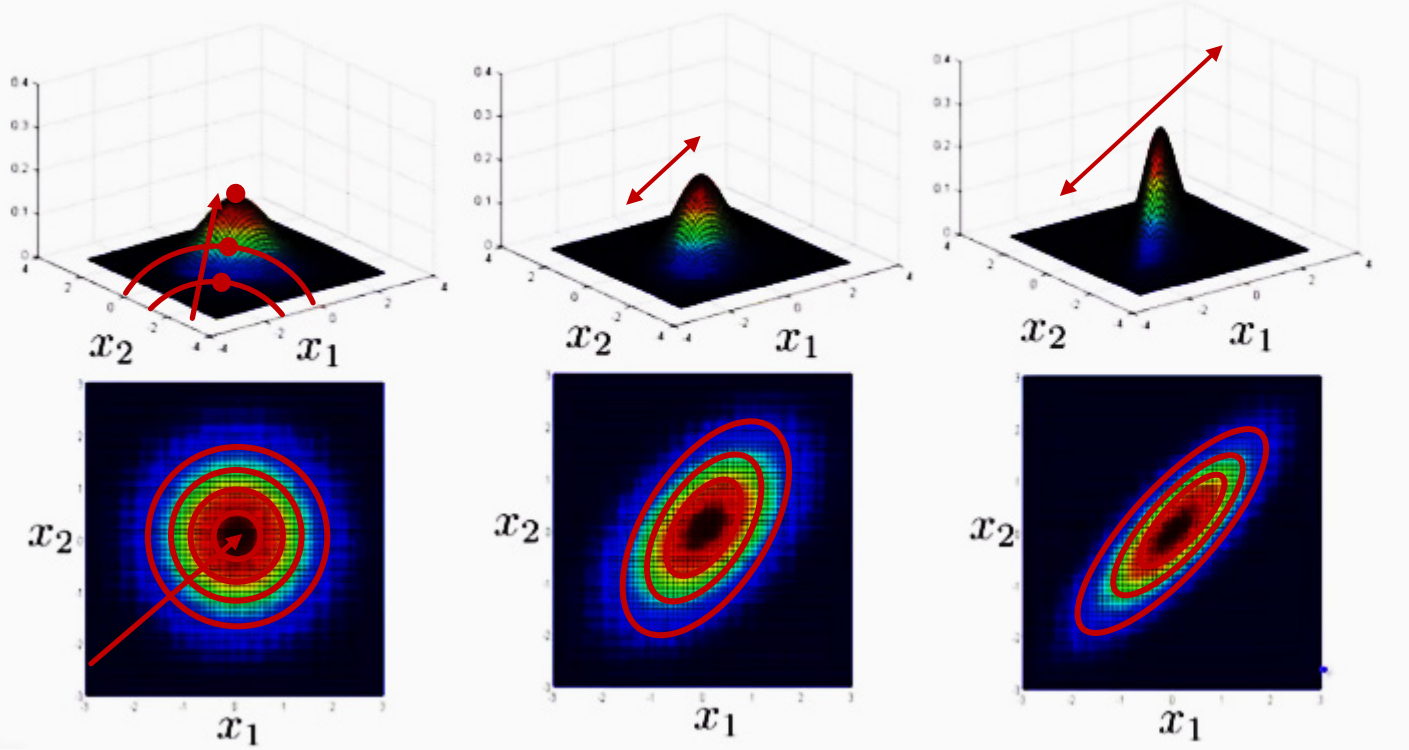


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.6 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$



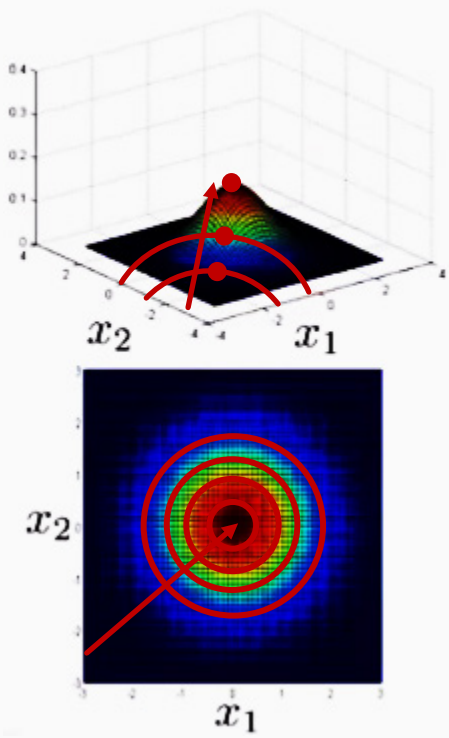
multivariate gaussian distribution can measure correlation amongst features:

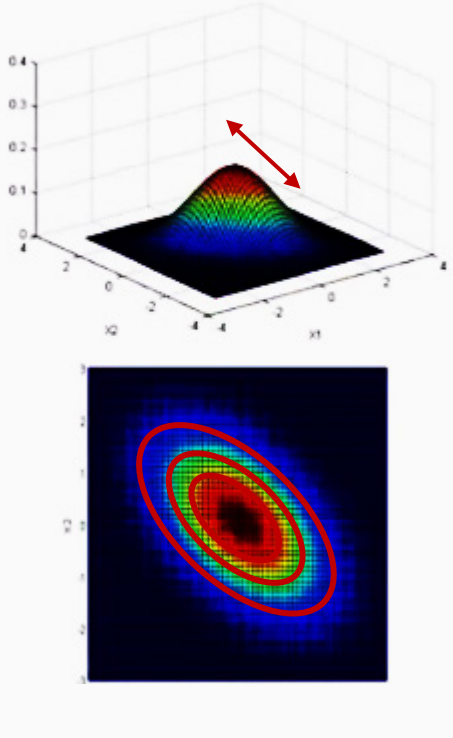
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

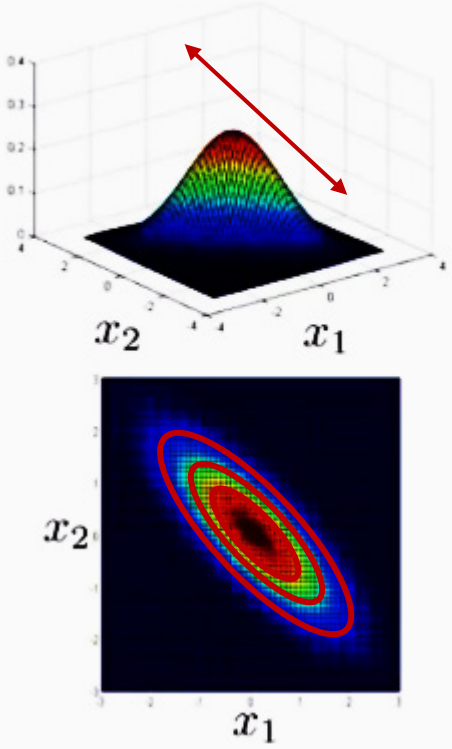


in contrast when sigma Σ is set to negative values:

$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$$

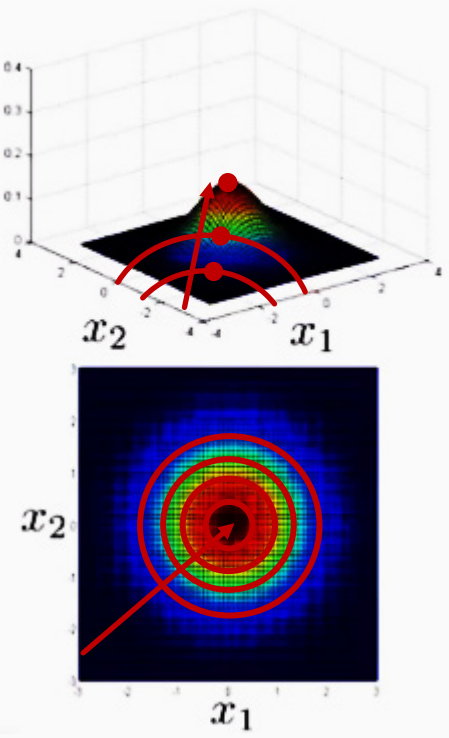


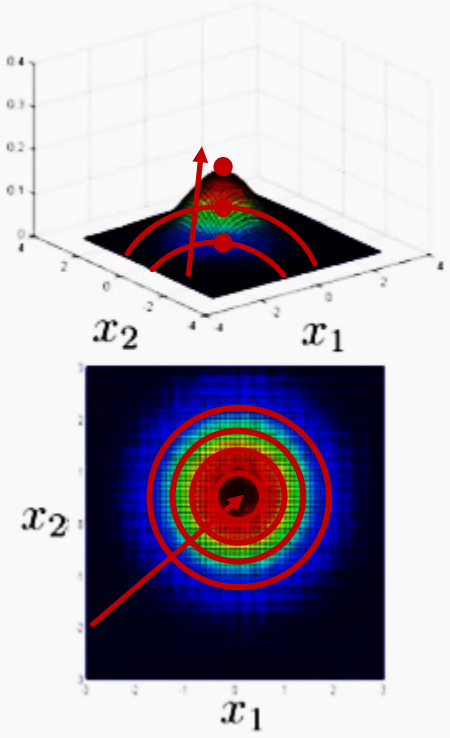


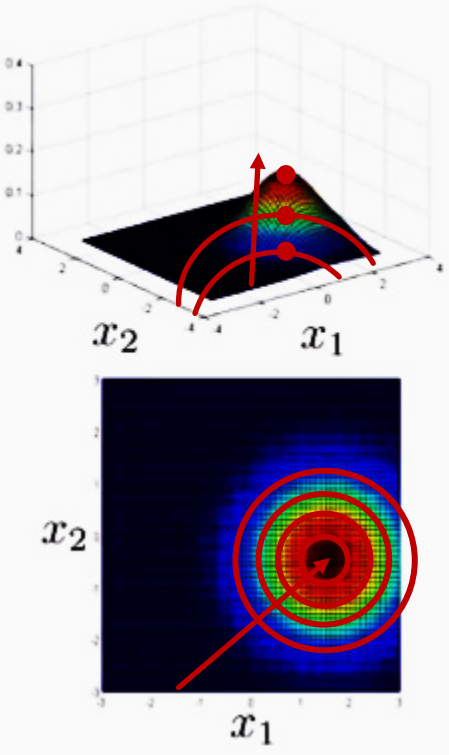


additionally, the μ parameter can also be varied:

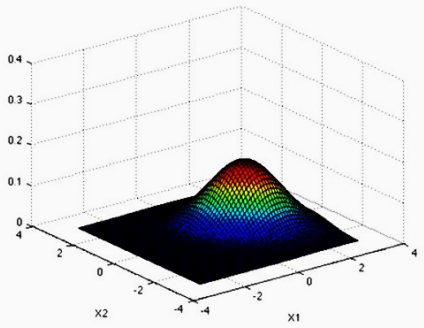
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mu = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

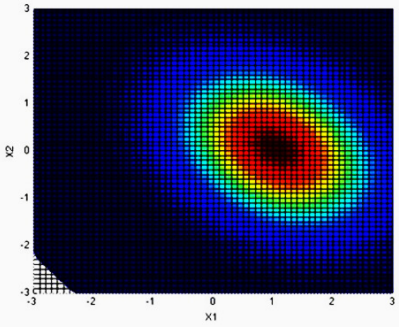






Consider the following multivariate Gaussian:





Which of the following are the μ and Σ for this distribution?

