Mathematical Approach of Machine Learning Algorithm II

– A Short Introduction –

Julien Lin

Department of Bioengineering | Imperial College London | 2015-2016

Part I

Introduction to

Supervised Machine Learning

Part I

Chapter 2

Classification

Classification

Classification is an approach that try to classify data by modelling relationships between a scalar dependent **discrete/categorical** output variable ***y*** given one or more independent input variable ***x*.**

I. Binary Classification: Logistic Regression

1) Example of Classification problem

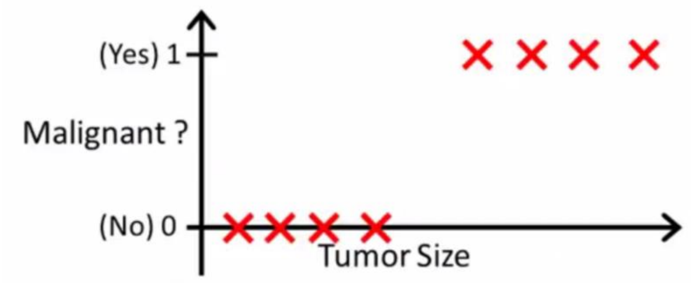
- Email: Try to classify whether the newly-received Email is a spam or not.
- Online Transaction: Try to classify whether the online transactions are fraudulent or not.
- Tumor: Try to classify whether the tumor is malignant or benign.

In all cases, the outcome variable ***y*** will have only two possible values, such as:

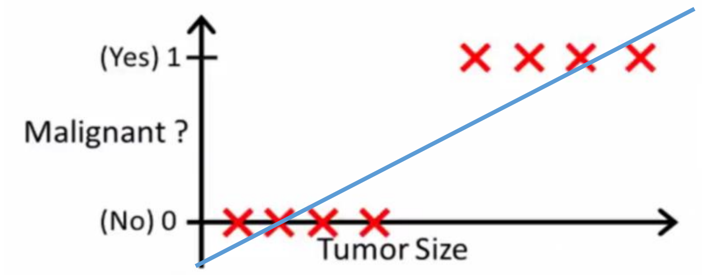
$$y = \begin{cases} \mathbf{0} : \text{"}\boldsymbol{Negative\ Class}\text{"} \ (e.g. benign\ tumor) \\ \mathbf{1} : \text{"}\boldsymbol{Positive\ Class}\text{"} \ (e.g. malignant\ tumor) \end{cases}$$

In the case of $\boldsymbol{y} \in \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$ $(in\ which\ \boldsymbol{y}\ can\ have\ more\ than\ 2\ solutions)$, this is called a **multi-class classification** problem.

Supposing that there is a set of data describing whether a tumour is malignant depending on its size such as



Issue: what will happen if a linear regression model is used to fit this dataset configuration?



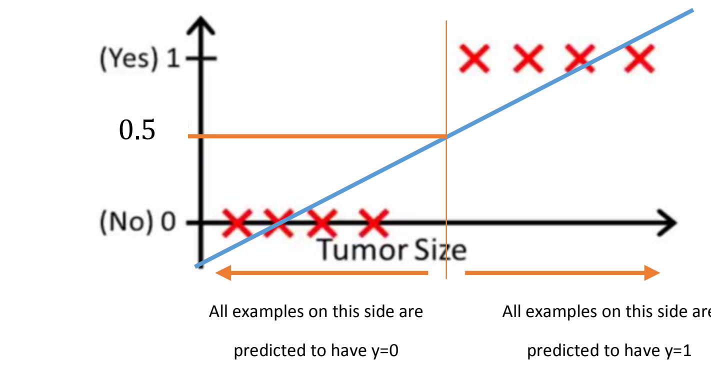
$h_\theta(\boldsymbol{x}) = \boldsymbol{\theta}^T\ \boldsymbol{x}$

Here, linear regression with a predictive function $h_\theta(\boldsymbol{x}) = \boldsymbol{\theta}^T\ \boldsymbol{x}$ is used to fit the training set. Suppose a threshold classifier output $h_\theta(x)$ at 0.5 such as

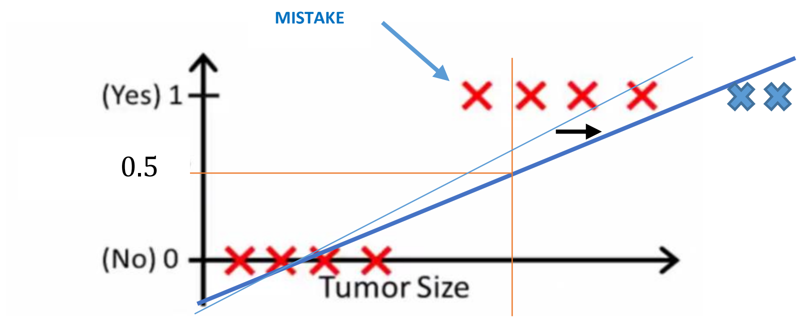
If $h_\theta(x)> 0.5$, then predict y=1 (i.e. the tumor is malignant)

If $h_\theta(x)< 0.5$, then predict y=0 (i.e. the tumor is benign)

$h_\theta(\boldsymbol{x}) = \boldsymbol{\theta}^T\ \boldsymbol{x}$



In this case, linear regression seems to fit the data very well in this classification problem. However, if linear regression is applied to another configuration such as



$h_\theta(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}$

Having two more training example on the right in this case causes linear regression to shift to the right. This result in a wrong hypothesis $h_\theta(x)$ when having a threshold at 0.5.

Thus applying linear regression to a classification problem might not be the most suitable model to fit the training data well. Indeed, in classification problem, y is either y=0 or y=1, but in linear regression $h_\theta(x)$ can output values >1 or <0 even if all training examples have labels y=0 or y=1.

The solution is to use a **classification algorithm** called **Logistic Regression** which have the property $0 \le h_\theta(x) \le 1$

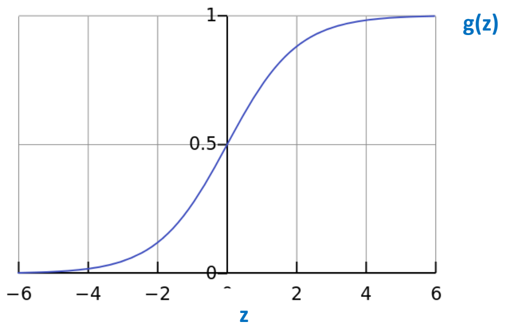
2) *Hypothesis/Predictive Function **h$_\theta$(x)** and Decision boundary*

In logistic regression, the predictive function $h_\theta(x)$= g($\boldsymbol{\theta}^T$ $\boldsymbol{x}$) with $0 \le h_\theta(x) \le 1$,

where $\boldsymbol{g(z)} = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{z}}}$ (called "sigmoid or logistic function") such as

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

Graphically represented by



With asymptotes at 0 and 1 and with y=0.5 when x=0.

Moreover, when $h_\theta(x)$ outputs some values, those values are going to be treat as the estimated probability that y=1 given an input x. For instance,

If x have two features such as $x = \begin{bmatrix} x_0 = 1 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ tumorSize \end{bmatrix}$ and suppose $h_\theta(x)$ outputs 0.7 (i.e. $h_\theta(x) = 0.7$), this can be interpreted as for a patient with features x, the probability that y=1 (i.e. malignant) is 0.7. This means that the patient have 70% chance of have a malignant tumor. Hence, $h_\theta(x)$ can be interpreted such as

$$h_\theta(x) = P(y = 1 \mid x; \theta)$$

$$\text{“}h_\theta(x) \textit{ is the probability that } y = 1, \textit{givenx, parameterized by } \theta\text{”}$$

Knowing that either y=0 or y=1,

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

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

Which means that by knowing the probability of y=1 (i.e malignant), it is thus possible to know the probability of y=0 (i.e. benign).

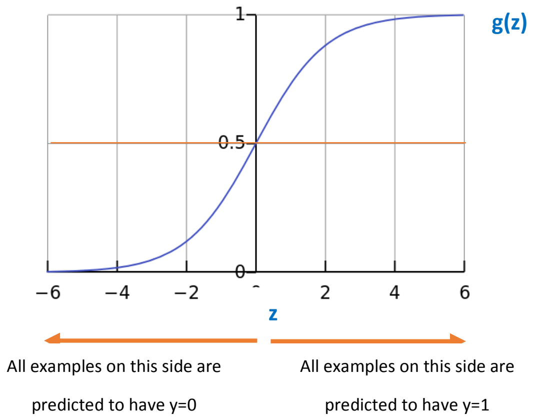
Thus, for

$$h_\theta(x) = P(y = 1 \mid x; \theta) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

Suppose

If $h_\theta(x)$> 0.5, then predict y=1

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



Thus,

$g(z) \geq 0.5$ when $z \geq 0 \Leftrightarrow h_\theta(x) = \mathrm{g}(\theta^T x) \geq 0.5\ whenever\ \ \theta^T x \geq 0$

$g(z) < 0.5$ when $z < 0 \Leftrightarrow h_\theta(x) = \mathrm{g}(\theta^T x) < 0.5\ whenever\ \ \theta^T x < 0$

<u>Now,</u>

Suppose $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ to which the parameters $\theta_0 = -3, \theta_1 = 1, and\ \theta_2 = 1$ have been defined such as $\boldsymbol{\theta} = \begin{bmatrix} \mathbf{-3} \\ \mathbf{1} \\ \mathbf{1} \end{bmatrix}$ and $z = -3 + x_1 + x_2$.

Suppose

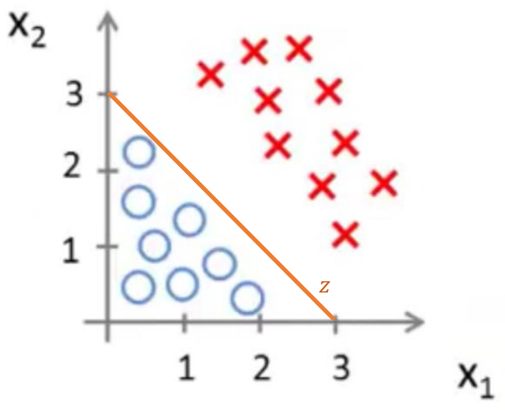
If $z = -3 + x_1 + x_2 \geq 0$, then predict the event “y=1”

If $z = -3 + x_1 + x_2 < 0$, then predict the event “y=0”

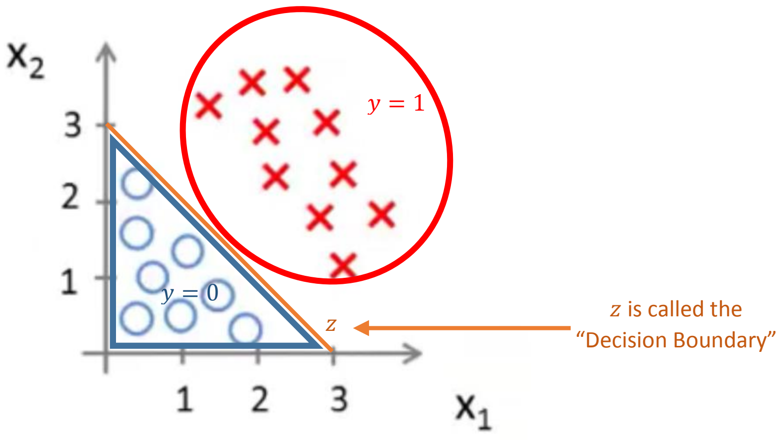
Thus, all desired elements that are wanted to be predicted with “y=1” need to be driven by the equation

$$-3 + x_1 + x_2 \geq 0 \Leftrightarrow x_1 + x_2 \geq 3$$

which is represented graphically by a straight line that pass through 3 and 3 on the x_1 and x_2:



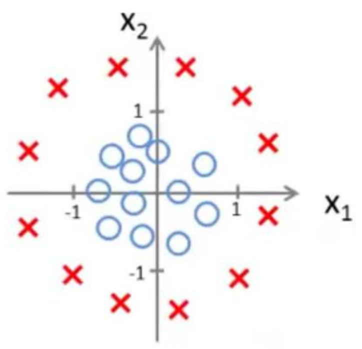
More concretely, all elements that are above this line will be predicted as “y=1” (i.e. $z = -3 + x_1 + x_2 \geq 0$) while all elements that are below this line will be predicted as “y=0” (i. e. $z = -3 + x_1 + x_2 < 0$).



The **Decision Boundary** z is defined by the equation $x_1 + x_2 = 3$ where correspond to a straight line (where $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) = 0.5$) that separate the region where the elements from the training set will be predicted as y=1 from the region where y=0. Here, z is a **Linear Decision Boundaries.**

<u>Issue:</u> What would be the case of a **Non-linear Decision Boundaries**?

Suppose the following training dataset:



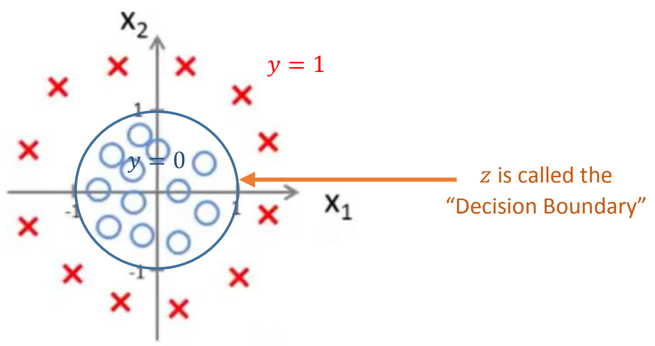
Which logistic regression to use for fitting our dataset? It is obvious here, that a polynomial function will be the best fit to define the decision boundary such as

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

With $\theta_0 = -1, \theta_1 = 0, \theta_2 = 0, \theta_3 = 1, \theta_4 = 1$; $\boldsymbol{\theta} = \begin{bmatrix} \mathbf{-1} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{1} \end{bmatrix}$,

Then, predict “y=1” if $-1 + x_1^2 + x_2^2 \geq 0 \Leftrightarrow x_1^2 + x_2^2 \geq 1$

Thus, the **Non-Linear Boundary** equation will be $x_1^2 + x_2^2 = 1$ (i.e. a circle of radius one) such as



Thus, by ending up with more complex **Non-Linear Boundary** equation such as

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1^2 x_2 + \theta_6 x_1^2 x_2^2 + \theta_6 x_1^3 + \cdots,$$

complex logistic regression model can be fitted to classify any sort of training dataset.

However, the Decision Boundary is mainly defined by the value of the parameters $\boldsymbol{\theta}$.

Issue: How do we define the parameters $\boldsymbol{\theta}$?

3) *Cost Function* **J(θ₀,θ₁)**

Knowing for Logistic Regression that

Training set: $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)} \ldots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$ with *m* the total number of example,

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \in R^{n=1} \; with \;\; x_0 = 1$$

$y \in \{0,1\}$, and $h_\theta(x) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\theta^T x}}$

The parameters $\boldsymbol{\theta}$ are chosen by minimizing J(**θ**).

From Linear Regression, $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}\,(h_\theta(x^{(i)}) - y^{(i)})^2$. In this equation, $\frac{1}{2}\,(h_\theta(x^{(i)}) - y^{(i)})^2$ is going to be defined as $cost(h_\theta(x^{(i)}), y^{(i)})$ such as

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

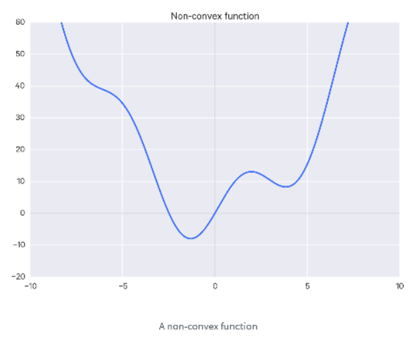
Or more simply,

$$\frac{1}{2}\,(h_\theta(x) - \mathrm{y})^2 = cost(h_\theta(\mathrm{x}), \mathrm{y})$$

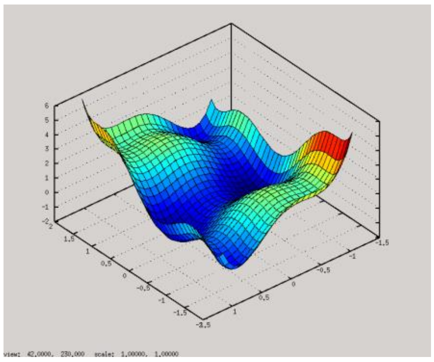
Thus,

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

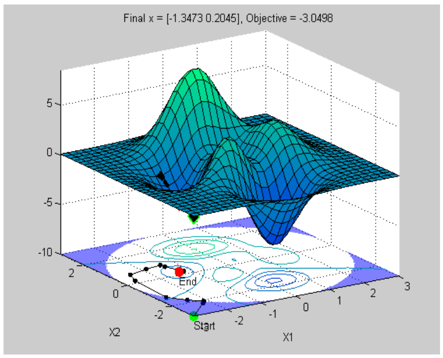
Since $h_\theta(x) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\theta^T x}}$ in logistic regression, $J(\boldsymbol{\theta})$ function will a **non-convex** function, which means that it will admit a global minimum, as well as several local minima potentially compared to Linear Regression such as



2D non − convex function



3D non − convex function



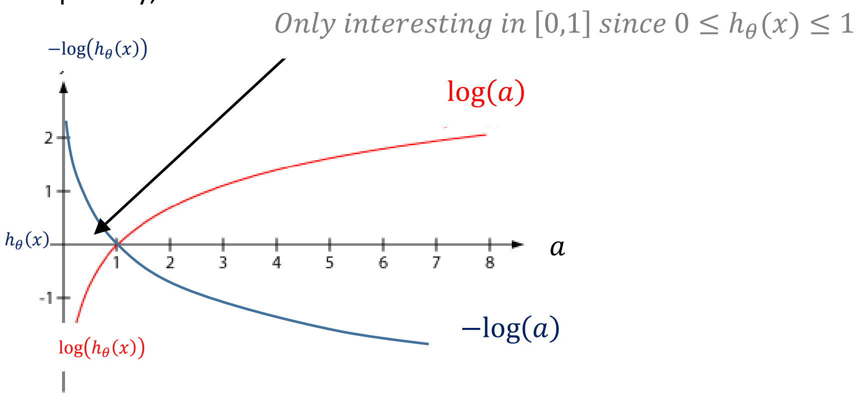
3D non − convex function

Hence, when applying gradient descent, the algorithm will not guarantee to find the global minimum but might end up in a local minimum instead.

