Introduction

What is Machine Learning?

Two definitions of Machine Learning are offered. Arthur Samuel described it as: "the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.

Tom Mitchell provides a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Example: playing checkers.

- E = the experience of playing many games of checkers
- T = the task of playing checkers.
- P = the probability that the program will win the next game.

Supervised Learning

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into **"regression"** and **"classification"** problems. In a **regression** problem, we are trying to predict results within a **continuous** output, meaning that we are trying to map input variables to some **continuous** function. In a **classification** problem, we are instead trying to predict results in a **discrete** output. In other words, we are trying to map input variables into **discrete** categories.

Example:

Given data about the size of houses on the real estate market, try to predict their price. Price as a function of size is a *continuous* output, so this is a regression problem.

We could turn this example into a classification problem by instead making our output about whether the house "sells for more or less than the asking price." Here we are classifying the houses based on price into two *discrete* categories.

Unsupervised Learning

Unsupervised learning, on the other hand, allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

We can derive this structure by **clustering** the data based on relationships among the variables in the data.

With unsupervised learning there is no feedback based on the prediction results, i.e., there is no teacher to correct you. It's not just about clustering. For example, associative memory is unsupervised learning.

Example:

Clustering: Take a collection of 1000 essays written on the US Economy, and find a way to automatically group these essays into a small number that are somehow similar or related by different variables, such as word frequency, sentence length, page count, and so on.

Associative: Suppose a doctor over years of experience forms associations in his mind between patient characteristics and illnesses that they have. If a new patient shows up then based on this patient's characteristics such as symptoms, family medical history, physical attributes, mental outlook, etc the doctor associates possible illness or illnesses based on what the doctor has seen before with similar patients. This is not the same as rule based reasoning as in expert systems. In this case we would like to estimate a mapping function from patient characteristics into illnesses.

Linear Regression with One Variable

Model Representation

Recall that in *regression problems*, we are taking input variables and trying to map the output onto a *continuous* expected result function.

Linear regression with one variable is also known as "univariate linear regression."

Univariate linear regression is used when you want to predict a **single output** value from a **single input** value. We're doing **supervised learning** here, so that means we already have an idea what the input/output cause and effect should be.

The Hypothesis(假设) Function

Our hypothesis function has the general form:

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

We give to hθ values for θ0 and θ1 to get our output 'y'. In other words, we are trying to create a function called hθ that is able to reliably map our input data (the x's) to our output data (the y's). (a univariate linear regression model)

Example:

x (input)	y (output)
0	4
1	7
2	7
3	8

Now we can make a random guess about our h_ϑ function: ϑ_0=2 and ϑ_1=2. The

hypothesis function becomes h_ϑ(x)=2+2x.

So for input of 1 to our hypothesis, y will be 4. This is off by 3.

Cost Function

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's compared to the actual output y's.

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

To break it apart, it is $\frac{1}{2}\bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x^{(i)}) - y^{(i)}$, or the difference between the predicted value and the actual value.

This function is otherwise called the "Squared error function", or Mean squared error (均方差). The mean is halved (1/(2m)) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out (抵消) the 1/2 term.

Now we are able to concretely measure the accuracy of our predictor function against the correct results we have so that we can predict new results we don't have.

要做的事情：选取 ϑ_0 和 ϑ_1，使得 cost function 值最小。

Gradient Descent

So we have our hypothesis function and we have a way of measuring how accurate it is. Now what we need is a way to automatically improve our hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields ϑ_0 and ϑ_1 (actually we are graphing the cost function for the combinations of parameters). This can be kind of confusing; we are moving up to a higher level of abstraction. We are not graphing x and y itself, but the guesses of our hypothesis function.

We put θ0 on the x axis and θ1 on the z axis, with the cost function on the vertical y axis. The points on our graph will be the result of the **cost function** using our hypothesis with those specific theta parameters.

We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum.

The way we do this is by taking the **derivative** (the line tangent(切线) to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down that derivative by the parameter α, called the learning rate.

The gradient descent equation is:

repeat until convergence:

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

for j=0 and j=1

Intuitively, this could be thought of as:

repeat until convergence:

$\vartheta_j := \vartheta_j - \alpha$[Slope of tangent aka derivative]

Gradient Descent for Linear Regression

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to (the derivation of the formulas can be found here):

repeat until convergence: {

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

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

}

Note：两个值的更新要同时进行（而不是先更新一个再计算另一个=>其他算法）

where m is the size of the training set, ϑ_0 a constant that will be changing simultaneously with ϑ_1 and $x^{(i)}$,$y^{(i)}$ are values of the given training set (data).