- $\mu = \begin{bmatrix}1\\0\end{bmatrix}, \Sigma = \begin{bmatrix}1 & 0.3\\0.3 & 1\end{bmatrix}$
- $\mu = \begin{bmatrix}0\\1\end{bmatrix}, \Sigma = \begin{bmatrix}1 & 0.3\\0.3 & 1\end{bmatrix}$
- $\mu = \begin{bmatrix}1\\0\end{bmatrix}, \Sigma = \begin{bmatrix}1 & -0.3\\-0.3 & 1\end{bmatrix}$

Correct Response

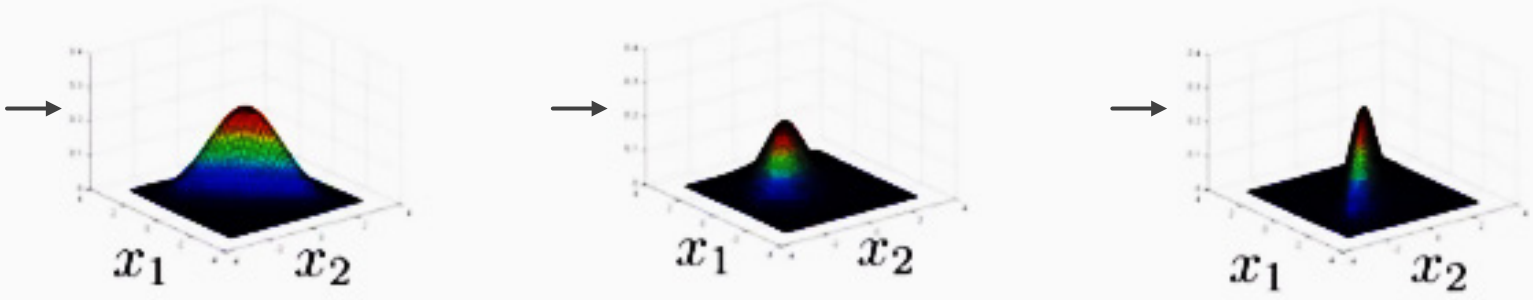
- $\mu = \begin{bmatrix}0\\1\end{bmatrix}, \Sigma = \begin{bmatrix}1 & -0.3\\-0.3 & 1\end{bmatrix}$

anaomaly detection using the multivariate gaussian distribution

multivariate gaussian (normal) distribution

parameters μ, Σ $\mu \in \mathbb{R}^2$ and $\sigma \in \mathbb{R}^2$

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\,|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\,\Sigma^{-1}(x-\mu)\right)$$



parameter fitting:

given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\} \leftarrow x \in \mathbb{R}^n$

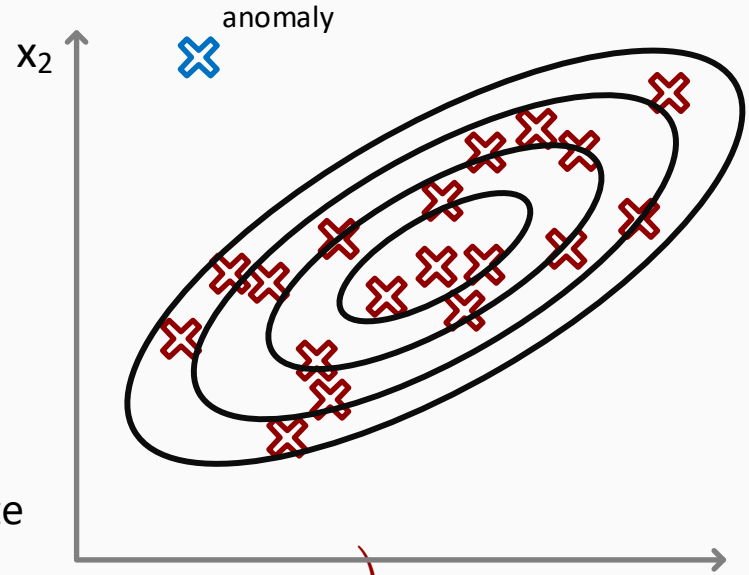
$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \quad \text{and} \quad \Sigma = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)}-\mu\right)\left(x^{(i)}-\mu\right)^T$$

anamoly detection with multvariate gaussian

1) Fit the model p(x) by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} \left(x^{(i)} - \mu\right)\left(x^{(i)} - \mu\right)^T$$



2) given a new example x, compute

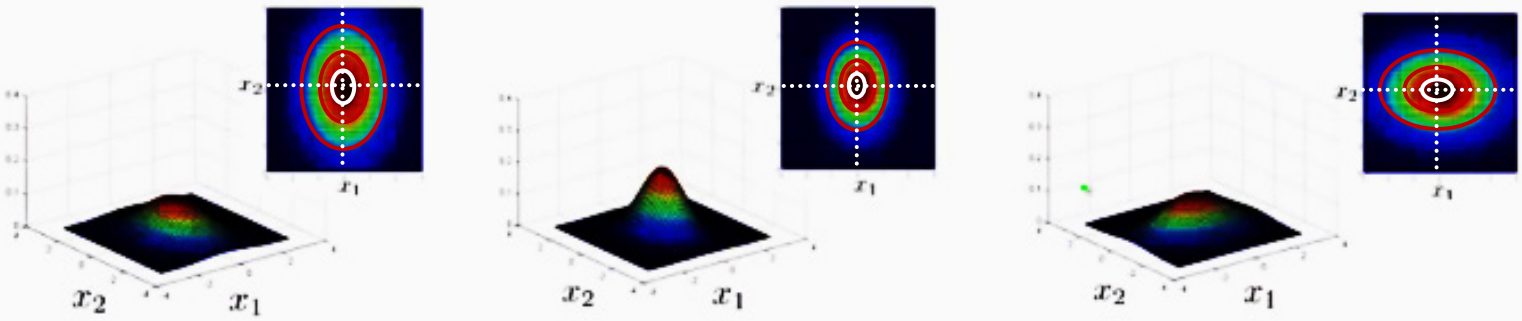
$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\,|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\,\Sigma^{-1}(x-\mu)\right)$$

flag an anomaly if $p(x) < \varepsilon$

relationship to the original single variable model

original model: $p(x) = p(x_1;\mu_1,\sigma_1^2) \times p(x_2;\mu_2,\sigma_2^2) \times \ldots \times p(x_n;\mu_n,\sigma_n^2)$

$$= \prod_{j=1}^{n} p\left(x_j;\mu_j,\sigma_j^2\right)$$



the original model corresponds to multivariate gaussians, where the contours of the gaussian are always axis aligned.

constraint: $\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_1^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_n^2 \end{bmatrix}$

original model

$$p(x_1; \mu_1, \sigma_1^2) \times \ldots \times p(x_n; \mu_n, \sigma_n^2)$$

manually create features to capture anomalies where x_1, x_2 take unusual combinations of values

$$\frac{x_1}{x_2} = x_5 \text{ or } \frac{(x_4)^2}{x_3} = x_6 \text{ etc...}$$

computationally cheaper and scales better to larger n's

n = 10,000, n = 100,000

sufficient model even if m (training set size) is small

multivariate gaussian

$$= \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

automatically captures correlations between multiple features

$$\Sigma = \mathbb{R}^{n \times n} \qquad \Sigma^{-1}$$

computationally more expensive

$$\Sigma \sim \frac{n^n}{2}$$

must have $m > n$ or else Σ is noninvertible $\rightarrow m \gg n$

Consider applying anomaly detection using a training set $\{x^{(1)}, \ldots, x^{(m)}\}$ where $x^{(i)} \in \mathbb{R}^n$. Which of the following statements are true? Check all that apply.

- [x] The original model $p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$ corresponds to a multivariate Gaussian where the contours of $p(x; \mu, \Sigma)$ are axis-aligned.

Correct Response

- [ ] Using the multivariate Gaussian model is advantageous when m (the training set size) is very small (m < n).

Correct Response

- [x] The multivariate Gaussian model can automatically capture correlations between different features in x.

Correct Response

- [x] The original model can be more computationally efficient than the multivariate Gaussian model, and thus might scale better to very large values of n (number of features).

Correct Response

recommender systems + collaborative filtering

working example: predicting movie ratings

users will rate movies using ~~one~~ zero (for computational efficiency) to five stars

movie	$user_1$	$user_2$	$user_3$	$user_4$
$movie_1$	5	5	0	0
$movie_2$	5	**?** *(4)*	**?** *(0)*	0
$movie_3$	**?** *(5)*	4	0	**?** *(0)*
$movie_4$	0	0	5	4
$movie_5$	0	0	5	**?** *(4)*

n_u = number of users

$n_u = \mathbf{4}$

n_m = number of movies

$n_m = \mathbf{4}$

$r(i,j) = 1$ if user j has rated the movie i

$y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j) = 1$)

In our notation, $r(i,j) = 1$ if user j has rated movie i, and $y^{(i,j)}$ is his rating on that movie.
Consider the following example (no. of movies $n_m = 2$, no. of users $n_u = 3$):

.	User 1	User 2	User 3
Movie 1	0	1	?
Movie 2	?	5	5

What is $r(2,1)$? How about $y^{(2,1)}$?

