# Machine Learning

**Nagubandi Krishna Sai**
**MS20BTECH11014**

**June 2021**

# Contents

# Chapter 1

# Fundamentals of Machine Learning

## 1.1 Supervised Learning

Supervised Learning gives "correct answers", the output values are same as real life values.
In Supervised Learning, we are given a set of data and we know what our correct output should look like, having an idea that there is relationship between the input and the output.
Supervised Learning problems has two types of problems,

1. Regression.
2. Classification.

### 1.1.1 Linear Regression

In regression type of problems, we are trying to predict results within a continuous output, means that we are trying to map input variables to some continuous function.
Linear regression has real-valued output, but the output will be same or near valued to the actual output.

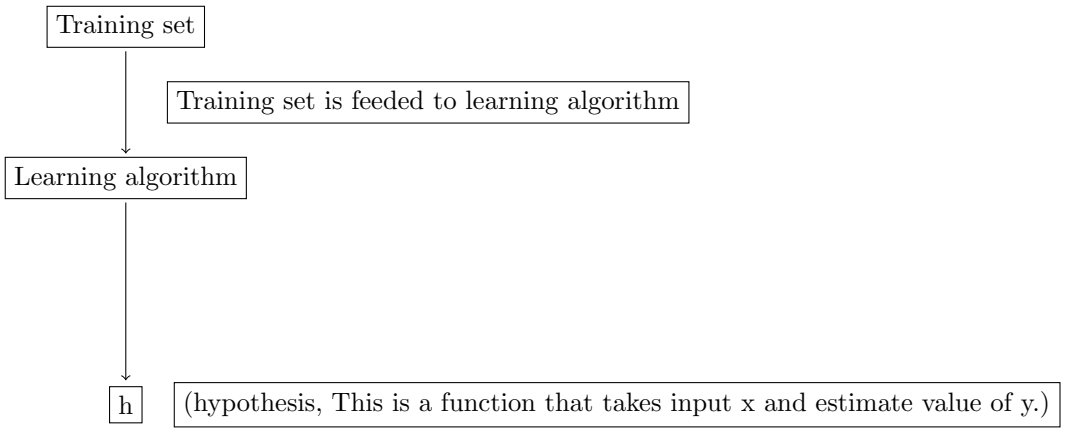
1. m = Number of training examples.
2. x's = "input" variable (or) feature.
3. y's = "output (or) target" variable.
4. (x,y) = one training example.
5. $(x^{(i)},y^{(i)}) = i^{th}$ training example.

Example :

| Size of feet$^2(x)$ | Price($) in 1000's (y) |
|---|---|
| 2104 | 460 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ⋮ | ⋮ |

Table 1.1: Training set of housing prices.

1. m = 47.
2. $x^{(1)} = 2104$ and $y^{(1)} = 460$.
3. $x^{(2)} = 1416$ and $y^{(2)} = 232$.




### 1.1.2 Hypothesis for Linear Regression

$$h_\theta(x) = \theta_0 + \theta_1.x \tag{1.1}$$

$\theta_i$'s = parameters.

This type of hypothesis mode is called "Linear regression with one variable" (or) "Univariate linear regression."

### 1.1.3 Cost function for Linear Regression

Cost function helps us know that how well to fit the best possible straight line over the given data.

Q⟩ *How to choose* $\theta_i$'s ?

1. Choose $\theta_i$ 's so that $h_\theta(x)$ is close to y for our training example (x,y).
2. minimise $\theta_0$,$\theta_1$ so, that [$h_\theta(x)$ - y] is small.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x^{(i)}) - y^{(i)}]^2 \tag{1.2}$$

J($\theta_0$,$\theta_1$) = Cost function (or) Squared error function.

### 1.1.4 Gradient descent for Linear Regression

Gradient descent is used to minimise cost function(J) in linear regression.
Gradient descent is used in many areas to minimise many functions in ML/AI.
**Gradient descent algorithm,**

$$\text{Repeat until convergence (minimum)} \left\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1),\ for\ j = 0, j = 1. \right. \tag{1.3}$$

1. := is Assignment operator.
2. $\alpha$ is learning rate.
3. $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ is derivative.

