# Machine learning Stanford Coursera

## Machine learning definition & Learning topic

▼ **Arthur Samuel (1959)**

- ***Machine Learning***: Field of study that gives computers the ability to learn without being explicitly programmed

## Example

▼ **Database mining**

- Large datasets from growth of automation/web
- **E.g.,** Web click data, medical records, biology, engineering

▼ **Application can't program by hand**

- **E.g**., Autonomous helicopter, handwriting recognition, most Natural Language Processing (NLP), Computer Vision.

▼ **Self-customizing programs**

- **E.g.,** Amazon, Netflix product recommendations

- **Understanding human learning (Brain, real AI)**

## Learning topics

### Machine learning algorithm

▼ **Supervised learning**

- **Linear regression**
- **Logistic regression**
- **Neural network**
- **Support Vector Machine (SVMs)**

▼ **Unsupervised learning**

- **K-means clustering**
- **Principal Component Analysis (PCA)**
- **Anomaly detection**

### Special applications / Special topics

- **Recommender system**
- **Large scale machine learning**

### Advise on building a machine learning system

- ***Bias/Variance, Regularization***
- ***Deciding what to work on next: Evaluations of learning algorithms, learning curves, Error analysis, ceiling analysis***

# Linear Regression

### WHEN is Linear regression useful? (NOT DONE)

> **Linear regression** is used to ***predict the real-value output***

- The continuous target features such as "Pricing of the house", "Student's grade",...

## Cost function (Linear regression)

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

**Cost function** equals to

- 1/2m * ***Square error between ("predicted result from hypothesis function h(x)" & "Value y data")***
- (Theta1, Theta2) are parameters of h(x)

▼ **Explaining the terms (1/2m)** in J(theta1, theta2)

- The (1/m) is to "average" the square error over the number of components
- The term (1/2) exist because, by the rule of thumb, ***2m seems to be more "natural"*** —> Author (Andrew Ng) prefers it & ***When minimizing J, It doesn't matter J or 2J***

# Gradient descent (Linear regression)

▼ **Objectives**

- Start with some (theta1, theta2,...) = (0, 0, ...)
- Keep changing (Theta1, Theta2,...) to reduce cost function J(theta1, theta2) until we hopefully end up at a minimum

**Gradient descent algorithm**

repeat until convergence {

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

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

}

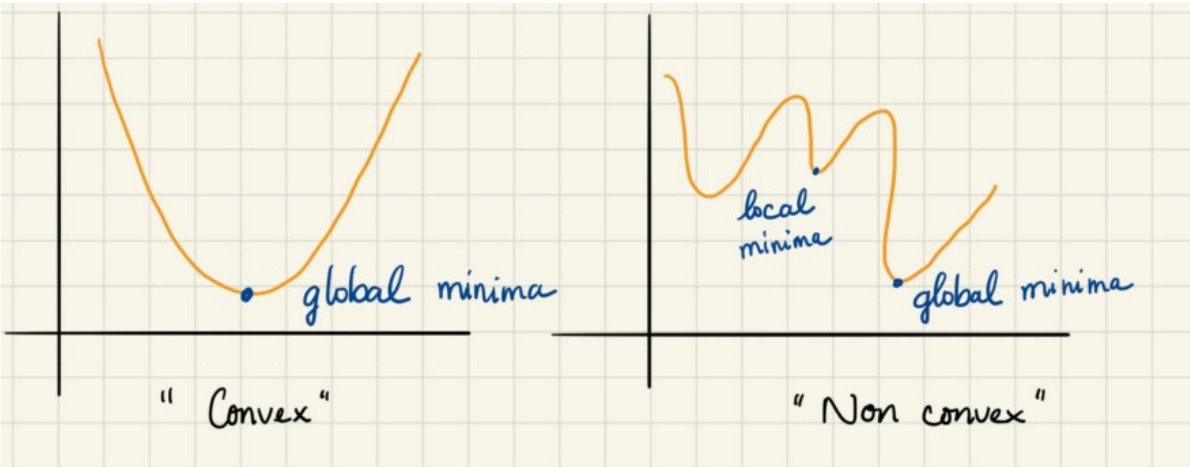
**Linear Regression Model**

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

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

▼ **NOTE!!!!!!!!!!**

- In "Gradient descent", The ***meanings of J(Theta1, Theta2)'*** is to ***decide the direction of the steps*** (Positive = left & Negative = right)
- **Learning rate (alpha)** ***decides the size of the step***
- Global minimum Vs Local minimum




(**Question!!!**) ***How to solve*** the problem when the ***cost function always converges to a local minimum*** instead of a global minimum?