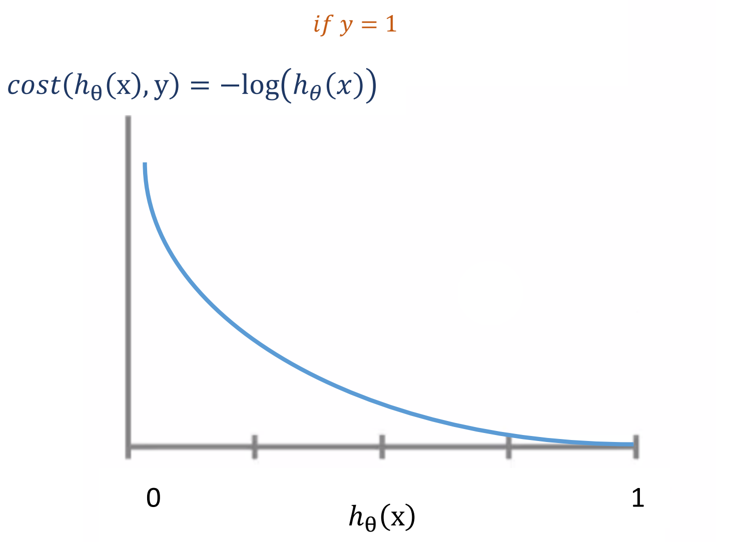
To solve this problem in logistic regression, $cost(h_\theta(x), y)$ is going to be defined as

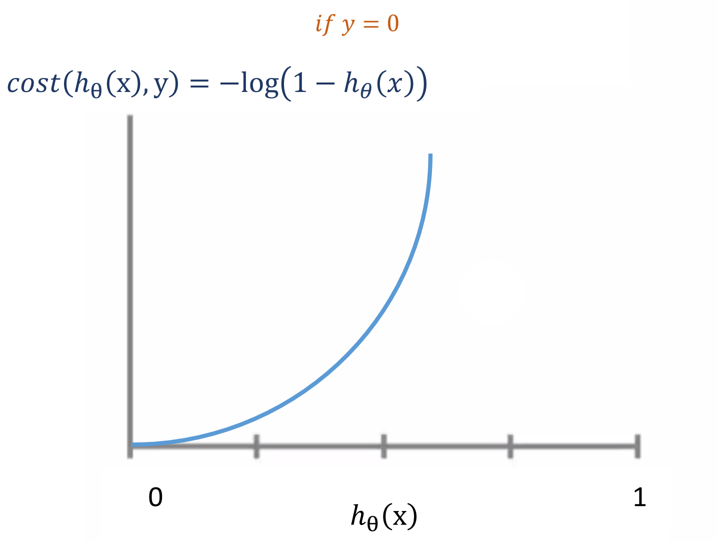
$$cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) \; if \; y = 1 \\ -\log(1 - h_\theta(x)) \; if \; y = 0 \end{cases}$$

Graphically,



$Since\ 0 \le h_\theta(x) \le 1,$





Graphical observations:

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

while $h_\theta(x) \to 0$, cost($h_\theta(x), y$) $\to \infty$.

Interpretation:

- If *the probability* $h_\theta(x) = 1$ (i.e. P(y=1|x; θ)=1) while y=1, this means that the output has been predicted **correctly**, then the cost would be 0 (i.e. **cost($h_\theta(x), y$)=0** ; no error in the prediction)
- If *the probability* $h_\theta(x) = 0$ (i.e. P(y=1|x; θ)=0) while y=1, this means that if the chance of having y=1 is equal to zero and it is **wrong** (e.g. explaining to a patient that his chance of having a malignant tumor is absolutely null but it turns out to be wrong), the learning algorithm will be penalized by a very large cost (i.e. **cost($h_\theta(x), y$)** $\to \infty$).

Graphical observations:

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

while $h_\theta(x) \to 1$, cost($h_\theta(x), y$) $\to \infty$.

Interpretation:

I. If *the probability* $h_\theta(x) = 0$ (i.e. P(y=0|x; θ)=0) while y=0, this means that the output that was supposed to be false has been predicted **correctly to be false**, then the cost would be 0 (i.e. **cost($h_\theta(x), y$)=0** ; no error in the prediction)

II. If *the probability* $h_\theta(x) = 1$ (i.e. P(y=0|x; θ)=1) while y=0, this means that if the chance of having y=0 is equal to 1 and it is actually **wrong** (e.g. explaining to a patient that the tumor that he will have will always be benign but it turns out to be wrong), the learning algorithm will be penalized by a very large cost (i.e. **cost($h_\theta(x), y$)** $\to \infty$).

Now,

Knowing that

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

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

The cost function $J(\boldsymbol{\theta})$ can be write out in a single equation with $cost(h_\theta(\mathrm{x}), \mathrm{y})$ in both case where y=1 and y=0:

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

$$cost(h_\theta(\mathrm{x}), \mathrm{y}) = \underbrace{\boxed{-y\log\left(h_\theta(x)\right)}}_{\boxed{for \; y = 1}}\underbrace{\boxed{- (1-y)\log\left(1 - h_\theta(x)\right)}}_{\boxed{for \; y = 0}}$$

If y=1

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

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

$$\boxed{cost(h_\theta(\mathrm{x}), \mathrm{y}) = -\log\left(h_\theta(x)\right)}$$

If y=0,

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

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

$$\boxed{cost(h_\theta(\mathrm{x}), \mathrm{y}) = -\log\left(1 - h_\theta(x)\right)}$$

This compressed equation will enable to derive the cost function $J(\boldsymbol{\theta})$ more easily when running gradient descent.

Thus,

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

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

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

With

$$h_\theta(x) = P(y = 1 \mid x; \theta) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

Then, the **optimization objective** will be to $minimize_{\boldsymbol{\theta}} \quad J(\boldsymbol{\theta})$ in order to get the optimal $\boldsymbol{\theta}$

4) *Gradient Descent*

Knowing that

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

With

$$h_\theta(x) = P(y=1 \mid x;\theta) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

The **optimization objective** for Logistic Regression will be to $minimize_{\boldsymbol{\theta}} \quad J(\boldsymbol{\theta})$ by using Gradient Descent Algorithm such as:

Repeat {

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

} (simultaneously update all $\boldsymbol{\theta_j}$)

Where

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

Thus,

$$\frac{\boldsymbol{\partial}}{\boldsymbol{\partial\theta_j}}\boldsymbol{J}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - \mathrm{y}^{(i)})x_j^{(i)}$$

Hence,

Repeat {

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

} (simultaneously update all $\boldsymbol{\theta_j}$)

With $\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta_0} \\ \boldsymbol{\theta_1} \\ \boldsymbol{\theta_2} \\ \cdots \\ \boldsymbol{\theta_n} \end{bmatrix}$

N.B. The Gradient Descent Equation for Logistics Regression is the same as in Linear Regression, except that in Logistic Regression, $h_\theta(\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$ whereas in Linear Regression, $h_\theta(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}$**.**

Then, the next step will be to plot $J(\boldsymbol{\theta})$ over the number of iterations that Gradient Descent run is used to monitor whether the algorithm is converging correctly or not (correct behaviour of $J(\boldsymbol{\theta})$ would be that it decreases after each step of Gradient Descent).

Although Gradient Descent Algorithm works well in large dataset, advance optimization algorithm enable can also be used for Logistic Regression Model to run more efficiently on a very large scale (e.g. very large amount of features). Advance Optimization such as Conjugate gradient, BFGS, and L-BFGS have the Advantages compared to Gradient Descent to be often faster than gradient descent while being more complex and there is no need to manually pick **.**

Conjugate gradient: https://en.wikipedia.org/wiki/Conjugate_gradient_method

BFGS: https://www.rtmath.net/help/html/9ba786fe-9b40-47fb-a64c-c3a492812581.htm

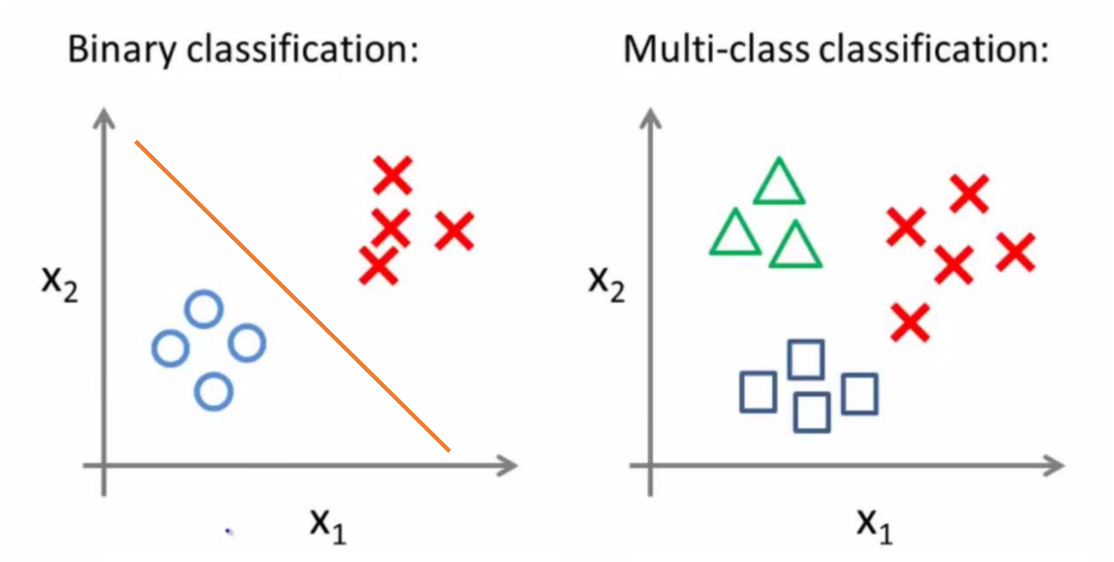
L-BFGS: https://en.wikipedia.org/wiki/Limited-memory_BFGS

II. Multiclass Classification: One-vs-All Classification

1) *Example of Classification problem*

- Email tagging/foldering: Work, Friends, Family, Hobby.
  In this examples, y=1 is assigned to *Work*, y=2 is assigned to *Friends,* y=3 is assigned to *Family,* y=1 is assigned to *Hobby.*
- Medical diagnosis: Not ill, Cold, Flu.
  In this examples, y=1 is assigned to *Not ill*, y=2 is assigned to *Cold,* y=3 is assigned to *Flu*
- Weather: Sunny, Cloudy, Rain, Snow.
  In this examples, y=1 is assigned to *Sunny*, y=2 is assigned to *Cloudy,* y=3 is assigned to *Rain,* y=4 is assigned to *Snow*

Compared to Logistic Regression which allow only a binary outcome such as y=1 or y=0, Multiclass classification has more than 2 outcomes such as $y \in \{2,3,\ldots,n\}$:



Examples that can be separated into 2 classes

Examples that can be separated into 3 classes

Issue: Knowing that in Binary classification that use a straight line to separate two classes, How does a learning algorithm learn to recognize more than 2 classes?

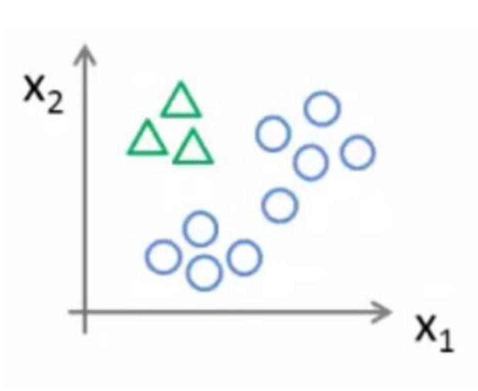
The solution is to use an algorithm that is call "One-vs-All Classification"

2) *Hypothesis/Predictive Function, Cost Function, and Gradient Descent*

Knowing that △ is defined as the class 1, □ as the class 2, and ✖ as the class 3,

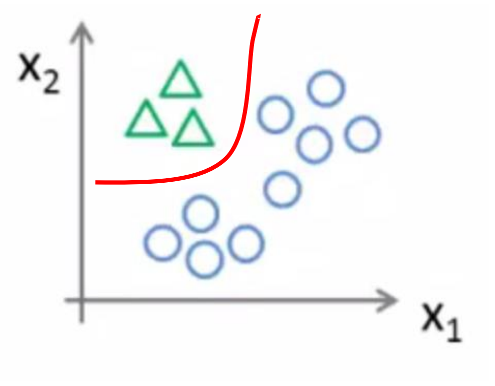
The strategy will be to turn the training set into three separate binary classification problems such as,

For △,

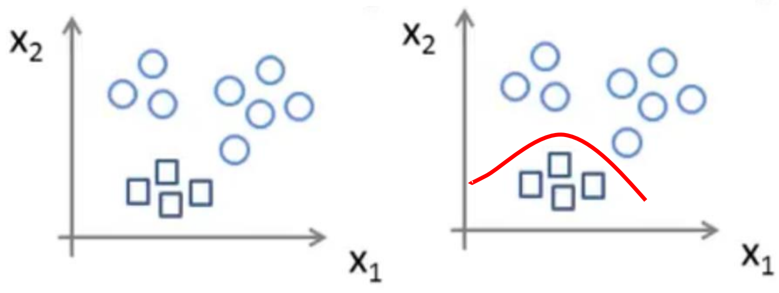


The training set is separate into 2 classes where class 2 (□) and 3 (✖) are going to be assigned as the negative class and class 1 (△) as the positive class.

Then, the next step is to train a simple logistic regression classifier $h_\theta^{(1)}(x)$ (i.e. predictive function) to get a decision boundary such as:

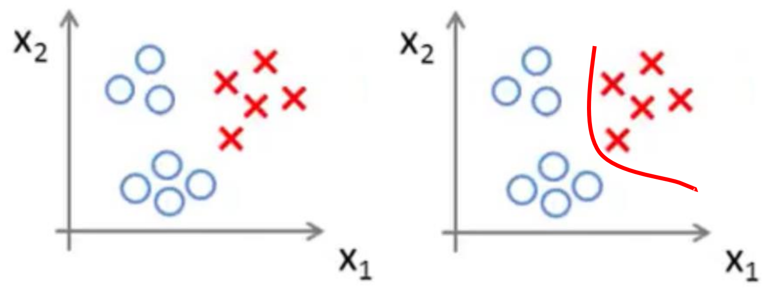


Same principle is applied to class 2 (□) and 3 (✖),



The training set is separate into 2 classes where class 1 (△) and 3 (✖) are going to be assigned as the negative class and class 2 (□) as the positive class.

Then, a simple logistic regression classifier $h_\theta^{(2)}(x)$ (i.e. predictive function) is trained in order to get a decision boundary.



The training set is separate into 2 classes where class 1 (△) and 2 (□) are going to be assigned as the negative class and class 3 (✖) as the positive class

Then, a simple logistic regression classifier $h_\theta^{(3)}(x)$ (i.e. predictive function) is trained in order to get a decision boundary.

Thus, once the three classifiers are fitted such as

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

The objective will be to train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i in order to predict the probability that y is equal to class i, given x, parameterized by θ such as:

$$h_\theta^{(1)}(x) = P(y = 1|x;\theta)$$

$$h_\theta^{(2)}(x) = P(y = 2|x;\theta)$$

$$h_\theta^{(3)}(x) = P(y = 3x;\theta)$$

Finally, to classify or make a prediction for a new input x, all three of the classifier are run on this new input x and then the class I that maximizes the three is picked (i.e. select the classifier that seems the most confident by giving the higher probability) such as:

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

By selecting the classifier that have the highest probability, its value y can be predicted, indicating the class to which it belong.

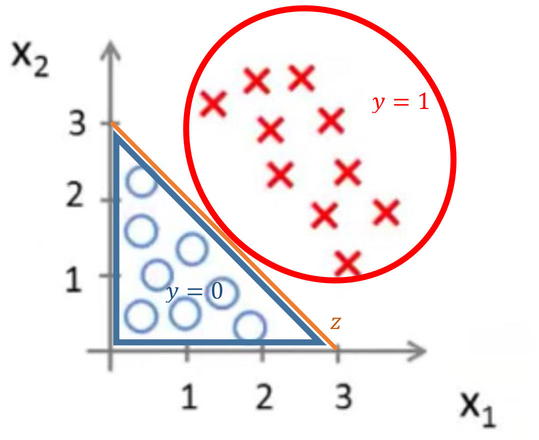
III. Neural Network

1) Examples with house classification problem

Issue: Knowing that Neural Networks is a classification algorithm, when should it be used instead of logistic regression?