- $r(2,1) = 0,\ y^{(2,1)} = 1$
- $r(2,1) = 1,\ y^{(2,1)} = 1$
- (selected) $r(2,1) = 0,\ y^{(2,1)} = \text{undefined}$

Correct Response

- $r(2,1) = 1,\ y^{(2,1)} = \text{undefined}$

content based recommendation systems

$n_u = 4, n_m = 5$
$x_0 = 1$

$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$

movie	user$_1$ $\theta^{(1)}$	user$_2$ $\theta^{(2)}$	user$_3$ $\theta^{(3)}$	user$_4$ $\theta^{(4)}$	x_1	x_2
movie$_1$ $x^{(1)}$	5	5	0	0	0.9	0
movie$_2$ $x^{(2)}$	5	**?**	**?**	0	1.0	0.01
movie$_3$ $x^{(3)}$	**? 4.95**	4	0	**?**	0.99	0
movie$_4$ $x^{(4)}$	0	0	5	4	0.1	1.0
movie$_5$ $x^{(5)}$	0	0	5	**?**	0	0.9

for each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$ $\longrightarrow$ $\theta^{(j)} \in \mathbb{R}^{n+1}$

predict user j as rating movie i with $\left(\theta^{(j)}\right)^T x^{(i)}$ stars

$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix} \leftrightarrow \theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$

$\left(\theta^{(1)}\right)^T x^{(3)} = 5 \times 0.99 = \mathbf{4.95}$

Consider the following set of movie ratings:

Movie	Alice (1)	Bob (2)	Carol (3)	David (4)	(romance)	(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

Which of the following is a reasonable value for $\theta^{(3)}$? Recall that $x_0 = 1$.

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

Correct Response

problem formulation

$r(i,j) = 1$ 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, predicated rating: $\left(\theta^{(j)}\right)^{T}\left(x^{(i)}\right) \longrightarrow \theta^{(j)} \in \mathbb{R}^{n+1}$
$m^{(j)} =$ number of movies rated by user j

to learn $\theta^{(j)}$:

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

optimization objective:

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

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

to learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$:

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

$$J(\theta^{(1)}, \dots, \theta^{(n_u)})$$

gradient descent update:

(for $k = 0$)

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

(for $k \neq 0$)

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

$\frac{1}{m^{(j)}}$ (crossed out)

$$\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \dots, \theta^{(n_u)})$$

collaborative filtering

changing the problem assuming the value of features $x_1, x_2, \dots, x_n$, etc. are unknown:

movie	user$_1$ $\theta^{(1)}$	user$_2$ $\theta^{(2)}$	user$_3$ $\theta^{(3)}$	user$_4$ $\theta^{(4)}$	x_1	x_2
movie$_1$ $x^{(1)}$	5	5	0	0	?	?
movie$_2$ $x^{(2)}$	5	?	?	0	?	?
movie$_3$ $x^{(3)}$	?	4	0	?	?	?
movie$_4$ $x^{(4)}$	0	0	5	4	?	?
movie$_5$ $x^{(5)}$	0	0	5	?	?	?

additional assumption that the values of parameter vectors $\theta^1, \theta^2, \theta^3, \theta^4$ are obtained:

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

the ability to obtain the values for the above parameter vectors make the computation of the unknown features $x_1, x_2, \ldots, x_n$, etc. possible

focusing on the first example $x^{(1)}$, the problem sums over all indices θ^j for which we have a feature vector $x_1^{(1)}$:

movie	user$_1$ $\theta^{(1)}$	user$_2$ $\theta^{(2)}$	user$_3$ $\theta^{(3)}$	user$_4$ $\theta^{(4)}$	x_1		x_2	
movie$_1$ $x^{(1)}$	5	5	0	0	**?**	**1.0**	**?**	**0.0**

$\theta^{(j)}$

$$\left(\theta^{(1)}\right)^T\left(x^{(1)}\right) \approx 5$$

$$\left(\theta^{(2)}\right)^T\left(x^{(2)}\right) \approx 5$$

$$\left(\theta^{(3)}\right)^T\left(x^{(3)}\right) \approx 0$$

$$\left(\theta^{(4)}\right)^T\left(x^{(4)}\right) \approx 0$$

$$x^{(1)} = \begin{bmatrix} 1 \\ \mathbf{1.0} \\ \mathbf{0.0} \end{bmatrix}$$

Consider the following movie ratings:

.	User 1	User 2	User 3	(romance)
Movie 1	0	1.5	2.5	?

Note that there is only one feature x_1. Suppose that:

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

What would be a reasonable value for $x_1^{(1)}$ (the value denoted "?" in the table above)?

- 0.5

Correct Response

- 1
- 2
- Any of these values would be equally reasonable.

optimization algorithm

formalizing the problem of learning $x^{(i)}$

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

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

summarize all indices J for which a rating is available for example i and minimize the squared error: choose features $x^{(i)}$ so the predicted value of how J rates example i will be similar to the actual value of $y^{(i,j)}$ that is actually observed in the rating of user J on example i

in summary, the term attempts to choose features $x^{(i)}$ so that for all users J that have provided a rating of example $x^{(i)}$, the algorithm predicts a value for how the user would have rated an example that is not too far from the actual rating observed. regularization can be added to control feature scales.

to expand the algorithm to learn the features for many examples as opposed to single example $x^{(i)}$, an additional summation is added:

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

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

Suppose you use gradient descent to minimize:

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

Which of the following is a correct gradient descent update rule for $i \neq 0$?

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

Correct Response

given the ratings of examples $r^{(i,j)}$ and $y^{(i,j)}$ and features $x^{(i)}, \dots, x^{(n_m)}$, the parameters can be estimated:

given $x^{(1)}, \dots, x^{(n_m)}$ (and movie ratings),

can estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$

given the parameters, the ratings of feature examples can be estimated:

given $\theta^{(1)}, \dots, \theta^{(n_u)}$,

can estimate $x^{(1)}, \dots, x^{(n_m)}$

the method is to randomly initialize $\theta^{(i)}$ to learn features $x^{(i)}$ and reiterate to converge to a reasonable set of features for examples and parameters for users:

guess $\theta \to x \to \theta \to x \to \theta \to x \to \cdots$

collaborative filtering refers to the observation that when the algorithm is run with a large set of users, the users are used to *collaborate* in the efforts to learn better features to be used by the algorithm for new users

collaborative filtering algorithm

given features $x^{(1)}, \dots, x^{(n_m)}$, estimate parameters $\theta^{(1)}, \dots, \theta^{(n_u)}$:

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

given parameters $\theta^{(1)}, \dots, \theta^{(n_u)}$, estimate features $x^{(1)}, \dots, x^{(n_m)}$:

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

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

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

both θ and x can be minimized simultaneously by combining their expressions seen above. it is important to note that the individual squared error formulas from the above two terms are identical in terms of estimating both $\theta^{(1)}, \dots, \theta^{(n_u)}$ and $x^{(1)}, \dots, x^{(n_m)}$

the first summation is summed over all movies i rated by user j

the second summation is conversely summed over all user j that have rated movie i

thus, the two squared error summations are combined in respects to summing all instances of (i,j) where $r(i,j) = 1$. additionally, the regularization parameters for each expression are comprised in the simultaneous minimization mechanism as they individually regularize both $x_k^{(j)}$ and $\theta_k^{(j)}$ altogether minimizing as one term:

$$\min_{\substack{x^{(1)},\dots,x^{(m)} \\ \theta^{(1)},\dots,\theta^{(n_u)}}} J\left(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}\right)$$

as opposed to the previous approach of random initialization of θ then x then θ etc...

guess $\theta \to x \to \theta \to x \to \theta \to x \to \cdots$

the $x_0 = 1$ convention is removed and thus $x \in \mathbb{R}^n$ as opposed to previously $x \in \mathbb{R}^{n+1}$ the same is assumed for θ_0 and thus $\theta \in \mathbb{R}^n$

collaborative filtering algorithm application

1. initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random variables
2. minimize $J\left(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}\right)$ using gradient descent
    1. (or an advanced optimization algorithm)
    2. e.g. for every $j = 1, \dots, n_u, i = 1, \dots, n_m$:

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

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

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

In the algorithm we described, we initialized $x^{(1)}, \dots, x^{(n_m)}$ and $\theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values. Why is this?

- This step is optional. Initializing to all 0's would work just as well.
- Random initialization is always necessary when using gradient descent on any problem.
- This ensures that $x^{(i)} \neq \theta^{(j)}$ for any i, j.
- This serves as symmetry breaking (similar to the random initialization of a neural network's parameters) and ensures the algorithm learns features $x^{(1)}, \dots, x^{(n_m)}$ that are different from each other.

Correct Response

collaborative filtering ⇩ low rank matrix factorization

vectorization: low rank matrix factorization

movie	$user_1$	$user_2$	$user_3$	$user_4$
$movie_1$	5	5	0	0
$movie_2$	5	?	?	0
$movie_3$	?	4	0	?
$movie_4$	0	0	5	4
$movie_5$	0	0	5	?

$n_m = 5$

$n_u = 4$

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

$y^{(i,j)}$

collaborative filtering → low rank matrix factorization

predicted ratings:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix} \begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \cdots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \cdots & (\theta^{(n_u)})^T(x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \cdots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

$$X = \begin{bmatrix} - & (x^{(1)})^T & - \\ - & (x^{(2)})^T & - \\ - & \vdots & - \\ - & (x^{(n_m)})^T & - \end{bmatrix} \quad \Theta = \begin{bmatrix} - & (\theta^{(1)})^T & - \\ - & (\theta^{(2)})^T & - \\ - & \vdots & - \\ - & (\theta^{(n_u)})^T & - \end{bmatrix}$$

$(\theta^{(1)})^T(x^{(1)}) \rightarrow X\Theta^T$

Let $X = \begin{bmatrix} - & (x^{(1)})^T & - \\ & \vdots & \\ - & (x^{(n_m)}) & - \end{bmatrix}$, $\Theta = \begin{bmatrix} - & (\theta^{(1)})^T & - \\ & \vdots & \\ - & (\theta^{(n_u)}) & - \end{bmatrix}$.