# Logistic regression

## When using it?

> **Logistic regression** is used to solve ***Classification problem***

▼ **Examples**

- Email: ***Spam / Not spam***?
- Online Transaction: Fraudulent ***(Yes / No)***?
- Tumor: ***Malignant / Benign***

# Hypothesis function

## Sigmoid / Logistic function

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

> h(x) = ***estimate probability that y=1 on input x***

- **Example**

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

$h_\theta(x) = 0.7$ $\quad y = 1$

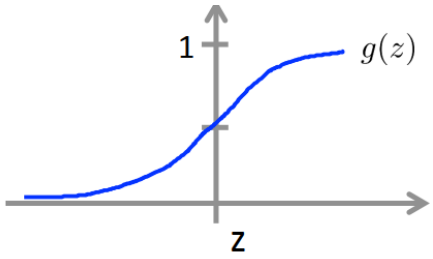
Tell patient that 70% chance of tumor being malignant

***If h(x) = 0.7*** —> It means that the ***patient has 70% chance of tumor being malignant***

## How to *validate the result of hypothesis function h(x)*

**Logistic regression**

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




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

$\theta^T x \geq 0$

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

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

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

$\theta^T x < 0$

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

- **Predict "y=1"** When **h(x) ≥ 0.5**
- **Predict "y=0"** When **h(x) < 0.5**

It means that

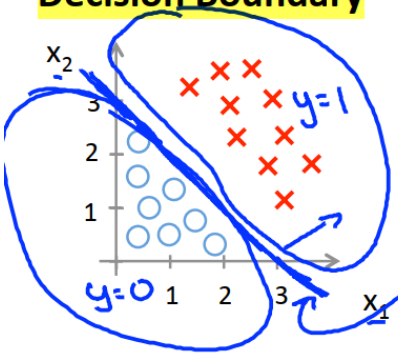
$$h_\theta(x) = g(\theta^T x)$$
$$g(z) = \frac{1}{1+e^{-z}}$$

> **h(x) ≥ 0.5** when **z ≥ 0** —> Model **predict "y=1"** **when z ≥ 0**

> **h(x) < 0.5** when **z < 0** —> Model **predict "y=0"** **when z < 0**

▼ **Example**

## Decision Boundary




Theta

$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

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

$\theta_0 = -3$, $\theta_1 = 1$, $\theta_2 = 1$

Decision boundary

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$\theta^T x$

$x_1 + x_2 \geq 3$

$x_1, x_2$

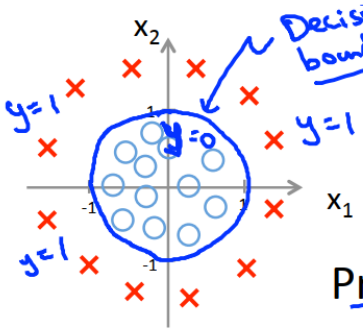
$h_\theta(x) = 0.5$

$x_1 + x_2 = 3$

$x_1 + x_2 < 3$

$y = 0$

## Non-linear decision boundaries




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

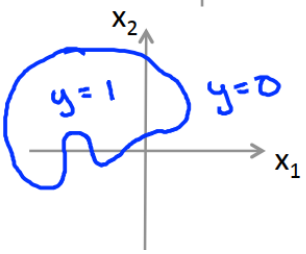
$\theta_0 = -1$, $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 1$, $\theta_4 = 1$

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

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

$x_1^2 + x_2^2 = 1$

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




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

## Cost function (Logistic regression)

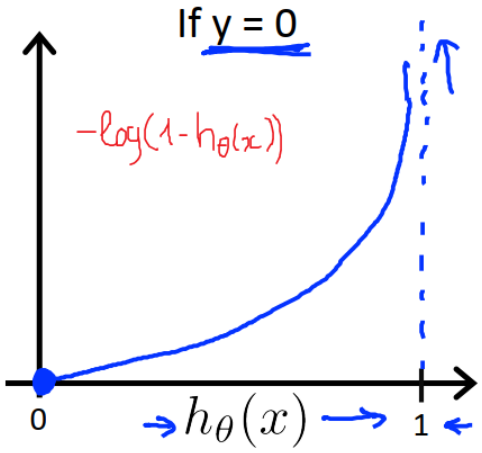
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

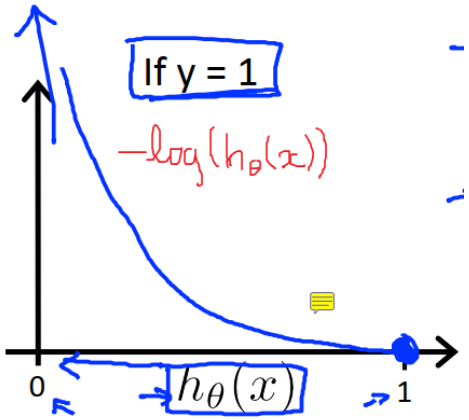
—> Conduct **(Simplify) to 1 cost function** based on value of y=1 or y=0

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

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

▼ **Cost function visualization**




▼ **REMARKS**

- **h(x)** _ Hypothesis function ***shows the probability*** that a data point ***getting value y=1 (YES)***
  - If Cost() ***increase*** —> ***h(x) closer to 0***

