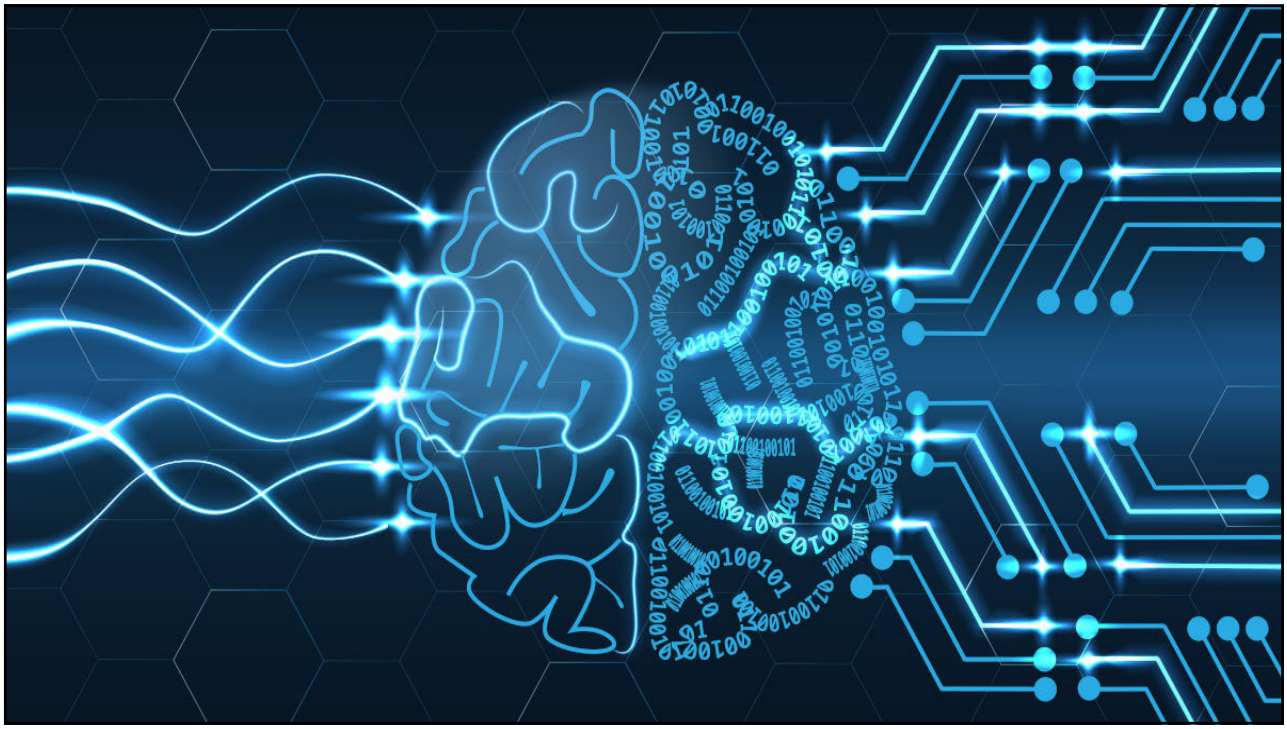
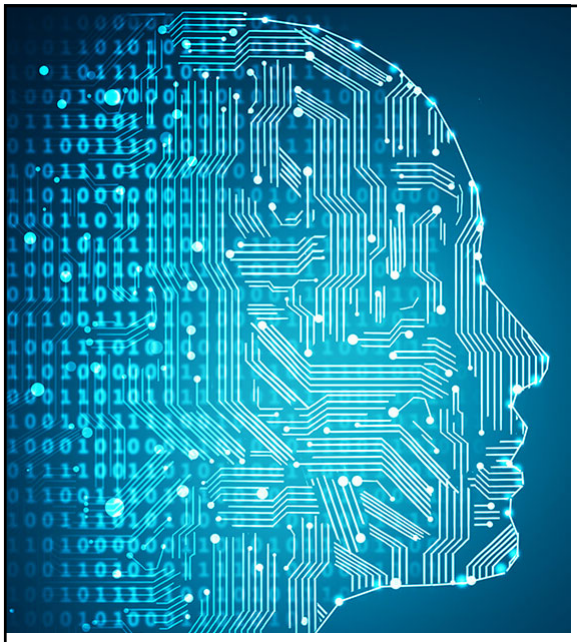


Machine Learning

By: Amin Jamili

Main Reference: Andrew NG Course



Introduction

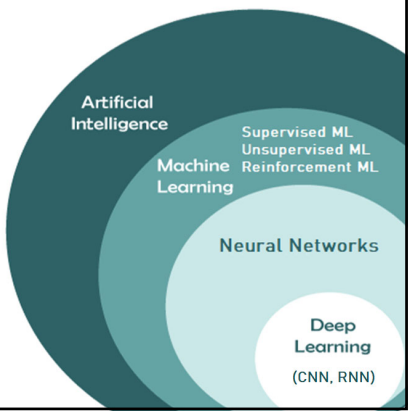
What is machine learning

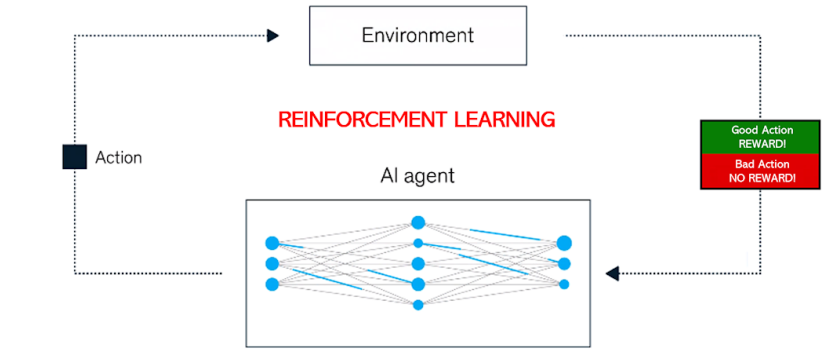
Machine Learning - Part 1
in abstract

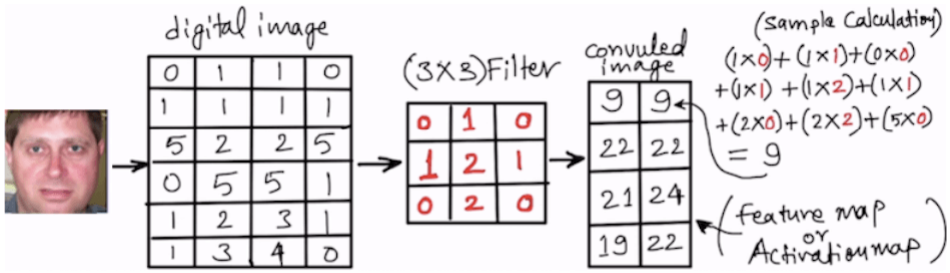
Machine Learning definition

- Machine learning is a field of artificial intelligence (AI) that focuses on developing algorithms that allow computer systems to automatically learn and improve from experience, **without being explicitly programmed.**

- The theory helps to understand the Practical advice for applying learning algorithms and debugging the algorithms

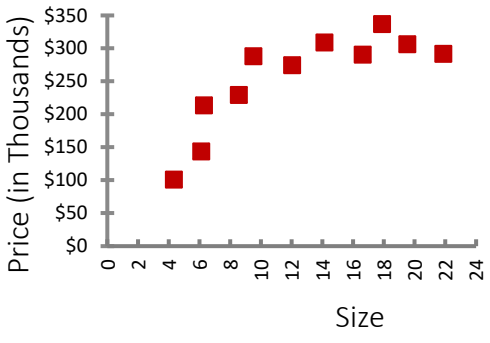




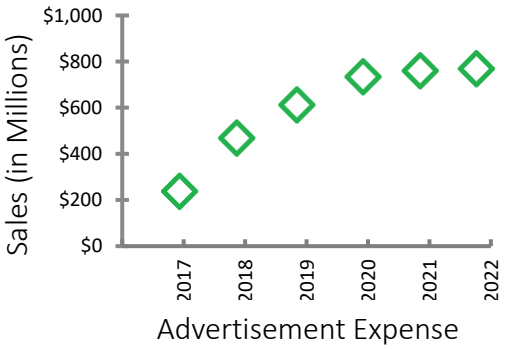


The Convolution step in CNN

Supervised Learning: Prediction



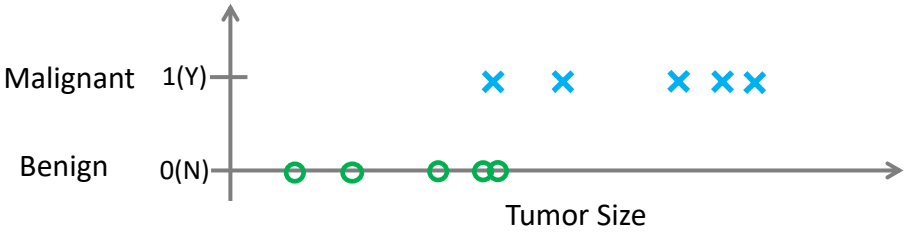
Housing price prediction.

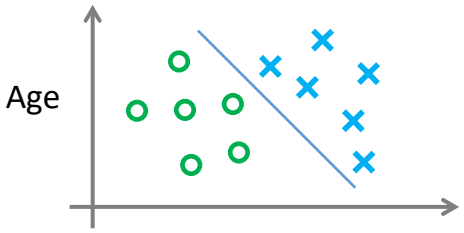


Company Sales Prediction

Supervised Learning: Classification

Breast cancer





- Uniformity of Cell Shape
- Uniformity of Cell Size

...

Other applications of Supervised Machine Learning

Time-series forecasting:

Practical examples: predicting stock prices, predicting traffic patterns, predicting demand for products or services.

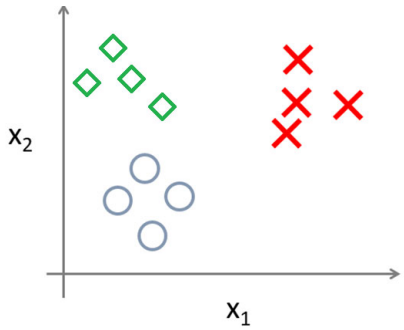
Object detection:

Practical examples: detecting and locating faces within an image, locating tumors within a medical image.

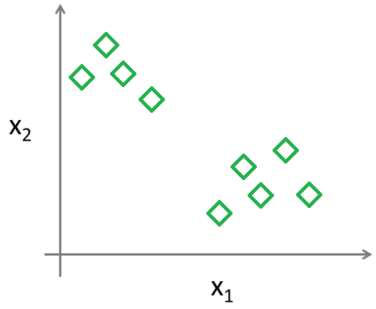
Natural language processing:

Practical examples: language translation, recognizing and responding to voice commands.

Supervised Learning



Unsupervised Learning



The main difference between supervised vs unsupervised learning is the need for **labelled training data**. Supervised machine learning relies on labelled input and output training data, whereas unsupervised learning processes unlabelled or raw data.

• Applications of Unsupervised learning

1.Clustering:
Customer segmentation, Data Anomaly detection

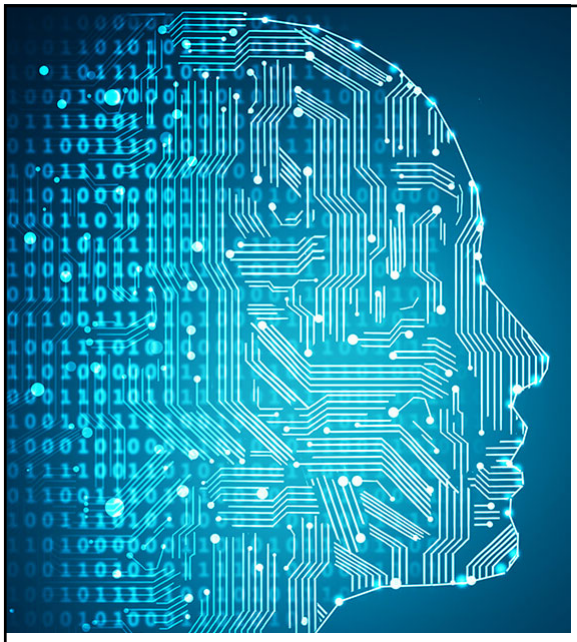
2. Dimensionality reduction:
Feature extraction (from high-dimensional data for classification usage), Data compression

3. Association rule learning:
Recommender systems (based on a user's past behavior or preference), Fraud detection

4. Generative models
Image generation, Data augmentation (for training machine learning models)

5.Anomaly detection:
Network traffic intrusion detection (for security breach), Medical diagnosis(To identify unusual patterns in medical data that may indicate a disease)

6.Topic modeling:
Text analysis (Identify topics in or summarization), Customer feedback analysis (Identify common themes in customer feedback to improve products or services)

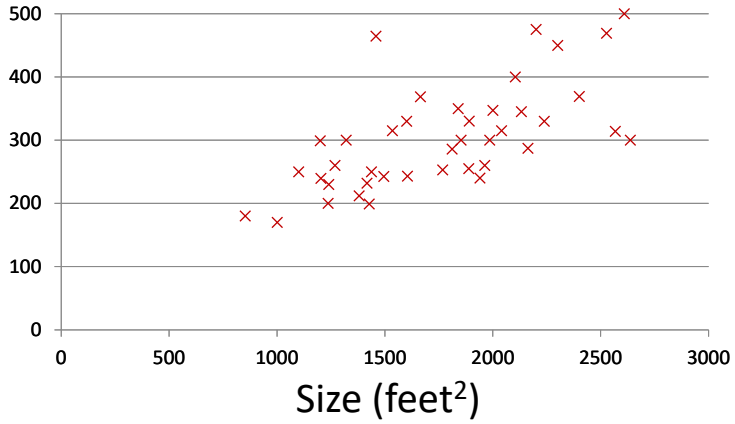


Machine Learning - Part 2
in abstract

Linear regression with one variable

Model representation and Cost Function

Housing Prices (Portland, OR)



Supervised Learning

Given the "right answer" for each example in the data.