Note that we have separated out the two cases for θ_j and that for $\theta 1$ we are multiplying $x^{(i)}$ at the end due to the derivative.

The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

梯度下降的问题：可能（由于初始值的选取）下降至局部最优解。

α 的选取：过小：收敛太慢；过大：可能无法收敛甚至导致发散。

当选取的 θ_j 位于局部最低点，那么由于此处偏导数为 0，θ_j 在下一步更新不变。

α 的值不需改变：在变化较抖的地方导数较大，在靠近局部最低点的地方导数值会变小。

“Batch” Gradient Descent(批量梯度下降)：每一步需要使用所有训练样本。

What’s Next

Instead of using linear regression on just one input variable, we'll generalize and expand our concepts so that we can predict data with multiple input variables. Also, we'll solve for $\theta 0$ and $\theta 1$ exactly without needing an iterative function like gradient descent.

Vectorization

Matlab、octave、C、java 等语言，有相应的库用于数值计算，线性代数等。
使用这些库来提高算法的效率。

Vectorization example：

$h\theta(x) = \sum_{j=0}^{n} \theta_j x_j = \theta^T x$

1) unvectorized implementation:

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

2) Vectorized implementation:

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

简单高效

一、分类问题 classification

Logistic regression is a method for classifying data into discrete outcomes.

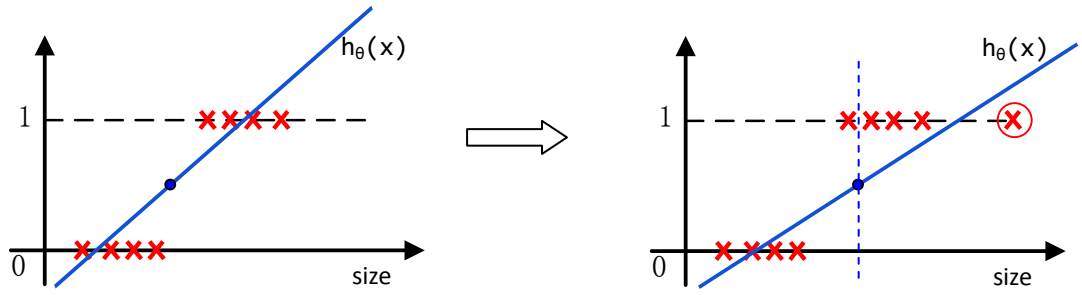
For example, we might use logistic regression to classify an email as spam or not spam.

$y \in \{0,1\}$ —— binary classification problem

0： “Negative class”
1： “Positive class”

多个分类的问题——multiclass classification problem。 例如： $y \in \{0,1,2,3\}$

考虑肿瘤分类的问题:



1） 增加的点并没有提供新的信息，但假设被改变
2） 产生远大于 1 和远小于 0 的预测数据，与实际数据范围不符。
→不适合使用线性回归

逻辑回归可以使 0≤$h_\theta(x)$≤1

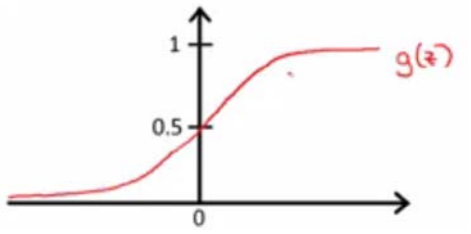
二、假设的表示

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

其中，$\boxed{g(z) = \frac{1}{1+e^{-z}}}$。 g 称为 **sigmoid function** （**S 型函数**）或 **logistic function**.

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

g 的形状：



约定 $h_\theta(x)$ 的含义：对于给定输入 x，y=1 的估计概率。

即：$h_\theta(x) = \mathrm{P}(y=1 \mid x;\theta)$ (probability that y = 1, given x, parameterized by θ)

三、Decision Boundary

根据预测的概率，如何决定输出的 y 何时为 1 何时为 0?