Gradient descent is nothing but the derivative of the Cost function.

$$Slope\ of\ cost\ function\ curve = \frac{\partial J(\theta_1)}{\partial \theta_1},\ when\ \theta_0 = 0. \tag{1.4}$$

**Learning rate,**

1. If $\alpha$ is too small, gradient descent can be slow. After many such operations(can be infinite times), the '$\theta_1$' could reach "global minimum".
2. If $\alpha$ is too large, gradient descent can overshoot the minimum. It may "fail to converge (or) even diverge".
3. If $\theta_1$ is at the local optima itself when we started or taken $\theta_1$, then there is no use of "$\alpha$ (or) gradient descent".

### 1.1.5 Linear Regression for multivariables

**Hypothesis,**

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n. \tag{1.5}$$

In the total context of Supervised learning, hypothesis is just predicting the output.
For convenience of notation, declare $x_0 = 1$ ($x_0^{(i)} = 1$).

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

The above matrix is 0 - indexed.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n . h_\theta(x) = \theta^\top x. \tag{1.6}$$

**Cost function,**

$$J(\theta) = J(\theta_0, \theta_1, \theta_2, ..., \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x^{(i)}) - y^{(i)}]^2 \tag{1.7}$$

**Gradient descent,**

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \theta_2, ..., \theta_n) \tag{1.8}$$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{1.9}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} [h_\theta(x^{(i)}) - y^{(i)}] x_j^{(i)} \tag{1.10}$$

**Feature scaling,**
Get every feature into approximately -1$\leq x_i \leq$ 1 *range.*
**Mean normalization,**
*Replace*$x_i$ *with* $x_i - \mu_i$ *to make features have approximately zero mean.*

### 1.1.6 Polynomial regression

**Hypothesis,**

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3. \tag{1.11}$$

| Housing price prediction |
|---|
| $x_1 = size$ |
| $x_2 = (size)^2$ |
| $x_3 = (size)^3$ |

Table 1.2: Features be-like in Polynomial regression.

### 1.1.7 Normal Equation

**Intuition,**

$$\frac{d}{d\theta} J(\theta) = 0.$$

**Cost function,**

$$\theta \, \epsilon \, \Re^{n+1}, \; J(\theta_0, \theta_1, \theta_2, ..., \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x^{(i)}) - y(i)]^2 \tag{1.12}$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = 0, \; (solve \; for \; \theta_0 \; , \theta_1 \; , ... \; , \theta_n) \tag{1.13}$$

**Example,**

| | Size (feet$^2$) | No.of Bed rooms | No.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 | 1852 | 2 | 1 | 36 | 178 |

Table 1.3: Sample Training set for Multi-Variate Linear regression.

1. n = number of Features.
2. $x_j^{(i)}$ = value of j in the $i^{th}$ training example.
3. $x_{(i)}$ = the input(features) of the $i^{th}$ training example.

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

X is m×(n+1)-dimensional matrix and Y is a m-dimensional vector.

$$\theta = (X^\top X)^{-1} X^\top Y \tag{1.14}$$

The above $\theta$ value is optimal $\theta$ value.
For Normal equation method, then no need to use **feature scaling.**
We use 'Gradient descent' and 'Normal equation' methods to minimise cost function.

| Gradient descent | Normal equation |
|---|---|
| 1⟩ Need to choose '$\alpha$'. | 1⟩ No need to choose '$\alpha$'. |
| 2⟩ Need many iterations. | 2⟩ Don't need to iterate. |
| 3⟩ Works well even, when 'n' is large. (n¿10000) | 3⟩ Need to compute n× n matrix inverse (X⊤X)$^{-1}$ |
| | 4⟩ Works Now if n is very large. |

Table 1.4: Why should we use the particular method? Advantages and Disadvantages of two methods.

$$\theta = (X^\top X)^{-1} X^\top Y \tag{1.15}$$

Q⟩ What is $X^\top X$ is non-invertible(singular/degenerate) ?
Reasons,

1. Redundant features (linearly dependent)
   - $x_1$ = Size in $feet^2$
   - $x_2$ = Size in $m^2$
   - $x_2 = (3.28)^2\ x_1$ , 1m = 3.28feet.
2. Too many features. (m≤n)
   - m = 10
   - n = 100, $\theta\ \epsilon\ \Re^{101}$, Delete some features (or) Regularization.