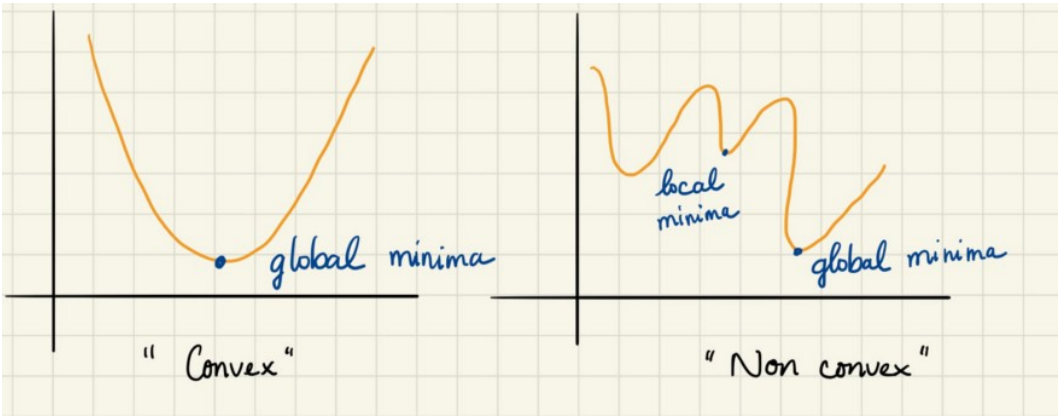
- If Cost() **decrease** —> ***h(x) closer to 1***

- **1 - h(x) _ *shows the probability*** that a data point ***getting value y=0 (NO)***
  - If Cost() ***increase*** —> ***h(x) closer to 1***
  - If Cost() **decrease** —> ***h(x) closer to 0***

▼ **NOTE!!!!!!!!!**

- ***Reason*** why **using "log(h(x))"** instead of directly use "h(x)" is

  —> It ***makes*** the ***cost function CONVEX*** —> Easy to determine GLOBAL MINIMUM




# Gradient descent (Logistic regression)

- The algorithm is similar to "Linear regression" gradient descest

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

—> It **becomes**

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

**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 $\theta_j$)

$\}$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$ for $i = 0$ to $n$

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

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

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

Algorithm looks identical to linear regression!

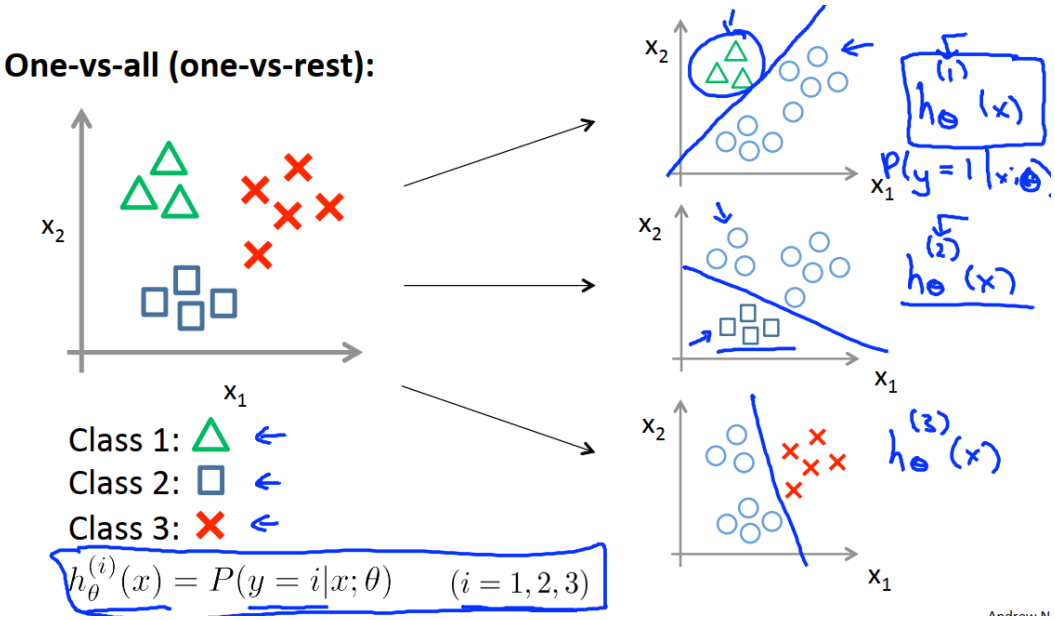
# Multi-classification problem

▼ **Problem**

- Email tagging: Work **(y=1)**, Friends **(y=2)**, Family **(y=3)**, Hobby **(y=4)**
- Medical diagram: Not ill **(y=1),** cold (**y=2),** flu **(y=3)**
- Whether: Sunny **(y=1)**, Cloudy **(y=2)**, Rain **(y=3),** Snow **(y=4)**

▼ **Hypothesis function & visualization**

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




▼ **Multi-classification process**

- Train the logistic regression classifier ***h_theta(i)(x)*** for ***each class "i"*** to ***predict the probability that y = i***

$$h_\theta^{(i)}(x)$$

- ***After training***, we got the ***"trained theta"*** —> ***How*** to ***make the prediction***?
  - Use the "trained theta" & input X to compute the hypothesis function h(x)
  - It results that for each data point (row), # of columns equals to # of classification terms (y=1, y=2, y=3,...) —> ***Just picking the max probability*** out of these probability columns

(**Remember**) ***h(x)*** shows the ***probability that value y equaled to each classification term (y=1 or y=2 or...)***

# Overfitting problem

▼ **Addressing Overfitting**

Options:

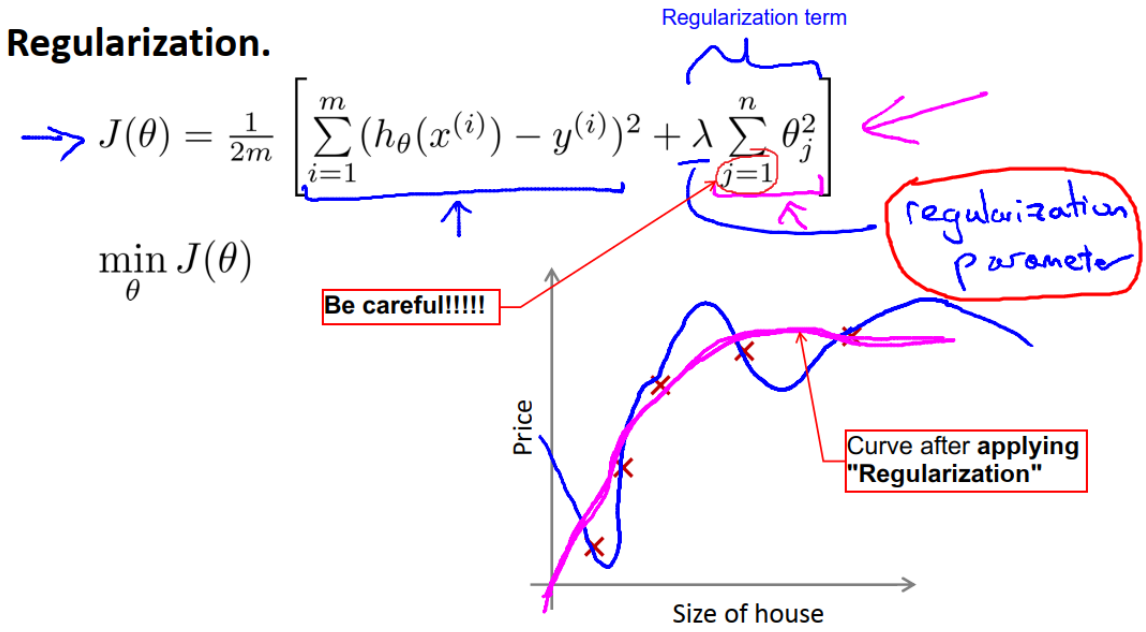
1. Reduce number of features.
   - — Manually select which features to keep.
   - — Model selection algorithm (later in course).
2. Regularization.
   - — Keep all the features, but reduce magnitude/values of parameters $\theta_j$.
   - — Works well when we have a lot of features, each of which contributes a bit to predicting $y$.

# Regularization

## Regularization in Cost function

▼ **Formula**

- Regularization is applied to "Cost function"




- "lambda" is **regularization parameter**
  - If ***lambda is set too large***
    - ***Fail to eliminate "Overfitting"*** problem

- Gradient descent will fail to converge

—> DO ***NOT set "lambda" too large***

▼ **NOTE!!!!!!!!!!**

- When performing regularization, ***avoid "theta_0" only run from theta_1 to theta_n*** (Rule of thumb)

## Regularization in Linear regression

▼ **Cost function (Linear regression)**

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

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

▼ **Gradient descent (In detail)**

**Gradient descent**

Repeat {

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

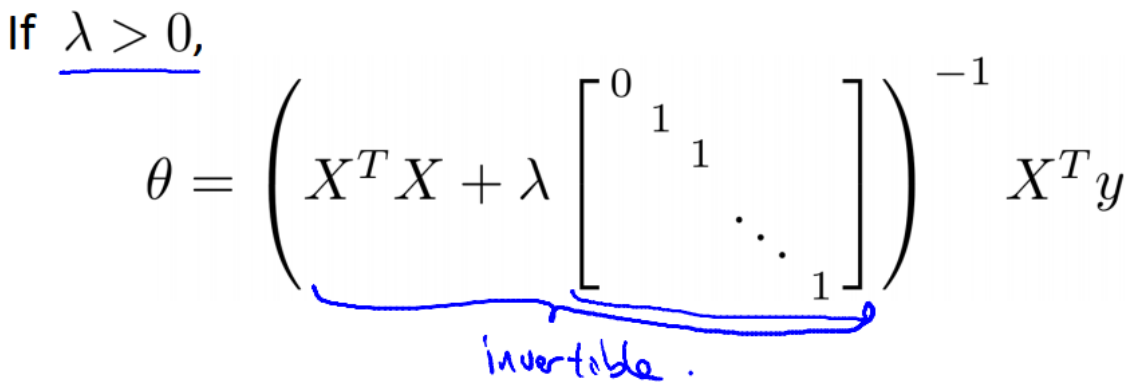
$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \ldots, n)$$

}

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

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

—> In short, normal equation for gradient descent




(**Remember**) ***Avoid theta_0, only perform from theta_1 to theta_n*** when applying ***Regularization***

## Regularization in Logistic regression

▼ **Cost function** (logistic regression)

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

—> Cost function ***applying Regularization***

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

▼ **Gradient descent (in detail)**

**Gradient descent**

Repeat {

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

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

}

$\theta_1 \ldots \theta_n$

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

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

# Neural network

## Neural networks Overview

- In ***logistic regression***:

```
X1  \
X2   ==>  z = XW + B ==> a = Sigmoid(z) ==> l(a,Y)
X3  /
```