Regression Problem

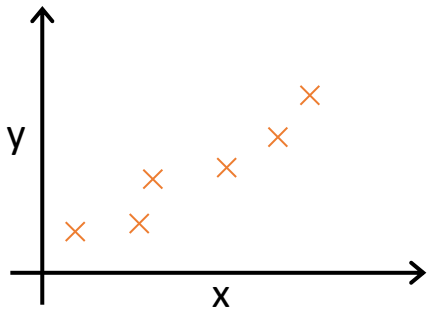
Predict real-valued output

Training set of housing prices (Portland, OR)

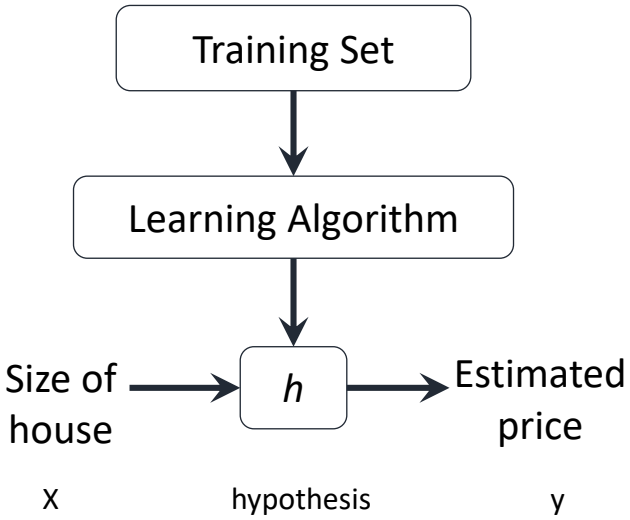
Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Notation:

- **m** = Number of training examples
- **x**'s = "input" variable / features
- **y**'s = "output" variable / "target" variable



Linear regression with one variable.



Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$ θ_i's: Parameters

Idea:

Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

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

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

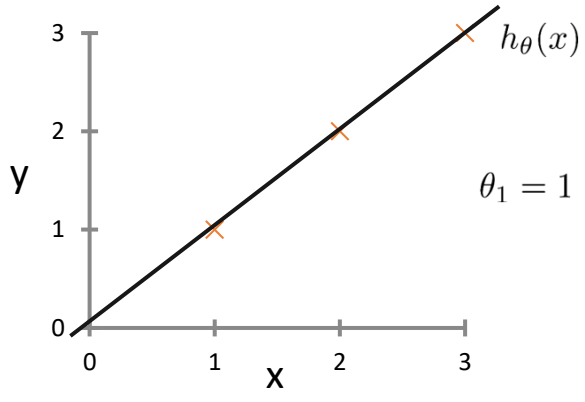
$$\theta_1$$

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

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

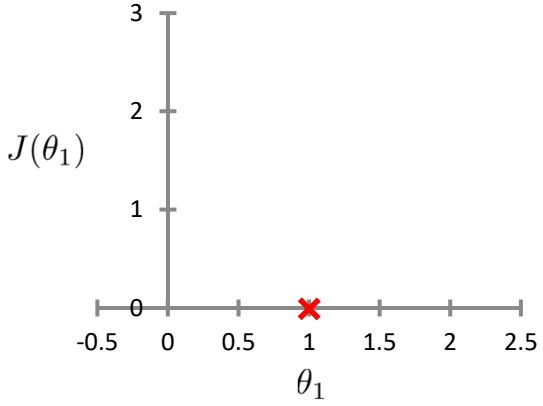
$h_\theta(x)$

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



$J(\theta_1)$

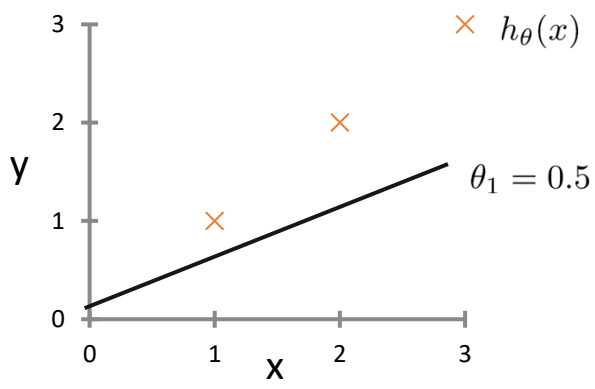
(function of the parameter θ_1)



$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 = 0$$

$h_\theta(x)$

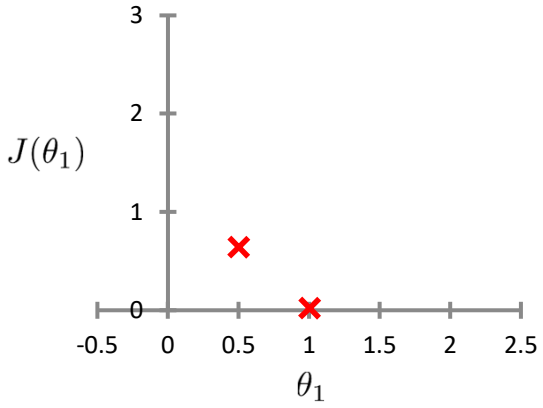
(for fixed θ_1, this is a function of x)



J(0.5)=0.58

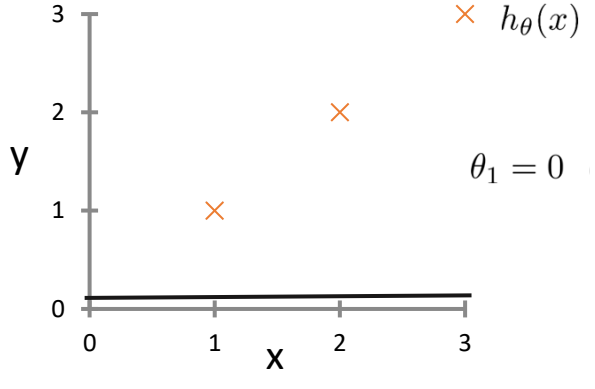
$J(\theta_1)$

(function of the parameter θ_1)



$h_\theta(x)$

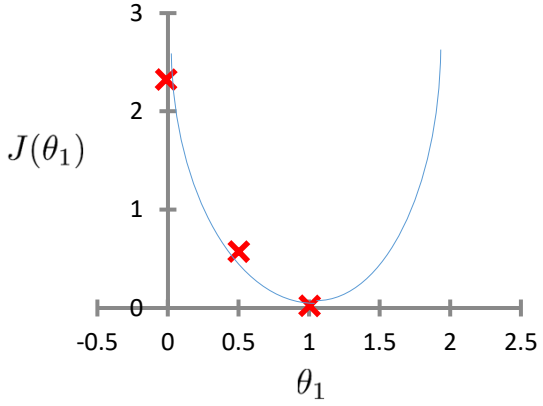
(for fixed θ_1, this is a function of x)



J(0)=2.3

$J(\theta_1)$

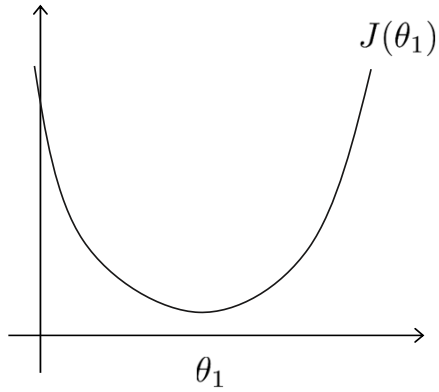
(function of the parameter θ_1)

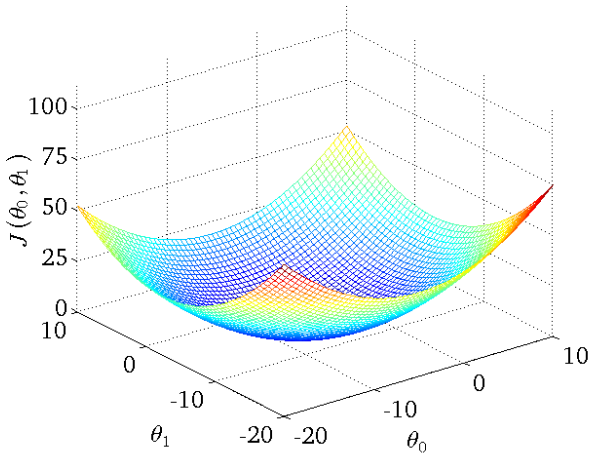


Minimize J(Θ)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)





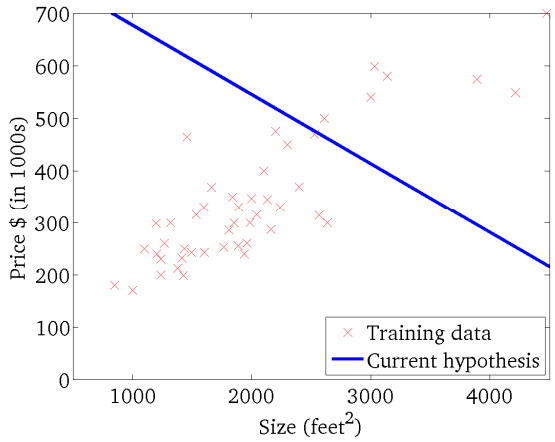
Surface Plot

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

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

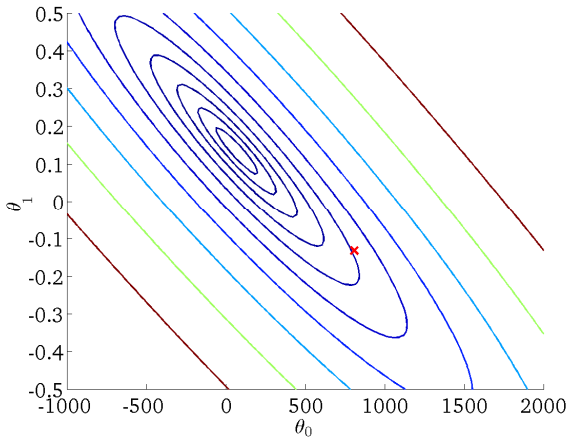
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

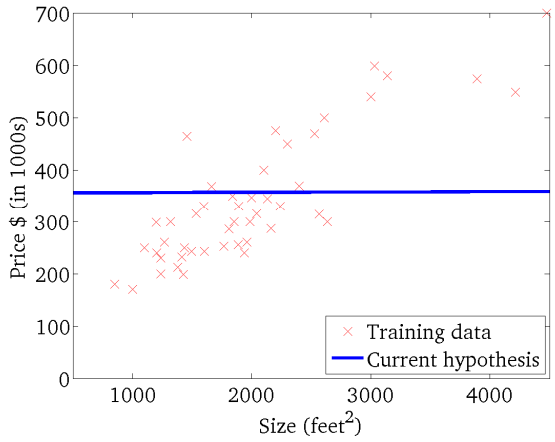
(function of the parameters θ_0, θ_1)



Contour Plot

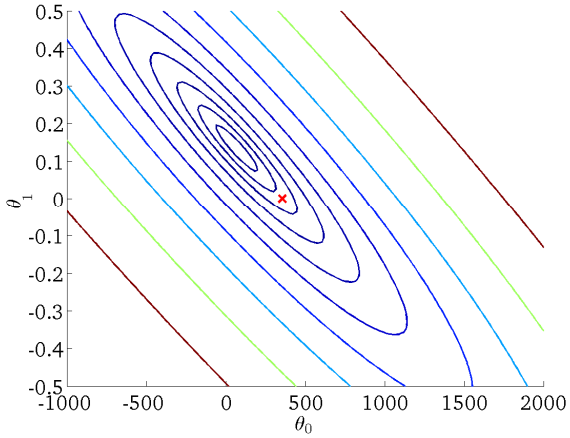
$h_\theta(x)$

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



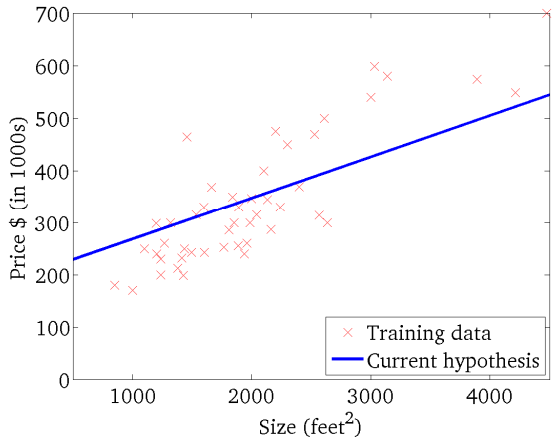
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



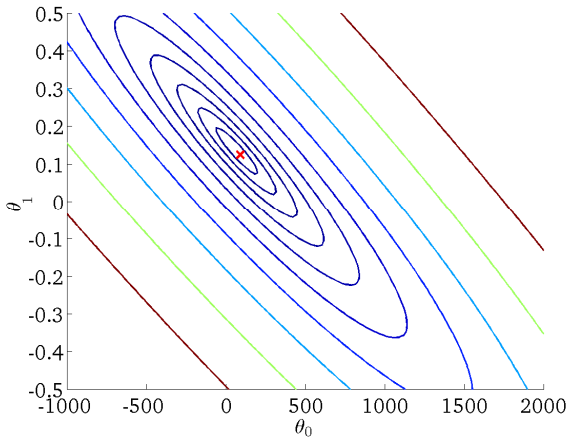
$h_\theta(x)$

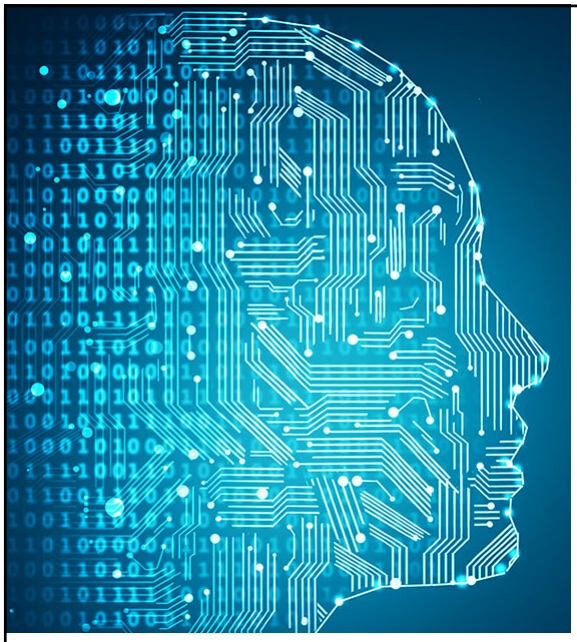
(for fixed θ_0, θ_1 this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)