$h_\theta(x)$ $h_\theta(x)$

↳ y=1 if ≥ 0.5; y=0 if < 0.5

由于 g(z)在 z>0 时大于 0.5，在 z<0 时小于 0.5，等价于 $\theta^T x$ ≥0 时预测值为 1，$\theta^T x$ <0 时预测值为 0。

直线 $\theta^T x = 0$ 称为 **Decision Boundary**。

例：

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$，得到 $\theta = [-1 \quad 0 \quad 0 \quad 1 \quad 1]^T$

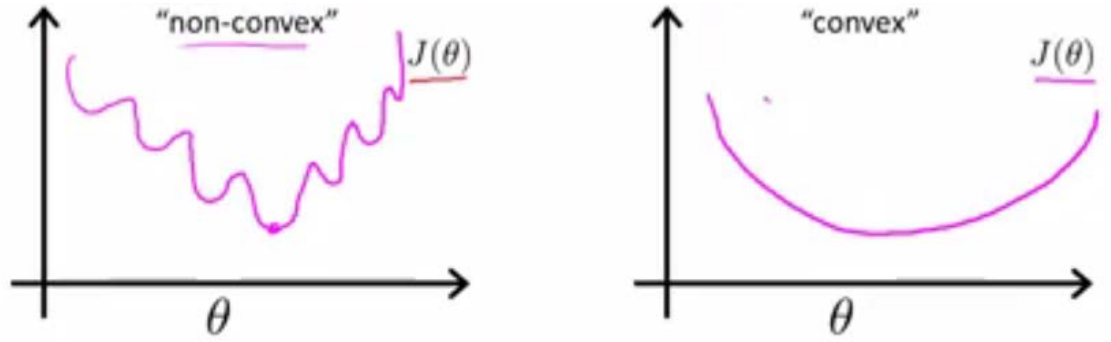
则当 $-1 + x_1^2 + x_2^2 \geq 0$ 时预测 y=1。 $\Rightarrow$ 平面上的圆

一、cost function

回忆线性回归的 cost function： $J(\theta)=\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(h_\theta(x^{(i)})-y^{(i)})^2$

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

如果对逻辑回归 $h_\theta(x)=\frac{1}{1+e^{-\theta^T x}}$ 使用这一 cost function，那么 $J(\theta)$ 将是参数 θ 的**非凸函数（non-convex function）**⇒多个局部最小值。



将逻辑回归的 cost function 设为：

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

具有性质：当 y=1， $h_\theta(x)$=1，则 Cost=0；

当 y=1， $h_\theta(x)\to 0$，则 $\mathrm{Cost}\to\infty$ 。

y=0 时相似。

二、simplified cost function and gradient descent

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

由于 $y\in\{0,1\}$，上面的 Cost function：

可写为等价形式：$\mathrm{Cost}(h_\theta(x),y)=-y\log(h_\theta(x))-(1-y)\log(1-h_\theta(x))$ 。

于是总的 Cost function 为：

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

$J(\theta)$ 是广泛使用的逻辑回归代价函数，并且 $J(\theta)$ 是一个凸函数。

则逻辑回归的过程为：找到 θ，使得 $J(\theta)$ 最小，来对新的输入 x 输出预测 $h_\theta(x)=\frac{1}{1+e^{-\theta^T x}}$ 。

⇒使用梯度下降来最小化 $J(\theta)$：

repeat{

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

（同时更新所有的 θ_j ）

}

将 $J(\theta)$ 与 $h_\theta(x)$ 带入可得以上等于：

repeat{

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

（同时更新所有的 θ_j ）（这里好像应该有 1/m，原幻灯片没有）

}

三、Advanced Optimization

Optimization algorithms：
- 梯度下降（Gradient decent）
- *共轭梯度算法（Conjugate gradient）
- *BFGS 变尺度法
- *L-BFGS 限制变尺度法

后三种更复杂的算法*，

优势：不需要手动选择 α -- 内部循环：line search algorithm
通常比梯度下降更快

缺点：更复杂

尽量使用现有库来进行数值计算！

Octave 中使用 optimize 库：

1)定义 costFunction：

function [jVal, gradient] = costFunction(theta)

...

2)设定 options：

options = optimset('GradObj', 'on', 'MaxIter', '100');

%'GradObj', 'on':设置梯度目标参数为打开-提供梯度; 'MaxIter','100': 设置迭代次数最多为 100

3)给定初始化 theta：

initialTheta = zeros(2,1) %至少是二维列向量

4)调用 fminunc：

[optTheta, functionVal, exitFlag]

= fminunc(@costFunction, initialTheta, options);