What is another way of writing the following:

$$\begin{bmatrix} (x^{(1)})^T(\theta^{(1)}) & \cdots & (x^{(1)})^T(\theta^{(n_u)}) \\ \vdots & \ddots & \vdots \\ (x^{(n_m)})^T(\theta^{(1)}) & \cdots & (x^{(n_m)})^T(\theta^{(n_u)}) \end{bmatrix}$$

- $X\Theta$
- $X^T\Theta$
- $X\Theta^T$ (Correct Response)
- $\Theta^T X^T$

using the learned features to find related movies

for each product i, the algorithm learns feature vector $x^{(i)} \in \mathbb{R}^n$

x_1 = romance, x_2 = action, x_3 = comedy, x_4 = ...

how to find moves j related to movie i:

small $\|x^{(i)} - x^{(j)}\|$ → movie j and i are 'similar'

5 most similar movies to movie i:

find the 5 movies j with the smallest distance between features $\|x^{(i)} - x^{(j)}\|$

implementational detail: mean normalization

consideration of an example where a user has not rated any movies

movie	$user_1$	$user_2$	$user_3$	$user_4$	**$user_5$**
$movie_1$	5	5	0	0	**?**
$movie_2$	5	**?**	**?**	0	**?**
$movie_3$	**?**	4	0	**?**	**?**
$movie_4$	0	0	5	4	**?**
$movie_5$	0	0	5	**?**	**?**

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

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

$n = 2$ features to learn and $\theta^{(5)} \in \mathbb{R}^2$ $\quad \theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ minimize $\frac{\lambda}{2}\left[\left(\theta_1^{(5)}\right)^2 + \left(\theta_2^{(5)}\right)^2\right]$

minimizing the regularization parameter encourages the algorithm to set $\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and thus will predict **$user_5$** as giving all 0 ratings for the examples → $\left(\theta^{(5)}\right)^T \left(x^{(i)}\right) = 0$

this problem is addressed with mean normalization

mean normalization

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$ stored in matrix as average $\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$

then subtract off the average rating matrix μ from matrix Y

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

$\rightarrow$ learn $\theta^{(j)}, x^{(i)}$

treat the new matrix Y as the orignial ratings of movies given by users

for user j, on movie i, predict: $\left(\theta^{(j)}\right)^T \left(x^{(i)}\right) + \boldsymbol{\mu_i}$

user$_5$: $\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\left(\theta^{(5)}\right)^T \left(x^{(i)}\right) + \boldsymbol{\mu_i}$

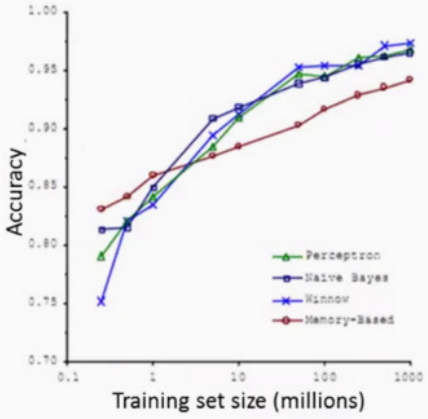
because minimization of the regularization parameter promoted a zero assignment to parameter vector $\theta^{(5)}$, mean normailzation will simplay apply the addition of μ_i vector for each movie rating example predicted for **user$_5$**

$$\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 5 & 5 & 0 & 0 & \mathbf{2.5} \\ 5 & ? & ? & 0 & \mathbf{2.5} \\ ? & 4 & 0 & ? & \mathbf{2} \\ 0 & 0 & 5 & 4 & \mathbf{2.25} \\ 0 & 0 & 5 & 0 & \mathbf{1.25} \end{bmatrix}$$

mean normalization is a method of feature scaling applied in machine learning algorithms. Unlike other applications of feature scaling, this method did not scale the movie ratings by dividing by the range (max – min value).

this is due to the ratings already having comparable scales (0 to 5 stars)

large scale machine learning



learning with large datasets

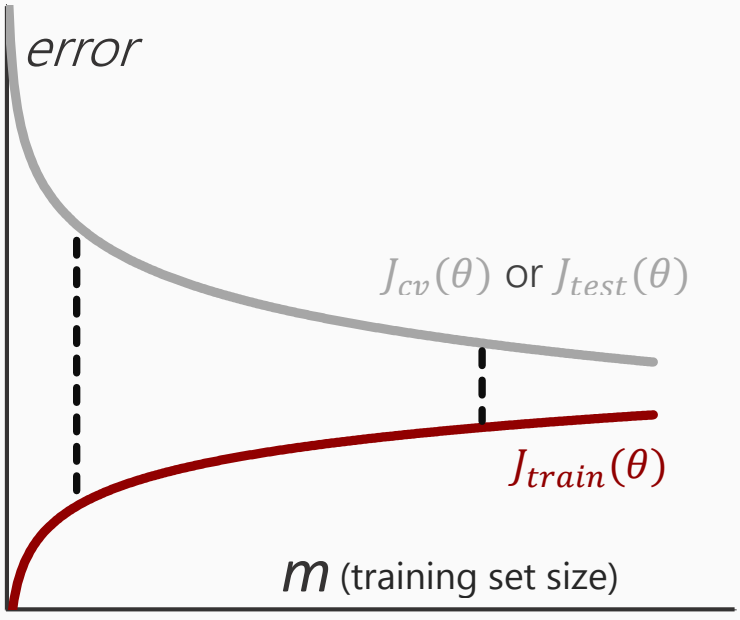
the importance of accessibility of large amounts of data, it has been noted over time that it is not necessarily who has the best algorithm that can solve a problem, but who has the most data

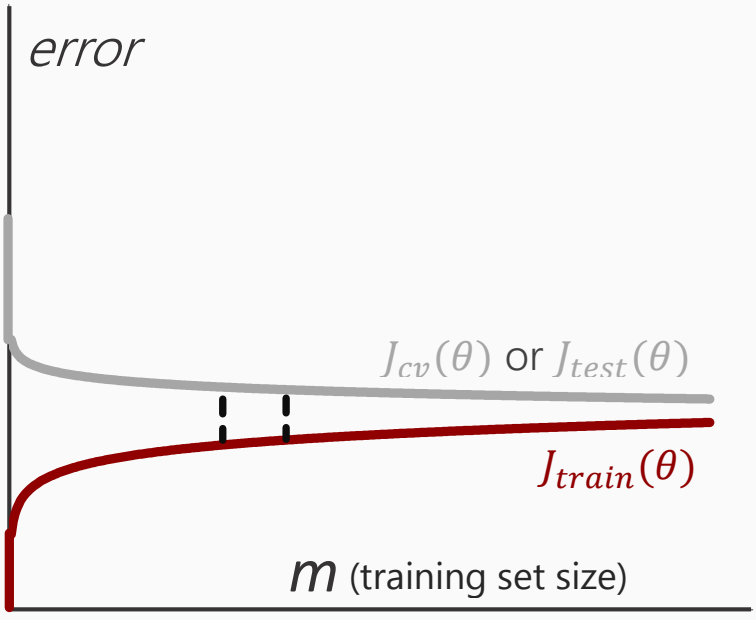
many cases provide datasets with millions of records

m = 100,000,000

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

the model for running iterations of gradients descent consequentially becomes very computationally expensive to implement. a sanity check should determine if the algorithm will work equally well on a much smaller subset of the data → m = 1,000. this can be evaluated by determining that the data still contains a high variance (left illustration) in a smaller sample but does not experience a high bias (right illustration) when m = 1,000





Suppose you are facing a supervised learning problem and have a very large dataset (m = 100,000,000). How can you tell if using all of the data is likely to perform much better than using a small subset of the data (say m = 1,000)?

- There is no need to verify this; using a larger dataset always gives much better performance.
- Plot $J_{\text{train}}(\theta)$ as a function of the number of iterations of the optimization algorithm (such as gradient descent).
- Plot a learning curve ($J_{\text{train}}(\theta)$ and $J_{\text{CV}}(\theta)$, plotted as a function of m) for some range of values of m (say up to m = 1,000) and verify that the algorithm has bias when m is small.
- Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.

Correct Response

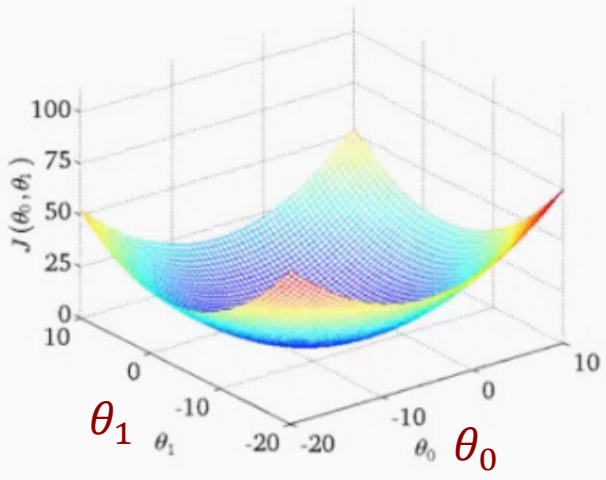
stochastic gradient descent

stochastic gradient descents allows an algorithm to scale to larger datasets

linear regression with gradient descent

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

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

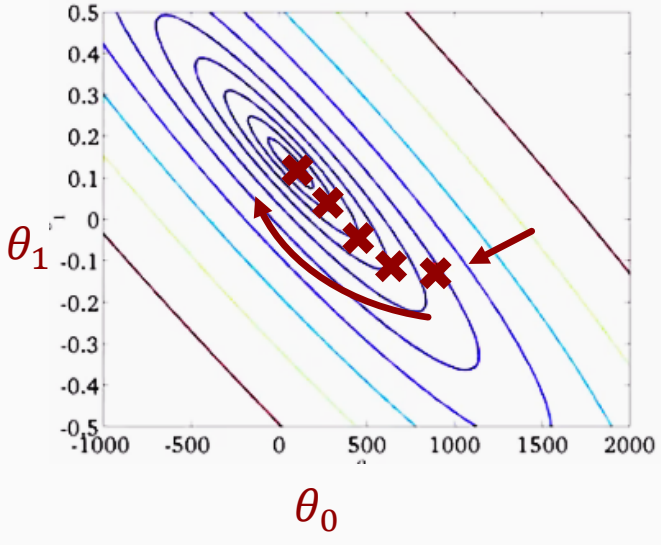


repeat {

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

(for every $j = 0, \ldots, n$)