Linear regression with one variable

Gradient descent

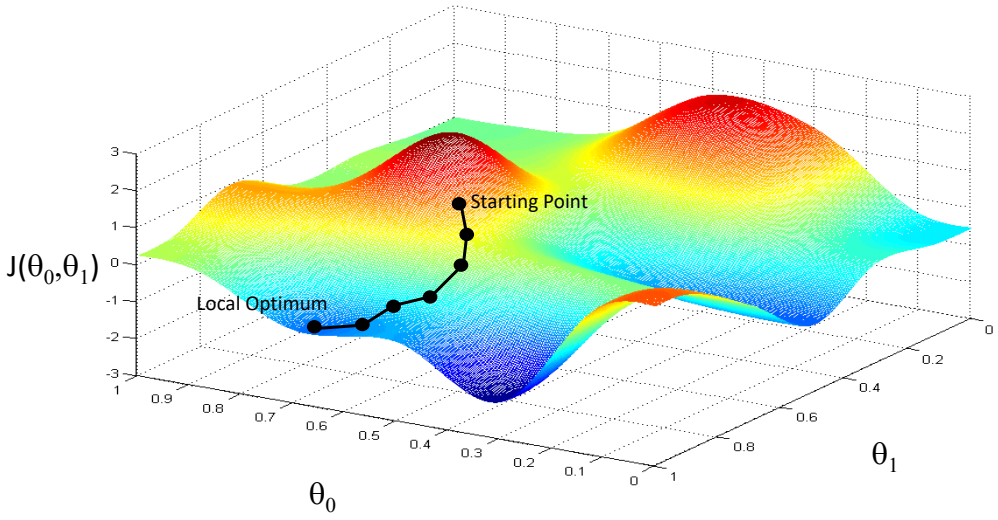
Machine Learning - Part 3
in abstract

Have some function $J(\theta_0, \theta_1)$

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

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum



Gradient descent algorithm

repeat until convergence {

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

}

Derivative

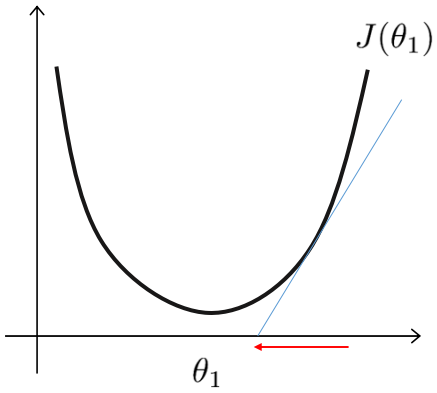
α= Learning Rate

Correct: Simultaneous update

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

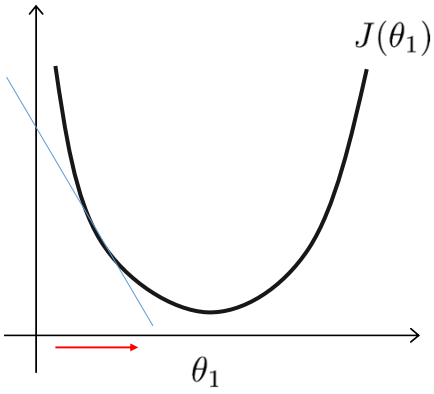
Incorrect:

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



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

>0



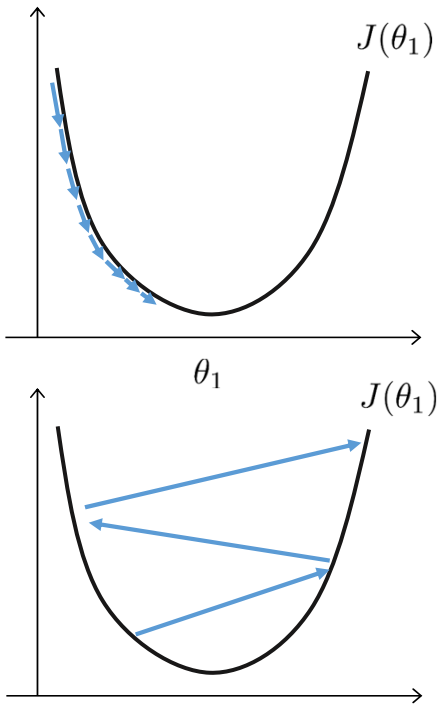
$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

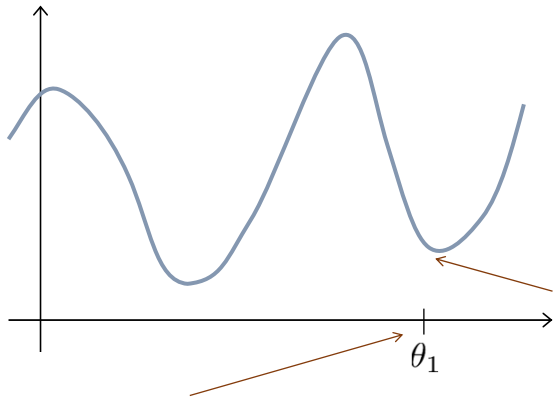
<0

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

If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.





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

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

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

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.

Gradient descent algorithm

repeat until convergence {

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

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

}

Linear Regression Model

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

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

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

$$U^{2\prime} = 2U'U$$

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

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

Gradient descent algorithm

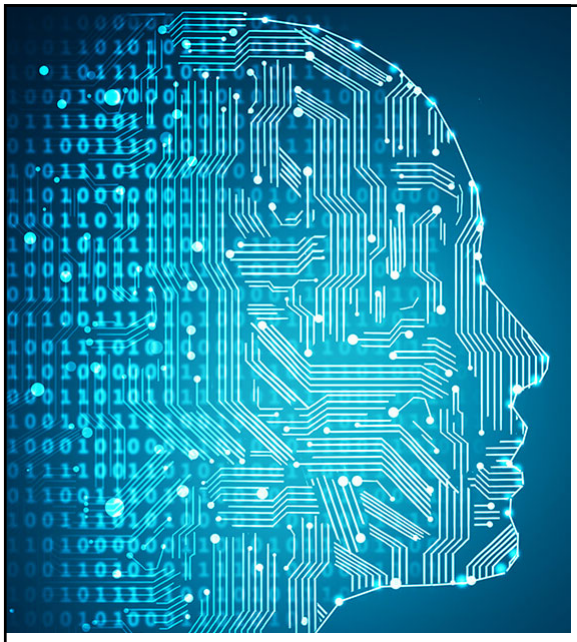
repeat until convergence {

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

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

}

update θ_0 and θ_1 simultaneously



Machine Learning - Part 4

in abstract

Linear Regression with multiple variables

Gradient Descent with Multiple features

Multiple features (variables).

Size (feet²)	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
…	…	…	…	…

Notation:

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.

Hypothesis:

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

Multivariate linear regression:

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

For convenience of notation, define $x_0 = 1$.

Cost function: $J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

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

Gradient Descent

Previously (n=1):

Repeat $\{$

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

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

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

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

$\}$

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

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

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

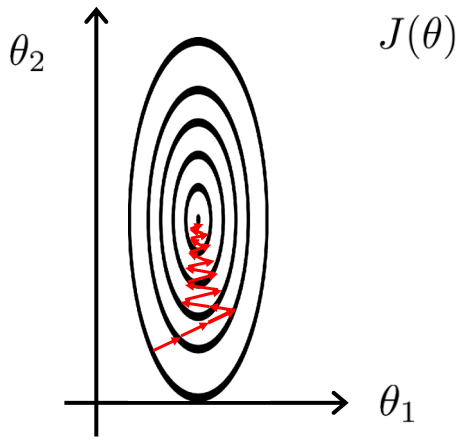
$\ldots$

Feature Scaling

Idea: Make sure features are on a similar scale.

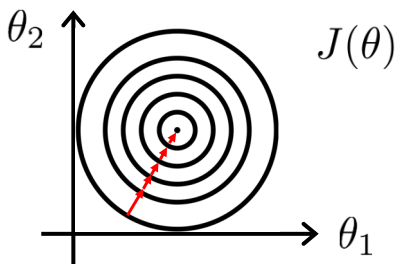
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)



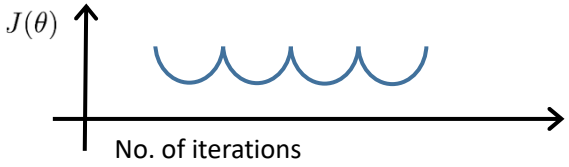
$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

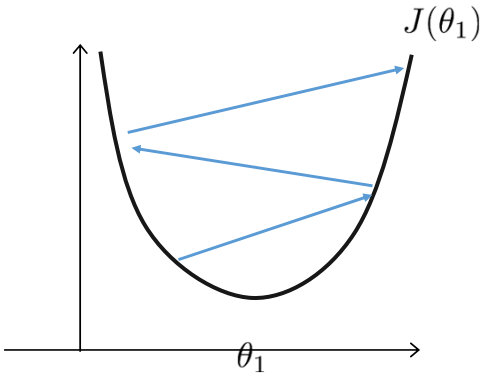
$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Making sure gradient descent is working correctly.
How to choose learning rate α ?

Debug the following Curve.





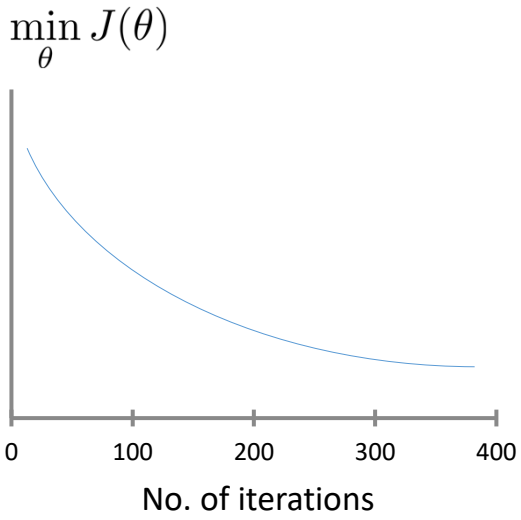
- If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.
- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Gradient descent

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

- "Debugging": How to make sure gradient descent is working correctly.

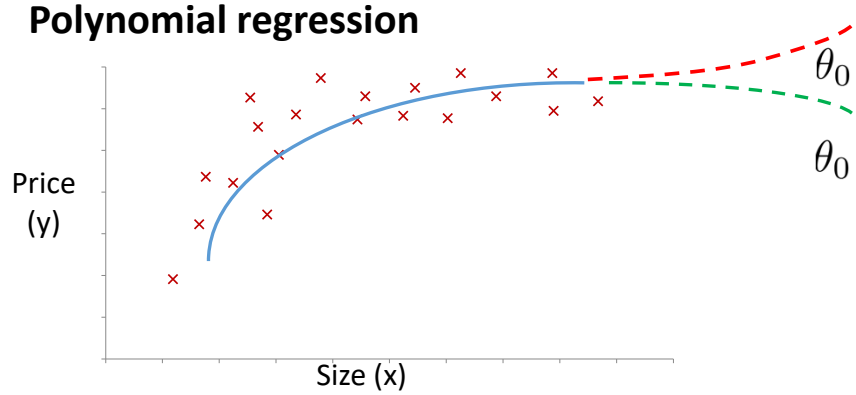
Making sure gradient descent is working correctly.



$J(\theta)$ should decrease after every iteration

Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

Polynomial regression



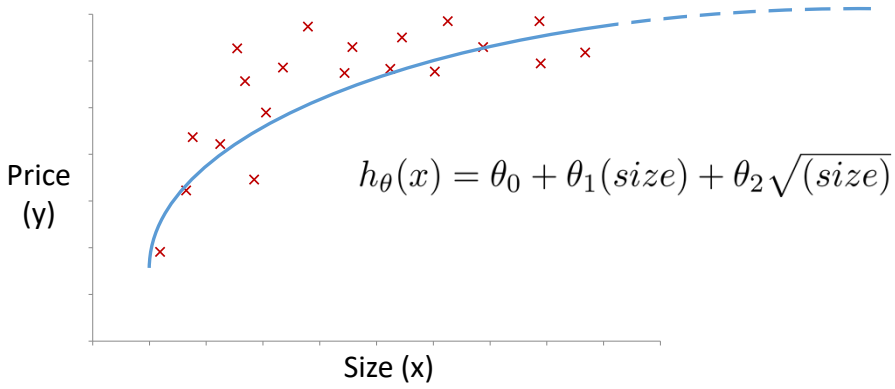
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

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

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

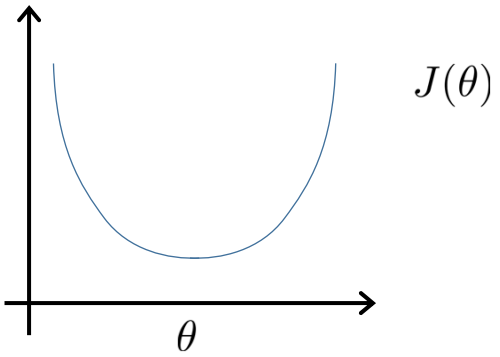
$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$

Choice of features



1. Gradient Descent

vs



2. Normal equation: Method to solve for θ analytically.

$$Y = aX + b$$

$$\hat{a} = \frac{\overline{xy} - \bar{x}.\bar{y}}{\overline{x^2} - \bar{x}^2} \qquad \hat{b} = \bar{y} - \hat{a}\bar{x}$$

Normal equation: Method to solve for θ analytically.

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	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

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

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

m training examples, n features.