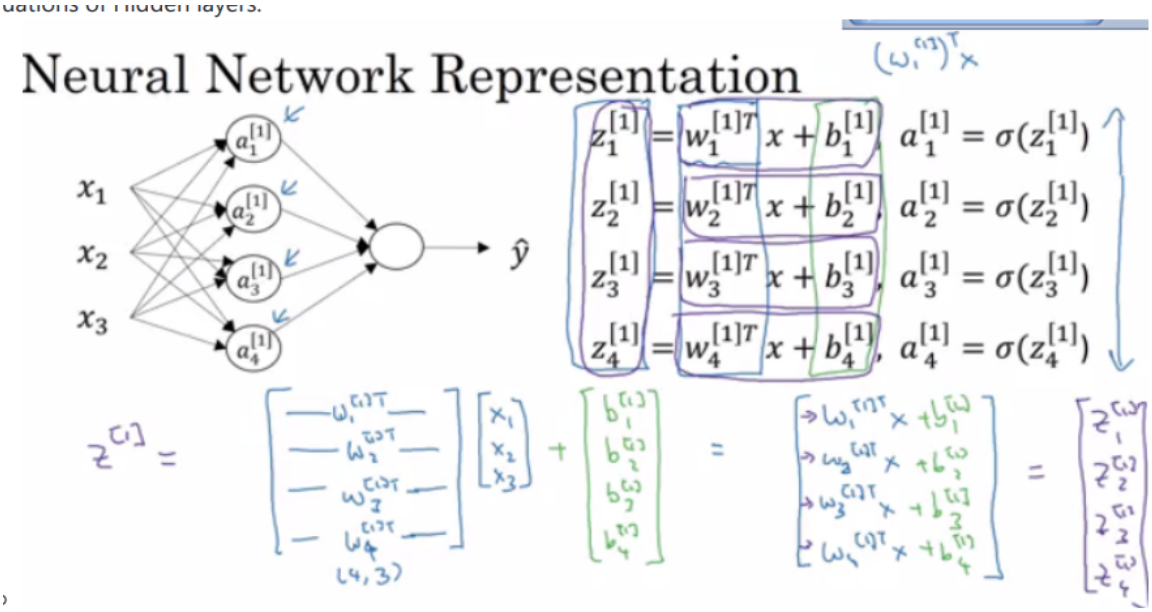
- In ***neural networks with one layer***:

```
X1  \
X2   =>  z1 = XW1 + B1 => a1 = Sigmoid(z1) => z2 = a1W2 + B2 => a2 = Sigmoid(z2) => l(a2,Y)
X3  /
```

- **NN** is a ***stack of logistic regression*** objects

## Neural Network Representation

- NN contains input layers, hidden layers, output layers
- **Equation** of hidden layers




▼ Here is some info about the last images

- noOfHiddenNeurons = 4
- Nx = 3

▼ The s**hape of the variables**

- **"W1"** is the ***matrix of the first hidden layer***, it has a shape of **(noOfHiddenNeurons, Nx)**
- **"b1"** is the ***matrix of the first hidden layer***, it has a shape of **(noOfHiddenNeurons, 1)**
- **"z1"** is the ***result of the equation*** z1 = W1*X + b, it has a shape of **(noOfHiddenNeurons, 1)**
- **"a1"** is the ***result of the equation*** a1 = sigmoid(z1) , it has a shape of **(noOfHiddenNeurons,1)**
- **"W2"** is the ***matrix of the second hidden layer***, it has a shape of **(1, noOfHiddenNeurons)**
- **"b2"** is the ***matrix of the second hidden layer***, it has a shape of **(1,1)**
- **"z2"** is the ***result of the equation*** z2 = W2*a1 + b, it has a shape of **(1,1)**
- **"a2"** is the ***result of the equation*** a2 = sigmoid(z2), it has a shape of **(1,1)**

# Activation functions

- **Sigmoid** can lead us ***to gradient decent problems*** where the updates are so slow.
- Sigmoid activation function range is [0,1]

$A = 1/(1 + np.exp(-z))$ # Where z is the input matrix

- **Tanh activation function** range is [-1,1]

  $A = (np.exp(z) - np.exp(-z))/(np.exp(z) + np.exp(-z))$ # z is input matrix
- It turns out that the ***tanh activation usually works better than the sigmoid activation function for hidden units***
  - Because the mean of its output is closer to zero, and so it centers the data better for the next layer.
- **Sigmoid or Tanh function disadvantage** is that ***if the input is too small or too high, the slope will be near zero which will cause us the gradient decent problem***
- One of the popular activation functions that solved the slow gradient descent is the **RELU function.**

  $RELU = max(0, z)$ # If z is negative, the slope is 0 & if z is positive, the slope remains linear
- So here is some basic rule for choosing activation functions, ***if your classification is between 0 and 1***, use the ***output activation as sigmoid and the others as RELU***.