### 1.1.8 Classification

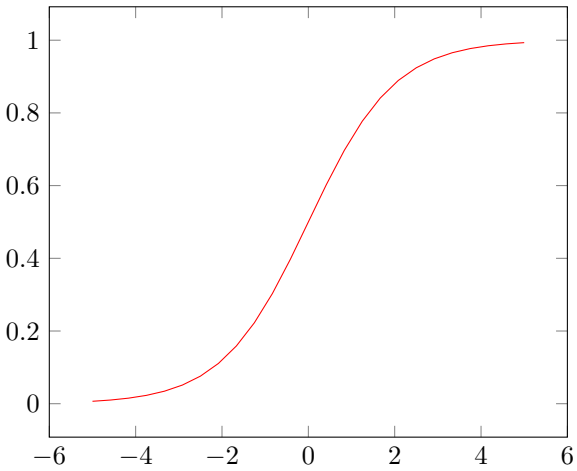
The output value 'y' is **discrete value.**
The algorithm used is **logistic regression.**

### 1.1.9 Logistic Regression Model

We want $0 \leq h_\theta(x) \leq 1$.

$$h_\theta(x) = g(\theta^\top x) \tag{1.16}$$
$$g(z) = \frac{1}{1+e^{-z}} \tag{1.17}$$
$$h_\theta(x) = g(\theta^\top x) \tag{1.18}$$
$$= \frac{1}{1+e^{-\theta^\top x}} \tag{1.19}$$

The above g(z) is called sigmoid function (or) logistic function.
**Graph,**



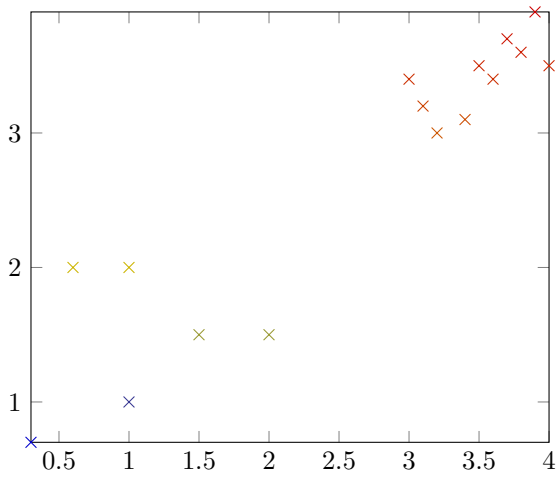

$$g(z) \geq 0.5, \ when \ z \geq 0. \tag{1.20}$$
$$h_\theta(x) = g(\theta^\top x) \geq 0.5, \ when \ \theta^\top x \geq 0. \tag{1.21}$$

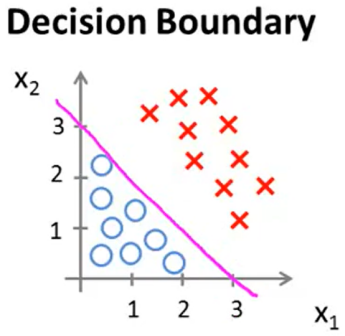
### 1.1.10 Interpretation of hypothesis output for Logistic Regression

$$h_\theta(x) = P(y=1|x;\theta) \tag{1.22}$$
$$P(y=0|x;\theta) + P(y=1|x;\theta) = 1 \tag{1.23}$$
$$P(y=0|x;\theta) = 1 - P(y=1|x;\theta) \tag{1.24}$$
$$= 1 - h_\theta(x) \tag{1.25}$$
$$\tag{1.26}$$

y = 0 (or) 1.