Gradient Descent

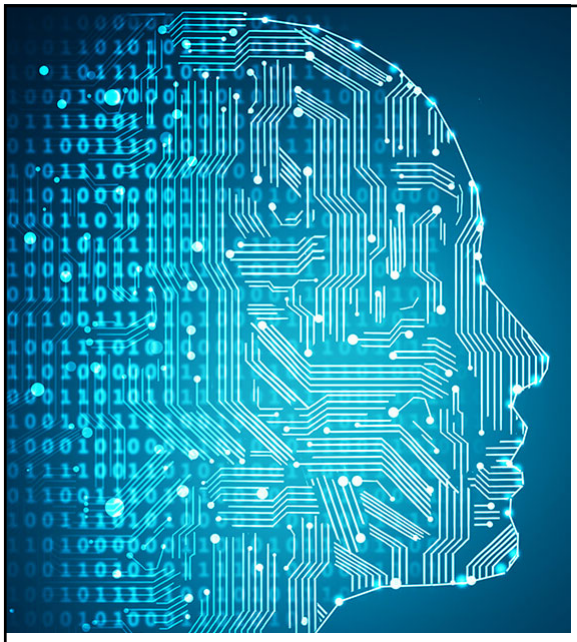
- Need to choose α.
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α.
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large.

$$O(n^3)$$

It is necessary to mention that most of the AI problems contains a huge number of features. As an example, as we see later in the image processing problem each pixel of an image is considered as a feature.



Logistic Regression

Classification

Machine Learning - Part 5
in abstract

Classification

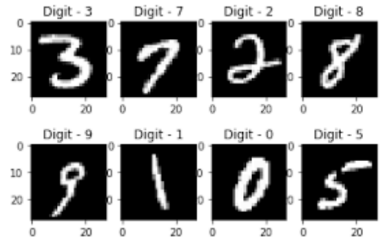
Email: Spam / Not Spam?
Online Transactions: Fraudulent (Yes / No)?
Tumor: Malignant / Benign ?
Image processing: Cat / Dog

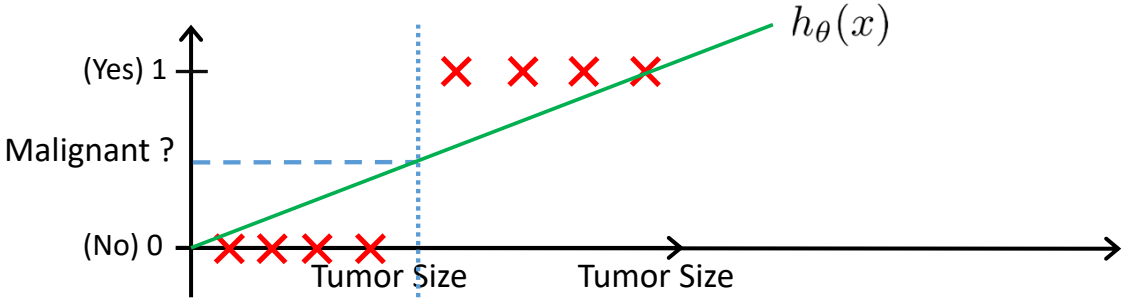
$y \in \{0, 1\}$

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

Handwritten digit recognition

$y \in \{1,\ldots,10\}$

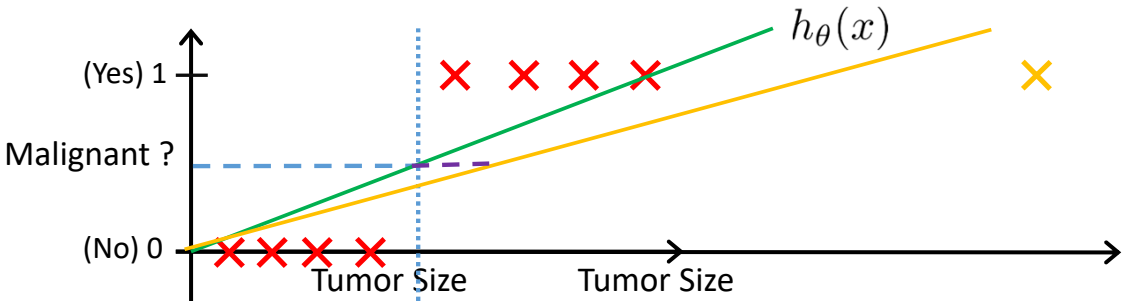




Threshold classifier output $h_\theta(x)$ at 0.5:

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

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



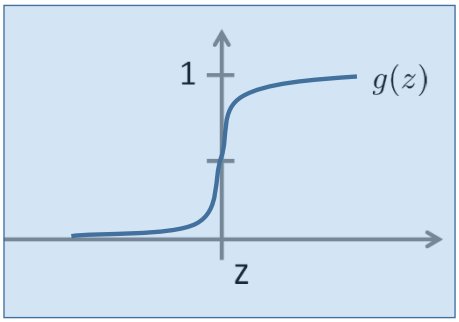
If $h_\theta(x) \geq 0.5$, predict "y = 1"

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

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

Logistic Regression Model

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



$when\ z \geq 0$
$g(z) \geq 0.5$

$when\ z < 0$
$g(z) \leq 0.5$

Want $0 \leq h_\theta(x) \leq 1$

$h_\theta(x) = g(\ \theta^T x\ )$

Interpretation of Hypothesis Output

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

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$

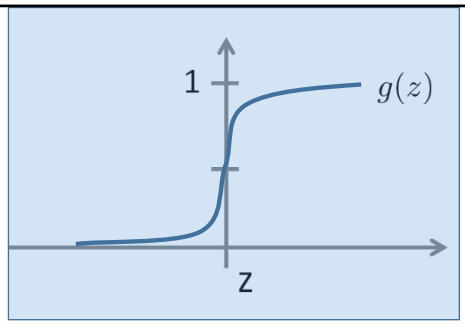
Tell patient that 70% chance of tumor being malignant

"probability that y = 1, given x, parameterized by θ"

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

Logistic regression

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



when $z \geq 0$
$g(z) \geq 0.5$

when $z < 0$
$g(z) \leq 0.5$

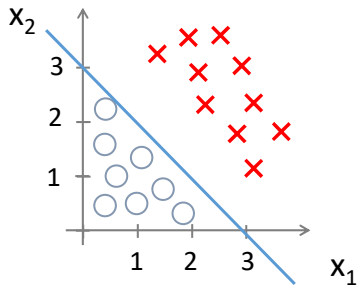
predict "$y = 1$" if $h_\theta(x) \geq 0.5$

or $\theta^T x \geq 0$

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

or $\theta^T x < 0$

Decision Boundary

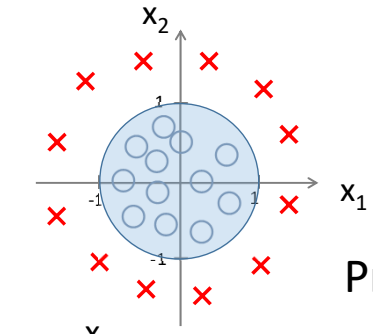


$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Predict "$y = 1$" if $\theta^T x \geq 0$ ⇒ $-3 + x_1 + x_2 \geq 0$

$$x_1 + x_2 \geq 3$$

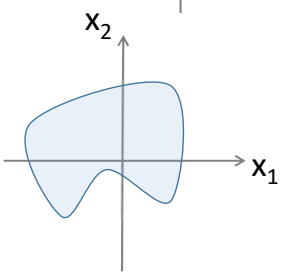
Non-linear decision boundaries



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

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

$$x_1^2 + x_2^2 \geq 1$$



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

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

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

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

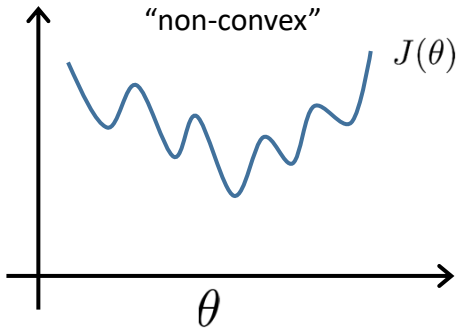
How to choose parameters θ ?

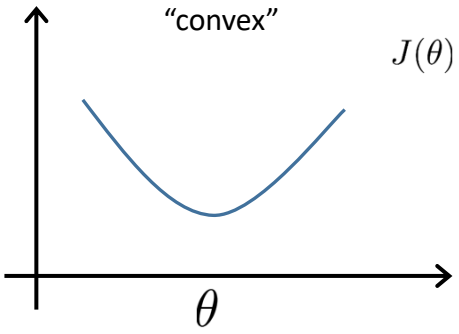
Cost function

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

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

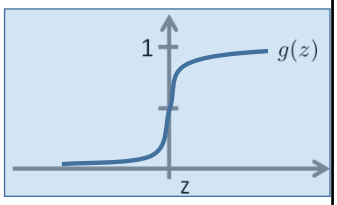
In logistic regression, the cost function is non-convex!



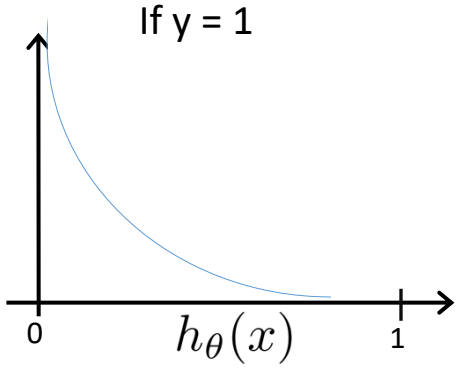


Logistic regression cost function

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



If y = 1



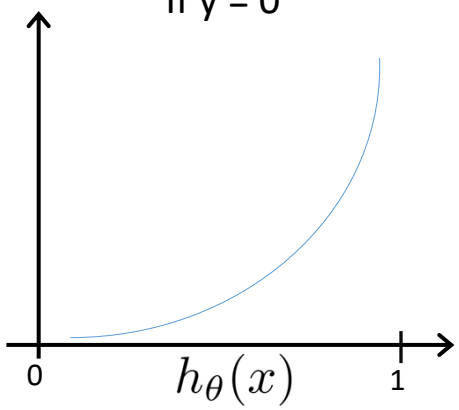
$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Logistic regression cost function

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

If y = 0



Logistic regression cost function

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

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

Note: $y = 0$ or 1 always

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

If y=1: ?

If y=0: ?

Gradient Descent

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

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

Algorithm looks identical to linear regression!

The cost function for logistic regression (binary cross-entropy loss) is:

$$J(\mathbf{w}, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\sigma(z^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma(z^{(i)})) \right]$$

where:

- m is the number of training examples,
- $y^{(i)}$ is the true label for the i-th example (0 or 1),
- $\sigma(z^{(i)})$ is the predicted probability for the i-th example.

Gradient Calculation

To update the weights $\mathbf{w}$ and bias b using gradient descent, we need to compute the gradients of the cost function J with respect to $\mathbf{w}$ and b.

Step 1: Derivative of the Sigmoid Function

The derivative of the sigmoid function $\sigma(z)$ with respect to z is:

$$\frac{d\sigma(z)}{dz} = \sigma(z) \cdot (1 - \sigma(z))$$

Step 2: Derivative of the Cost Function with Respect to z

The cost function J depends on z through $\sigma(z)$. Using the chain rule, the derivative of J with respect to $z^{(i)}$ for a single example is:

$$\frac{\partial J}{\partial z^{(i)}} = \sigma(z^{(i)}) - y^{(i)}$$

This is because:

$$\frac{\partial J}{\partial z^{(i)}} = \frac{\partial J}{\partial \sigma(z^{(i)})} \cdot \frac{\partial \sigma(z^{(i)})}{\partial z^{(i)}}$$

$$\frac{\partial J}{\partial \sigma(z^{(i)})} = -\frac{y^{(i)}}{\sigma(z^{(i)})} + \frac{1 - y^{(i)}}{1 - \sigma(z^{(i)})}$$

$$\frac{\partial \sigma(z^{(i)})}{\partial z^{(i)}} = \sigma(z^{(i)})(1 - \sigma(z^{(i)}))$$

Multiplying these together simplifies to:

$$\frac{\partial J}{\partial z^{(i)}} = \sigma(z^{(i)}) - y^{(i)}$$

Step 3: Gradients with Respect to $\mathbf{w}$ and b

Using the chain rule again, the gradients of J with respect to $\mathbf{w}$ and b are:

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{1}{m} \sum_{i=1}^{m} (\sigma(z^{(i)}) - y^{(i)}) \mathbf{x}^{(i)}$$

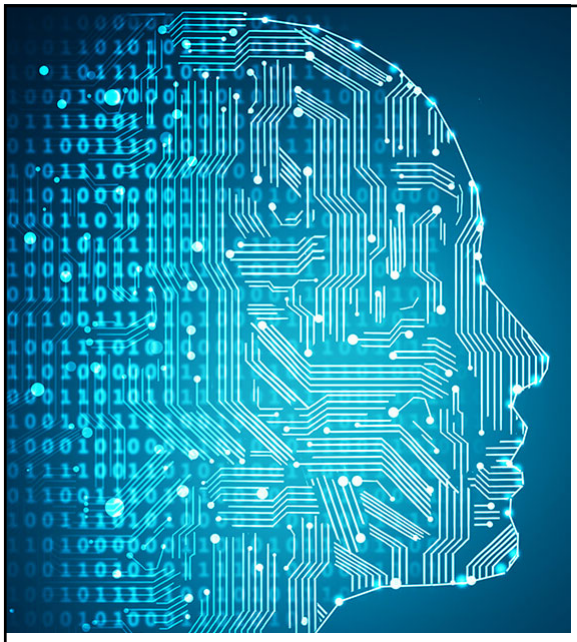
$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (\sigma(z^{(i)}) - y^{(i)})$$

These gradients are used to update $\mathbf{w}$ and b in gradient descent:

$$\mathbf{w} := \mathbf{w} - \alpha \frac{\partial J}{\partial \mathbf{w}}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

where α is the learning rate.