Here, the classification problem will be to classify all the houses that will be sold out in the next 6 months.

Logistic regression works well when there are only two features (x_2 is plotted against x_1) in the polynomial term,



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

With n=2

But, what if instead of having only $x = \begin{bmatrix} x_1: size \\ x_2: number\ of\ bedrooms \end{bmatrix}$ but more than two features such as

$x = \begin{bmatrix} x_1: size \\ x_2: number\ of\ bedrooms \\ x_3: number\ of\ floors \\ x_4: age\ of\ the\ house \\ \dots \\ x_{100} \end{bmatrix}$ with n=100, the number of total original features. Moreover, if all the **quadratic** terms are included to drawn the decision boundary z, there will be a lot of additional features such as

$$\begin{bmatrix} x_1^2 & x_1 x_2 & x_1 x_2 & \dots & x_1 x_{100} \\ & x_2^2 & x_2 x_3 & \dots & x_2 x_{100} \\ & & & \dots & \dots \\ & & & \dots & x_{100}^2 \end{bmatrix}$$

Which results in end up with ~ 5000 features. If the number of original features n=100, the number of quadratic features grows approximately as order n,

$$O(n^2) = \frac{n^2}{2}$$

What if all the **cubic** terms were added such as the total number of features will rise to $O(n^3)$

What if all the **polynomia**l terms were added such as the total number of feature will rise to $O\left(n^d\right)$

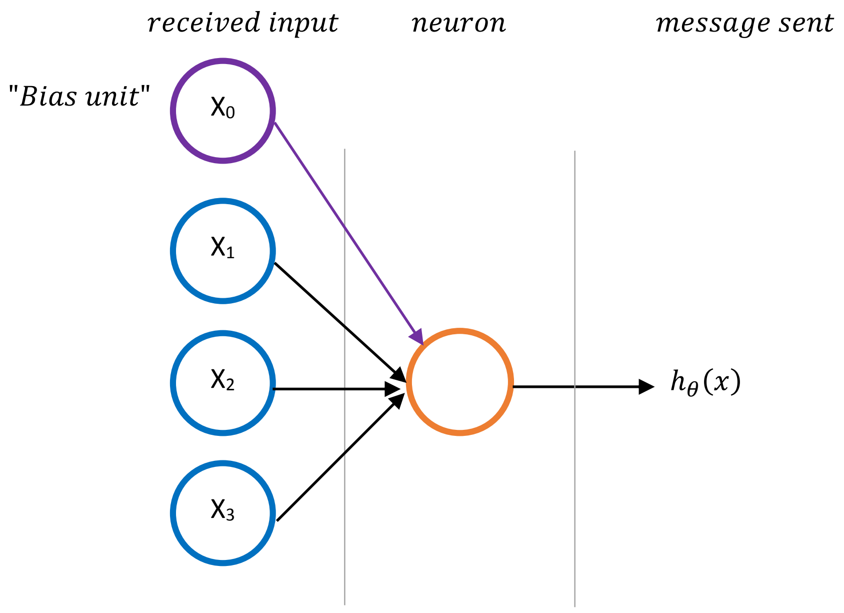
Adding such a huge number of polynomial features to the hypothesis function might is computationally expensive and might result in overfitting.

A simple logistic regression is not appropriate to learning a complex non-linear hypotheses due to the large number of additional features

One cost-effective solution will be to use neural networks.

2) Model Representation and hypothesis function $h_\Theta(x)$

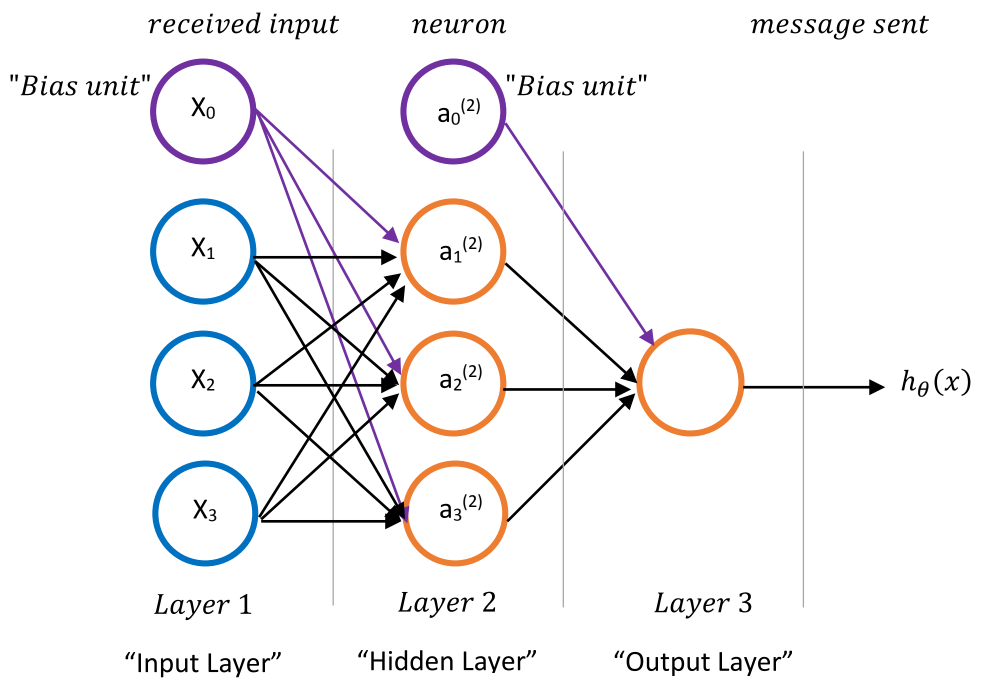
Neural network is an algorithm that tend to mimic brain's function. Similar to a neuron is biology which receive information, process, and sent messages to other neurons via its axon, a **single** neuron is modelled as a **logistic units** such as:



Where $h_\theta(\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$ is a Sigmoid or Logistic activation function, $\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta_0} \\ \boldsymbol{\theta_1} \\ \boldsymbol{\theta_2} \\ \cdots \\ \boldsymbol{\theta_n} \end{bmatrix}$ are the parameters or the "***weights***",

$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_0} \\ \boldsymbol{x_1} \\ \cdots \\ \boldsymbol{x_n} \end{bmatrix}$ the input received from several other neurons. $\boldsymbol{x_0}$ is known as a bias unit, characterised by $\boldsymbol{x_0} = 1$.

For a **neural network,**



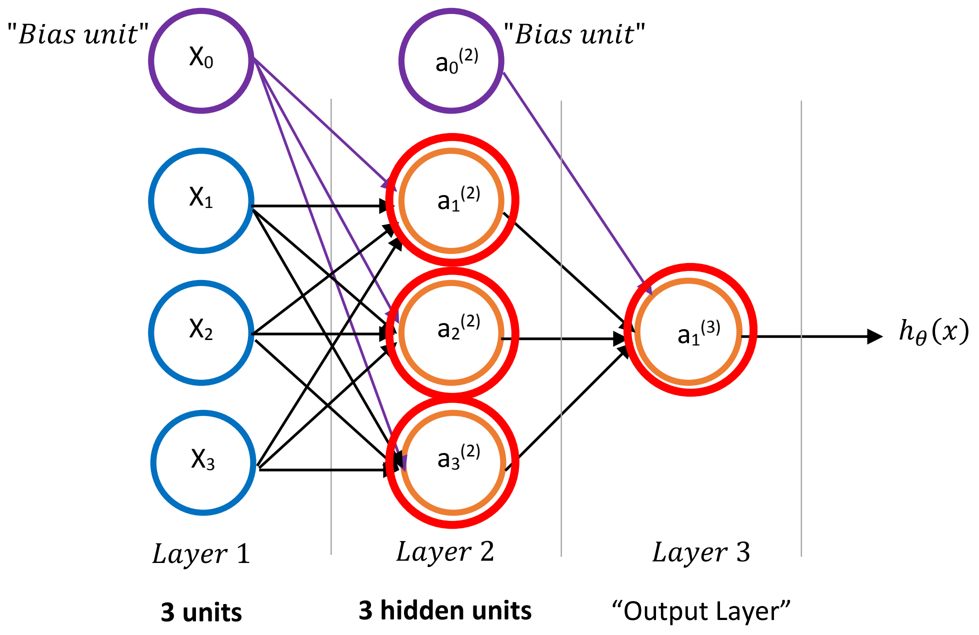
Where $a_1^{(2)}$, $a_2^{(2)}$, $a_3^{(2)}$ are neurons in a so-called "Hidden Layer". The term "Hidden" refers to the fact that this layer is not seen when applying this algorithm to the training set.

In more details,

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

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

For $a_1^{(2)}$, $a_2^{(2)}$, $a_3^{(2)}$



$$a_1^{(2)} = g(\Theta_{1,0}^{(1)}x_0 + \Theta_{1,1}^{(1)}x_1 + \Theta_{1,2}^{(1)}x_2 + \Theta_{1,3}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{2,0}^{(1)}x_0 + \Theta_{2,1}^{(1)}x_1 + \Theta_{2,2}^{(1)}x_2 + \Theta_{2,3}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{3,0}^{(1)}x_0 + \Theta_{3,1}^{(1)}x_1 + \Theta_{3,2}^{(1)}x_2 + \Theta_{3,3}^{(1)}x_3)$$

With $\Theta^{(1)}$ is a 3x4 matrix dimension. If network has s_j units in layer j, s_{j+1} units in layer j+1, then $\Theta^{(j)}$ will be of dimension $\boldsymbol{s_{j+1} * (s_j + 1)}$

Thus,

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{1,0}^{(2)}a_0^{(2)} + \Theta_{1,1}^{(2)}a_1^{(2)} + \Theta_{1,2}^{(2)}a_2^{(2)} + \Theta_{1,3}^{(2)}a_3^{(2)})$$

$h_\Theta(x) = g(\Theta_{1,0}^{(2)}a_0^{(2)} + \Theta_{1,1}^{(2)}a_1^{(2)} + \Theta_{1,2}^{(2)}a_2^{(2)} + \Theta_{1,3}^{(2)}a_3^{(2)})$ looks similar to a logistic regression, except that the features $a_i^{(j)}$ are from a hidden layers instead of the input features x_i. The features $a_i^{(j)}$ are themselves are learned as functions of the input.

3) *Forward propagation: Vectorized implementation*

Knowing that

$$a_{\mathbf{1}}^{(2)} = g(\Theta_{\mathbf{1},0}^{(1)}x_0 + \Theta_{\mathbf{1},1}^{(1)}x_1 + \Theta_{\mathbf{1},2}^{(1)}x_2 + \Theta_{\mathbf{1},3}^{(1)}x_3)$$

$$a_{\mathbf{2}}^{(2)} = g(\Theta_{\mathbf{2},0}^{(1)}x_0 + \Theta_{\mathbf{2},1}^{(1)}x_1 + \Theta_{\mathbf{2},2}^{(1)}x_2 + \Theta_{\mathbf{2},3}^{(1)}x_3)$$

$$a_{\mathbf{3}}^{(2)} = g(\Theta_{\mathbf{3},0}^{(1)}x_0 + \Theta_{\mathbf{3},1}^{(1)}x_1 + \Theta_{\mathbf{3},2}^{(1)}x_2 + \Theta_{\mathbf{3},3}^{(1)}x_3)$$

And

$$z_{\mathbf{1}}^{(2)} = \Theta_{\mathbf{1},0}^{(1)}x_0 + \Theta_{\mathbf{1},1}^{(1)}x_1 + \Theta_{\mathbf{1},2}^{(1)}x_2 + \Theta_{\mathbf{1},3}^{(1)}x_3$$

$$z_{\mathbf{2}}^{(2)} = \Theta_{\mathbf{2},0}^{(1)}x_0 + \Theta_{\mathbf{2},1}^{(1)}x_1 + \Theta_{\mathbf{2},2}^{(1)}x_2 + \Theta_{\mathbf{2},3}^{(1)}x_3$$

$$z_{\mathbf{3}}^{(2)} = \Theta_{\mathbf{3},0}^{(1)}x_0 + \Theta_{\mathbf{3},1}^{(1)}x_1 + \Theta_{\mathbf{3},2}^{(1)}x_2 + \Theta_{\mathbf{3},3}^{(1)}x_3$$

such as

$$a_{\mathbf{1}}^{(2)} = g(z_{\mathbf{1}}^{(2)})$$

$$a_{\mathbf{2}}^{(2)} = g(z_{\mathbf{2}}^{(2)})$$

$$a_{\mathbf{3}}^{(2)} = g(z_{\mathbf{3}}^{(2)})$$

In this case $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_0} \\ \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \boldsymbol{x_3} \end{bmatrix}$ **and** $\boldsymbol{z}^{(2)} = \begin{bmatrix} z_{\mathbf{1}}^{(2)} \\ z_{\mathbf{2}}^{(2)} \\ z_{\mathbf{3}}^{(2)} \end{bmatrix}$.

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

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

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

$$\therefore \boldsymbol{a}^{(2)} = g(\Theta^{(1)}\boldsymbol{a}^{(1)})$$

Taking in account the bias unit $a_{\mathbf{0}}^{(2)} = 1, \boldsymbol{a}^{(2)} \in \boldsymbol{R}^{4}$

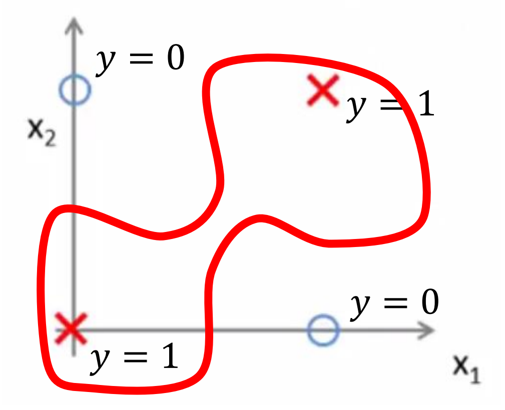
$$\boldsymbol{z}^{(3)} = \Theta^{(2)}\boldsymbol{a}^{(2)}.$$

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

The process of computing $h_\theta(x)$ is called **Forward Propagation**. Forward propagation starts with the activations of the input-units before “forward-propagating” them to the hidden layer and compute the activations of this hidden layers before “forward-propagating” them to the output layer and compute the activation of this output layer

4) *Neural network classification example*

Considering X₁ , X₂ are binary (0 or 1) coordinate such as (X₁ , X₂)



How does neural network predict y=1 or y=0 to each point of coordinate (0, 0), (1, 0), (0, 1), (1, 1)?

The solution would be to classify such as

- Y= x_1 XOR x_2 , which means that x_1 , x_2 are true only if one example of x_1 or x_2 are equal to 1.
- Y= x_1 XNOR x_2= NOT(x_1 XOR x_2), which means that x_1 , x_2 are true only if one example of x_1 or x_2 are equal to 0

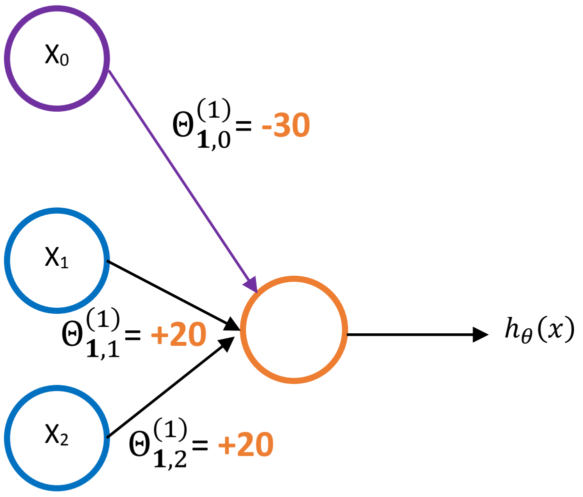
This means $h_\theta(x) = 1$ when either both coordinate x_1 $\boldsymbol{AND}$ x_2 are 0 , ***OR*** $h_\theta(x) = 1$ when both x_1 $\boldsymbol{AND}$ x_2 are 1.

Example: AND

$x_1, x_2 \in \{\mathbf{0}, \mathbf{1}\}$

$y = x_1 \; AND \; x_2$ would be equal to 1 if and only if $x_1 = x_2 = 1$

Suppose $\Theta^{(1)}_{\mathbf{1},0}$= **-30** , $\Theta^{(1)}_{\mathbf{1},1}$= **+20**, $\Theta^{(1)}_{\mathbf{1},2}$= **+20** such as



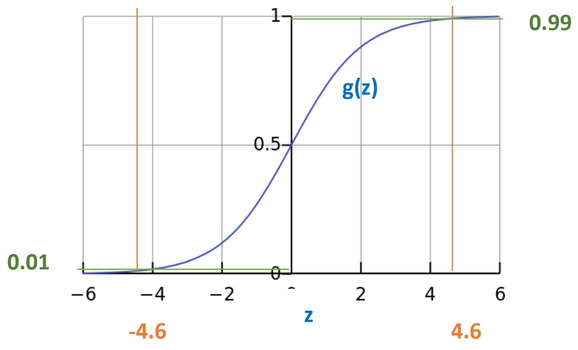
X₀
$\Theta^{(1)}_{\mathbf{1},0}$= -30
X₁
$\Theta^{(1)}_{\mathbf{1},1}$= +20
X₂
$\Theta^{(1)}_{\mathbf{1},2}$= +20
$h_\theta(x)$

Thus,

$$h_\theta(x) = \boldsymbol{g}(-\mathbf{30} + \mathbf{20}x_1 + \mathbf{20}\, x_2)$$

Issue: What is the logical function that the neural function compute? Is it possible to have a one-unit network to compute this logical AND function?