**Decision boundary,**




For, the above diagram decision boundary will be a line separating the two output values of y(y=0 (or) y=1).

$$h_\theta(x) \geq 0.5 \rightarrow y = 1. \tag{1.27}$$

$$h_\theta(x) < 0.5 \rightarrow y = 0. \tag{1.28}$$

$$h_\theta(x) = g(\theta^\top x) \tag{1.29}$$

$$= g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \tag{1.30}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \qquad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

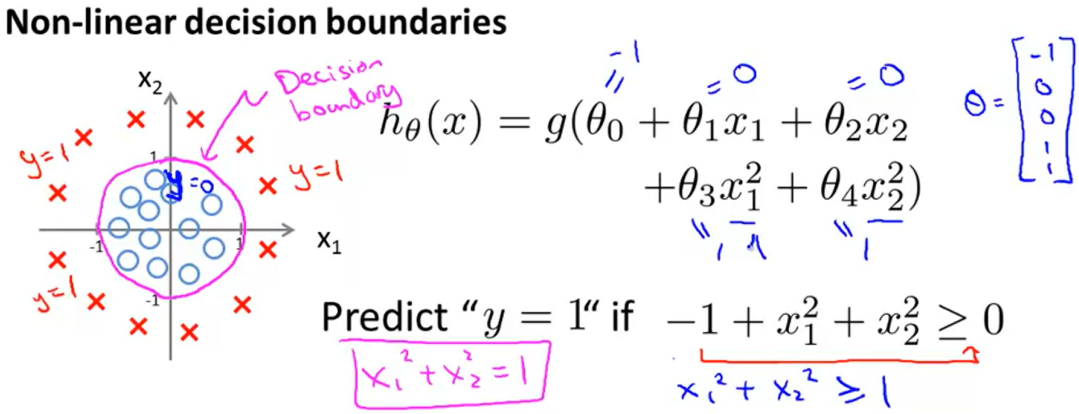
**Example,**

Let,

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

$$y = 1, if\ \theta^\top x \geq 0 \tag{1.31}$$
$$-3 + x_1 + x_2 \geq 0 \tag{1.32}$$
$$x_1 + x_2 \geq 3,\ y = 1 \tag{1.33}$$
$$x_1 + x_2 < 3,\ y = 0. \tag{1.34}$$

**Non-Linear Decision Boundary,**




So, in the above non-linear classification, the **decision boundary is a circle** of radius of 1unit.

$$Inside\ circle,\ y = 0 \tag{1.35}$$
$$Outside\ circle,\ y = 1. \tag{1.36}$$

### 1.1.11 Cost function for Logistic Regression

**Training set,**

| $x^{(1)}$ | $y^{(1)}$ |
|---|---|
| $x^{(2)}$ | $y^{(2)}$ |
| $x^{(3)}$ | $y^{(3)}$ |
| $\vdots$ | $\vdots$ |
| $x^{(m)}$ | $y^{(m)}$ |

Table 1.5: Training set of m-examples for Logistic Regression

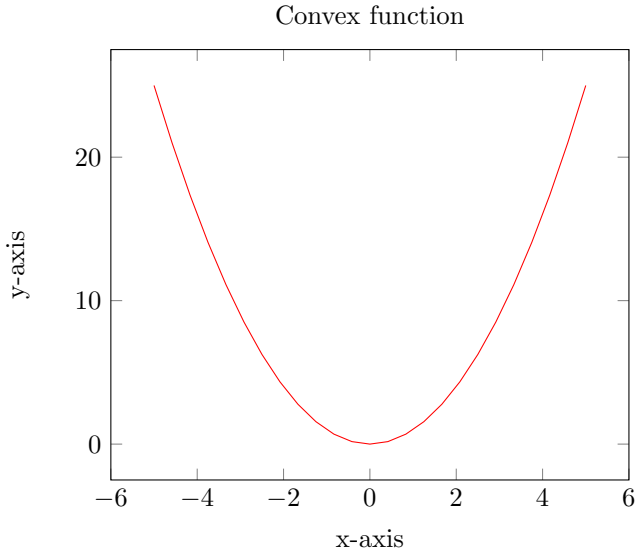
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \epsilon \Re^{n+1} \; - \; n \; features. \; x_0 = 1, \; y \; \epsilon \; 0, 1.$$