## Why do we need non-linear activation functions?

- ***If we removed the activation function*** from our algorithm that can be ***called the linear activation function.***
- If removing the activation function, ***whatever hidden layers you add***, the activation will be ***always linear like logistic regression*** —> it's ***useless in a lot of complex problems***

# Gradient descent for Neural Network

- In this section, we will ***have the full backpropagation*** of the neural network

▼ **NN parameters**

- **n[0] = Nx**
- **n[1] = NoOfHiddenNeurons**
- **n[2] = NoOfOutputNeurons** = 1
- **W1** shape is **(n[1], n[0])**
- **b1** shape is **(n[1], 1)**
- **W2** shape is **(n[2], n[1])**
- **b2** shape is **(n[2], 1)**

- **Cost function**

**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:

Result of the **"ith example"**

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

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

**"kth output unit"** in model

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

## Gradient descent

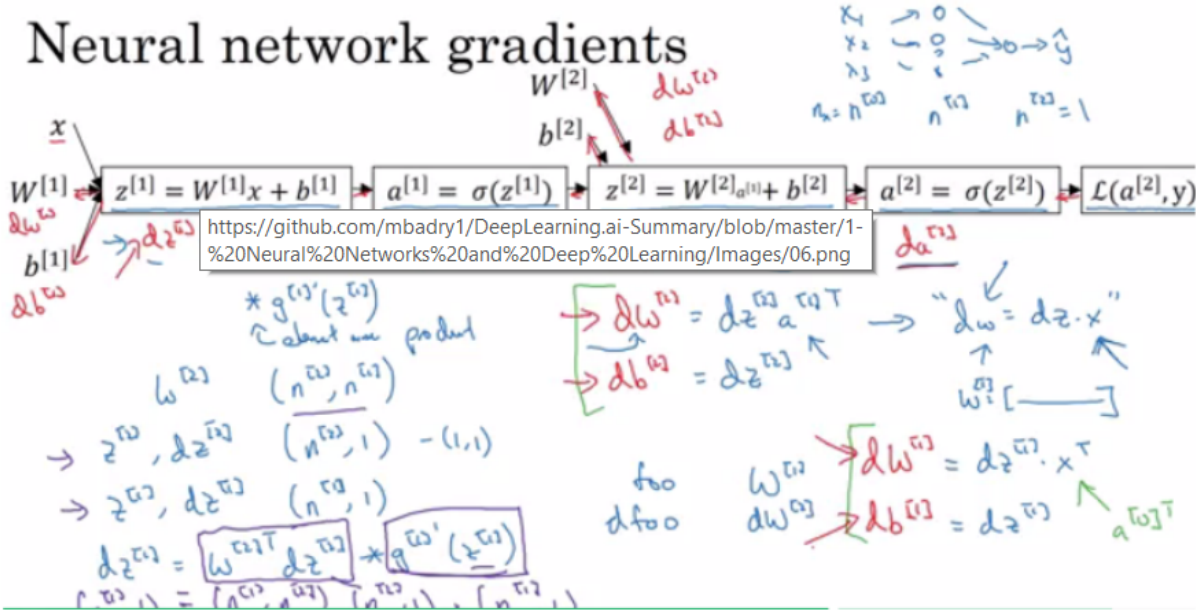
```
Repeat:
                Compute predictions (y'[i], i = 0,...m)
                Get derivatives: dW1, db1, dW2, db2
                Update: W1 = W1 - LearningRate * dW1
                              b1 = b1 - LearningRate * db1
                              W2 = W2 - LearningRate * dW2
                              b2 = b2 - LearningRate * db2
```

▼ **Forward propagation**

```
Z1 = W1A0 + b1    # A0 is X
A1 = g1(Z1)
Z2 = W2A1 + b2
A2 = Sigmoid(Z2)      # Sigmoid because the output is between 0 and 1
```

▼ **Backpropagation**

```
dZ2 = A2 - Y      # derivative of cost function we used * derivative of the sigmoid function
dW2 = (dZ2 * A1.T) / m
db2 = Sum(dZ2) / m
dZ1 = (W2.T * dZ2) * g'1(Z1)  # element wise product (*)
dW1 = (dZ1 * A0.T) / m   # A0 = X
db1 = Sum(dZ1) / m
# Hint there are transposes with multiplication because to keep dimensions correct
```




# Random initialization

- ***If we initialize all the weights with zeros in NN it won't work*** (initializing bias with zero is OK):
  - All hidden units will be completely identical (symmetric) - compute exactly the same function

- On each gradient descent iteration, all the hidden units will always update the same

- **Initialize the W's with a small random numbers**

```
W1 = np.random.randn((2,2)) * 0.01    # 0.01 to make it small enough
b1 = np.zeros((2,1))                  # its ok to have b as zero, it won't get us to the symmetry breaking
problem
```

# Support vector machine

▼ **Definition**

- **Support vector machine (SVM)** is also a ***supervised algorithm*** that sometimes ***gives a cleaner*** AND sometimes a ***more powerful way of learning complex non-linear functions***

(**NOTE**) ***SVM*** is ***so sensitive with outliers*** data —> ***Solving this problem*** is the ***reason*** why ***parameter C exist***

▼ The ***main idea behind*** **Support Vector Machines** (**SVM**)

1. ***Start with data*** in a relatively ***low dimension***
2. ***Move*** data into a ***higher dimension***
3. Find ***Support Vector Classifier (SVC)*** that ***separates*** the higher dimensional data ***into 2 groups***

(**Question**) In step 2, ***how*** do we ***decide how to transform the data***?