Knowing that



IF both $x_1 = 0\ AND\ x_2 = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(-30 + 20 * 0 + 20 * 0) = \boldsymbol{g}(-30)$

Graphically, $\boldsymbol{g}(-30) \approx \mathbf{0}$ **,** $h_\theta(x) \approx \mathbf{0}$

IF both $x_1 = 0\ AND\ x_2 = 1$, **THEN** $h_\theta(x) = \boldsymbol{g}(-30 + 20 * 0 + 20 * 1) = \boldsymbol{g}(-10)$

Graphically, $\boldsymbol{g}(-10) \approx \mathbf{0}$ **,** $h_\theta(x) \approx \mathbf{0}$

IF both $x_1 = 1\ AND\ x_2 = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(-30 + 20 * 1 + 20 * 0) = \boldsymbol{g}(-10)$

Graphically, $\boldsymbol{g}(-10) \approx \mathbf{0}$ **,** $h_\theta(x) \approx \mathbf{0}$

IF both $x_1 = 1\ AND\ x_2 = 1$, **THEN** $h_\theta(x) = \boldsymbol{g}(-30 + 20 * 1 + 20 * 1) = \boldsymbol{g}(+10)$

Graphically, $\boldsymbol{g}(+10) \approx \mathbf{1}$ **,** $h_\theta(x) \approx \mathbf{1}$

Summary,

x_1	x_2	$h_\theta(x)$
0	0	0
0	1	0
1	0	0
1	1	1

AND Logical function

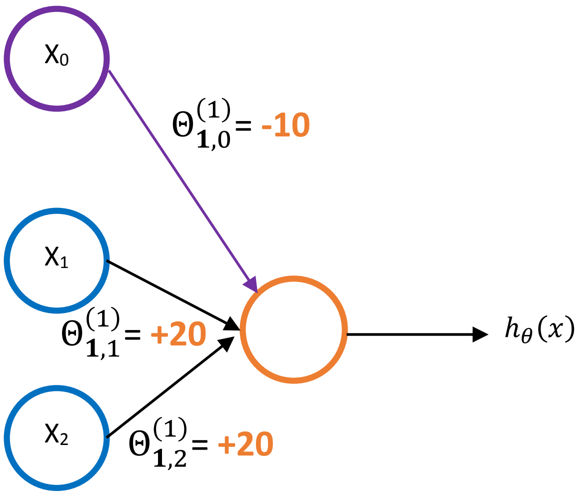
Thus, $h_\theta(x) \approx x_1\ \boldsymbol{AND}\ x_2$

Example: OR

$x_1, x_2 \in \{\mathbf{0}, \mathbf{1}\}$

$y = x_1\ OR\ x_2$ would be equal to 1 if and only if $x_1 or\ x_2 = 1$

Suppose $\Theta^{(1)}_{\mathbf{1},0}$= **-10** , $\Theta^{(1)}_{\mathbf{1},1}$= **+20**, $\Theta^{(1)}_{\mathbf{1},2}$= **+20** such as



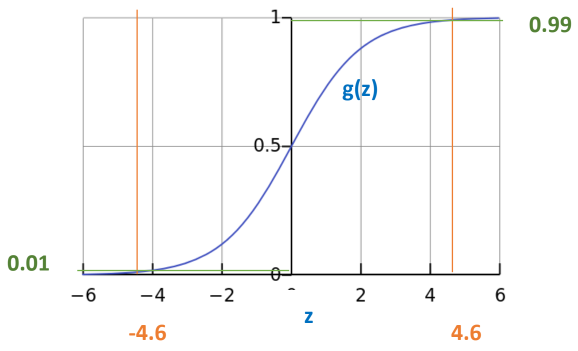
X_0
X_1
X_2
$\Theta^{(1)}_{1,0}$ = -10
$\Theta^{(1)}_{1,1}$ = +20
$\Theta^{(1)}_{1,2}$ = +20
$h_\theta(x)$

Thus,

$$h_\theta(x) = \boldsymbol{g}(-10 + 20x_1 + 20\,x_2)$$

Issue: What is the logical function that the neural function compute? Is it possible to have a one-unit network to compute this logical OR function?

Knowing that



IF both $x_1 = 0\ AND\ x_2 = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(-10 + 20 * 0 + 20 * 0) = \boldsymbol{g}(-10)$

Graphically, $\boldsymbol{g}(-10) \approx \mathbf{0}$ **,** $h_\theta(x) \approx \mathbf{0}$

IF both $x_1 = 0\ AND\ x_2 = 1$, **THEN** $h_\theta(x) = \boldsymbol{g}(-10 + 20 * 0 + 20 * 1) = \boldsymbol{g}(10)$

Graphically, $\boldsymbol{g}(10) \approx \mathbf{1}$ **,** $h_\theta(x) \approx \mathbf{1}$

IF both $x_1 = 1\ AND\ x_2 = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(-10 + 20 * 1 + 20 * 0) = \boldsymbol{g}(10)$

Graphically, $\boldsymbol{g}(10) \approx \mathbf{1}$**,** $h_\theta(x) \approx \mathbf{1}$

IF both $x_1 = 1\ AND\ x_2 = 1$, **THEN** $h_\theta(x) = \boldsymbol{g}(-10 + 20 * 1 + 20 * 1) = \boldsymbol{g}(+30)$

Graphically, $\boldsymbol{g}(+30) \approx \mathbf{1}$ **,** $h_\theta(x) \approx \mathbf{1}$

Summary,

x_1	x_2	$h_\theta(x)$
0	0	0
0	1	1
1	0	1
1	1	1

OR Logical function

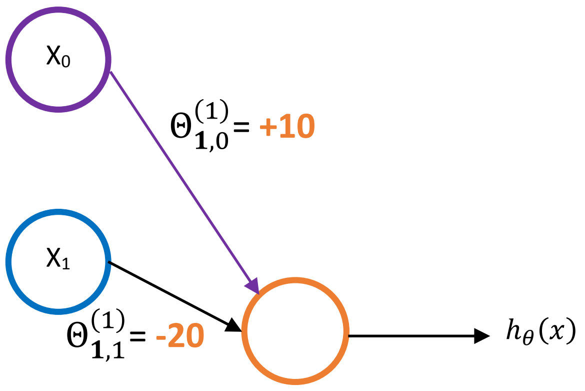
Thus, $h_\theta(x) \approx x_1\ \boldsymbol{OR}\ x_2$

Example: NOT

$x_1 \in \{\mathbf{0}, \mathbf{1}\}$

$y = \boldsymbol{NOT}\ x_1$ would be equal to 1 if and only if $x_1 = 0$

Suppose $\Theta^{(1)}_{\mathbf{1},0}$ = **+10** , $\Theta^{(1)}_{\mathbf{1},1}$ = **-20**, such as

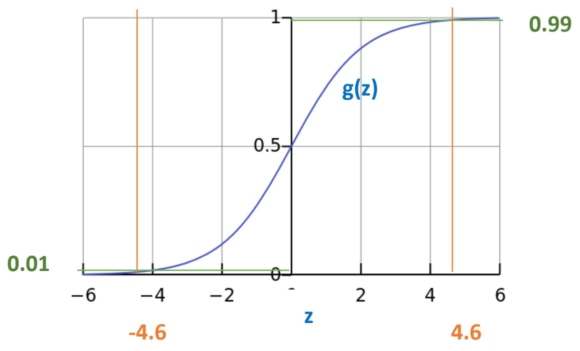

X₀
X₁
$\Theta^{(1)}_{\mathbf{1},0}$ = +10
$\Theta^{(1)}_{\mathbf{1},1}$ = -20
$h_\theta(x)$

Thus,

$$h_\theta(x) = \boldsymbol{g}(\mathbf{10} - \mathbf{20}x_1)$$

Issue: What is the logical function that the neural function compute? Is it possible to have a one-unit network to compute this logical NOT function?

Knowing that



IF $x_1 = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(+\mathbf{10} - \mathbf{20} * 0) = \boldsymbol{g}(\mathbf{10})$

Graphically, $\boldsymbol{g}(\mathbf{10}) \approx \mathbf{1}$ **,** $h_\theta(x) \approx \mathbf{1}$

IF $x_1 = 1$ **, THEN** $h_\theta(x) = \boldsymbol{g}(+\mathbf{10} - \mathbf{20} * 1) = \boldsymbol{g}(-\mathbf{10})$

Graphically, $\boldsymbol{g}(-\mathbf{10}) \approx \mathbf{0}$ **,** $h_\theta(x) \approx \mathbf{0}$

Summary,

x_1	$h_\theta(x)$
0	1
1	0

NOT Logical function

Thus, $h_\theta(x) \approx \boldsymbol{NOT}\ x_1$

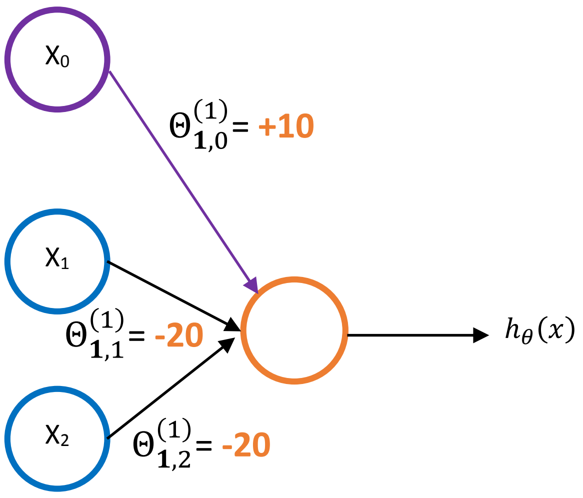
The only way to end up negative is to put a very large weight to x_1.

Example: NOT (AND)

$x_1, x_2 \in \{\mathbf{0}, \mathbf{1}\}$

$y = NOT\ (x_1\ AND\ x_2) = NOT(x_1)\ AND\ NOT(x_2)$ would be equal to 1 if and only if $x_1 = x_2 = 0$

Suppose $\Theta^{(1)}_{\mathbf{1},0}$= 10 , $\Theta^{(1)}_{\mathbf{1},1}$= -20, $\Theta^{(1)}_{\mathbf{1},2}$= -20 such as



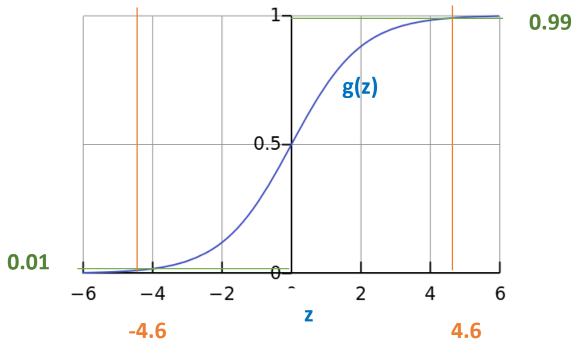
X₀
$\Theta^{(1)}_{\mathbf{1},0}$= +10
X₁
$\Theta^{(1)}_{\mathbf{1},1}$= -20
X₂
$\Theta^{(1)}_{\mathbf{1},2}$= -20
$h_\theta(x)$

Thus,

$$h_\theta(x) = \boldsymbol{g}(\mathbf{10} - \mathbf{20}x_1 - \mathbf{20}\ x_2)$$

Issue: What is the logical function that the neural function compute? Is it possible to have a one-unit network to compute this logical NOT function?

Knowing that



IF both $x_1 = 0\ AND\ x_2 = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(\mathbf{10} - \mathbf{20} * 0 - \mathbf{20} * 0) = \boldsymbol{g}(\mathbf{10})$

Graphically, $\boldsymbol{g}(\mathbf{10}) \approx \mathbf{1}$ **,** $h_\theta(x) \approx \mathbf{1}$

IF both $x_1 = 0\ AND\ x_2 = 1$, **THEN** $h_\theta(x) = \boldsymbol{g}(\mathbf{10} - \mathbf{20} * 0 - \mathbf{20} * 1) = \boldsymbol{g}(-\mathbf{10})$

Graphically, $\boldsymbol{g}(-\mathbf{10}) \approx \mathbf{0}$ **,** $h_\theta(x) \approx \mathbf{0}$

IF both $x_1 = 1\ AND\ x_2 = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(\mathbf{10} - \mathbf{20} * 1 - \mathbf{20} * 0) = \boldsymbol{g}(-\mathbf{10})$

Graphically, $\boldsymbol{g}(-\mathbf{10}) \approx \mathbf{0}$**,** $h_\theta(x) \approx \mathbf{0}$

IF both $x_1 = 1\ AND\ x_2 = 1$, **THEN** $h_\theta(x) = \boldsymbol{g}(\mathbf{10} - \mathbf{20} * 1 - \mathbf{20} * 1) = \boldsymbol{g}(-\mathbf{30})$

Graphically, $\boldsymbol{g}(-\mathbf{30}) \approx \mathbf{0}$ **,** $h_\theta(x) \approx \mathbf{0}$

Summary,

x_1	x_2	$h_\theta(x)$
0	0	1
0	1	0
1	0	0
1	1	0

NOT(AND) Logical function

Thus, $h_\theta(x) \approx NOT\ (x_1\ \boldsymbol{AND}\ x_2)$

Example: NOT (OR)

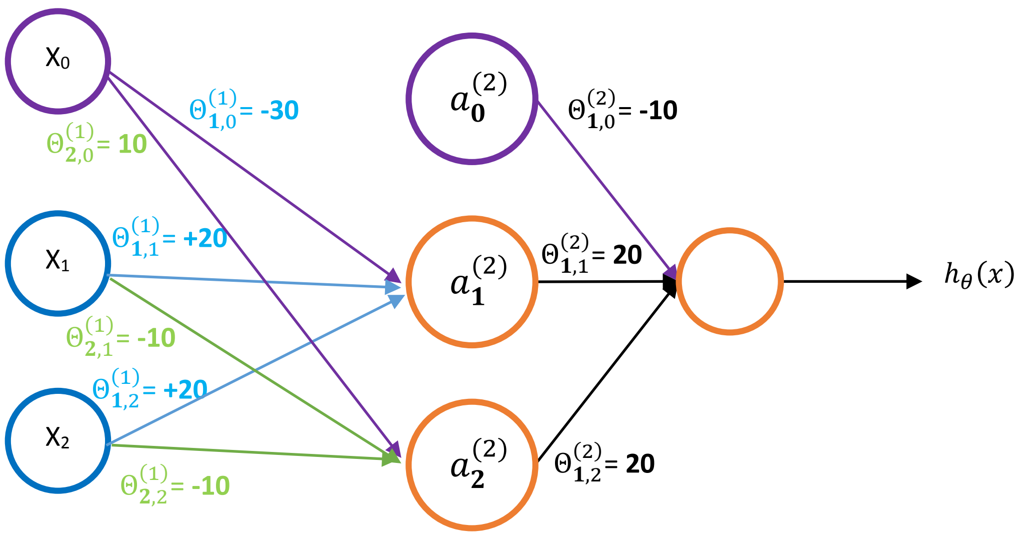
$x_1, x_2\ \in \{\mathbf{0}, \mathbf{1}\}$

$y = \ NOT\ (x_1\ OR\ x_2) = \ NOT(x_1)\ AND\ NOT(x_2)$ would be equal to 1 if and only if $x_1 or\ x_2 = 0$

Suppose $\Theta^{(1)}_{\mathbf{1,0}}$= -30 , $\Theta^{(1)}_{\mathbf{1,1}}$= 20, $\Theta^{(1)}_{\mathbf{1,2}}$= 20 (i.e. weight computing the AND function)

Suppose $\Theta^{(1)}_{\mathbf{2,0}}$= 10 , $\Theta^{(1)}_{\mathbf{2,1}}$= -20, $\Theta^{(1)}_{\mathbf{2,2}}$= -20 (i.e. weight computing the NOT(AND) function)

Suppose $\Theta^{(2)}_{\mathbf{1,0}}$= -10 , $\Theta^{(2)}_{\mathbf{1,1}}$= 20, $\Theta^{(2)}_{\mathbf{1,2}}$= 20 (i.e. weight computing the OR function)


X_0
X_1
X_2
$\Theta^{(1)}_{1,0}$= -30
$\Theta^{(1)}_{2,0}$= 10
$\Theta^{(1)}_{1,1}$= +20
$\Theta^{(1)}_{2,1}$= -10
$\Theta^{(1)}_{1,2}$= +20
$\Theta^{(1)}_{2,2}$= -10
$a_0^{(2)}$
$a_1^{(2)}$
$a_2^{(2)}$
$\Theta^{(2)}_{1,0}$= -10
$\Theta^{(2)}_{1,1}$= 20
$\Theta^{(2)}_{1,2}$= 20
$h_\theta(x)$

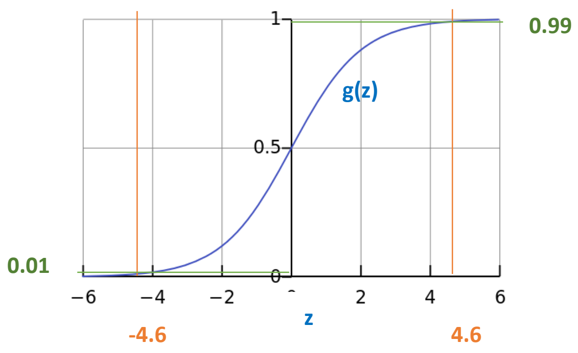
Thus,

$$a_{\mathbf{1}}^{(2)} = \boldsymbol{g}(-30 + 20x_1 + 20\ x_2)$$

$$a_{\mathbf{2}}^{(2)} = \boldsymbol{g}(10 - 20x_1 - 20\ x_2)$$

Issue: What is the logical function that the neural function compute? Is it possible to have a one-unit network to compute this logical NOT function?