For linear regression,

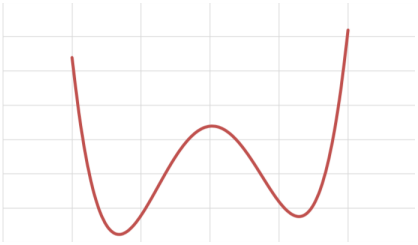
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x^{(i)} - y^{(i)}]^2 \tag{1.37}$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}, y^{(i)}) \tag{1.38}$$

$$Cost(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2}[h_\theta(x^{(i)} - y^{(i)}]^2 \tag{1.39}$$



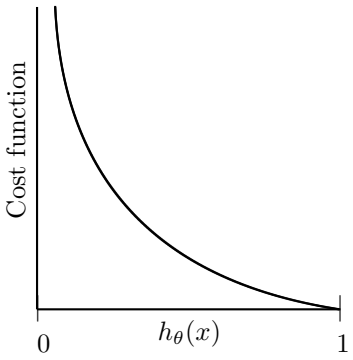
The above graph is a convex.



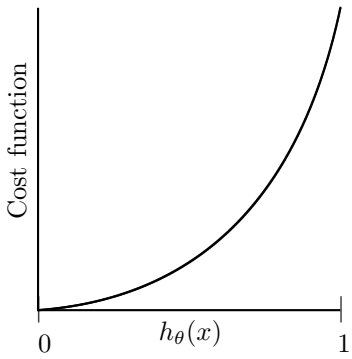
The below graph is a non-convex.
If we use the cost function of linear regression in logistic regression, the the we would get non-convex cost function, because the **hypothesis is** $\frac{1}{1+e^{-\theta^\top x}}$.
For Logistic regression,

$$Cost(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -\log(1 - h_\theta(x)), & if\ y = 0 \\ -\log(h_\theta(x)), & if\ y = 1. \end{cases} \tag{1.40}$$

If y=1,




If y=0,




**Simplified Cost function,**

$$Cost(h_\theta(x^{(i)}), y^{(i)}) = -(1-y)\log(1-h_\theta(x)) - y\log(h_\theta(x)).\ \forall\ y\ \epsilon\ \{0,1\}. \tag{1.41}$$

$$If\ y = 1 : Cost(h_\theta(x), y) = -\log(h_\theta(x)). \tag{1.42}$$

$$If\ y = 0 : Cost(h_\theta(x), y) = -\log(1-h_\theta(x)). \tag{1.43}$$

$$Costfunction = J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}, y^{(i)}) \tag{1.44}$$

$$= \frac{1}{m}[-\sum_{i=1}^{m}(1-y^{(i)})\log(1-h_\theta(x^{(i)})) + y^{(i)}\log(h_\theta(x^{(i)}))] \tag{1.45}$$

### 1.1.12 Gradient Descent for Logistic Regression

$$\text{Repeat until convergence (minimum)}\begin{cases}\theta_j := \theta_j - \alpha\dfrac{\partial}{\partial\theta_j}J(\theta) \end{cases} \tag{1.46}$$

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^{(i)}) - y^{(i)}]x_j^{(i)} \tag{1.47}$$