%optTheta 返回的最佳参数，functionVal 差距，exitFlag 是否收敛

使用 optimize 进行逻辑回归：

实现函数 costFunction：（theta 为 n+1 维列向量）

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ)];
    gradient(1) = [code to compute ∂/∂θ0 J(θ)];
    gradient(2) = [code to compute ∂/∂θ1 J(θ)];
    ⋮
    gradient(n+1) = [code to compute ∂/∂θn J(θ) ];
```

四、Multiclass Classification: One-vs-all

例：email 分类：work、friends、family、。。

weather：Sunny、Cloudy、Rain、Snow

=>可使用 One-vs-all 方法

思想：对没个 class 计算分类时，将其余类别的样本都作为 negative。

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

2)对于一个新的输入 x，所进行的预测是选择最大的 $h_\theta^{(i)}(x)$ 对应的 i

Regularization

-- prevent models from over-fitting the training data

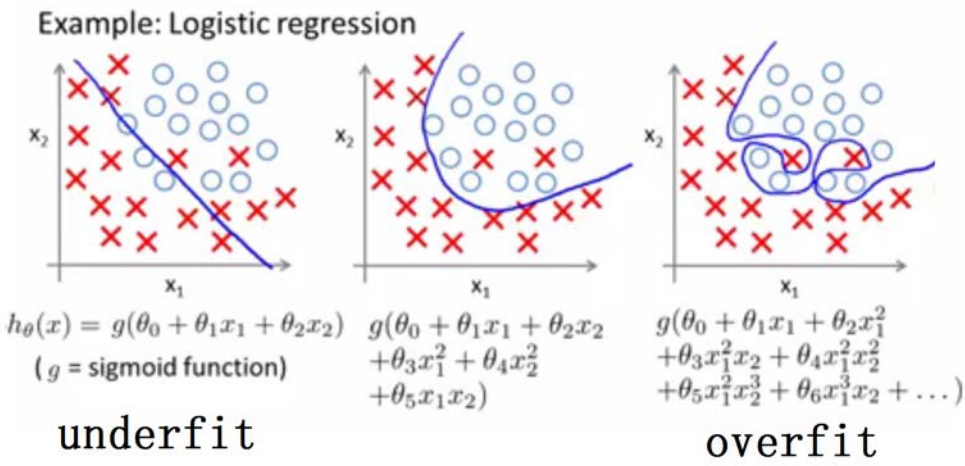
一、The Problem of Overfitting

欠拟合（under-fitting）、
high bias

过度拟合（Over-fitting）、
high variance(方差)

欠拟合：选取的特征过少（假设是有偏的），不能很好地拟合训练样本。

过度拟合：选取的特征过多，使得假设能很好地拟合训练集合，但是不能泛化（generalize）到新样本（预测新样本）。



解决过度拟合：

（绘出假设的图像，如果有很多扭曲的地方可能说明出现过度拟合。但对于很多个特征向量的情况不太适用：难以可视化）

1）减少特征的数量：可能会丢掉一些有用特征

- 手动选取特征
- Model selection algorithm -- 能够自动选取保留哪些特征

2）正则化（regularization）：保留所有的特征，但是减小参数的权重。对于很多特征的情形很有效。

二、Cost Function

思想：使一些参数很小，从而使得假设“简单”，减少过度拟合的可能。

例如：对于假设 $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$，要减小 θ_3 和 θ_4，可以在 cost function 中增加两项：

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

于是最小化 cost function 的过程会使得 θ_3 和 $\theta_4 \approx 0$。

对于一般的情况，我们事先不知道需要减少那些特征的影响，因此对每个参数都增加项：

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

（注意在上式中被正规化的项不含 θ_0，**即在 Matlab 中不含 theta(1)**）

λ：regularization parameter（正规化参数）-- 控制拟合训练数据与避免过度拟合之间的平衡。

三、Regularized Linear Regression

由新的 cost function： $J(\theta)=\frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2+\lambda\sum_{j=1}^{n}\theta_j{}^2]$

得到正规化的线性回归过程：

repeat{

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

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

（where $j=1,2,\ldots,n$）

}

对于正规方程方法，$\theta=(\mathbf{X}^T\mathbf{X}+\lambda\begin{bmatrix}0&0&\ldots&0\\0&1&&\ldots\\\ldots&&\ldots&0\\0&\ldots&0&1\end{bmatrix}_{(n+1)\times(n+1)})^{-1}\mathbf{X}^T\mathrm{y}$ 。

四、Regularized Logistic Regression

cost function：

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

新的逻辑回归过程：

repeat{