}



reiterating to find the global minimum

in the circumstance where $m = 300{,}000{,}000$, the method of **'batch gradient descent'** implies that all 300,000,000 iterations would be stored in the machine's memory in order to take the first step towards finding the global minimum in gradient descent

a separate algorithm that applies a single training example in one iteration as opposed to every training example in all iterations addresses computational cost of the above

batch gradient descent

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

repeat {

$$\theta_j := \theta_j - a\underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}}_{\frac{\partial}{\partial\theta_j}J_{train}(\theta)}$$

(for every $j = 0, \dots, n$)

}

stochastic gradient descent

$$\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)$$

1. randomly shuffle dataset
2. repeat {

for $i = 1, \dots, m$ {

$$\theta_j := \theta_j - a\underbrace{\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}}_{\frac{\partial}{\partial\theta_j}\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)}$$

(for $j = 0, \dots, n$)

}

}

stochastic gradient descent scans through the training examples, first looking at the first example $\left(x^{(1)}, y^{(1)}\right)$ and taking a gradient descent step only in respects to the first example. the algorithm will proceed to the second training example $\left(x^{(2)}, y^{(2)}\right)$ and perform the same process of advancing another step in respects to the cost of the second example. similarly, the algorithm will proceed to the third training example $\left(x^{(3)}, y^{(3)}\right)$ in attempt to fit it slightly better, and so on... (random shuffling necessary)

the algorithm essentially takes the form of summation in batch gradient descent $\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$ and instead applies the metric in a single iteration $\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$ through stochastic gradient descent to benign making improvements to the algorithm with each individual iteration.

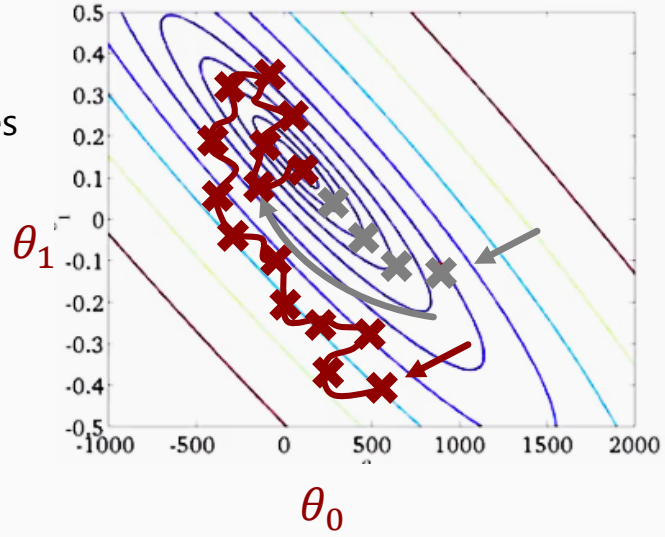
stochastic gradient descent

1. randomly shuffle (reorder) training examples
2. repeat {
   for $i = 1, \ldots, m$ {

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

   (for $j = 0, \ldots, n$)

   }
}



The nature of **stochastic gradient descent** does **not** converge in the same sense that **batch gradient descent** is expected to. stochastic gradient descent will tend to wander around the parameter space in efforts to find the global minimum as opposed to moving in a more linear fashion. Regardless, the algorithm will still arrive at a respectable hypothesis

if *m* is extraordinarily large (e.g. *m* = 300,000,000), the first iteration of stochastic gradient descent might actually be enough to form a suitable hypothesis forming a parameter that arrives at the global minimum (opposed to multiple reiterations)

Which of the following statements about stochastic gradient descent are true? Check all that apply.

- [x] When the training set size m is very large, stochastic gradient descent can be much faster than gradient descent.

Correct Response

- [x] The cost function $J_{\text{train}}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)})^2$ should go down with every iteration of batch gradient descent (assuming a well-tuned learning rate α) but not necessarily with stochastic gradient descent.

Correct Response

- [ ] Stochastic gradient descent is applicable only to linear regression but not to other models (such as logistic regression or neural networks).

Correct Response

- [x] Before beginning the main loop of stochastic gradient descent, it is a good idea to "shuffle" your training data into a random order.

Correct Response

mini · batch gradient descent

batch gradient descent: use all m examples in each iteration

stochastic gradient descent use 1 single example in each iteration

mini-batch gradient descent: use b examples in each iteration

$b = \text{mini} - \text{batch size}$ typically batches range from 2 – 100

in the case of 10 examples ($b = 10$)

get $b = 10$ examples $(x^{(i)}, y^{(i)}), \dots, (x^{(i+9)}, y^{(i+9)})$

$$\theta_j := \theta_j - a\frac{1}{10}\sum_{k=i}^{i+9}\left(h_\theta(x^{(k)}) - y^{(k)}\right)x_j^{(k)}$$

$$i := i + 10$$

when comparing stochastic gradient descent (1 example) to mini-batch gradient descent (b examples), the reason behind using mini-batch gradient descent is when **vectorization** is possible.

this allows the user of a numerical linear algebra library to **parallelize** the

formally: $b = 10$, $m = 1{,}000$

1. repeat {

for $i = 1, 11, 21, 31, \dots, 991$ {

$$\theta_j := \theta_j - a\frac{1}{10}\sum_{k=i}^{i+9}\left(h_\theta(x^{(k)}) - y^{(k)}\right)x_j^{(k)}$$

(for every $j = 0, \dots, n$)

}

}

Suppose you use mini-batch gradient descent on a training set of size m, and you use a mini-batch size of b. The algorithm becomes the same as batch gradient descent if:

- b = 1
- b = m / 2
- b = m

Correct Response

- None of the above

stochastic gradient descent convergence

checking for gradient descent convergence

batch gradient descent:

plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent:

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

stochastic gradient descent:

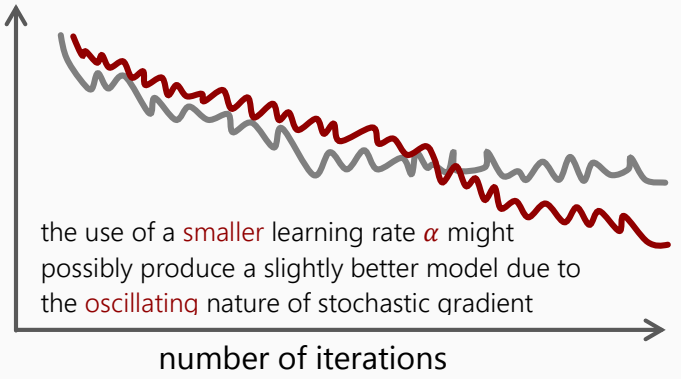
$$\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

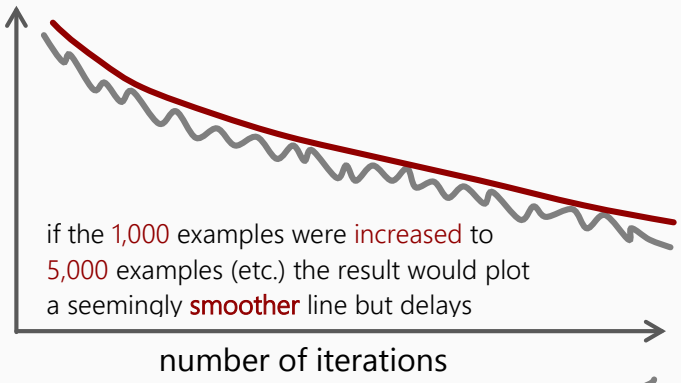
while the algorithm is learning, compute $\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)$ prior to updating θ using the training example $\left(x^{(i)}, y^{(i)}\right)$

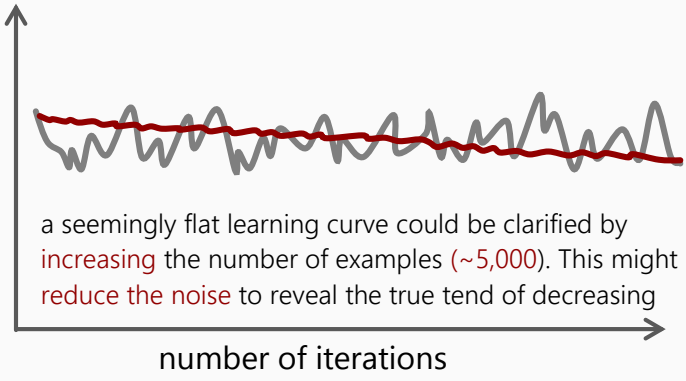
every 1,000 iterations (example), plot $\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)$ averaged over the last 1,000 examples processed by the algorithm

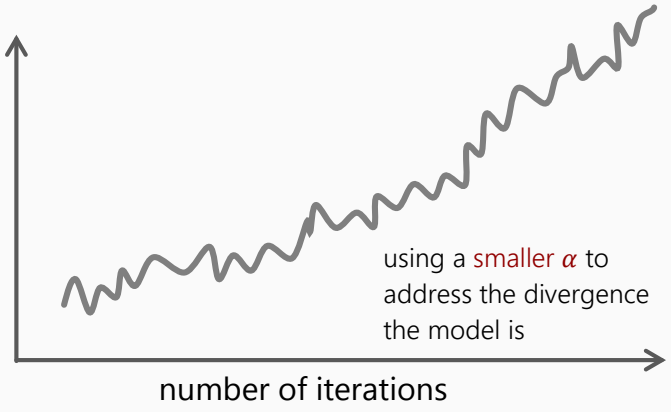
as stochastic gradient descent is scanning through the training set, before θ is updated using a particular training example $\left(x^{(i)}, y^{(i)}\right)$, compute how well the hypothesis is performing on training example $\left(x^{(i)}, y^{(i)}\right)$

plot $\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right)$, averaged over the last 1,000 (example number) examples:



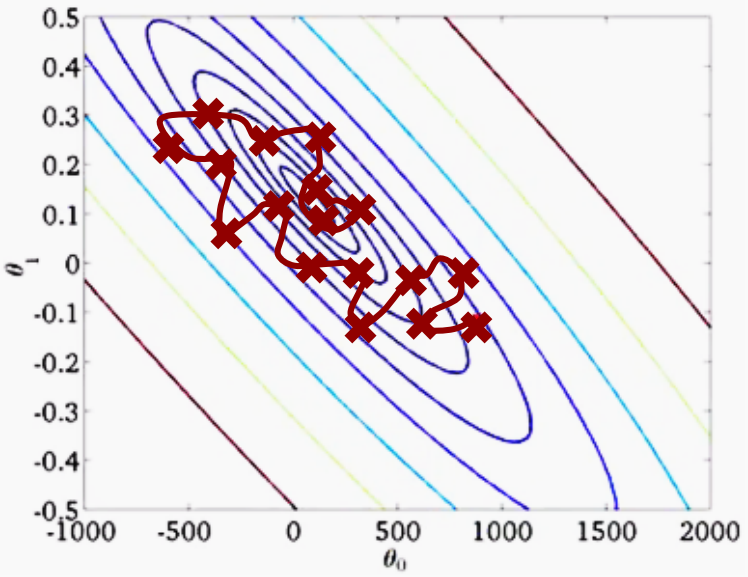






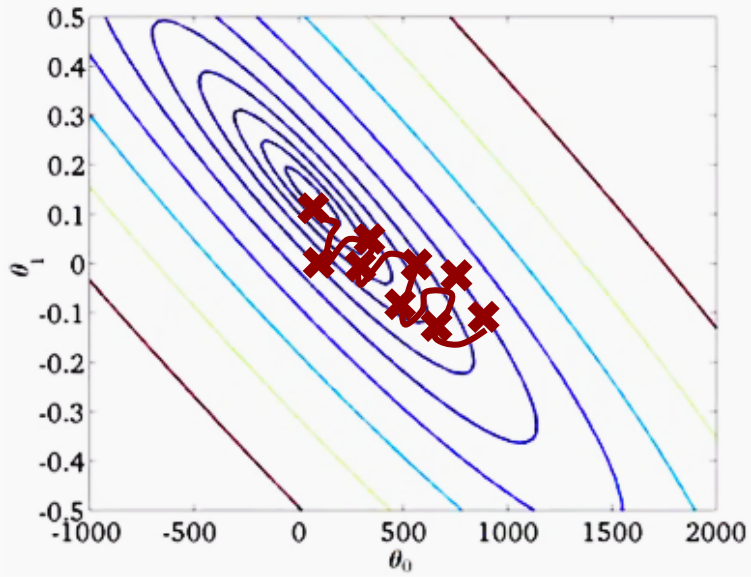
stochastic gradient descent

typical behavior of stochastic gradient descent is representative of the model wandering around the parameter space while converging but never actually settles on the global minimum. instead, the algorithm will essentially wander around the global minimum indefinitely (close convergence but not quite exact)



$$\text{cost}\left(\theta\left(x^{(i)}, y^{(i)}\right)\right) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

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



1. randomly shuffle (reorder) training examples
2. repeat {
   for $i = 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }

learning rate α is typically held constant. α can be slow decreased over time if θ is needing to converge (e.g. $\alpha = \frac{const1}{interationNumber + const2}$)

in effect, the algorithm with a properly adjusted α will experience smaller and smaller oscillations under convergence and arrive at an ultimately suitable global minimum

Which of the following statements about stochastic gradient descent are true? Check all that apply.

- [ ] Picking a learning rate α that is very small has no disadvantage and can only speed up learning.

Correct Response

- [x] If we reduce the learning rate α (and run stochastic gradient descent long enough), it's possible that we may find a set of better parameters than with larger α.

Correct Response

- [ ] If we want stochastic gradient descent to converge to a (local) minimum rather than wander of "oscillate" around it, we should slowly increase α over time.

Correct Response

- [x] If we plot $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ (averaged over the last 1000 examples) and stochastic gradient descent does not seem to be reducing the cost, one possible problem may be that the learning rate α is poorly tuned.

Correct Response

online learning

shipping example

shipping service website where a user comes, specifies the origin and destination for shipping. The package is offered to ship for a certain asking price, and the user sometimes choose to user the shipping service $(y = 1)$ or not $(y = 0)$

features x capture properties of the user, the origin, the destination, and the asking price. the aim is to learn $p(y = 1|x; \theta)$ to optimize the shipping price offered

logistic regression approach

repeat indefinitely {
obtain (x, y) corresponding to the user
update θ using (x, y): $(x^{(i)}, y^{(i)})$
$\theta_j := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \quad (j = 0, \dots, n)$
}

the notion of using (x, y) opposed to the typically denoted $(x^{(i)}, y^{(i)})$ is due to the presence of streaming data and no fixed training set being used. Each example in the expression is used for learning and subsequently discarded for the next examples received in real-time, or near real-time.

The significance of this type of online learning algorithm is the **adaptability** to changing user preferences.

product search example (learning to search)

users search for "android phone 1080p camera"

100 phones in the online store of which will **return 10 results** (*10 training examples per search)*

x = features of phone, how many words in user query match description, etc…

$y = 1$ if user clicks on link, $y = 0$ otherwise

learn $p(y = 1|x; \theta)$ → predicted click-through-rate (CTR)

algorithm used to show the user 10 phones they are most likely to click on

other examples

special offers to show user; news article selection; product recommendation, etc…

Some of the advantages of using an online learning algorithm are:

- [x] It can adapt to changing user tastes (i.e., if $p(y|x;\theta)$ changes over time).

Correct Response

- [ ] There is no need to pick a learning rate α.

Correct Response

- [x] It allows us to learn from a continuous stream of data, since we use each example once then no longer need to process it again.

Correct Response

- [ ] It does not require that good features be chosen for the learning task.

Correct Response

map reduce and data parallelism

map reduce

assuming a sample of $m = 400$ (actual application would imply a sample more relative to $m = 400{,}000{,}000$; reduced for formula clarity in the following example)

batch gradient descent:

$$\theta_j := \theta_j - a\frac{1}{400}\sum_{i=1}^{400}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

machine 1: use $\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(100)}, y^{(100)}\right)$ $\quad \text{temp}_j^{(1)} = \sum_{i=1}^{100}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

machine 2: use $\left(x^{(101)}, y^{(101)}\right), \ldots, \left(x^{(200)}, y^{(200)}\right)$ $\quad \text{temp}_j^{(2)} = \sum_{i=101}^{200}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

machine 3: use $\left(x^{(201)}, y^{(201)}\right), \ldots, \left(x^{(300)}, y^{(300)}\right)$ $\quad \text{temp}_j^{(3)} = \sum_{i=201}^{300}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

machine 4: use $\left(x^{(301)}, y^{(301)}\right), \ldots, \left(x^{(400)}, y^{(400)}\right)$ $\quad \text{temp}_j^{(4)} = \sum_{i=301}^{400}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

combine:

$$\theta_j := \theta_j - a\frac{1}{400}\sum_{i=1}^{400}\left(\text{temp}_j^{(1)} + \text{temp}_j^{(2)} + \text{temp}_j^{(3)} + \text{temp}_j^{(4)}\right) \qquad (j = 0, \ldots, n)$$

map reduce is an applicable technique when the learning algorithm can be expressed as computing sums of functions over the training set

application example photo ocr

problem description and pipeline

photo optical character recognition (OCR)

photo ocr pipeline

1. text detection
2. character segmentation
3. character classification → S n r
    1.

When someone refers to a "machine learning pipeline," he or she is referring to:

- A PhotoOCR system.
- A character recognition system.
- A system with many stages / components, several of which may use machine learning.

Correct Response

- An application in plumbing. (Haha.)

supervised learning for pedestrian detection

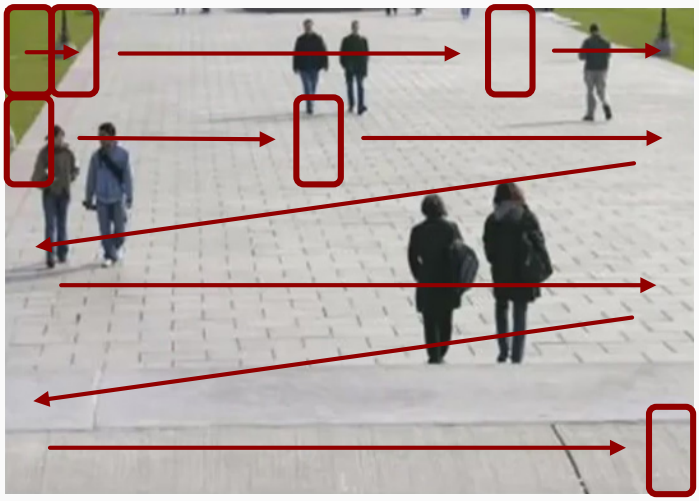
x = pixels in an 82x36 image patch

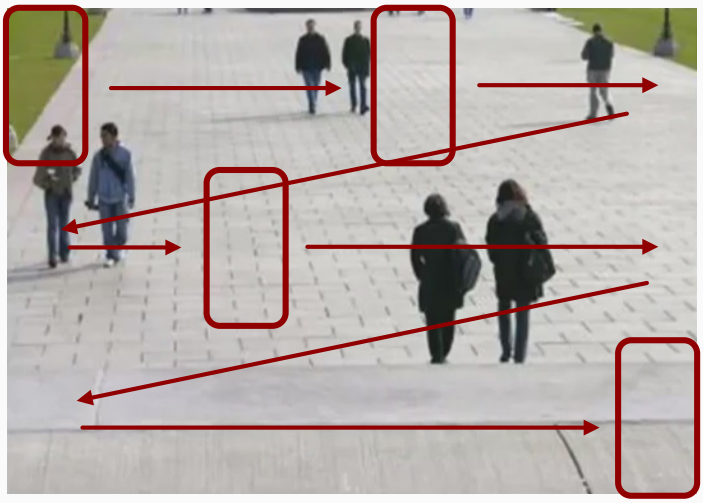
positive examples ($y = 1$)

negative examples ($y = 0$)

sliding window detection

step-size/stride is determined by pixel length and scaled upward after each iteration