Logistic Regression

Multi-class classification

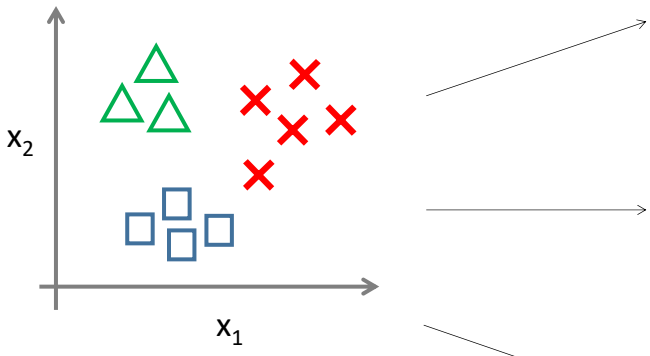
Machine Learning - Part 6
in abstract

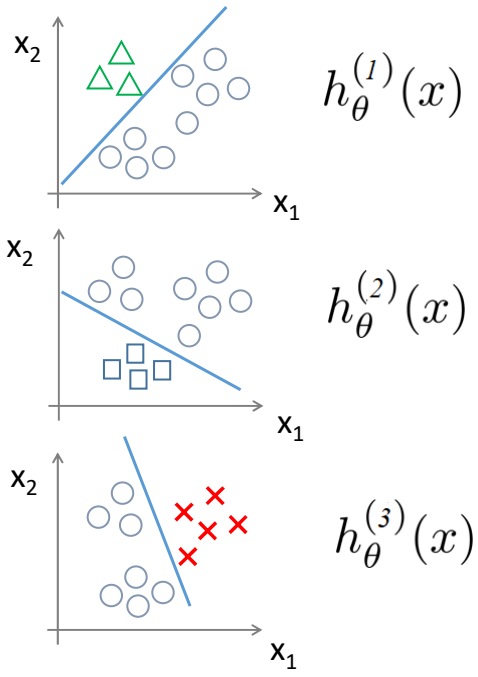
Solving Strategies

Strategy 1: Transformation to binary
It refers to strategies for reducing the problem of multiclass classification to multiple binary classification problems.
Like: One-vs.-rest, One-vs.-one

Strategy 2: Extension from binary
It refers to strategies of extending the existing binary classifiers to solve multi-class classification problems.
Like: neural networks, support vector machines, decision trees, k-nearest neighbors, naive Bayes and extreme learning machines

One-vs-all (one-vs-rest):





Class 1: △

Class 2: □

Class 3: ✕

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

One-vs-all

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes $\max_i h_\theta^{(i)}(x)$

One-vs-one

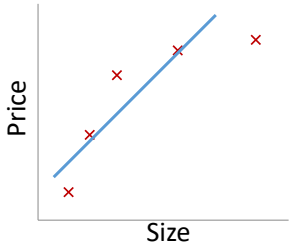
In this method we train K (K − 1) / 2 binary classifiers for a K-way multiclass problem.

فرمول جایگشت

n آیتم داریم و می‌خواهیم تعداد روش‌هایی که می‌توان k آیتم را از میان آن‌ها با ترتیب معین انتخاب کرد، به دست آوریم:

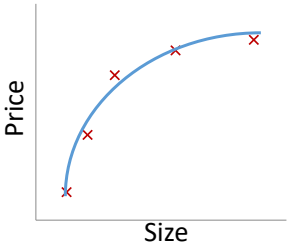
$$P(n, k) = \frac{n!}{(n-k)!}$$

Overfitting vs Under fitting problem



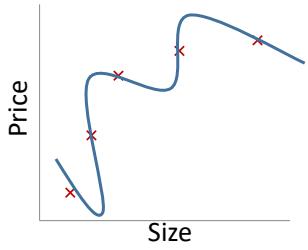
$\theta_0 + \theta_1 x$

Under fit / High Bias



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

Just Right

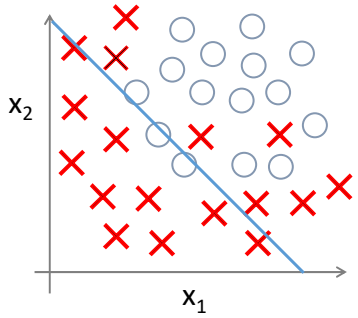


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Over fit / High Variance

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

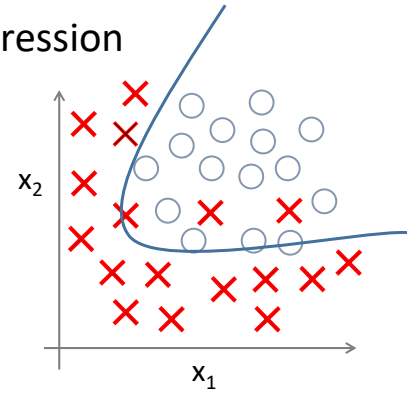
Example: Logistic regression



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

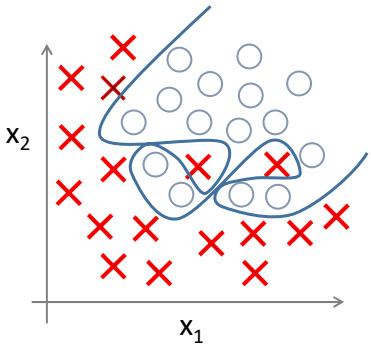
(g = sigmoid function)

Under fit / High Bias



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

Just Right



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

Over fit / High Variance

Addressing overfitting:

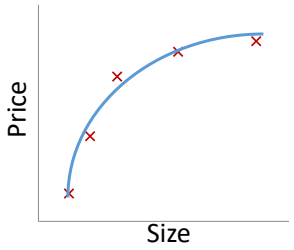
Options:

1. Reduce number of features.
   - Manually select which features to keep.
   - Model selection algorithm (like resampling method).

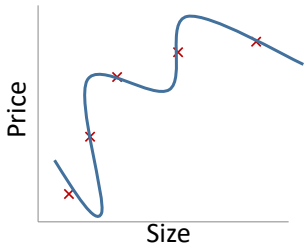
   i.e. by **splitting the data into multiple subsets** and **iteratively training and validating the model** on different combinations of these subsets.
2. Regularization.
   - Keep all the features, but reduce magnitude/values of parameters θ_j.

   Works well when we have a lot of features, each of which contributes a bit to predicting y.
3. Increasing the number of examples.

Intuition of Regularization



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



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

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 10000 \times \theta_3^2 + 10000 \times \theta_4^2$$

Regularization.

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

- "Simpler" hypothesis
- Less prone to overfitting

Housing:

- Features: $x_1, x_2, \ldots, x_{100}$
- Parameters: $\theta_0, \theta_1, \theta_2, \ldots, \theta_{100}$

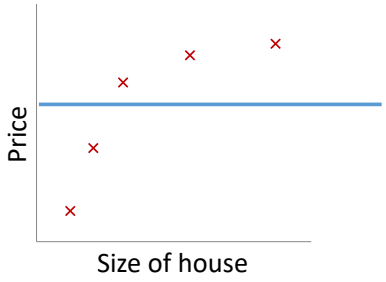
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

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

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 for too large for our problem, say $\lambda = 10^{10}$)?



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

Regularized Gradient descent

Repeat $\{$

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

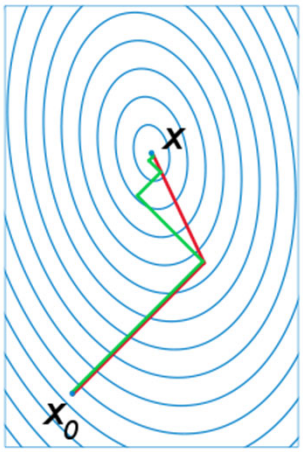
$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} - \frac{\lambda}{m} \theta_j \right]$$

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

$\}$

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

Gradient descent vs Conjugate gradient



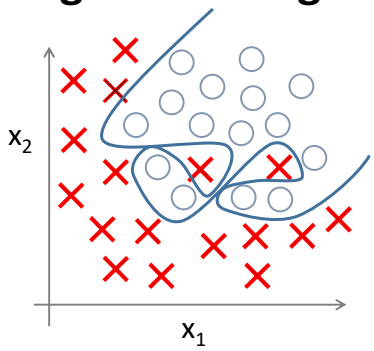
First-order vs second-order derivatives

Gradient descent with optimal step size (in green) and conjugate vector (in red) for minimizing a quadratic function

Other well-known optimization algorithms:

- BFGS
- L-BFGS

Regularized logistic regression.



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

Cost function:

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

Gradient descent

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

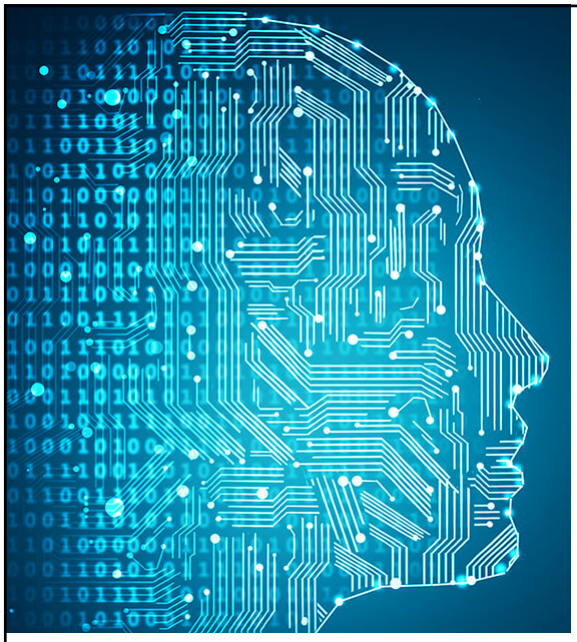
Repeat $\{$

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

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} - \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \dots, n)$$

$\}$

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



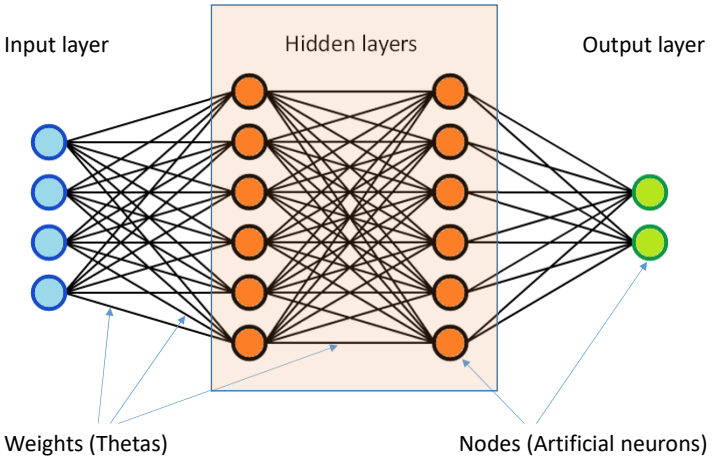
Neural Networks: Representation

Details of Backpropagation Algorithm

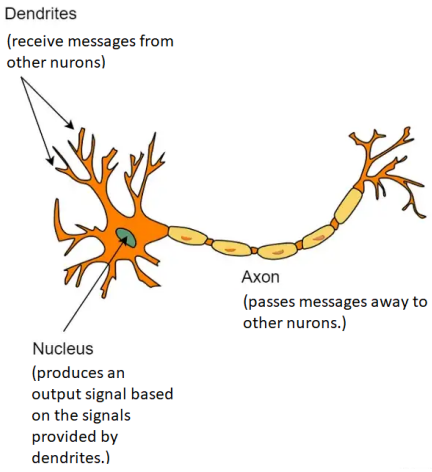
Machine Learning - Part 7

in abstract

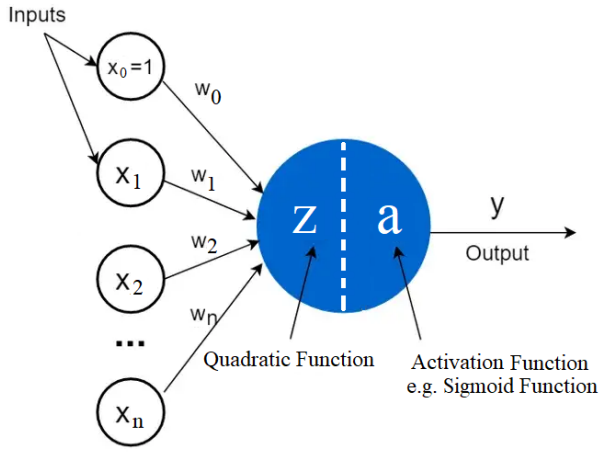
The Architecture of a Neural Network



Biological neuron vs Artificial neuron



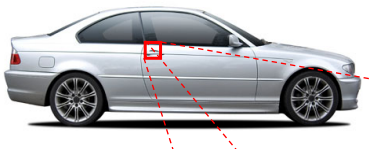
Biological neuron.



Artificial neuron

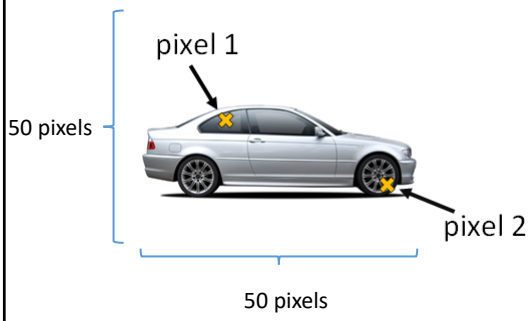
What is this?