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

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

（where $j=1,2,\ldots,n$）

}

（$h_\theta(x)=\frac{1}{1+e^{-\theta^T x}}$，因此与线性回归并不是相同的算法）

Neural Networks: Representation

Di Zhao, 2016-4-5

zhaodi01@mail.ustc.edu.cn

1 Non-linear Hypotheses

考虑一个较复杂的分类问题，例如有100个特征。那么如果要包含二次方项，例如x_1^2, x_1x_2 等，需要计算$O(n^2)$个特征约5000项；如果要计算三次方项，则需要计算$O(n^3)$个特征，约130000项。例如计算机视觉问题，按照每个像素的亮度作为特征，则50×50分辨率就有2500个特征。

=> 使用非线性假设

2 Neurons and the Brain

Neural Networks: Origins - Algorithms that try to mimic the brain. Widely used in 80s and 90s; popularity diminished in late 90s.

Now: state-of-the-art technique for many applications.

neuro-rewiring experiments: 切断听觉和触觉皮质与相应器官的连接后，与视觉器官相连，则这些皮质最终会学会处理视觉信号。

同一块脑组织可学会处理听觉、视觉、触觉信号=> 同一个算法也许可以处理这些不同任务

生物学的神经元（Figure 1）

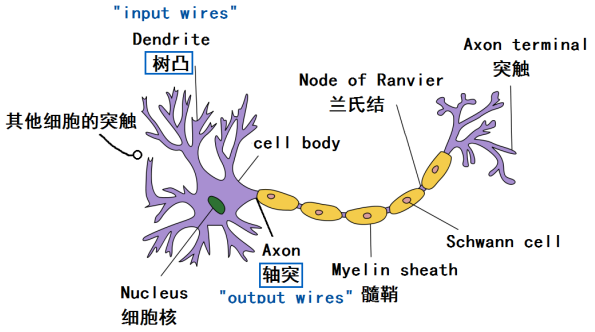


Figure 1: Neuron

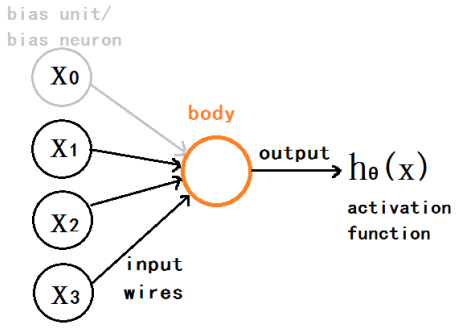


Figure 2: Neuron model: logistic unit

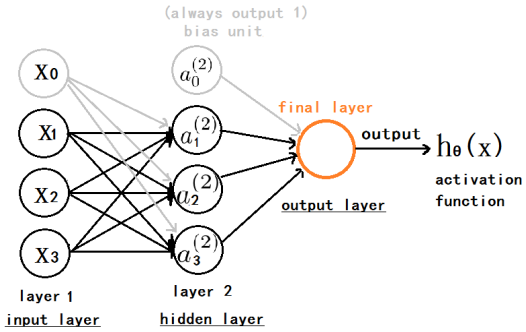


Figure 3: Neuron Network

3 Model Representation

在计算机实现上，将一个神经元建模为一个逻辑单元（Figure 2）.

其中，$x = [x_0\ x_1\ x_2\ x_3]^T$, $\theta = [\theta_0\ \theta_1\ \theta_2\ \theta_3]^T$。parameter θ 在神经网络中也称为"weights".

神经网络就是将单个的神经元紧密联系在一起（Figure 3）.

记号：

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

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

则：

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

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

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

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

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

因此，对于Figure 3中的例子，有3个输入单元、3个hidden units，$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$.

3.1 Forward propagation - vectorized

令 $a^{(1)} = x = [x_0\ x_1\ x_2\ x_3]^T$, 令

$$z^{(2)} = \Theta^{(1)}x = \Theta^{(1)}a^{(1)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \begin{bmatrix} \Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3 \\ \Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3 \\ \Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3 \end{bmatrix}$$

（上标表示相关的layer）。

则$a^{(2)} = g(z^{(2)}) \in \mathbb{R}^3$.($g$作用于矩阵每个元素)