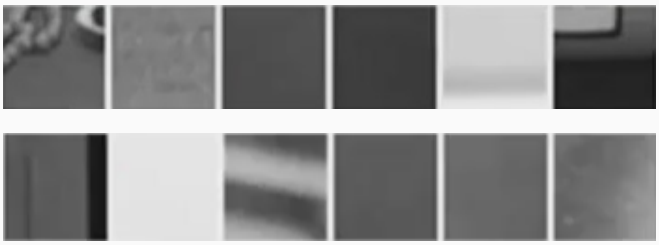




the process of training a sliding window classifier in order to detect appropriately positive examples ($y = 1$) in the problem of detecting pedestrians within the presence of a given images

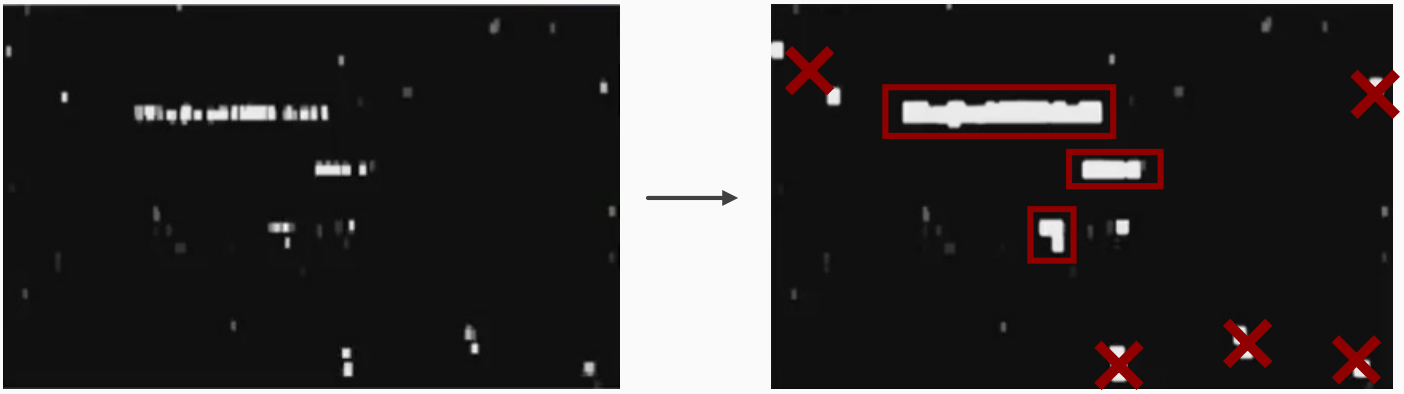
text detection

positive examples ($y = 1$)



negative examples ($y = 0$)

in a text detection algorithm, the process scans through an image using sliding windows and maps out the algorithm's determination of text presence based on the black and white coding (center image). the algorithm subsequently performs mathematical "expansion" in order to amplify the areas identified as containing text



a heuristic will be applied in a manner suitable for expectation of what constitutes text (e.g. text boxes should be much wider than they are tall, implying that smaller rectangular text identification areas are deemed to not represent the presence of text)

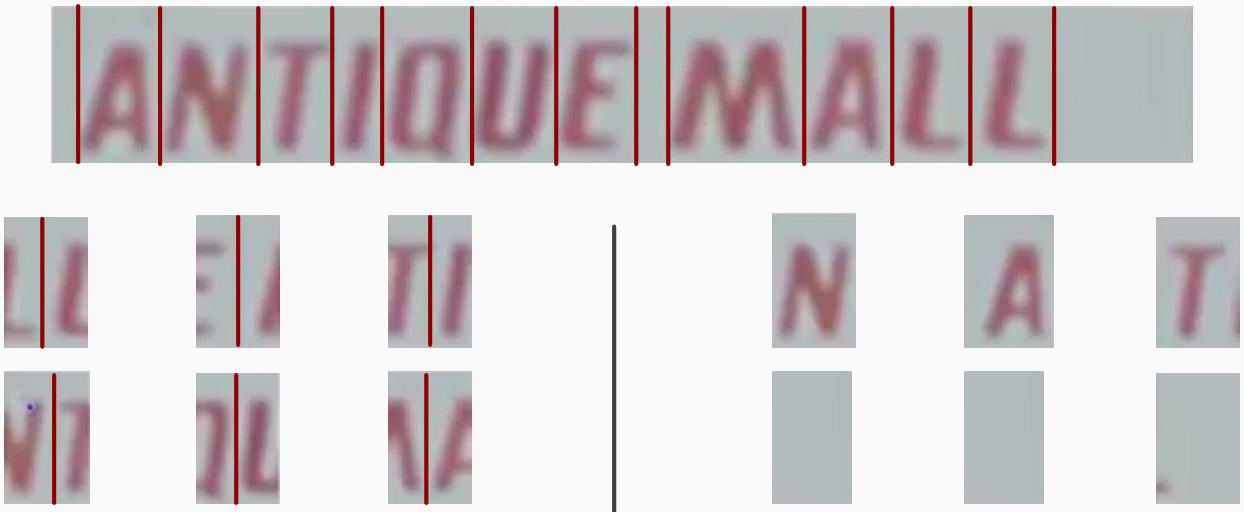
Suppose you are running a text detector using 20x20 image patches. You run the classifier on a 200x200 image and when using sliding window, you "step" the detector by 4 pixels each time. (For this problem assume you apply the algorithm at only one scale.) About how many times will you end up running your classifier on a single image? (Pick the closest answer.)

- About 100 times.
- About 400 times.
- About 2,500 times.

Correct Response

- About 40,000 times.

1 dimension window for character segmentation



positive examples ($y = 1$) negative examples ($y = 0$)

the next step will determine whether or not there is a split between any of the detected training examples. positive examples will represent those segments that contain splits and the negative examples to the algorithm will label images being absent of a split

getting lots of data and artificial data

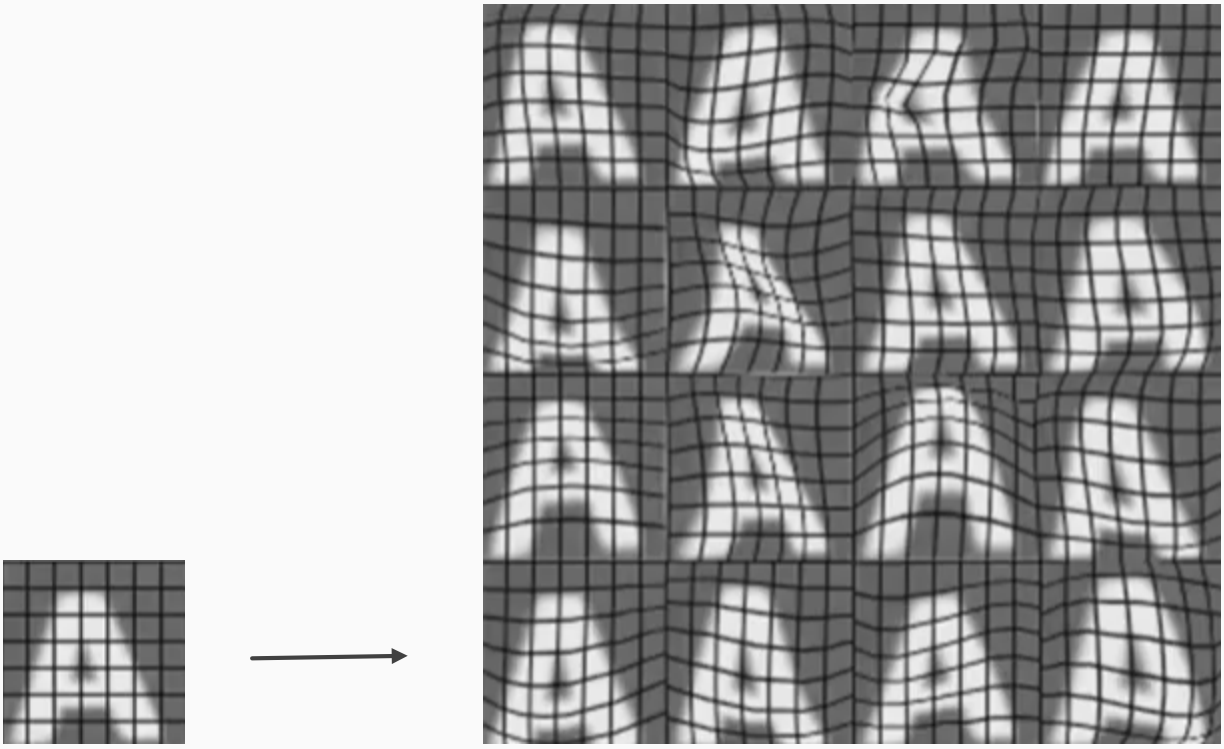
artificial data sysnthesis via photo ocr

real data

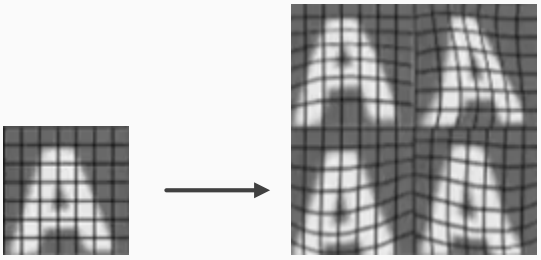
synthetic data

in the example of photo ocr, data can be taken in the form of many different fonts and scripts and synthesized against random backgrounds to create larger training sets

synthesizing data by introducing distortions

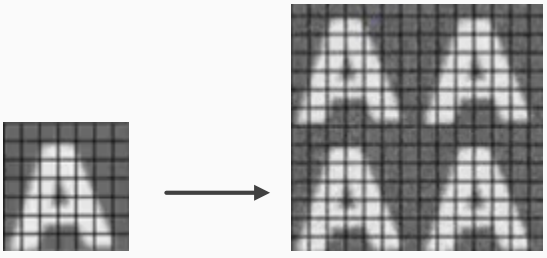


the same methods can be applied to image distortions and audio clips alike
distortion introduced should be a representation of the type of noise/distortion likely present to the test set



audio:
background noise,
poor cellphone

it is typically not valuable to add random/meaningless noise to the dataset



x_i =intensity (brightness)of pixel i

Suppose you are training a linear regression model with m examples by minimizing:

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

Suppose you duplicate every example by making two identical copies of it. That is, where you previously had one example $(x^{(i)}, y^{(i)})$, you now have two copies of it, so you now have 2m examples. Is this likely to help?