—> SVM uses something called ***"Kernel function"*** to systematically ***find SVC in higher dimensions*** (step 3)

## Kernels

▼ **Definition**

- **Kernel** is used due to set of ***mathematical functions*** used in Support Vector Machine ***provides the window to manipulate the data***
- **Kernel function** generally ***transforms the training set of data*** so that a ***non-linear decision surface*** is able to transform to a linear equation in a ***higher***

***number of dimension spaces.***

- ***Choosing the right kernel is crucial***, because if the transformation is incorrect, then the model can have very poor results

## Kernel functions

- **Polynomial kernel function**
- **Radial basis function (RBF) kernel function**
- **Gaussian kernel**

▼ **Kernel tricks**

- Is the trick to calculate the high-dimensional relationships without actually transforming the data to a higher dimension, is called "The Kernel Trick"

▼ ***Reference link*** to continue

https://towardsdatascience.com/svm-and-kernel-svm-fed02bef1200

https://www.youtube.com/watch?v=efR1C6CvhmE&ab_channel=StatQuestwithJoshStarmer

# Cost function

▼ **Remind**

- **Logistic regression**

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

A B

- **Hypothesis function** (Logistic regression)

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

## Cost function of SVM

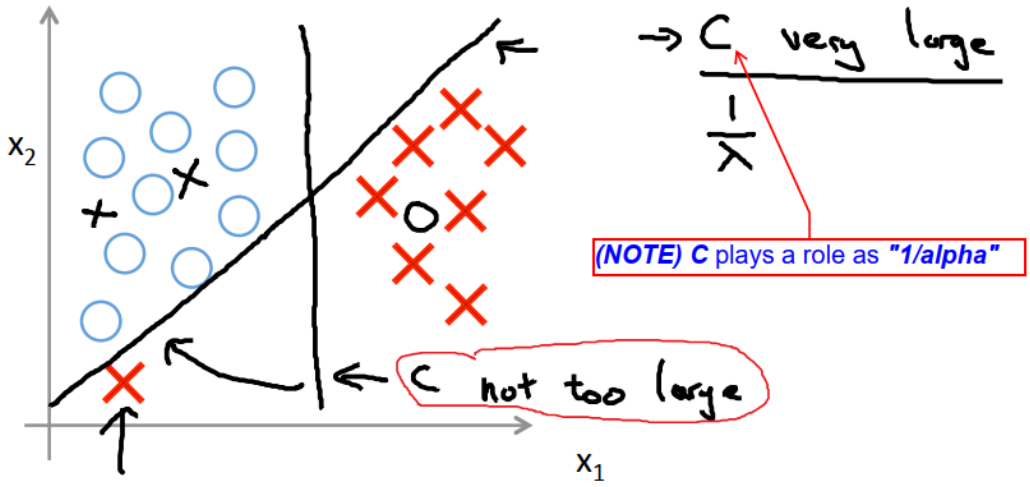
- Cost of SVM is the ***improvement*** based on ***both Logistic regression & Hypothesis function***

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

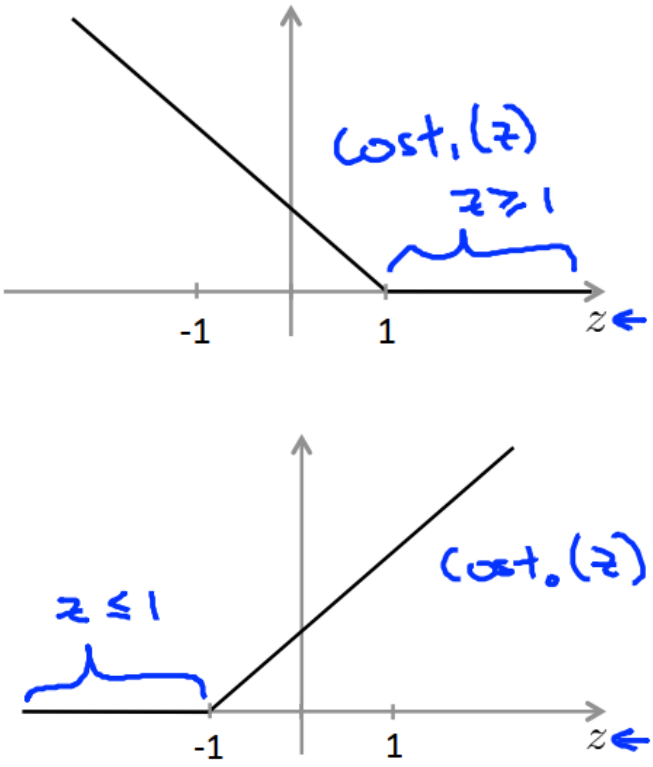
▼ **Parameter C** plays a role ***as (1 / alpha)***

***How*** to ***choose the value of C?***

- ***If C is very large*** —> **Decision boundary** will ***NOT ignore some outliers*** (Decrease the performance of algorithm)
- If C is large enough —> **Decision boundary** will ***ignore some outliers***