### 1.1.13 Optimization algorithm

1. Gradient descent.
2. Conjugate gradient.
3. BFGS.
4. L - BFGS.

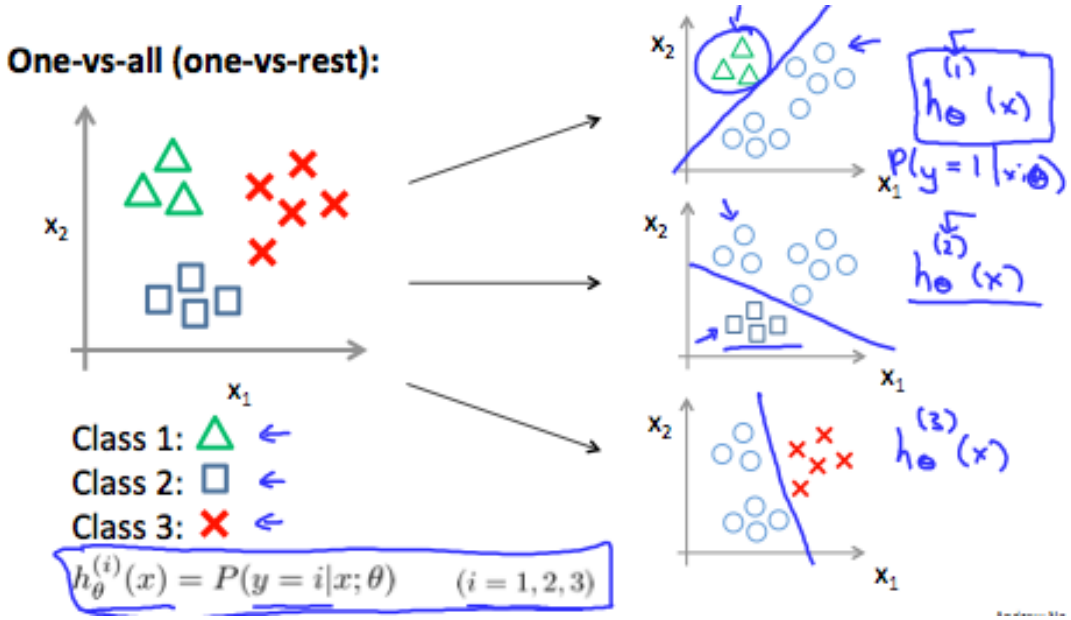
These are the 4 algorithms to minimise **cost function.**
Advantages of the **last three advanced optimization algorithm.**

- No need to manually pick $\alpha$.
- Often faster than Gradient descent.
- They themselves choose $\alpha$, for faster convergence.

### 1.1.14 Multiclass Classification : One-vs-All

$$
\begin{aligned}
y &\in \{0, 1...n\} \\
h_\theta^{(0)}(x) &= P(y = 0|x; \theta) \\
h_\theta^{(1)}(x) &= P(y = 1|x; \theta) \\
&\vdots \\
h_\theta^{(n)}(x) &= P(y = n|x; \theta) \\
\text{prediction} &= \max_i(h_\theta^{(i)}(x))
\end{aligned}
$$

To summarize,




1. Train a logistic regression classifier $h_\theta(x)$ for each class to predict the probability that y=i.
2. To make a prediction on a new x, pick the class that maximizes $h_\theta(x)$

### 1.1.15 Problem of Overfitting,


**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).

Similar for Logistic Regression.

**Addressing Overfitting,**

1. Reduce number of features.
   - Manually select which features to keep.
   - Model selection algorithm.
2. Regularization
   - Keep all features, but reduce magnitude/values of parameters $_j$.
   - Works well when we have a lot of features, each of which contributes a bit to predict $y^{(i)}$.

### 1.1.16 Regularization

Small values for parameters $\theta_0, \theta_1, \theta_2, ..., \theta_n$.

1. Simpler hypothesis.
2. Less prone to overfitting.

**Regularized Cost function,**

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left[ [h_\theta(x^{(i)} - y^{(i)}]^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right] \tag{1.48}$$

If '$\lambda$' is extremely large, then the cost function will become underfitting (doesn't fit to our training data).