Knowing that



For $a_1^{(2)}$

IF both $x_1 = 0\ AND\ x_2 = 0$, **THEN** $a_1^{(2)} = g(-30 + 20 * 0 + 20 * 0) = g(-30)$

Graphically, $g(-30) \approx 0\ ,\ a_1^{(2)} \approx 0$

IF both $x_1 = 0\ AND\ x_2 = 1$, **THEN** $a_1^{(2)} = g(-30 + 20 * 0 + 20 * 1) = g(-10)$

Graphically, $g(-10) \approx 0\ ,\ a_1^{(2)} \approx 0$

IF both $x_1 = 1\ AND\ x_2 = 0$, **THEN** $a_1^{(2)} = g(-30 + 20 * 1 + 20 * 0) = g(-10)$

Graphically, $g(-10) \approx 0\ ,\ a_1^{(2)} \approx 0$

IF both $x_1 = 1\ AND\ x_2 = 1$, **THEN** $a_1^{(2)} = g(-30 + 20 * 1 + 20 * 1) = g(+10)$

Graphically, $g(+10) \approx 1\ ,\ a_1^{(2)} \approx 1$

For $a_2^{(2)}$

IF both $x_1 = 0\ AND\ x_2 = 0$, **THEN** $a_2^{(2)} = g(10 - 20 * 0 - 20 * 0) = g(10)$

Graphically, $g(10) \approx 1\ ,\ a_2^{(2)} \approx 1$

IF both $x_1 = 0\ AND\ x_2 = 1$, **THEN** $a_2^{(2)} = g(10 - 20 * 0 - 20 * 1) = g(-10)$

Graphically, $g(-10) \approx 0\ ,\ a_2^{(2)} \approx 0$

IF both $x_1 = 1\ AND\ x_2 = 0$, **THEN** $a_2^{(2)} = g(10 - 20 * 1 - 20 * 0) = g(-10)$

Graphically, $g(-10) \approx 0,\ a_2^{(2)} \approx 0$

IF both $x_1 = 1\ AND\ x_2 = 1$, **THEN** $a_2^{(2)} = g(10 - 20 * 1 - 20 * 1) = g(-30)$

Graphically, $g(-30) \approx 0\ ,\ a_2^{(2)} \approx 0$

Thus,

$$\boldsymbol{h_\theta(x) = g\left(-10 + 20a_1^{(2)} + 20\, a_2^{(2)}\right)}$$

IF both $a_1^{(2)} = 0\ AND\ a_2^{(2)} = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(-10 + 20 * 0 + 20 * 0) = \boldsymbol{g}(-10)$

Graphically, $g(-10) \approx 0$, $h_\theta(x) \approx 0$

IF both $a_1^{(2)} = 0\ AND\ a_2^{(2)} = 1$, **THEN** $h_\theta(x) = \boldsymbol{g}(-10 + 20 * 0 + 20 * 1) = \boldsymbol{g}(10)$

Graphically, $g(10) \approx 1$, $h_\theta(x) \approx 1$

IF both $a_1^{(2)} = 1\ AND\ a_2^{(2)} = 0$, **THEN** $h_\theta(x) = \boldsymbol{g}(-10 + 20 * 1 + 20 * 0) = \boldsymbol{g}(10)$

Graphically, $g(10) \approx 1$, $h_\theta(x) \approx 1$

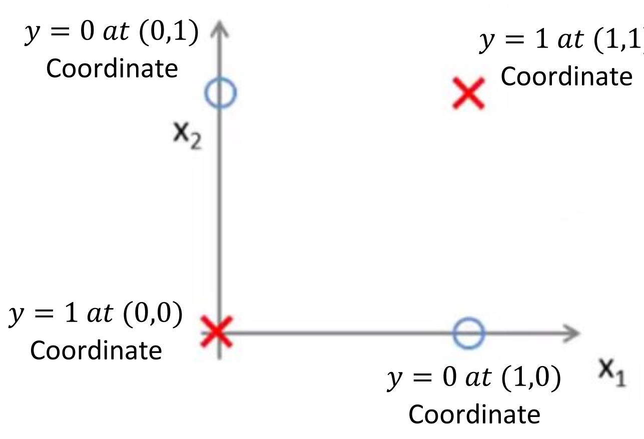
Summary,

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

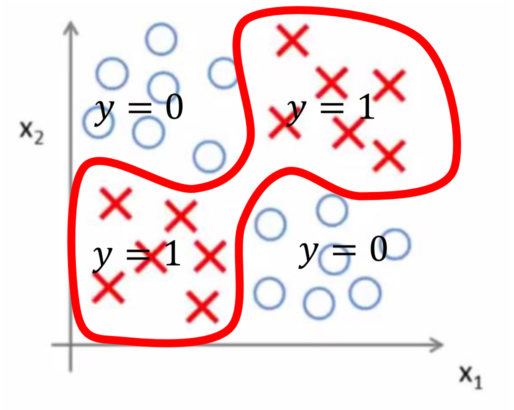
Thus, $h_\theta(x) \approx NOT\ (x_1\ \boldsymbol{OR}\ x_2)$ **NOT(OR)** Logical function

This means $h_\theta(x) = 1$ when either both x_1 $\boldsymbol{AND}$ x_2 are 0, $\boldsymbol{OR}$ $h_\theta(x) = 1$ when both x_1 $\boldsymbol{AND}$ x_2 are 1.

Concretely, $h_\theta(x)$ outputs 1 exactly at the following (0, 0) and (1, 1) coordinates:



Same principal can be applied to more complex dataset



To compute more complex function, neural network often ends up having more hidden layers.

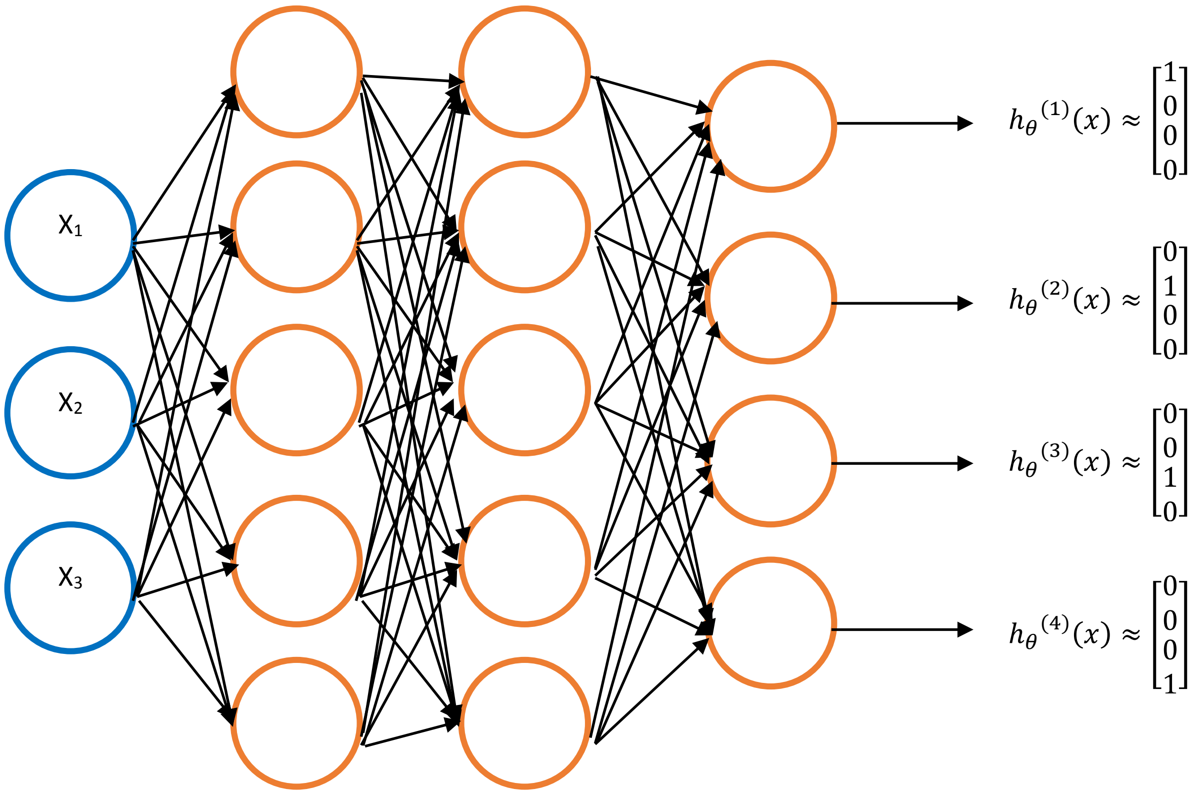
SUMMARY

	AND	**OR**	**NOT(AND)**	**NOT(OR)**
$h_\theta(x)=\begin{bmatrix} x_1=0\ and\ x_2=0 \\ x_1=1\ and\ x_2=0 \\ x_1=0\ and\ x_2=1 \\ x_1=1\ and\ x_2=1 \end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$	$\begin{bmatrix}0\\1\\1\\1\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\\1\end{bmatrix}$

5) *Neural network for multi-class classification*

In multiclass classification, each output variable $h_\theta{}^{(i)}(x)$ refers to different classes such as

$h_\theta{}^{(1)}(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ when pedestrian , $h_\theta{}^{(2)}(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ when car, $h_\theta{}^{(2)}(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ when monorcycle, etc…



X_1
X_2
X_3
$h_\theta{}^{(1)}(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$
$h_\theta{}^{(2)}(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$
$h_\theta{}^{(3)}(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$
$h_\theta{}^{(4)}(x) \approx \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

Training set: $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, …, $(x^{(m)}, y^{(m)})$ where $x^{(i)}$can be an input feature and $y^{(i)}$ is the output pedestrian when $h_\theta{}^{(1)}(x)$ predicts $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, a car when predicted $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, a monocycle when predict $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc…

With $y \in R^K$ where $S_l = \mathrm{K}$ is the number of output units. $h_\theta(x) \in R^K$.

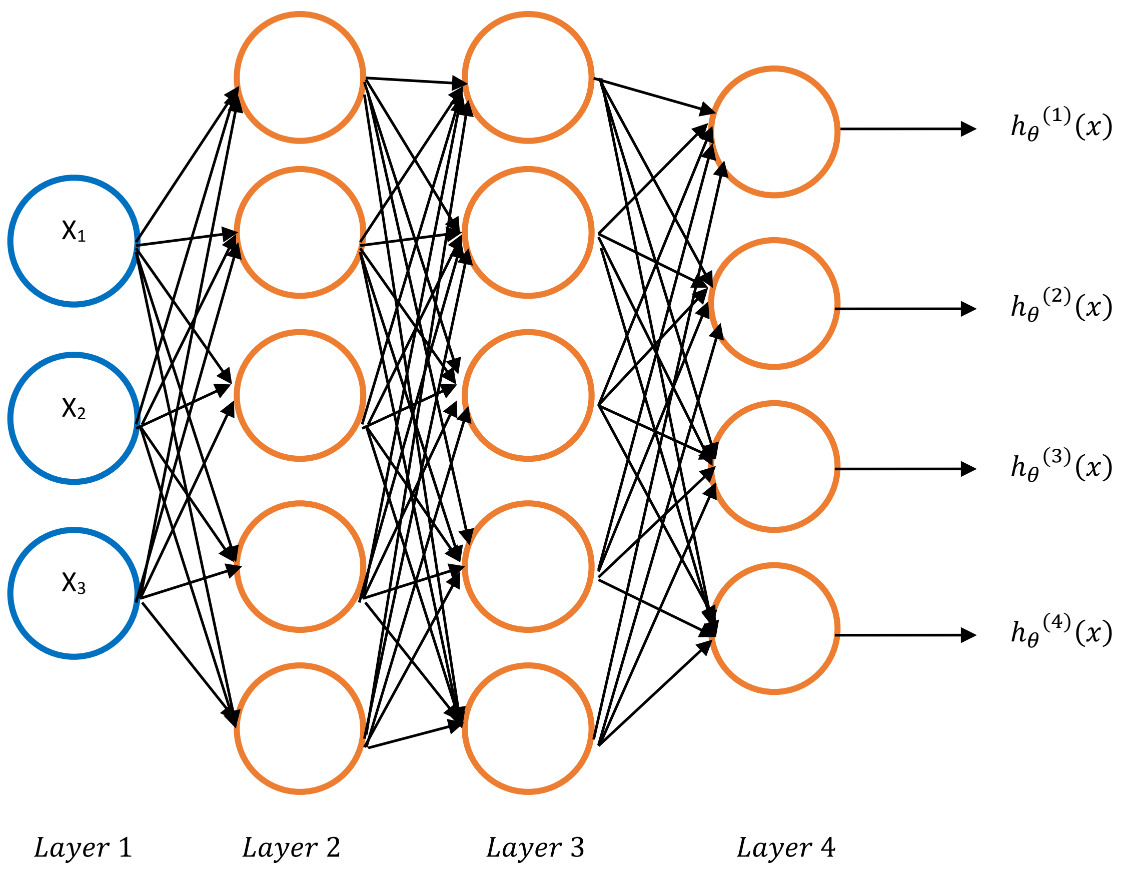
Multiclass classification (K classes) is used instead of binary classification ($y = 0\ or 1$, $S_l = \mathrm{K} = 1$ output unit) when $k > 2$

6) Cost Function

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

L = total number of layers in network

S_l = number of units (not counting bias unit) in layer l. $S_1 = 3, S_2 = 5, S_4 = 4$



The cost function used in **regularised logistic regression** (binary classification) is:

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

The cost function used in **regularised neural network** (multiclass classification) is:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} \mathrm{y}^{(i)}{}_k \log\left(h_{\boldsymbol{\Theta}}\left(x^{(i)}\right)\right)_k + (1 - \mathrm{y}^{(i)}{}_k) \log\left(1 - h_{\boldsymbol{\Theta}}\left(x^{(i)}\right)\right)_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(\boldsymbol{\Theta}_{ji}^{(l)}\right)^2$$

Sum of eachlogistic regression cost function over the K

Sum over layers

Sum over each column

With $h_\theta(x) \in R^K$ and $\left(h_{\boldsymbol{\Theta}}\left(x^{(i)}\right)\right)_i = i^{th}\ output.$

Sum over each row

7) Backpropagation Algorithm

Suppose:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} \mathrm{y}^{(i)}{}_k \log\left(h_{\boldsymbol{\Theta}}(x^{(i)})\right)_k + (1-\mathrm{y}^{(i)}{}_k)\log\left(1-h_{\boldsymbol{\Theta}}(x^{(i)})\right)_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(\boldsymbol{\Theta}_{ji}^{(l)}\right)^2$$

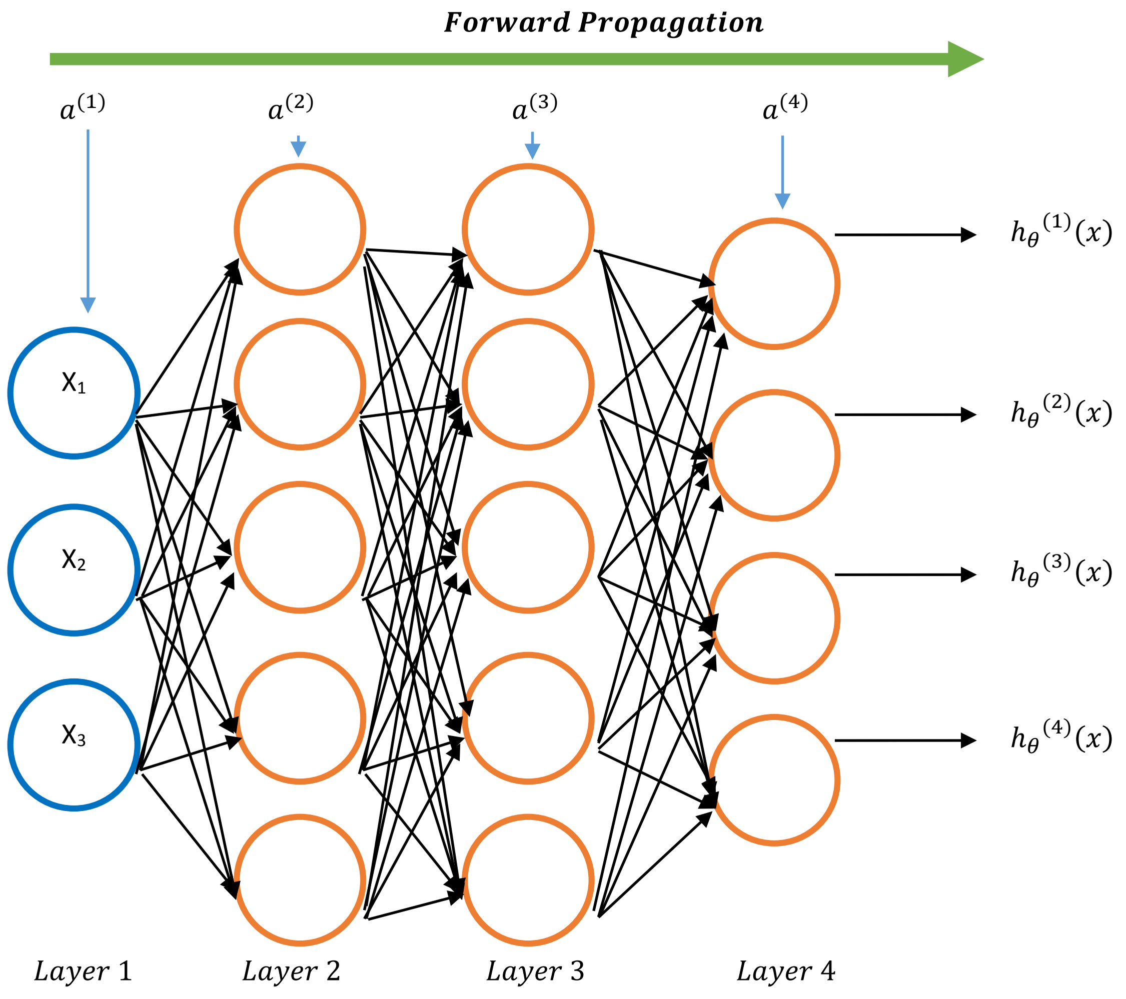
Optimization objectives: $minimise_{\boldsymbol{\Theta}}\, J(\boldsymbol{\Theta})$

Need to do gradient computation by writing code for:

- $J(\boldsymbol{\Theta})$
- $\frac{\partial}{\partial \boldsymbol{\Theta}_{ji}^{(l)}} J(\boldsymbol{\Theta})$

To get $J(\boldsymbol{\Theta})$, the first step will be to do a **Forward propagation** to find $h_{\boldsymbol{\Theta}}(x)$ such as:

$a^{(1)} = x$
$z^{(2)} = \boldsymbol{\Theta}^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})$ (add $a_0{}^{(2)}$)
$z^{(3)} = \boldsymbol{\Theta}^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)})$ (add $a_0{}^{(3)}$)
$z^{(4)} = \boldsymbol{\Theta}^{(3)} a^{(3)}$
$a^{(4)} = g(z^{(4)}) = h_{\boldsymbol{\Theta}}(x)$ → Once $h_{\boldsymbol{\Theta}}(x)$ is found, $J(\boldsymbol{\Theta})$ can be computed.