- Yes, because increasing the training set size will reduce variance.
- Yes, so long as you are using a large number of features (a "low bias" learning algorithm).
- No. You may end up with different parameters θ, but they are unlikely to do any better than the ones learned from the original training set.
- No, and in fact you will end up with the same parameters θ as before you duplicated the data.

Correct Response

important aspects of obtaining more data

ensure a low bias classifier is used before expending effort (plot learning curves to verify). E.g. increase the number of features/number of hidden units in a neural network until a low bias classifier is obtained

determine and value the difficulty of obtaining as much as 10x current data

- artificial data synthesis
- collect and label data manually
- "crowd source" (e.g. amazon mechanical turk)

You've just joined a product group that has been developing a machine learning application for the last 12 months using 1,000 training examples. Suppose that by manually collecting and labeling examples, it takes you an average of 10 seconds to obtain one extra training example. Suppose you work 8 hours a day. How many days will it take you to get 10,000 examples? (Pick the closest answer.)

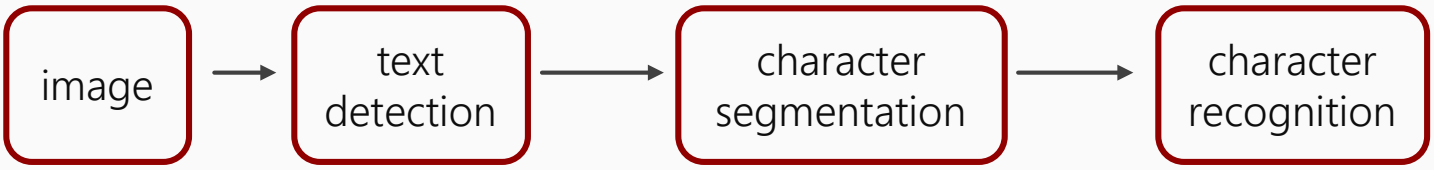
- About 1 day.
- About 3.5 days.

Correct Response

- About 28 days.
- About 200 days.

ceiling analysis: part of the pipeline to work on next

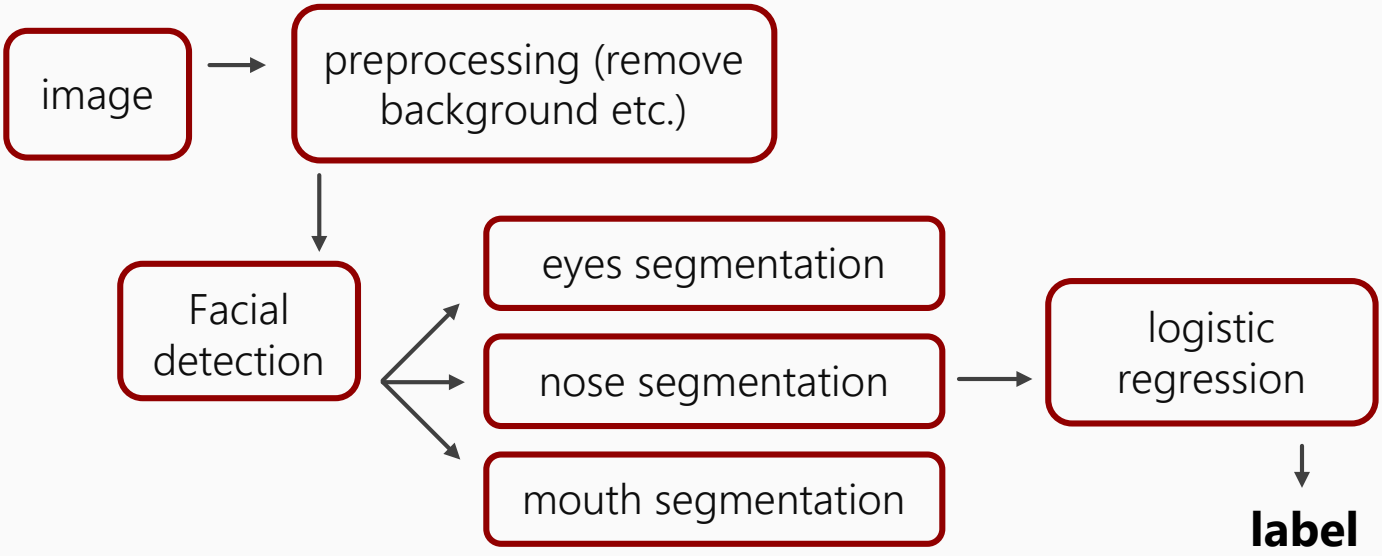
estimating errors due to each component (ceiling analysis)



determining part of the pipeline to spent the most time improving (resource allocation)

component	accuracy	improvement
overall system	72%	-
text detection	89%	17% ↑
character segmentation	90%	1% ↑
character recognition	100%	10% ↑

ceiling analysis: facial recognition example pipeline



component	accuracy	improvement
overall system	85%	-
preprocessing	85.1%	0.1% ↑
face detection	91%	5.9% ↑
eyes segmentation	95%	4% ↑
nose segmentation	96%	1% ↑
mouth segmentation	97%	1% ↑
logistic regression	100	3% ↑

it is important to remain conscious of time-spent to performance gains

machine learning formulae and expressions

linear regression

hypothesis:

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

parameters:

$$\theta_0, \theta_1, \ldots, \theta_n$$

cost function:

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

goal:

$$\min_{\theta_0, \theta_1, \ldots, \theta_n} J(\theta_0, \theta_1, \ldots, \theta_n)$$

partial derivative gradient descent for multivariate linear regression:

repeat until convergence {

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

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

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

}

normal equation:

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

regularized multivariate linear regression:

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

$$\min_\theta J(\theta)$$

regularized gradient descent for multivariate linear regression: (for all j)

repeat until convergence {

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

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

(update θ_j for $j = ~~0~~ 1, 2, 3 \ldots, n$ simultaneously)

}

alternative notation for θ_j update: $\theta_j := \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

regularized normal equation:

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

logistic regression

hypothesis:

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

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T X}} \quad \rightarrow \quad \text{sigmoid/logistic function}$$

interpretation:

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

cost function:

$$\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(h_\theta\left(x^{(i)}\right)\right) & \text{if } y = 1 \\ -\log\left(1 - h_\theta\left(x^{(i)}\right)\right) & \text{if } y = 0 \end{cases}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

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

partial derivative gradient descent for multivariate logistic regression:

repeat until convergence {

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

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

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

}

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

regularized multivariate logistic regression:

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

regularized gradient descent for multivariate logistic regression: (for all j)

repeat until convergence {

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

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

(update θ_j for $j = 0, 1, 2, 3 \ldots, n$ simultaneously)

}

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

neural networks

backpropagation algorithm

gradient computation

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{i=1}^{K} y_k^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right)\log\left(1 - \left(h_\theta\left(x^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_1}\sum_{j=1}^{s_1+1}\left(\theta_{ji}^{(l)}\right)^2$$

$$\min_\theta J(\theta)$$

gradient checking

$$\frac{\partial}{\partial\theta}J(\theta) \approx \frac{J(\theta+\varepsilon) - (\theta-\varepsilon)}{2\varepsilon}$$

train · validation · test error

training error:

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)^2$$

cross validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right), y_{cv}^{(i)}\right)^2$$

test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}\left(h_\theta\left(x_{test}^{(i)}\right), y_{test}^{(i)}\right)^2$$

precision:

$$\frac{\text{true positives}}{\#\ \textbf{predicted}\ \text{positives}} = \frac{\text{true positives}}{\text{true positives} + \textbf{false positives}}$$

recall:

$$\frac{\text{true positives}}{\#\ \textbf{actual}\ \text{positives}} = \frac{\text{true positives}}{\text{true positives} + \textbf{false negatives}}$$

support vector machines

optimization objective:

$$\min_\theta \frac{1}{m} C\sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + \left(1 - y^{(i)}\right)\text{cost}_0\left(\theta^T x^{(i)}\right) + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

hypothesis output:

$$h_\theta x = \begin{cases} 1 \text{ if } \theta^T x \text{ is } \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

decision boundary:

$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$

kernels

similarity:

$$f_1 = \text{similarity}(x, \ell^{(1)}) = \exp\left(-\frac{\left\|x - \ell^{(1)}\right\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}\left(x_j - \ell_j^{(1)}\right)^2}{2\sigma^2}\right)$$

k-means algorithm

optimization objective:

$$J\left(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - \mu_c{}^{(i)}\right\|^2$$

anomaly detection algorithm

gaussian distribution:

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$$p(x) = \prod_{j=1}^{n} p\left(x_j; \mu_j, \sigma_j^2\right) = \prod_{j=1}^{n}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{\left(x_j - \mu_j\right)^2}{2\sigma_j^2}\right)$$

multivariate gaussian distribution:

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\,|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu)^T\,\Sigma^{-1}(x-\mu)\right)$$

recommender systems

optimization objective:

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

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

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

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

simultaneous gradient descent update:

(for $k = 0$)

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

(for $k \neq 0$)

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

collaborative filtering optimization algorithm:

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

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

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

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

collaborative filtering algorithm:

given features $x^{(1)}, \dots, x^{(n_m)}$, estimate parameters $\theta^{(1)}, \dots, \theta^{(n_u)}$:

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

given parameters $\theta^{(1)}, \dots, \theta^{(n_u)}$, estimate features $x^{(1)}, \dots, x^{(n_m)}$:

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

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

$$J\left(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}\right)$$

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

collaborative filtering algorithm update:

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

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