Repeat {

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

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

}

The term $\frac{\lambda}{m}\theta_j$ performs our regularization. With some manipulation our update rule can also be represented as:

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \tag{1.49}$$

The first term in the above equation, $1 - \alpha\frac{\lambda}{m}$ will always be less than 1. Intuitively you can see it as reducing the value of $\theta_j$ by some amount on every update. Notice that the second term is now exactly the same as it was before.
**Normal equation after regularization,**

$$\theta = (X^\top X) + \lambda . L^{-1} X^\top Y \tag{1.50}$$

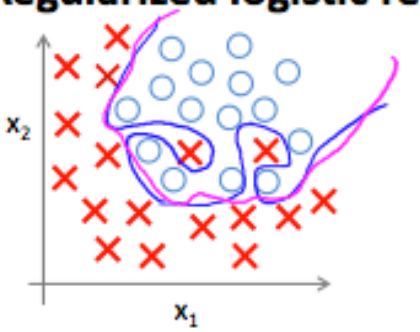
$$where\ L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

### 1.1.17 Regularized Logistic Regression

**Cost function,**

$$J(\theta) = \frac{1}{m}[-\sum_{i=1}^{m}(1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) + y^{(i)} \log(h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{1.51}$$

**Regularized logistic regression.**




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

The second sum, $\sum_{j=1}^{n} \theta_j^2$ means to explicitly exclude the bias term, $\theta_0$. I.e. the vector is indexed from 0 to n (holding n+1 values, $\theta_0$ through $\theta_n$), and this sum explicitly skips $\theta_0$, by running from 1 to n, skipping 0. Thus, when computing the equation, we should continuously update the two following equations:

Repeat {

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

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

}

## 1.2 Neural Networks

### 1.2.1 Model representation I

Let's examine how we will represent a hypothesis function using neural networks. At a very simple level, neurons are basically computational units that take inputs (dendrites) as electrical inputs (called **"spikes"**) that are channeled to outputs (axons). In our model, our dendrites are like the input features $x_1 \cdots x_n$, and the output is the result of our hypothesis function. In this model our $x_0$ input node is sometimes called the **"bias unit".** It is always equal to 1. In neural networks, we use the same logistic function as in classification, $\frac{1}{1+e^{-\theta^T x}}$, yet we sometimes call it a sigmoid (logistic) activation function. In this situation, our **"theta" parameters** are sometimes called **"weights".**
A simple representation looks like :

$$[\ x_0 x_1 x_2\ ] \rightarrow [\ ] \rightarrow h_\theta(x)$$

Our input nodes (layer 1), also known as the **"input layer"**, go into another node (layer 2), which finally outputs the hypothesis function, known as the **"output layer".**
We can have intermediate layers of nodes between the input and output layers called the **"hidden layers."**
In this example, we label these intermediate or **"hidden"** layer nodes $a_0^2, \cdots, a_n^2 a$ and call them **"activation units."**

1. $a_i^{(j)}$ = "activation" of unit i in layer j
2. $\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer j+1

If we had one hidden layer, it would look like :

$$\begin{bmatrix} x_0 x_1 x_2 x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} a_2^{(2)} a_3^{(2)} \end{bmatrix} \rightarrow h_\theta(x)$$

The values for each of the **"activation"** nodes is obtained as follows :

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \tag{1.52}$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \tag{1.53}$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \tag{1.54}$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}) \tag{1.55}$$

This is saying that we compute our activation nodes by using a 3×4 matrix of parameters. We apply each row of the parameters to our inputs to obtain the value for one activation node. Our hypothesis output is the logistic function applied to the sum of the values of our activation nodes, which have been multiplied by yet another parameter matrix $\Theta^{(2)}$ containing the weights for our second layer of nodes.
Each layer gets its own matrix of weights, $\Theta^{(j)}$.
The dimensions of these matrices of weights is determined as follows :
If network has $s_j$ units in layer $j$ and $s_{j+1}$ units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.
The +1 comes from the addition in $\Theta^{(j)}$ of the **"bias nodes,"** $x_0$ and $\Theta_0^{(j)}$. In other words the output nodes will not include the bias nodes while the inputs will.