In order to compute $\frac{\partial}{\partial \boldsymbol{\Theta}_{ji}^{(l)}} J(\boldsymbol{\Theta})$, **Backpropagation** is used.

The principal is to compute **each** $\delta_j^{(l)}$, *the "error of a neural unit j in layer l"* for **each** *activation* $a_j^{(l)}$.

Concretely,

For each output unit, $h_\theta(x)$ (layer L=4), the difference between the activation unit $a_j^{(4)}$ and the actual value y_j

$$\delta_j^{(4)} = a_j^{(4)} - y_j = \left(h_{\boldsymbol{\Theta}}(x)\right)_j - y_j$$

Vectorized script of this equation will be:

$$\delta^{(4)} = a^{(4)} - y$$

After computing $\delta_j^{(4)}$ for the 4th layer, $\delta_j^{(3)}$ *and* $\delta_j^{(2)}$ for the 3th and the 2nd layers need to be computed using element-wise matrix multiplication such as

$$\delta^{(3)} = \left(\boldsymbol{\Theta}^{(3)}\right)^{\boldsymbol{T}} \delta^{(4)} .* \boxed{g'(z^{(3)})} \leftarrow \text{Derivative that can be represented as } a^{(3)} .* (1 - a^{(3)})$$

$$\delta^{(2)} = \left(\boldsymbol{\Theta}^{(2)}\right)^{\boldsymbol{T}} \delta^{(3)} .* \boxed{g'(z^{(2)})} \leftarrow \text{Derivative that can be represented as } a^{(2)} .* (1 - a^{(2)})$$

No $\delta^{(1)}$ since the 1st layer is the input layers where the features has been collected.

Then, ignoring the regularization term λ or when λ=0

$$\frac{\partial}{\partial \boldsymbol{\Theta}_{ji}^{(l)}} J(\boldsymbol{\Theta}) = a_j^{(l)} \delta_i^{(l+1)}$$

In more details,

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

Backpropagation algorithm

1) $Set\ \Delta_{ij}^{(l)} = 0$ for all l, i, j ($\Delta_{ij}^{(l)}$ are used as a cumulator)
2) $For\ i = 1\ to\ m$
   a. Set $a^{(i)} = x^{(i)}$
   b. Perform forward propagation to compute $a^{(l)}\ for\ l = 2, 3, \dots, L$
   c. Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
   d. Compute $\delta^{(L-1)}, \delta^{(L-1)}, \dots, \delta^{(2)}$. No $\delta^{(1)}$
   e. Compute Gradient Descent $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

      Or its Vectorized Implementation $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} \left(a^{(l)}\right)^T$
3) Compute:

   $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \boldsymbol{\Theta}_{ji}^{(l)}$ if j≠0

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

   N.B: $\frac{\partial}{\partial \boldsymbol{\Theta}_{ji}^{(l)}} J(\boldsymbol{\Theta}) = D_{ij}^{(l)}$

N.B. Matlab Code Example using Advanced optimization routine

Neural Network (L=4):

$\boldsymbol{\Theta}^{(1)}, \boldsymbol{\Theta}^{(2)}, \boldsymbol{\Theta}^{(3)}$- matrices (Theta1, Theta2, Theta3)

$\boldsymbol{D}^{(1)}, \boldsymbol{D}^{(2)}, \boldsymbol{D}^{(3)}$- matrices (D1, D2, D3)

fminunc is an advanced optimization algorithm; **gradient, theta** and **initialTheta** are vectors $\in \boldsymbol{R}^{n+1}$

Objective: "Unroll" parameters by reshaping matrices into vectors so that those matrices end up being in a format suitable such as

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

        …

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

Suppose a neural network such as

$S_1 = 10, S_2 = 10, S_3 = 1$

$\boldsymbol{\Theta}^{(1)} \in \boldsymbol{R}^{10*11}, \boldsymbol{\Theta}^{(2)} \in \boldsymbol{R}^{10*11}, \boldsymbol{\Theta}^{(3)} \in \boldsymbol{R}^{1*11}$

$\boldsymbol{D}^{(1)} \in \boldsymbol{R}^{10*11}, \boldsymbol{D}^{(2)} \in \boldsymbol{R}^{10*11}, \boldsymbol{D}^{(3)} \in \boldsymbol{R}^{1*11}$

"Unroll" those elements of $\boldsymbol{\Theta}$ and $\boldsymbol{D}$ matrices) into a long single vectors such as

```
thetaVec=[Theta1(:); Theta2(:), Theta3(:)];

DVec = [D1(:);D2(:);D3(:) ];
```

If going back from the vector representations to the matrix representations:

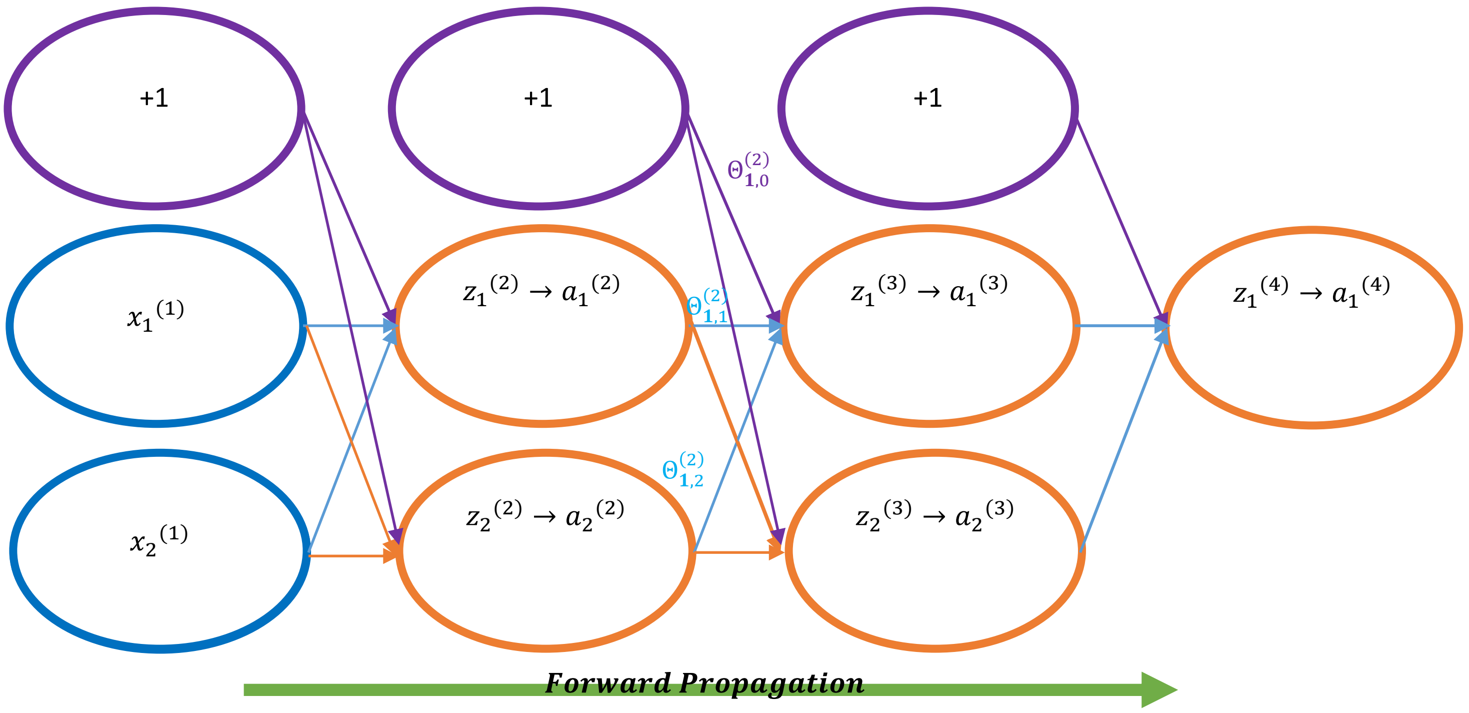
```
Theta1= reshape (thetaVec(1:110),10,11);

Theta2= reshape (thetaVec(111:220),10,11);

Theta3= reshape (thetaVec(221:231),1,11);
```

8) Graphically Representation of Forward and Back propagation

Forward propagation


+1
+1
+1
$x_1{}^{(1)}$
$x_2{}^{(1)}$
$z_1{}^{(2)} \rightarrow a_1{}^{(2)}$
$z_2{}^{(2)} \rightarrow a_2{}^{(2)}$
$z_1{}^{(3)} \rightarrow a_1{}^{(3)}$
$z_2{}^{(3)} \rightarrow a_2{}^{(3)}$
$z_1{}^{(4)} \rightarrow a_1{}^{(4)}$
$\Theta^{(2)}_{1,0}$
$\Theta^{(2)}_{1,1}$
$\Theta^{(2)}_{1,2}$
Forward Propagation

Where *for instance*, $z_1{}^{(3)} = \Theta^{(2)}_{1,0} * 1 + \Theta^{(2)}_{1,1} * a_1{}^{(2)} + \Theta^{(2)}_{1,2} * a_2{}^{(2)}$

Backpropagation

Knowing that

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

Focusing on a single example $x^{(i)}$, $\mathrm{y}^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\mathrm{cost(i)} = \mathrm{y}^{(i)} \log(h_{\boldsymbol{\Theta}}(x^{(i)})) + (1-\mathrm{y}^{(i)}) \log(1-h_{\boldsymbol{\Theta}}(x^{(i)}))$$

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} cost(i)]$$

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

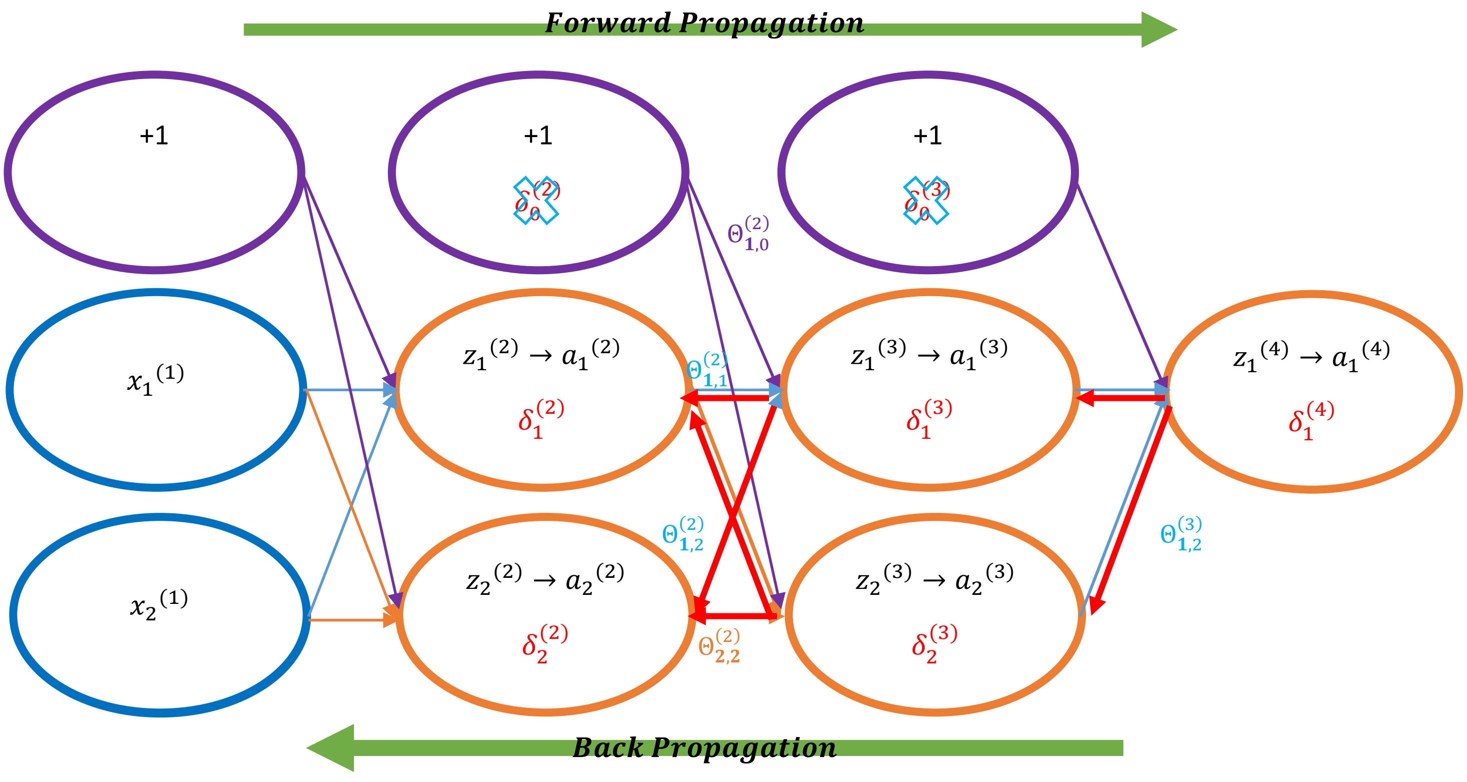
Formally,

$$\delta_j^{(l)} = \frac{\partial}{\partial \boldsymbol{z}_j^{(l)}} cost(i) \text{ (for } j \geq 0)$$

Since,

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

After having defined $\delta_1^{(4)}$, it is possible to propagate this further backward and compute $\delta_1^{(3)}$ and $\delta_2^{(3)}$, and this end up with $\delta_1^{(2)}$ and $\delta_2^{(2)}$

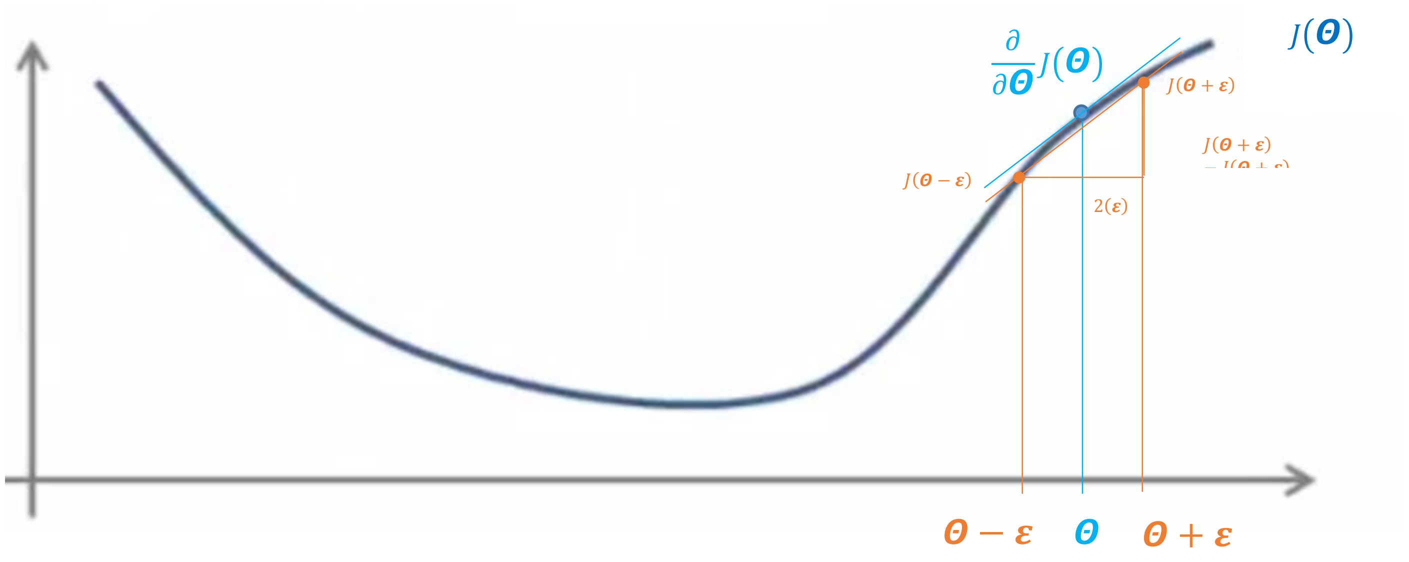


Where *for instance*, $\delta_2^{(2)} = \Theta_{1,2}^{(2)} * \delta_1^{(3)} + \Theta_{2,2}^{(2)} * \delta_2^{(3)}$ and $\delta_2^{(3)} = \Theta_{1,2}^{(3)} * \delta_1^{(4)}$

9) *Gradient Checking*

Gradient checking is used to assess whether the Forward and Backpropagation algorithm is working correctly (i.e. if $J(\boldsymbol{\Theta})$ is decreasing after each iteration).