▼ **Visualization**

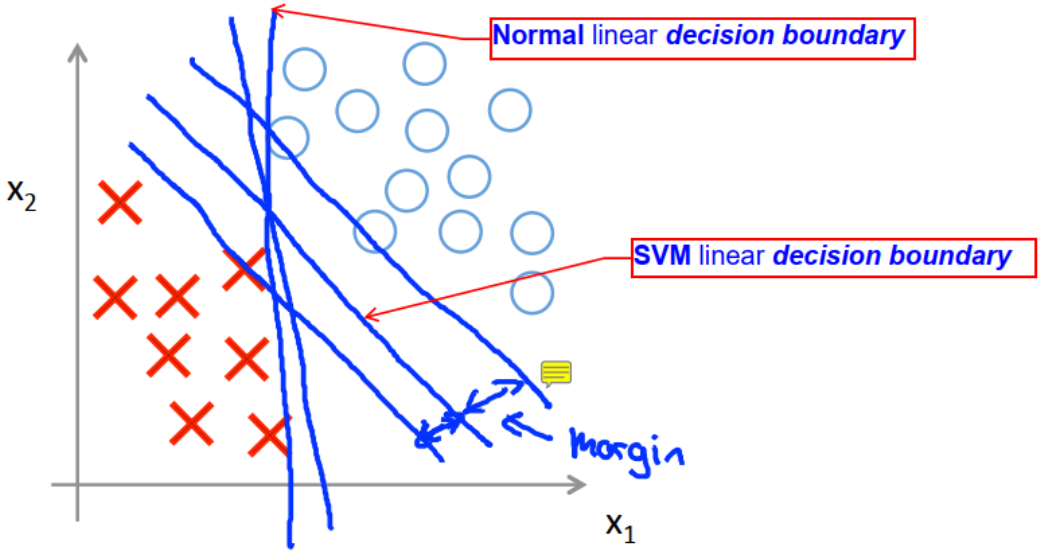



—> **Hence**

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

▼ **Reason calling SVM** is **"large margin classifier"**

- There is a l***arge distance*** between ***decision boundary of SVM*** and ***2 separated type of data***
  —> Seem to be more robust than the other linear decision boundary

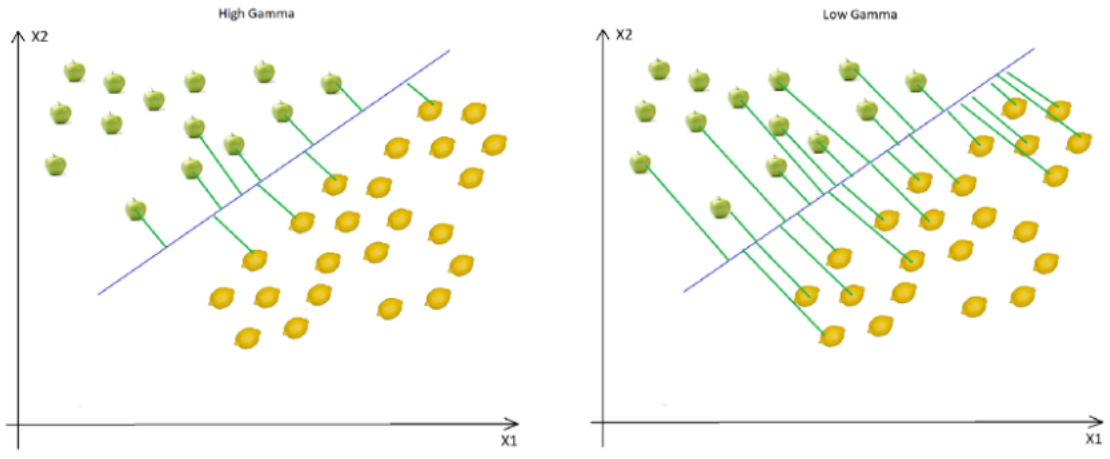



Large margin classifier

## Application of SVM in Python

▼ Most important parameters

1. **kernel**: The most common kernels are **rbf** (this is the default value), **poly** or **sigmoid**, **Gaussian**, ...
2. **C**: This is ***regularization parameter*** (talking about this above)
3. **gamma**: Defines ***how far the influence of a single training example reaches***




4. **degree**: used only if the chosen kernel is poly to set the degree

▼ Example of ***applying SVM in python***

```
# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', C = 0.1, gamma = 0.1)
classifier.fit(X_train, y_train)
```

## Determine features important in Linear SVM

- The weight obtained from ***"svm.coef_"*** represents the ***list of coefficients*** between each feature with the target classification
- The example below showcases the top 15 words which are highly correlated with detecting spam email

```
In [45]: print("\nFinding the weight of each word in Porter's vocab")
         vocab_df["weights"] = clf.coef_[0, :]
         top15 = vocab_df.sort_values("weights", axis=0, ascending=False).head(15)
         print(top15)
```

```
Finding the weight of each word in Porter's vocab
      index     vocab   weights
1190   1191       our  0.500614
297     298     click  0.465916
1397   1398     remov  0.422869
738     739  guarante  0.383622
1795   1796     visit  0.367710
155     156  basenumb  0.345064
476     477    dollar  0.323632
1851   1852      will  0.269724
1298   1299     price  0.267298
1263   1264     pleas  0.261169
1066   1067      most  0.257298
1088   1089      nbsp  0.253941
965     966        lo  0.253467
698     699        ga  0.248297
791     792      hour  0.246404
```

(**NOTE**) Only used in ***linear kernel***