We see this:



But the camera sees this:

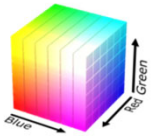
194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50



50 x 50 pixel images = 2500 pixels

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

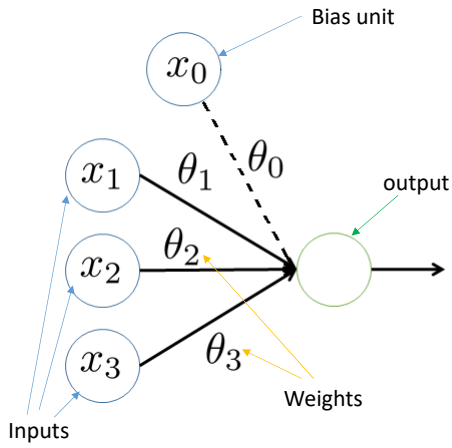
{0, 1, 2, …, 255}



7500 pixels if RGB

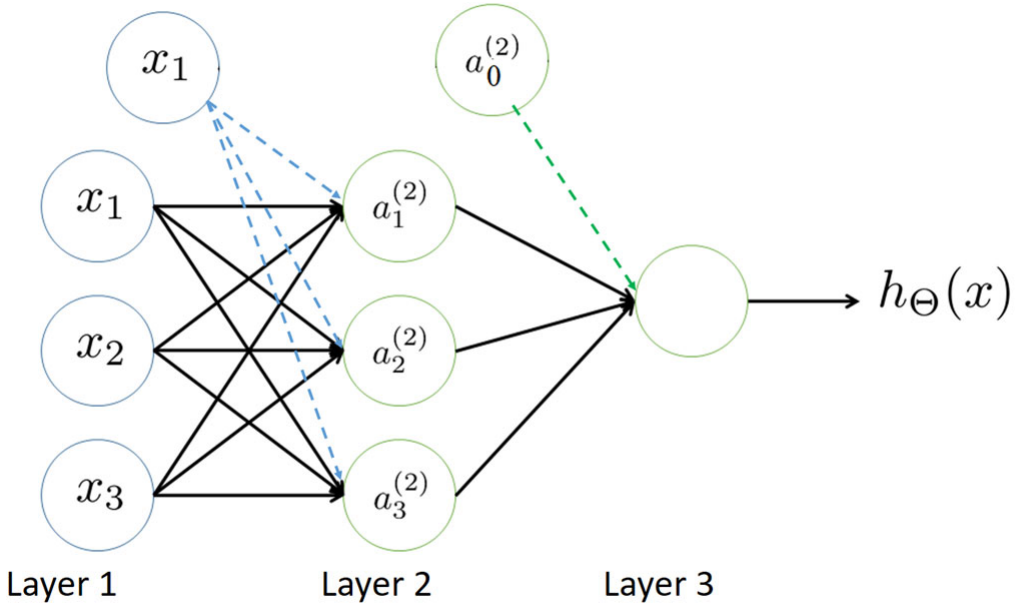
Quadratic features ($x_i \times x_j$): ≈3 million features

Neuron model: Logistic unit

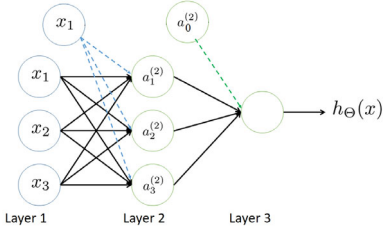


$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Sigmoid (logistic) activation function.



Neural Network



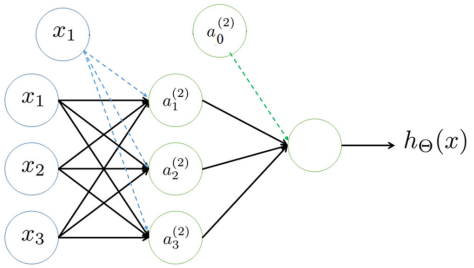
$a_i^{(j)}$ = "activation" of unit i in layer j

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

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

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

$$z^{(2)} = \Theta^{(1)}x$$
$$a^{(2)} = g(z^{(2)})$$

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

$$z^{(3)} = \Theta^{(2)}a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

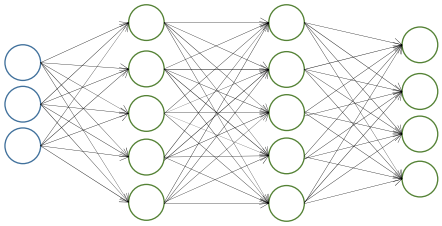
Multiple output units: One-vs-all.


Pedestrian


Car


Motorcycle


Truck



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

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

when pedestrian when car when motorcycle

Neural Network (Classification)

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

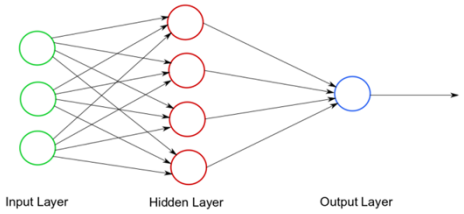
L = total no. of layers in network

s_l = no. of units (not counting bias unit) in layer l

Binary classification

$y = 0$ or 1

1 output unit

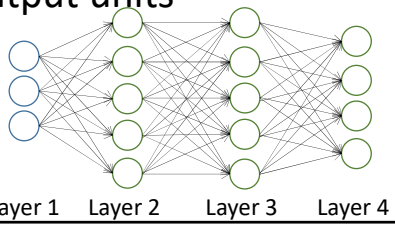


Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units



Cost function

Logistic regression:

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

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log((h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

k is the index of class.

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

l is the index of layers.

Gradient computation

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

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

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

Backpropagation algorithm

To compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$, one can

(1) Use chain rule separately for **each weight** Or

(2) Use Backpropagation algorithm to compute the gradient of **each layer** denoted by δ^l

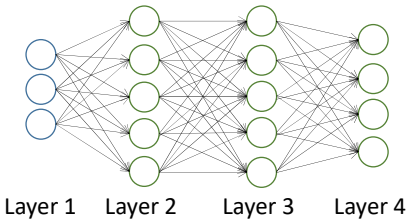
The first method is inefficient because of enormous number of duplicate calculations, however Back propagation method benefits by not computing unnecessary intermediate values

Gradient computation

Given one training example (x, y):

Forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



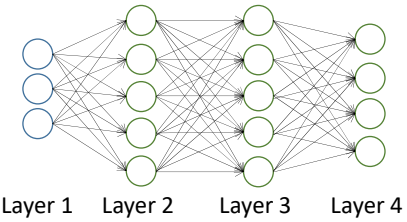
Gradient computation: Backpropagation algorithm

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

For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Actual value of output



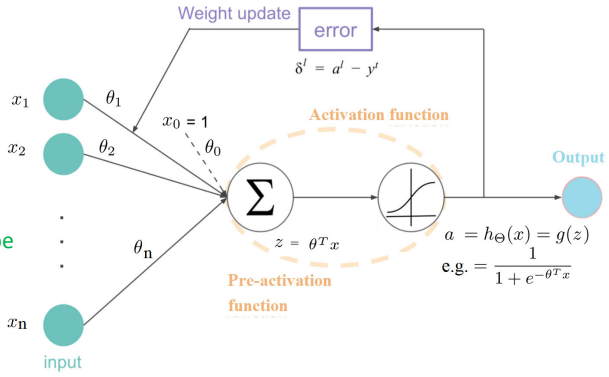
$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$
$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

For Math computation details go to: https://www.jeremyjordan.me/neural-networks-training/

Gradient computation: Backpropagation algorithm

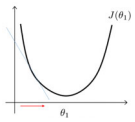
$$\delta_j^{(4)} = a_j^{(4)} - y_j$$



II

The activation function may not be differentiable everywhere.

$$\delta^{(4)} = \frac{\partial J}{\partial a^{(4)}} \sim \frac{\partial J}{\partial z^{(4)}}$$

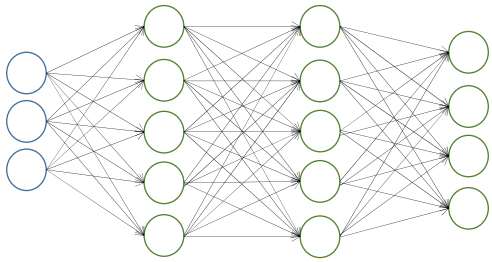


The derivative of the cost function which evaluate the accuracy of the network's predictions, with respect to the activations at the output layer **represents how sensitive the cost function is to changes in the activations**. At the **minimum cost**, the deviation is near to zero.

$$\delta^{(3)} = \frac{\partial J}{\partial z^{(3)}} = \frac{\partial J}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial z^{(3)}}$$

II

$$\delta^{(4)}$$



We know

$$z^{(4)} = \Theta^{(3)} a^{(3)} = \Theta^{(3)} g\left(z^{(3)}\right)$$

$$\frac{\partial z^{(4)}}{\partial z^{(3)}} = \Theta^{(3)}\, g'\left(z^{(3)}\right)$$

So we proved that: $\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* g'\left(z^{(3)}\right)$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

The goal is to compute the gradient of the loss function $J(\Theta)$ with respect to the weight $\Theta_{ij}^{(l)}$. Using the chain rule, this gradient can be expressed as:

$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = \frac{\partial J(\Theta)}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{ij}^{(l)}}$$

- $\frac{\partial J(\Theta)}{\partial z_i^{(l+1)}}$: This is the error term $\delta_i^{(l+1)}$, which measures how much the weighted input $z_i^{(l+1)}$ to node i in layer $l+1$ affects the loss $J(\Theta)$.
- $\frac{\partial z_i^{(l+1)}}{\partial \Theta_{ij}^{(l)}}$: This is the activation $a_j^{(l)}$, because $z_i^{(l+1)}$ is computed as:

  $$z_i^{(l+1)} = \Theta_{i0}^{(l)} a_0^{(l)} + \Theta_{i1}^{(l)} a_1^{(l)} + \cdots + \Theta_{ij}^{(l)} a_j^{(l)} + \dots$$

  Taking the derivative of $z_i^{(l+1)}$ with respect to $\Theta_{ij}^{(l)}$ gives:

  $$\frac{\partial z_i^{(l+1)}}{\partial \Theta_{ij}^{(l)}} = a_j^{(l)}$$

Backpropagation algorithm (to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

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

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

For $i = 1$ to m

For each training example, accumulate the gradient of the loss function with respect to each weight $\Theta_{ij}^{(l)}$

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

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

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

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

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

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$

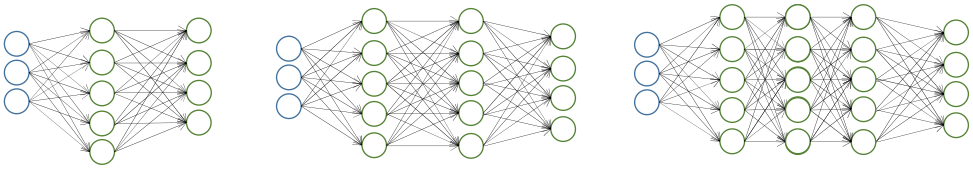
$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

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

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

Training a neural network

1. Pick a network architecture (connectivity pattern between neurons)



No. of input units: Dimension of features $x^{(i)}$
No. output units: Number of classes
Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

2. Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$ (i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

Training a neural network

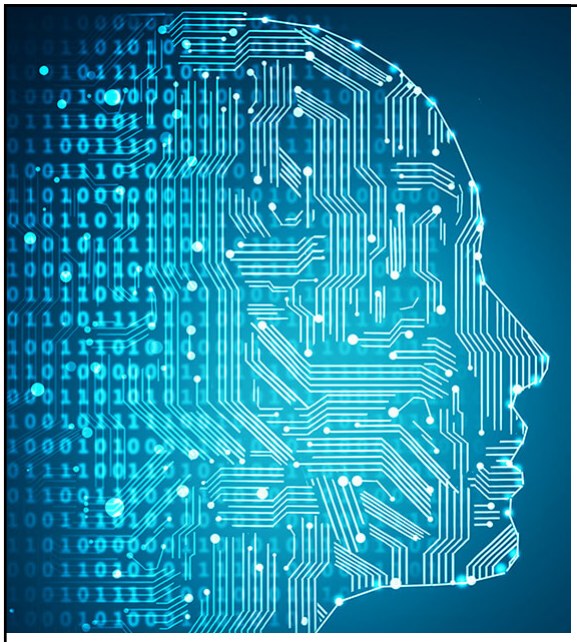
3. Implement forward propagation using example $(x^{(i)}, y^{(i)})$
4. Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$
5. Compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

Training a neural network

6. Check the gradients $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed by backpropagation vs. using numerical estimate of gradient of $J(\Theta)$, computed by the following equation:

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

7. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ and update them.



Machine Learning - Part 8

in abstract

Advice for applying machine learning

Debugging and error analyzing

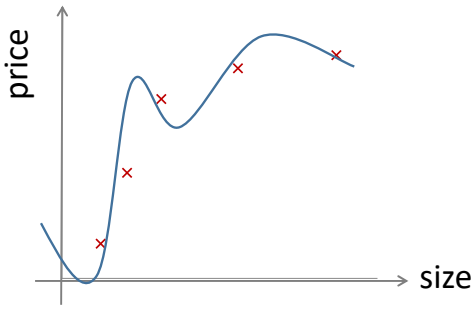
Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices.

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

However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc.})$
- Try decreasing λ
- Try increasing λ

Overfitting example



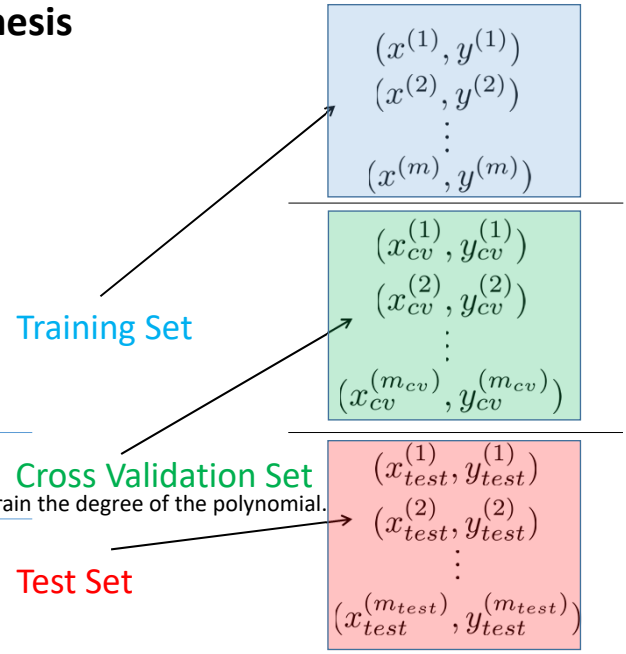
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

Evaluating your hypothesis

Dataset:

	Size	Price
60%	2104	400
	1600	330
	2400	369
	1416	232
	3000	540
	1985	300
20%	1534	315
	1427	199
20%	1380	212
	1494	243



Train/validation/test error

Training error:

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

Cross Validation error:

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

Test error:

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

Model selection

1. $h_\theta(x) = \theta_0 + \theta_1 x$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$