Considering,



In order to check whether the gradient is working properly, the derivative needs to be estimated such as

$$\frac{\partial}{\partial \boldsymbol{\Theta}} J(\boldsymbol{\Theta}) \approx \frac{J(\boldsymbol{\Theta}+\varepsilon)-J(\boldsymbol{\Theta}-\varepsilon)}{2\varepsilon} \text{ with usually } \boldsymbol{\varepsilon} = \mathbf{10^{-4}} \text{ and } \boldsymbol{\Theta} \in \boldsymbol{R}^n$$

In more details,

Suppose,

$$\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n]$$

$$\frac{\partial}{\partial \boldsymbol{\theta}_1} J(\boldsymbol{\theta}) \approx \frac{J(\boldsymbol{\theta}_1+\varepsilon, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n) - J(\boldsymbol{\theta}_1-\varepsilon, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \boldsymbol{\theta}_2} J(\boldsymbol{\theta}) \approx \frac{J(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2+\varepsilon, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n) - J(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2-\varepsilon, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n)}{2\varepsilon}$$

$$\dots$$

$$\frac{\partial}{\partial \boldsymbol{\theta}_n} J(\boldsymbol{\theta}) \approx \frac{J(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n+\varepsilon) - J(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n-\varepsilon)}{2\varepsilon}$$

N.B. Matlab Code Example**:**

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

Where **thetaPlus** represents $\begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \cdots \\ \boldsymbol{\theta}_i + \varepsilon \\ \cdots \\ \boldsymbol{\theta}_n \end{bmatrix}$, **thetaMinus** represents $\begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \cdots \\ \boldsymbol{\theta}_i - \varepsilon \\ \cdots \\ \boldsymbol{\theta}_n \end{bmatrix}$,

gradApprox represents $\frac{J(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n+\varepsilon) - J(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \dots, \boldsymbol{\theta}_n-\varepsilon)}{2\varepsilon}$

Then, this for loop is implemented to compute and check whether **gradApprox** $\approx$**DVec**

DVec is the derivative $\frac{\partial}{\partial \boldsymbol{\theta}_n} J(\boldsymbol{\theta})$ from the backpropagation by unrolling $\boldsymbol{D}^{(1)}, \boldsymbol{D}^{(2)}, \boldsymbol{D}^{(3)}$.

Once **gradApprox** $\approx$**DVec** is confirmed it is important to turn off gradient checking before running backpropagation algorithm. If gradient checking is turned on (i.e. computing **gradApprox**) in every iteration of gradient descent, the code will be <u>very</u> slow.

10) Random initialization of $\boldsymbol{\Theta}$

For gradient descent and advanced optimization, it is important to initialize the value of $\boldsymbol{\Theta}$

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

If set **initialTheta** = zeros(n,1) such as $\boldsymbol{\Theta}_{ji}^{(l)}$=0 for all i, j, l

➔ The problem would be that after each update, parameters corresponding to inputs which are going to each of two hidden units are going to be identical such as

$$a_1{}^{(2)} = a_2{}^{(2)} \text{ and } \delta_1^{(2)} = \delta_2^{(2)}$$

$$\boldsymbol{\Theta}_{01}^{(1)} = \boldsymbol{\Theta}_{02}^{(1)} \text{ and } \frac{\partial}{\partial \boldsymbol{\Theta}_{01}^{(1)}} J(\boldsymbol{\Theta}) = \frac{\partial}{\partial \boldsymbol{\Theta}_{02}^{(1)}} J(\boldsymbol{\Theta})$$

In order to break this symmetric problem, each value of $\boldsymbol{\Theta}_{ji}^{(l)}$ will be initialized to a random number between $[-\boldsymbol{\varepsilon};\ \boldsymbol{\varepsilon}]$ such as

$$-\boldsymbol{\varepsilon} \leq \boldsymbol{\Theta}_{ji}^{(l)} \leq \boldsymbol{\varepsilon}$$

N.B. Matlab Code Example**:**

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

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

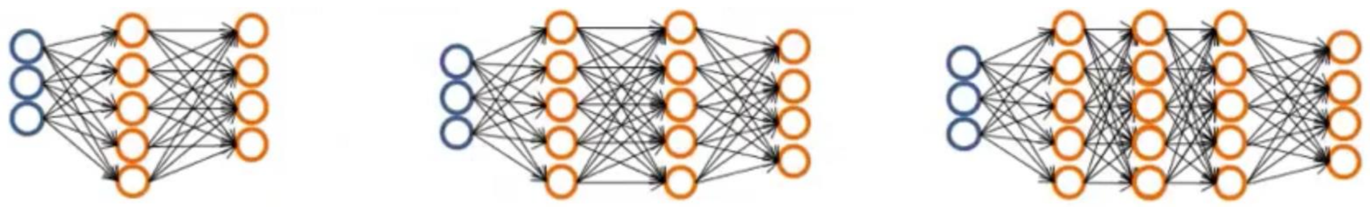
Random matrix 1X11 between 0 and 1

11) Training and Overfitting

Training using neural network

1. Randomly initial weights ($-\boldsymbol{\varepsilon} \leq \boldsymbol{\Theta}_{ji}^{(l)} \leq \boldsymbol{\varepsilon}$)
2. Implement forward propagation to get $h_{\boldsymbol{\Theta}}(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\boldsymbol{\Theta})$
4. Implement backpropagation to compute partial derivatives $\frac{\partial}{\partial \boldsymbol{\Theta}_{jk}^{(l)}} J(\boldsymbol{\Theta})$
5. Use gradient checking to compare $\frac{\partial}{\partial \boldsymbol{\Theta}_{jk}^{(l)}} J(\boldsymbol{\Theta})$ computed using backpropagation vs. numerical estimate of gradient of $J(\boldsymbol{\Theta})$. Do not forget to disable gradient checking code.
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\boldsymbol{\Theta})$ as a function of parameters $\boldsymbol{\Theta}$. $J(\boldsymbol{\Theta})$ is usually non-convex but ending up in a local minimum do not make a huge difference than ending in global minimum in pratice

Issue: When training the neural network algorithm, how many hidden layers should it be included in the model depending on the number of input units (dimension of features $\boldsymbol{x}^{(i)}$) and the number of output units (number of classes)?



By default, 1 hidden layer or if >1 hidden layer, it is essential to have the same number of hidden units in every layer.

IV. Support Vector Machine

Support Vector Machine (SVM) is a supervised learning models that perform lineaer and non-linear classification using “Kernel”. Compared to logistic regression and neural network, SVM can sometimes give a cleaner, more efficient way of learning complex non-linear functions.

1) Optimization Objectives

Based on Logistic Regression Optimization Objectives:

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

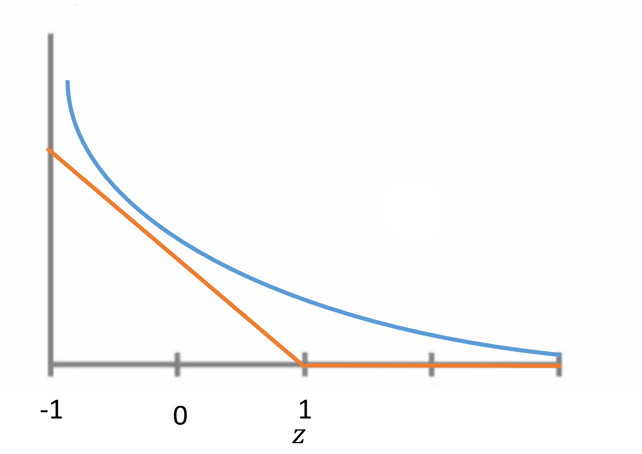
Support vector machine Optimization Objectives:

$$minimizing_{\boldsymbol{\theta}}\, J(\boldsymbol{\theta}) = \frac{1}{\cancel{m}}\sum_{i=1}^{m}[\mathrm{y}^{(i)}(cost_1\left(\theta^T x^{(i)}\right) + \left(1-\mathrm{y}^{(i)}\right)cost_0\left(\theta^T x^{(i)}\right)] + \frac{\lambda}{2\cancel{m}}\sum_{i=1}^{n}\theta_j^2$$

With $cost_1\left(\theta^T x^{(i)}\right) = -\log\left(h_\theta\left(x^{(i)}\right)\right)$ and $cost_0\left(\theta^T x^{(i)}\right) = -\log\left(1-h_\theta(x)\right)$ such as

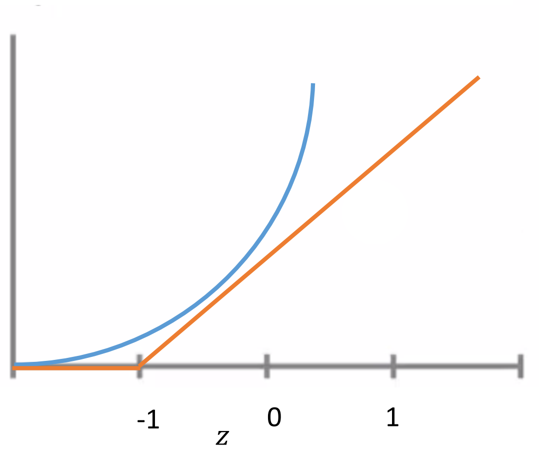
$cost_1\left(\theta^T x^{(i)}\right) = -\log\left(h_\theta\left(x^{(i)}\right)\right)$

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



$cost_0\left(\theta^T x^{(i)}\right) = -\log\left(1-h_\theta(x)\right)$

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



<u>Now,</u>

Knowing that $\overset{\mathbf{A}}{\boxed{\frac{1}{m}\sum_{i=1}^{m}[\mathrm{y}^{(i)}(cost_1\left(\theta^T x^{(i)}\right) + \left(1-\mathrm{y}^{(i)}\right)cost_0\left(\theta^T x^{(i)}\right)]}} + \overset{\mathbf{B}}{\boxed{\frac{\lambda}{2m}\sum_{i=1}^{n}\theta_j^2}}$

To control the trade-off between the **minimizing goal** and the **regularization goal**,

It is possible to reshape the equation

$$\mathbf{A} + \lambda\mathbf{B}$$

into

$$\mathbf{CA} + \mathbf{B}$$

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

Thus, the optimization objectives of Support Vector Machine can be re-written as:

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

And its hypothesis function is

$$h_\theta(x) = \begin{cases} 1, & if\ \theta^T \geq 0 \\ 0, & otherwise \end{cases}$$

2) SVM as a Large Margin Classifier

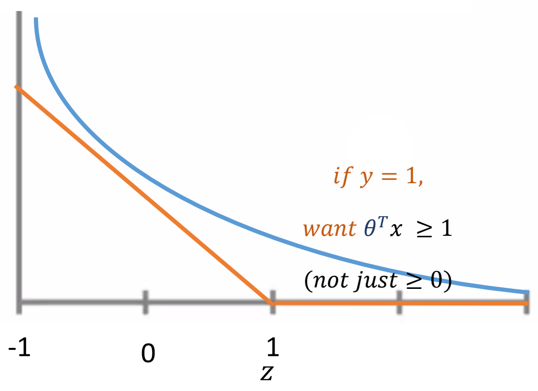
To summarize

Optimization Objectives:

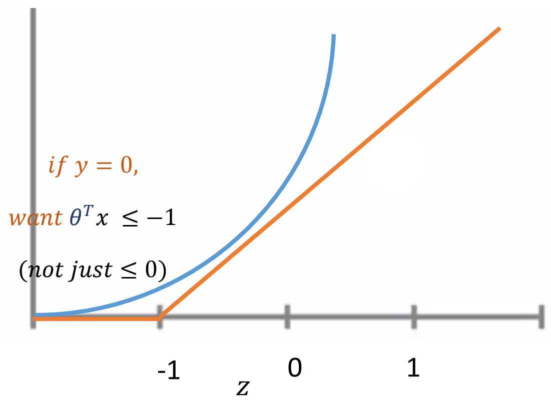
$$minimizing_{\boldsymbol{\theta}}\, J(\boldsymbol{\theta}) = \boldsymbol{C}\sum_{i=1}^{m}[\mathrm{y}^{(i)}\left(cost_1\left(\theta^T x^{(i)}\right) + \left(1-\mathrm{y}^{(i)}\right)cost_0\left(\theta^T x^{(i)}\right)\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Graphical Representation:

$$cost_1\left(\theta^T x^{(i)}\right) = -\log\left(h_\theta\left(x^{(i)}\right)\right)$$



$$cost_0\left(\theta^T x^{(i)}\right) = -\log\left(1 - h_\theta(x)\right)$$

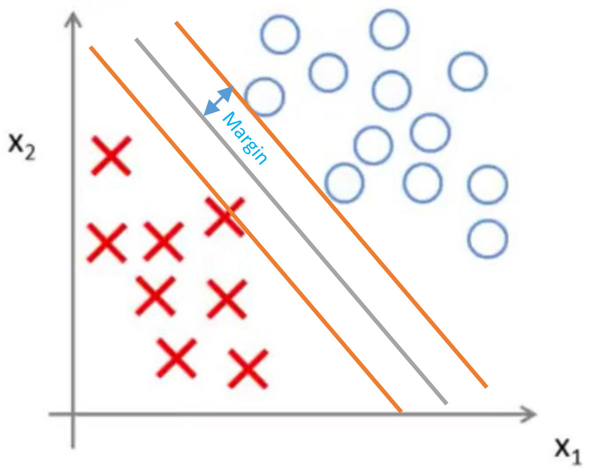


To make this cost function small

If having a positive example y=1, $cost_1(z) = 0\ \Leftrightarrow \mathrm{z} \geq 1$

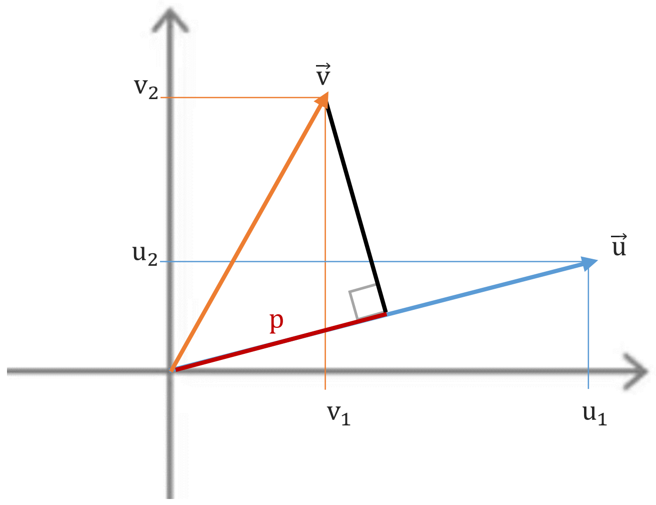
If having a negative example y=0, $cost_0(z) = 0\ \Leftrightarrow \mathrm{z} \leq -1$

Thus, to solve $\boldsymbol{C}\sum_{i=1}^{m}[\mathrm{y}^{(i)}\left(cost_1\left(\theta^T x^{(i)}\right) + \left(1-\mathrm{y}^{(i)}\right)cost_0\left(\theta^T x^{(i)}\right)\right] \approx 0$, this draws a decision boundary with a margin as large minimum distance as possible from any of the training examples such as:



Issue: How the margins are obtain when minimizing the cost function of SVM?