### 1.2.2 Model representation II

we'll do a vectorized implementation of the above functions. We're going to define a new variable $z_k^{(j)}$ that encompasses the parameters inside our g function.

$$a_1^{(2)} = g(z_1^{(2)}) \tag{1.56}$$
$$a_2^{(2)} = g(z_2^{(2)}) \tag{1.57}$$
$$a_3^{(2)} = g(z_3^{(2)}) \tag{1.58}$$

In other words, for layer j=2 and node k, the variable z will be :

$$z_k^{(2)} = \Theta_{k,0}^{(1)} x_0 + \Theta_{k,1}^{(1)} x_1 + \cdots + \Theta_{k,n}^{(1)} x_n \tag{1.59}$$

The vector representation of x and $z^j$ is :

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \vdots \\ z_n^{(j)} \end{bmatrix}$$

Setting x = $a^{(1)}$, we can rewrite the equation as :

$$z^{(j)} = \theta^{(j-1)} a^{(j-1)} \tag{1.60}$$

We are multiplying our matrix $\Theta^{(j-1)}$ with dimensions $s_j \times$ (n+1)(where $s_j$ is the number of our activation nodes) by our vector $a^{(j-1)}$ with height (n+1). This gives us our vector $z^{(j)}$ with height $s_j$. Now we can get a vector of our activation nodes for layer j as follows :

$$a^{(j)} = g(z^{(j)}) a \tag{1.61}$$

Where our function g can be applied element-wise to our vector $z^{(j)}$
We can then add a bias unit (equal to 1) to layer j after we have computed $a^{(j)}$. This will be element $a_0^{(j)}$ and will be equal to 1. To compute our final hypothesis, let's first compute another z vector :

$$z^{(j+1)} = \Theta^{(j)} a^{(j)} \tag{1.62}$$

We get this final z vector by multiplying the next theta matrix after $\Theta^{(j-1)}$ with the values of all the activation nodes we just got. This last theta matrix $\Theta^{(j)}$ will have only one row which is multiplied by one column $a^{(j)}$ so that our result is a single number. We then get our final result with :

$$h_\Theta(x) = a^{(j+1)} = g(z^{(j+1)}) \tag{1.63}$$

Notice that in this last step, between layer j and layer j+1, we are doing exactly the same thing as we did in logistic regression. Adding all these intermediate

layers in neural networks allows us to more elegantly produce interesting and more complex non-linear hypothesis.

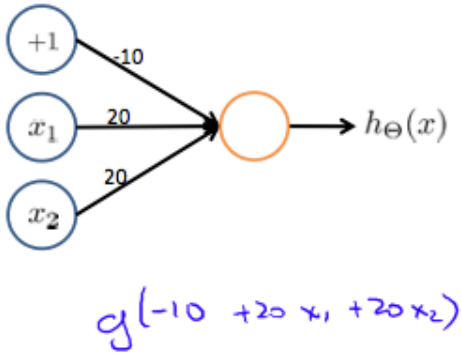
**Examples,**

A simple example of applying neural networks is by predicting $x_1$ AND $x_2$, which is the logical 'and' operator and is only true if both $x_1$ and $x_2$ are 1.

$$
\begin{aligned}
h_\Theta(x) &= g(-30 + 20x_1 + 20x_2) \\
x_1 = 0 \;\; &and \;\; x_2 = 0 \;\; then \;\; g(-30) \approx 0 \\
x_1 = 0 \;\; &and \;\; x_2 = 1 \;\; then \;\; g(-10) \approx 0 \\
x_1 = 1 \;\; &and \;\; x_2 = 0 \;\; then \;\; g(-10) \approx 0 \\
x_1 = 1 \;\; &and \;\; x_2 = 1 \;\; then \;\; g(10) \approx 1
\end{aligned}
$$

So we have constructed one of the fundamental operations in computers by using a small neural network rather than using an actual AND gate. Neural networks can also be used to simulate all the other logical gates. The following is an example of the logical operator 'OR', meaning either $x_1$ is true or $x_2$ is true, or both :

**Example: OR function**




| $x_1$ | $x_2$ | $h_\Theta(x)$ |
|---|---|---|
| 0 | 0 | g(-10) ≈ 0 |
| 0 | 1 | g(10) ≈ 1 |
| 1 | 0 | ≈ 1 |
| 1 | 1 | ≈ 1 |