Add $a_0^{(2)} = 1$ (则$a^{(2)} \in \mathbb{R}^4$):

于是$z^{(3)} = \Theta^{(2)}a^{(2)}$,
$h_\Theta(x) = a^{(3)} = g(z^{(3)})$.
这一过程称为forward propagation。

3.2 Neural network learning its own features

从Figure 3中可以看到，神经网络和逻辑回归比较相似。区别是逻辑回归中使用原有的特征作为输入；而神经网络使用训练结果前一层的作为下一层的输入 => learn its own features. 通过这种方式可以学习到一些更复杂有趣的特征，得到更好的假设。

network *architecture*: 神经元之间如何连接。(分为几层、每层几个节点)

Neural Networks: Representation - Applications

Di Zhao, 2016-4-9

zhaodi01@mail.ustc.edu.cn

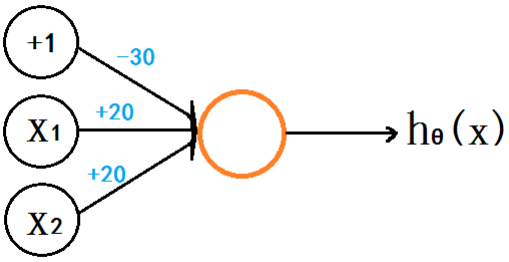
1 Non-linear classification example: XOR/XNOR

$x_1, x_2 \in \{0, 1\}$, 实现函数:
$y = x_1$ **XOR** x_2,
$y = x_1$ **NXOR** $x_2 = NOT(x_1$ **XOR** $x_2)$

1.1 example: AND

已知sigmoid function的特点：当$z \geqslant 4.0$时$g(z) \approx 1.0$，当$z \leqslant -4.0$时$g(z) \approx 0$。

构建两层的Nuron（input和output），第二层只有$a_1^{(1)}$一个节点。且$\Theta_1^{(1)} = [-30, 20, 20]^T$。则$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$。（Figure 1）



可以得到真值表

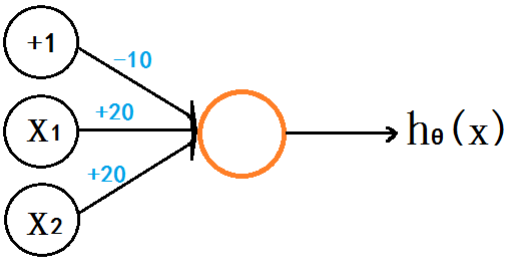
x_1	x_2	z	$h_\Theta(x)$
0	0	-30	≈ 0
0	1	-10	≈ 0
1	0	-10	≈ 0
1	1	10	≈ 1

Figure 1: AND function

从真值表可以看出，通过图1的神经网络实现了一个**AND**函数。

1.2 example: OR

类似地，可以构造**OR**函数：



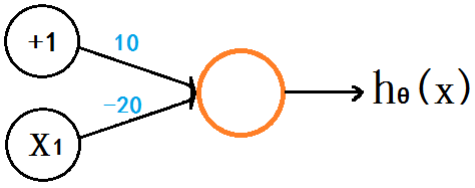
x_1	x_2	z	$h_\Theta(x)$
0	0	-10	≈ 0
0	1	10	≈ 1
1	0	10	≈ 1
1	1	30	≈ 1

Figure 2: OR function

从真值表可以看出，通过图2的神经网络实现了一个**OR**函数。

1.3 example: NOT

函数NOT的构造：



x_1	z	$h_\Theta(x)$
0	10	≈ 1
1	-10	≈ 0

Figure 3: NOT function

1.4 Put together: x_1 NXOR x_2

注意到：x_1 **XOR** x_2 $=$ (x_1 **OR** x_2) **AND** (**NOT** (x_1 **AND** x_2)),
x_1 **NXOR** x_2 $=$ (x_1 **NOR** x_2) **OR** (x_1 **AND** x_2)

因此可以组合得到**NXOR**函数(Figure 4)，以及真值表：

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

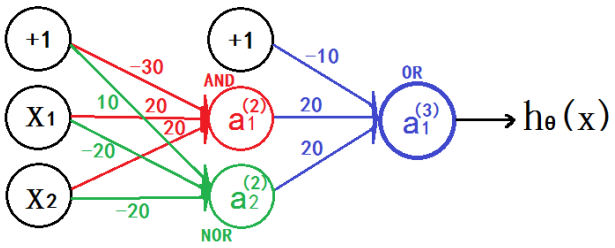


Figure 4: NXOR function

2 Multiclass Classification: one-vs-all

假设我们希望计算一个分类问题，有4个要计算的类别，那么$h_\Theta(x) \in \mathbb{R}^4$. 变成了一个向量。并且我们希望 $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$，$h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$，等等。（每个元素的值代表是否属于该类别）

相应地，训练集：$(x^{(1)}, y^{(1)})$，$(x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})$ 中，y不再是属于集合$\{1, 2, 3, 4\}$的单个值；

$$y^{(i)} \in \{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \}。$$

神经网络的结构类似于：