Reminder:



$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ and $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$\|u\|$ is the length of vector u such as $\|u\| = \sqrt{{u_1}^2 + {u_2}^2}$

P is the length of the projection of v onto u. p can be positive or negative.

$$u^T.v = p.\|u\|$$

Or

$$u^T.v = v^T.u = u_1v_1 + u_2v_2$$

Now,

For n=2

$$minimizing_{\boldsymbol{\theta}}\ \frac{1}{2}\sum_{i=1}^{n}\theta_j^2 = \frac{1}{2}(\theta_1^2+\theta_2^2) = \frac{1}{2}\left(\sqrt{(\theta_1^2+\theta_2^2)}\right)^2 = \frac{1}{2}\|\theta\|^2$$

With $\sqrt{(\theta_1^2+\theta_2^2)} = \|\theta\|$

Thus, what SVM is doing in the optimization objectives is to minimize the squared norm of the squared length of the parameter vector $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$.

Now,

Since we want,

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

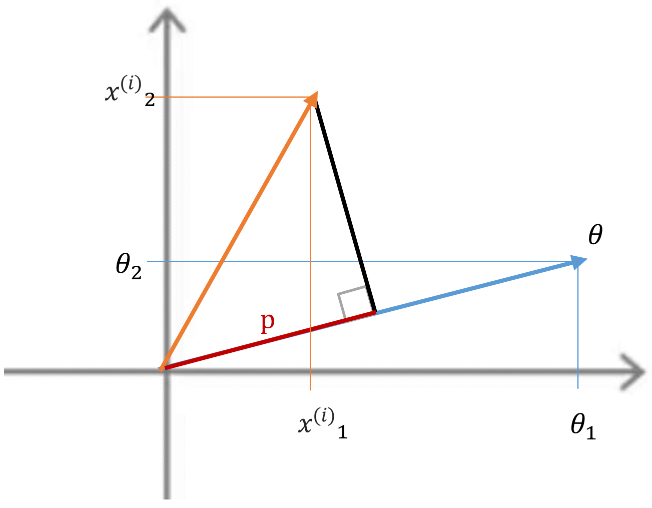
$\theta^T x^{(i)} \leq -1 \ \ if\ y^{(i)} = 0$

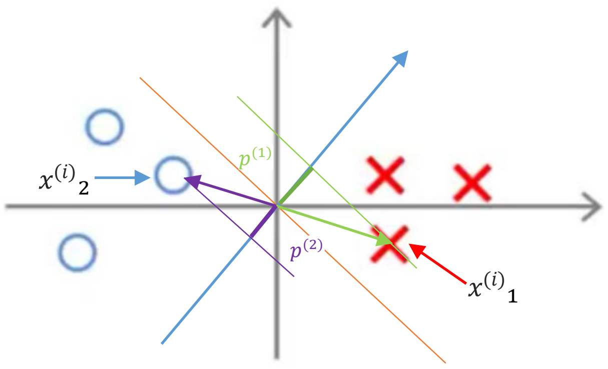
Suppose,

$\theta^T x^{(i)} = p^{(i)}.\|\theta\| = {x^{(i)}}_1\theta_1 + {x^{(i)}}_2\theta_2$ where $p^{(i)}$ *is the projection of* $x^{(i)}$ *onto the vector* θ

$p^{(i)}.\|\theta\| \geq 1 \ \ if\ y^{(i)} = 1$

$p^{(i)}.\|\theta\| \leq -1 \ \ if\ y^{(i)} = 0$



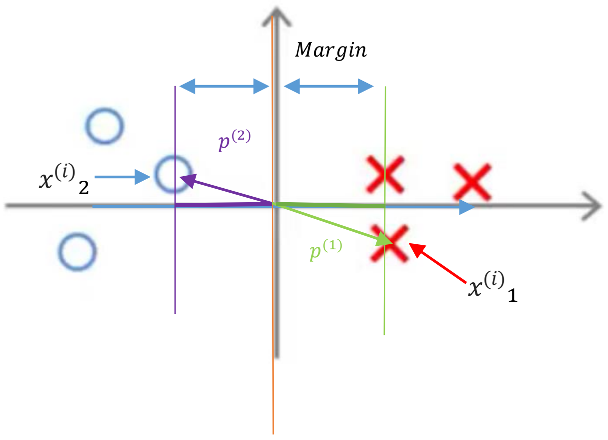


$With\ \theta_0 = 0$

In this case,

$p^{(i)}.\,||\theta|| \geq 1$ if and only if $||\theta||$ is large

$p^{(i)}.\,||\theta|| \leq -1$ if and only if $||\theta||$ is large



$With\ \theta_0 = 0$

In this case,

$p^{(i)}.\,||\theta|| \geq 0$ if and only if $||\theta||$ is small

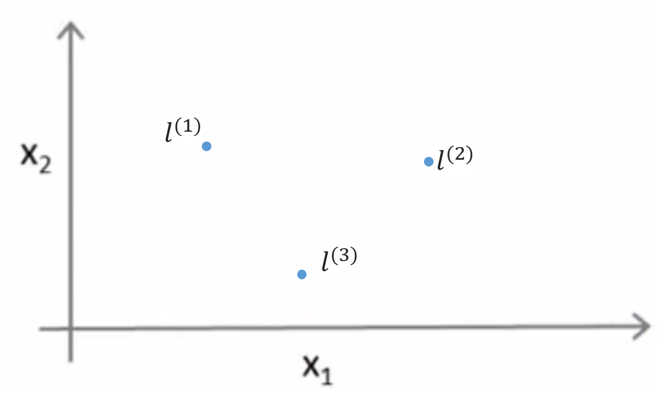
$p^{(i)}.\,||\theta|| \leq 0$ if and only if $||\theta||$ is small

This enables a large margin

When simplified with $\theta_0 = 0$, this means that the SVM hypothesis function pass by the coordinate (0,0)

3) SVM with Kernels

SVM can be adapted in order to develop complex nonlinear classifiers by using **Kernels.**
When describing a decision boundary on a complex dataset, incorporating a large number of feature into the hypothesis function become computationally expensive. The alternative will be to use **Kernels**



Given x, the aim of **Kernels** is to define new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

In more details,
Given x, compute:

$$f_1 = similarity(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$f_2 = similarity(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

$$f_3 = similarity(x, l^{(3)}) = \exp\left(-\frac{\|x - l^{(3)}\|^2}{2\sigma^2}\right)$$

…

$$f_i = \underbrace{\boxed{similarity(x, l^{(i)})}}_{\text{Kernel}} = \underbrace{\boxed{\exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)}}_{\text{Gaussian Kernels}} = \exp\left(-\frac{\sum_{j=1}^{n}\|x_j - l_j^{(i)}\|^2}{2\sigma^2}\right)$$

$similarity(x, l^{(i)})$ means that

- If x *is close to* $l^{(i)}, x \approx l^{(i)}$:

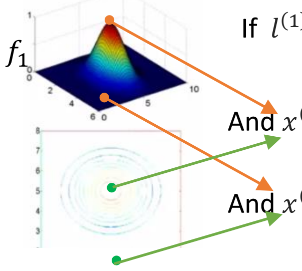
$$f_i \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

- If x *is far from* $l^{(i)}$:

$$f_i \approx \exp\left(-\frac{Large\ number^2}{2\sigma^2}\right) \approx 0$$

Graphically,

$\sigma^2 = 1$

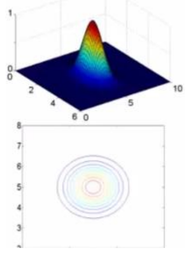


If $l^{(1)} = \begin{bmatrix}3\\5\end{bmatrix}, f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

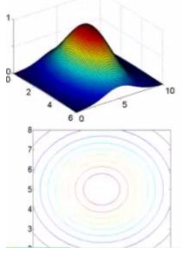
And $x^{(i)} = \begin{bmatrix}3\\5\end{bmatrix}, f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$

And $x^{(i)} = \begin{bmatrix}10\\32\end{bmatrix}, f_1 \approx \exp\left(-\frac{Large\ number^2}{2\sigma^2}\right) \approx 0$

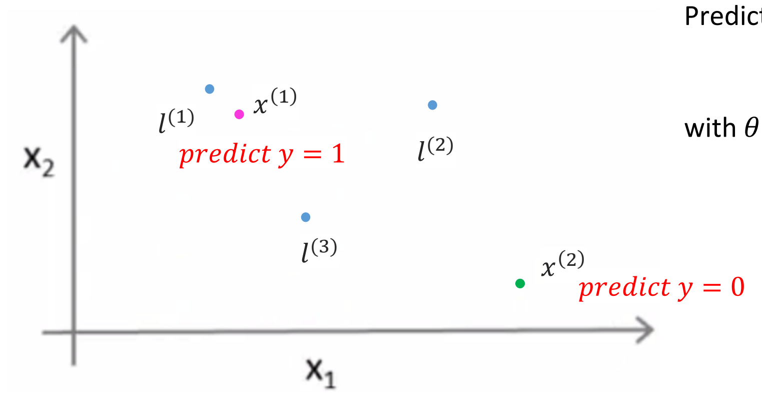
$\sigma^2 = 0.5$



$\sigma^2 = 2$



<u>Now,</u>



Predict "y=1" when $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

with $\theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 1 \\ \theta_2 = 1 \\ \theta_3 = 0 \end{bmatrix}$

IF for $x^{(1)}$, $f_1 \approx 1, f_2 \approx 0, f_3 \approx 0$ such as

$$\theta_0 + \theta_1 * 1 + \theta_2 * 0 + \theta_3 * 0 = -0.5 + 1 = 0.5 \geq 0$$

Then for $x^{(1)}$, predict y=1 when close to $l^{(1)}$ but predict y=0 when close to $l^{(2)}$ or $l^{(3)}$

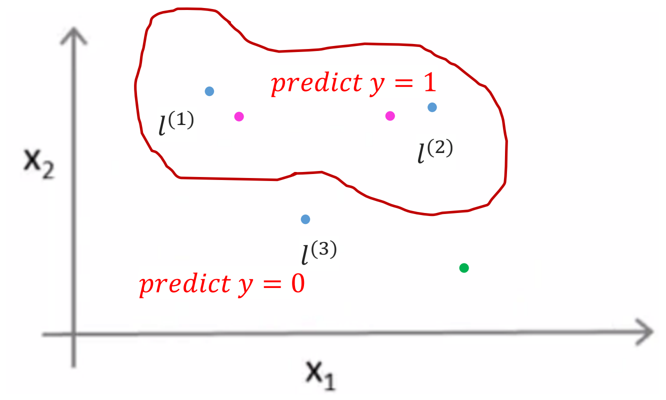
IF for $x^{(2)}$, $f_1 \approx 0, f_2 \approx 0, f_3 \approx 0$ such as

$$\theta_0 + \theta_1 * 0 + \theta_2 * 0 + \theta_3 * 0 = -0.5 + 1 = -0.5 \leq 0$$

Then for $x^{(2)}$, predict y=0 when close to $l^{(1)}$, $l^{(2)}$ or $l^{(3)}$

Etc…

If all the training example are predicted y=1 when they are closed to $l^{(1)}$ and $l^{(2)}$ systematically when predict y=0 when they are far from these landmarks, then a decision boundary will looks like:

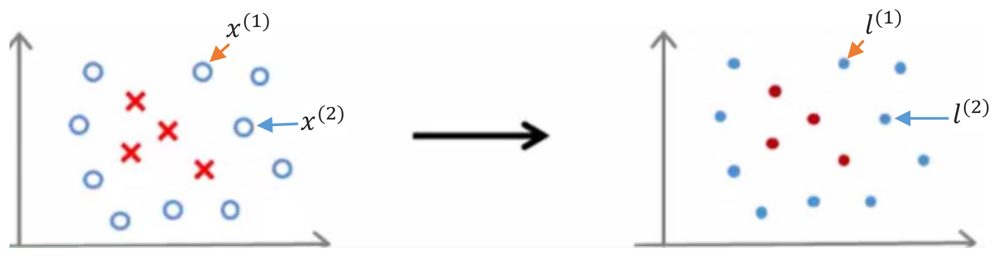


Thus, all new examples that will be inside the boundary will be predicted as y=1 while all the examples that will be outside the boundary will be predicted as y=0

<u>**Issue**:</u> Where to place $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?

In practice, the landmarks $l^{(i)}$ are actually put at the <u>same</u> locations as the training examples.

Graphically,



By using this methodology,

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

Thus for given example x,

$$\boxed{\begin{matrix} f_1 \\ \\ f_2 \\ \\ \\ \\ f_m \end{matrix}}\;\begin{matrix} = similarity(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) \\ = similarity(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right) \\ \dots \\ = similarity(x, l^{(m)}) = \exp\left(-\frac{\|x - l^{(m)}\|^2}{2\sigma^2}\right) \end{matrix}$$

With $f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \dots \\ f_m \end{bmatrix}$ and $f_0 = 1$

Then for training example $(x^{(i)}, y^{(i)})$:

$$x^{(i)} \left\{ \begin{matrix} f_1^{(i)} = similarity(x^{(i)}, l^{(1)}) \\ f_2^{(i)} = similarity(x^{(i)}, l^{(2)}) \\ \dots \\ f_i^{(i)} = similarity(x^{(i)}, l^{(i)}) = \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1 \\ \dots \\ f_m^{(i)} = similarity(x^{(i)}, l^{(m)}) \end{matrix} \right.$$

With $f^{(i)} = \begin{bmatrix} f^{(i)}{}_0 \\ f^{(i)}{}_1 \\ f^{(i)}{}_2 \\ \dots \\ f^{(i)}{}_m \end{bmatrix}$ and $f^{(i)}{}_0 = 1$

<u>Now</u>,

It is possible to apply Kernels to SVM

This features vector $f^{(i)}$ (i.e. similarity function) that represents the training example can be used with SVM such as:

Given x, the aim is to define the features $f \in R^{m+1}$ in order to predict "y=1" if $\theta^T f \geq 0$

With $\theta^T f = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \dots + \theta_m f_m$ amd $\theta \in R^{m+1}$

Thus, the optimization objectives of SVM with Kernels will be:

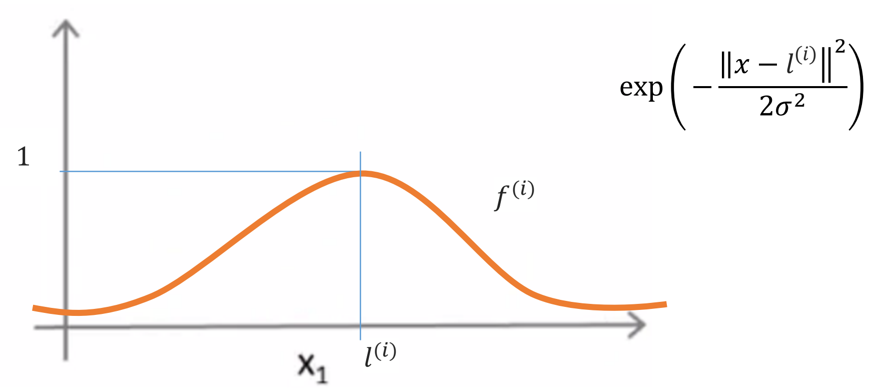
$$min_\theta\, \boldsymbol{C} \sum_{i=1}^{m} [y^{(i)} (cost_1(\theta^T \boldsymbol{f}^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T \boldsymbol{f}^{(i)})] + \frac{1}{2} \sum_{i=1}^{\boldsymbol{m}} \theta_j^2$$

Where n=m and $\sum_{i=1}^{m} \theta_j^2$ can be written in its vectorised form $\theta^T\theta$. When using SVM with Kernels, $\theta^T\theta$ is usually modified for obtaining better efficiency into $\theta^T \boldsymbol{M} \theta$ where $\boldsymbol{M}$ represents the matrix of all training example.

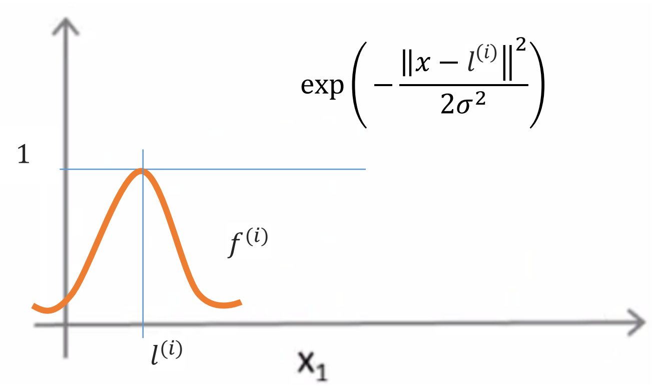
Kernels can be used with other learning algorithms (i.e. classifier such as logistic regression or neural networks) but it does not usually work well.

4) SVM parameters

- Large C ($= \frac{1}{\lambda}$): Lower Bias, Higher Variance (small λ)
- Small C ($= \frac{1}{\lambda}$): Lower Variance, Higher Bias (large λ)
- Large σ^2: Higher Bias, Lower Variance



- Small σ^2: Higher Variance, Lower Bias



5) SVM in practice

It is possible to use SVM software package (e.g. liblinear, libsvm,…) to solve for parameters θ.

What to specify when using those packages?

- The choice of parameter C
- Choice of kernel (i.e. similarity function):
    - No kernel (also called "linear kernel"): predict "y=1" if $\theta^T x \geq 0$ with $x \in R^{n+1}$
    - Gaussian kernel:
        - $f_i = similarity(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$
        - Need to choose σ^2
        - $x \in R^n$
        - $feature\ scaling$ can be performed using the Gaussian kernel
    -

N.B. Matlab code example of Gaussian kernel:

function f = kernel (x1,x2)

$$f = \exp\left(-\frac{\left\|x - l^{(i)}\right\|^2}{2\sigma^2}\right)$$

return