   ⋮
10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$

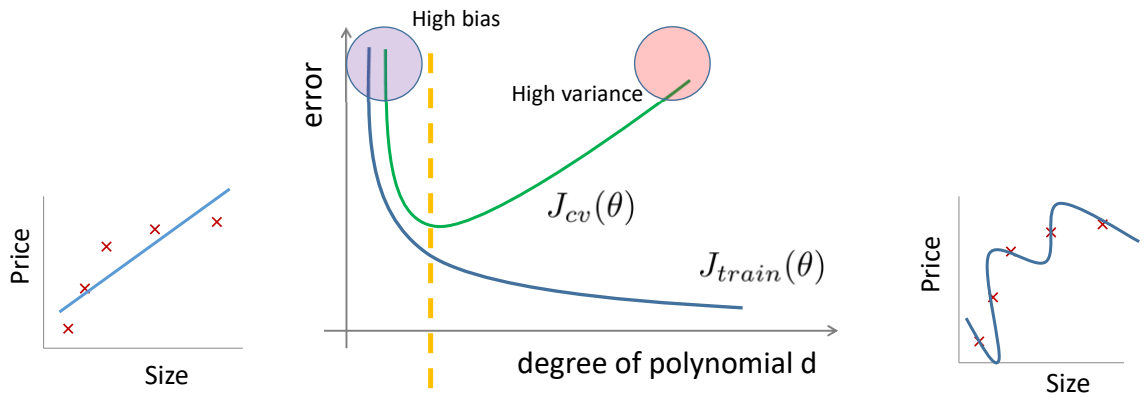
Pick the model with minimum $J_{cv}(\theta)$ e.g. polynomial with grade 4

and then Estimate generalization error for test set $J_{test}(\theta^{(4)})$

Bias/variance

Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

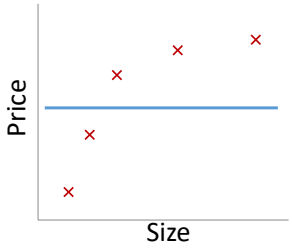
Cross validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$



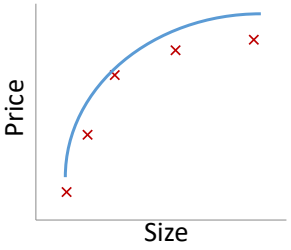
Regularization term

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

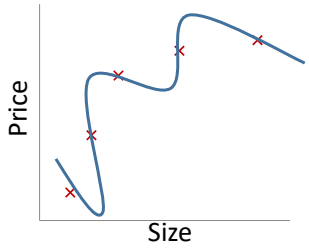
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$



Large λ
High bias (underfit)
$\lambda = 10000.\ \theta_1 \approx 0, \theta_2 \approx 0, \ldots$
$h_\theta(x) \approx \theta_0$



Intermediate λ
"Just right"



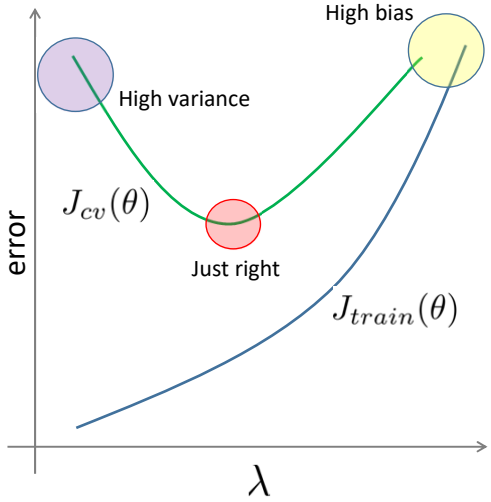
Small λ
High variance (overfit)

Bias/variance as a function of the regularization parameter λ

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

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

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



Choosing the regularization parameter λ

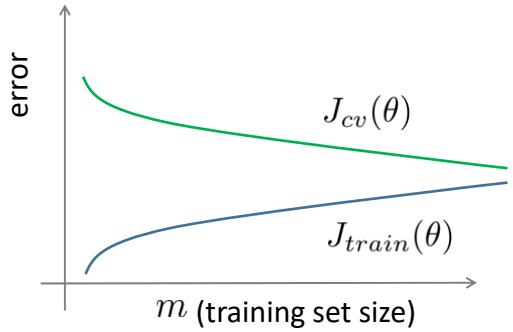
1. Try $\lambda = 0$ $\quad \min_\theta J(\theta) \Rightarrow \theta^{(1)} \Rightarrow J_{cv}(\theta^{(1)})$
2. Try $\lambda = 0.01$ $\quad \min_\theta J(\theta) \Rightarrow \theta^{(2)} \Rightarrow J_{cv}(\theta^{(2)})$
3. Try $\lambda = 0.02$ $\quad \min_\theta J(\theta) \Rightarrow \theta^{(3)} \Rightarrow J_{cv}(\theta^{(3)})$
4. Try $\lambda = 0.04$ $\quad \min_\theta J(\theta) \Rightarrow \theta^{(4)} \Rightarrow J_{cv}(\theta^{(4)})$
5. Try $\lambda = 0.08$ $\quad \min_\theta J(\theta) \Rightarrow \theta^{(5)} \Rightarrow J_{cv}(\theta^{(5)})$ Pick (say) $\theta^{(5)}$

⋮

12. Try $\lambda = 10$ $\quad \min_\theta J(\theta) \Rightarrow \theta^{(10)} \Rightarrow J_{cv}(\theta^{(10)})$

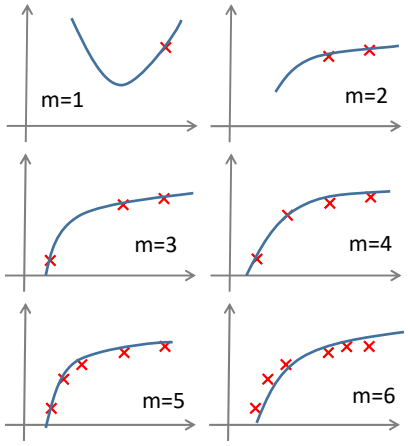
Learning curves
Impact of data set size

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

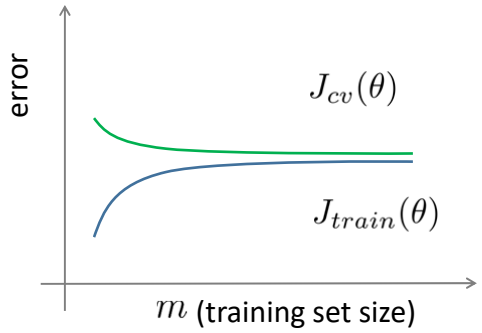


$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

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

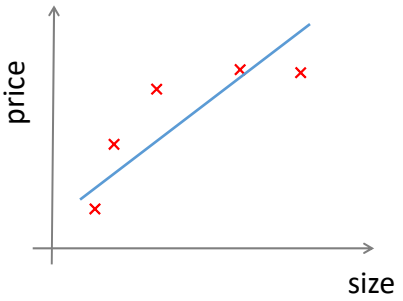


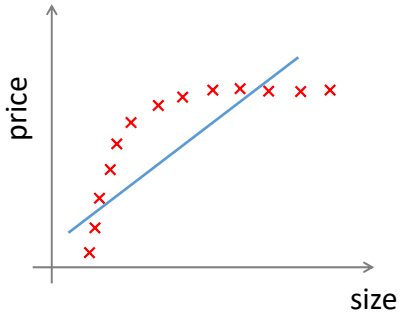
High bias



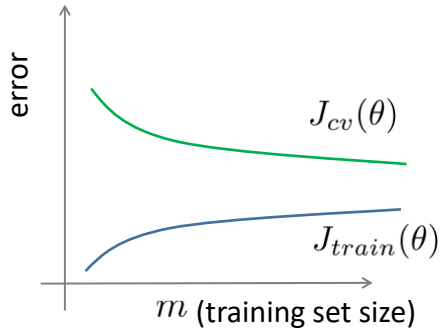
If a learning algorithm is suffering from high bias, getting more training data **will not (by itself) help** much.

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



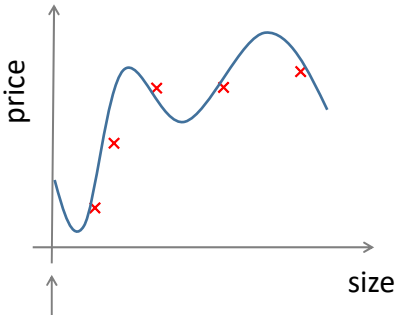


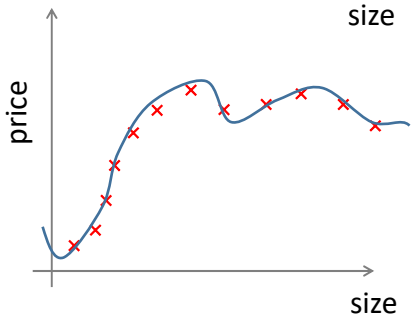
High variance



If a learning algorithm is suffering from high variance, getting more training data is likely to help.

$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{100} x^{100}$
(and small λ)





Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → **Fixes high variance**
- Try smaller sets of features → **Fixes high variance**
- Try getting additional features → **Fixes high bias**
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc})$ → **Fixes high bias**

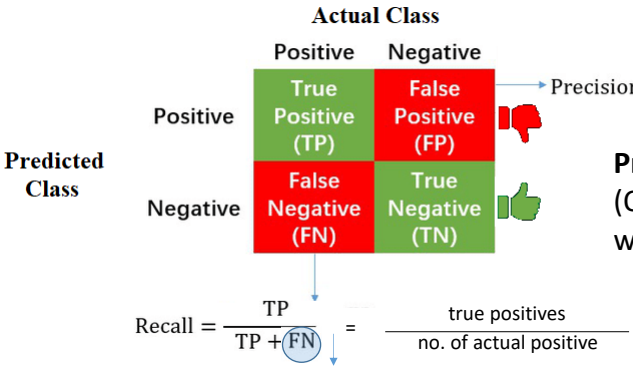
- Try decreasing λ → **Fixes high bias**
- Try increasing λ → **Fixes high variance**

Recommended approach

- Consider necessary features to make a good prediction, specifically, check if a human expert can confidently predict by the data.
- Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross-validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Error analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

Precision/Recall

$y = 1$ in presence of rare class that we want to detect



Precision
(Of all patients where we predicted $y = 1$, what fraction actually has cancer?)

Recall
(Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

Trading off precision and recall

Logistic regression: $0 \leq h_\theta(x) \leq 1$

Predict 1 if $h_\theta(x) \geq 0.5$

Predict 0 if $h_\theta(x) < 0.5$

Suppose we want to predict $y = 1$ (cancer) only if very confident.

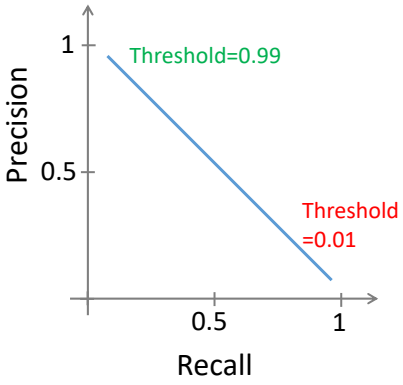
Predict 1 if $h_\theta(x) \geq 0.75$
Predict 0 if $h_\theta(x) < 0.75$

Higher precision

Suppose we want to avoid missing too many cases of cancer (avoid false negatives).

Predict 1 if $h_\theta(x) \geq 0.25$
Predict 0 if $h_\theta(x) < 0.25$

Higher recall



More generally: Predict 1 if $h_\theta(x) \geq$ threshold.

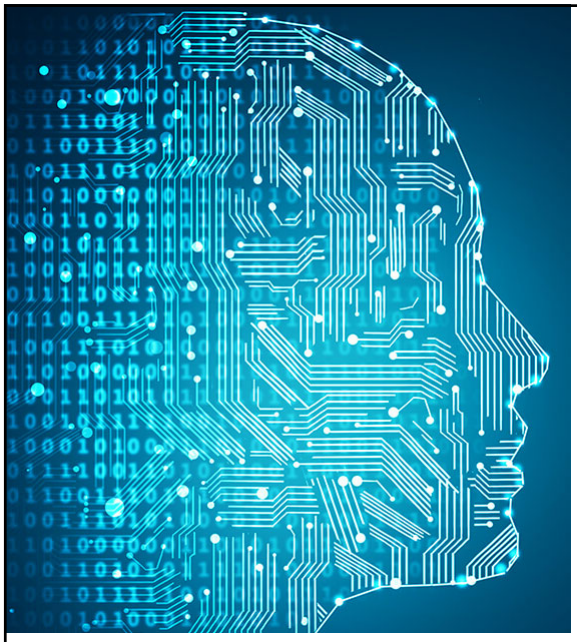
F_1 Score (F score)

How to compare precision/recall numbers?

	Precision(P)	Recall (R)
Algorithm 1	0.5	0.4
Algorithm 2	0.7	0.1
Algorithm 3	0.02	1.0

Average: $\frac{P+R}{2}$

F_1 Score: $2\frac{PR}{P+R}$



Support Vector Machines

Linear and Nonlinear Model

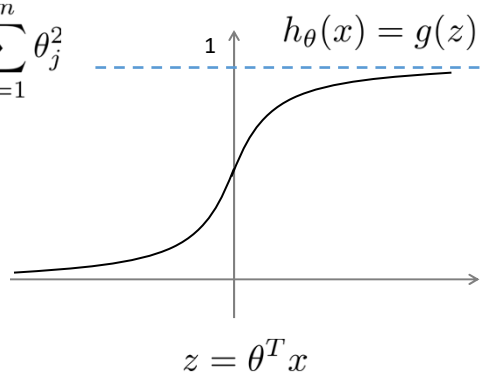
Machine Learning - Part 9

in abstract

Alternative view of logistic regression

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

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



If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

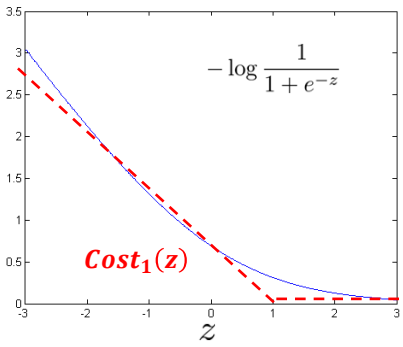
If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

Alternative view of logistic regression

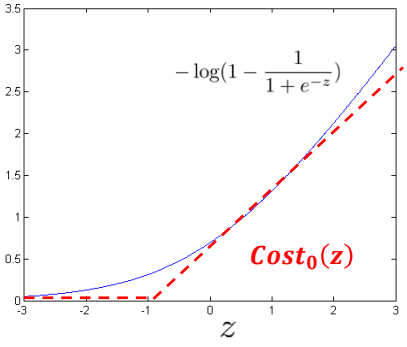
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1 - h_\theta(x)))$

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

If $y = 1$ (want $\theta^T x \gg 0$):



If $y = 0$ (want $\theta^T x \ll 0$):



Support vector machine

Logistic regression:

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

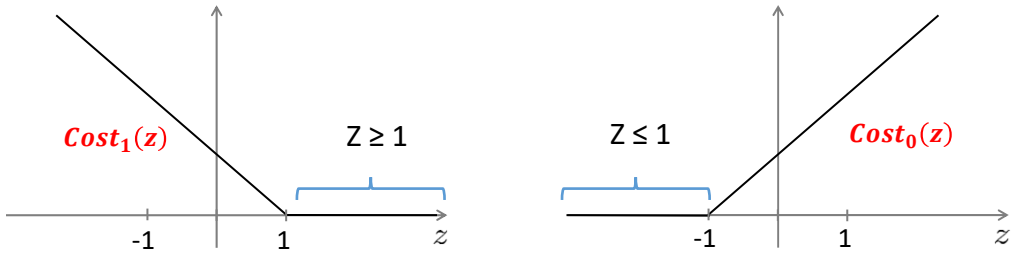
Support vector machine:

$$C = \frac{1}{\lambda}$$

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Support Vector Machine

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$



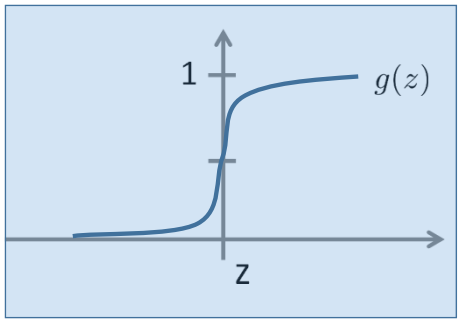
If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

Reminder:
Logistic regression

$$h_\theta(x) = g(\theta^T x)$$

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



$when\ z \geq 0$
$g(z) \geq 0.5$

$when\ z < 0$
$g(z) \leq 0.5$

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

or $\theta^T x \geq 0$

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

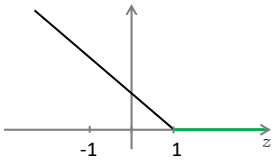
or $\theta^T x < 0$

SVM Decision Boundary

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

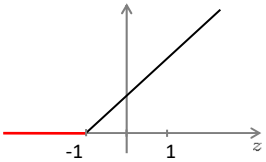
Whenever $y^{(i)} = 1$:

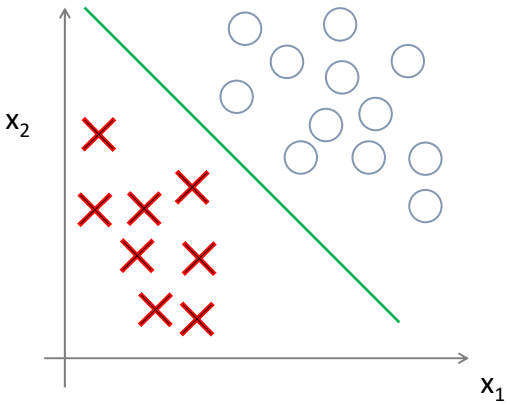
$\theta^T x \geq 1$



Whenever $y^{(i)} = 0$:

$\theta^T x \leq -1$



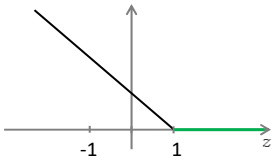


SVM Decision Boundary

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

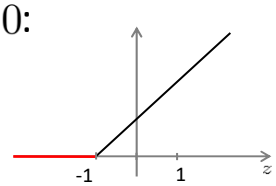
Whenever $y^{(i)} = 1$:

$\theta^T x \geq 1$



Whenever $y^{(i)} = 0$:

$\theta^T x \leq -1$



$$\min_\theta C\,\mathbf{A} + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

S.t.

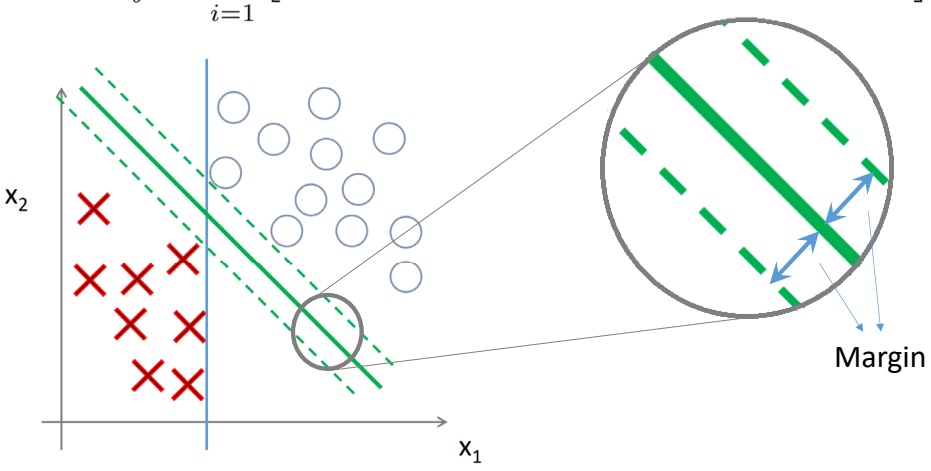
$\theta^T x \geq 1$ **if** $y^{(i)} = 1$

$\theta^T x \leq -1$ **if** $y^{(i)} = 0$

when the regularization parameter C in SVM is small, the algorithm prioritizes maximizing the margin. This is because a small value of C allows for more misclassifications, which means that the model is less likely to overfit the data.

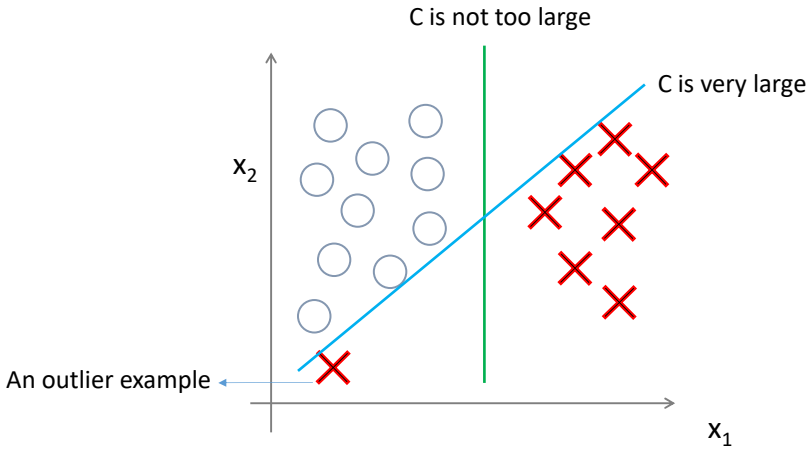
SVM Decision Boundary: Linearly separable case

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

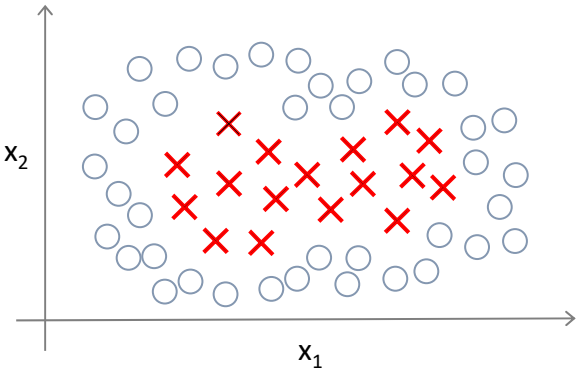


Large margin classifier

Large margin classifier in presence of outliers



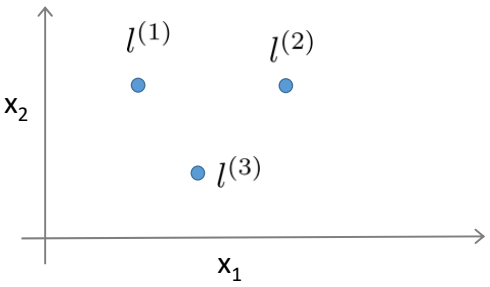
Non-linear Decision Boundary



$$\text{Predict } y = 1 \text{ if}$$
$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

Kernel



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

Given x: $f_i = \text{similarity}(x, l^{(i)}) = \underbrace{\exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)}_{\text{Gaussian kernel}}$

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x = l^{(1)}$:

$$f_1 = \exp\left(-\frac{0^2}{2\sigma^2}\right) = 1$$

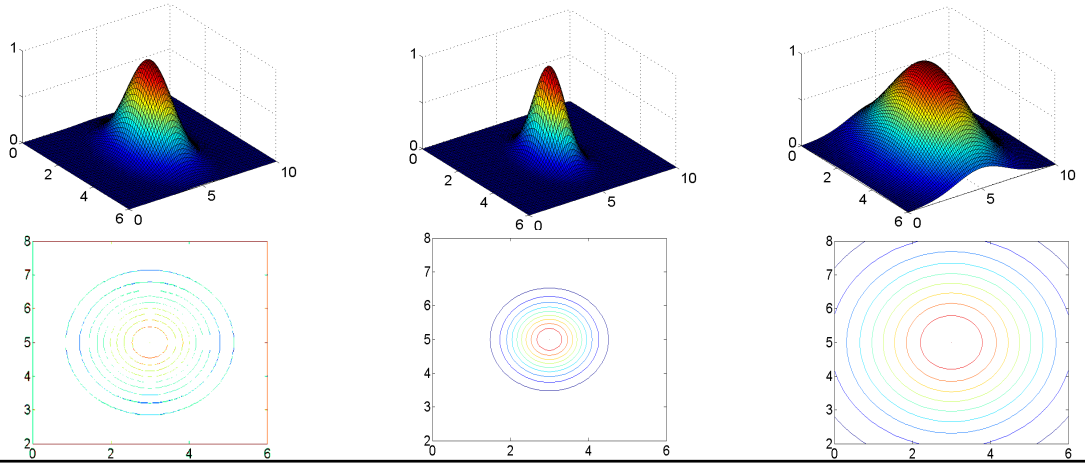
If x is far from $l^{(1)}$:

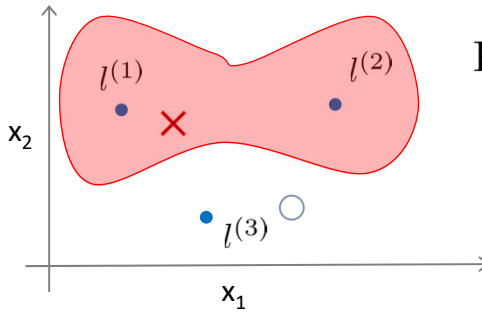
$$f_1 = \exp\left(-\frac{\text{large number}^2}{2\sigma^2}\right) \approx 0$$

Example:

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$\sigma^2 = 1$ $\sigma^2 = 0.5$ $\sigma^2 = 3$





Predict "1" when ×

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

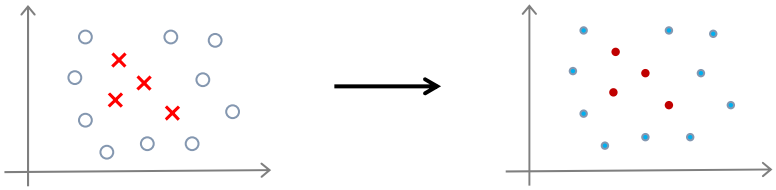
$\theta_0 = -0.5 \quad \theta_1 = 1 \quad \theta_2 = 1 \quad \theta_3 = 0$

× $f_1 \approx 1 \quad f_2 \approx 0 \quad f_3 \approx 0 \quad \theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0 = 0.5$

○ $f_1 \approx 0 \quad f_2 \approx 0 \quad f_3 \approx 1 \quad \theta_0 + \theta_1 \times 0 + \theta_2 \times 0 + \theta_3 \times 1 = -0.5$

Choosing the landmarks

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,
choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

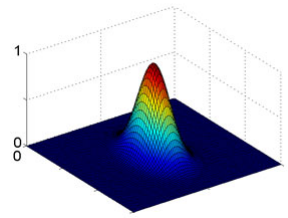
Given example x : $f_1 = \text{similarity}(x, l^{(1)})$
$f_2 = \text{similarity}(x, l^{(2)})$
$\ldots$

SVM parameters:

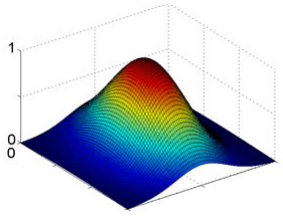
C ($= \frac{1}{\lambda}$). Large C: Lower bias, high variance.
Small C: Higher bias, low variance.

σ^2 Small σ^2: Features f_i vary less smoothly.
Lower bias, higher variance.

The reason: In this case, the kernel function is more sensitive to local variations in the data, which results in a hypothesis that fits the training data set.



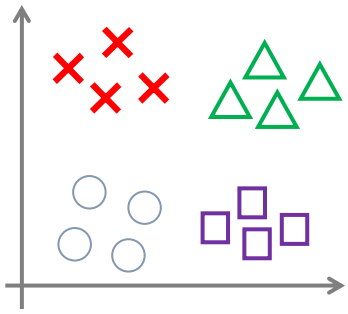
Large σ^2: Features f_i vary more smoothly.
Higher bias, lower variance.



Using a SVM

1. Find the $\text{similarity}(x, l)$ like linear kernel, Gaussian kernel, Polynomial kernel, String kernel, etc, which satisfy the "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge.
2. Do perform feature scaling before using the kernel method.
3. In the case of Multi-class classification, many SVM packages already have built-in multi-class classification functionality.
Otherwise, use one-vs.-all method

$$y \in \{1, 2, 3, \ldots, K\}$$



Logistic regression vs. SVMs

n =number of features ($x \in \mathbb{R}^{n+1}$), m =number of training examples

- If n is large (relative to m):
  Use logistic regression, or SVM without a kernel ("linear kernel")
- If n is small, m is intermediate:
  Use SVM with Gaussian kernel
- If n is small, m is large:
  Create/add more features, then use logistic regression or SVM without a kernel

Neural network likely to work well for most of these settings, but may be slower to train.