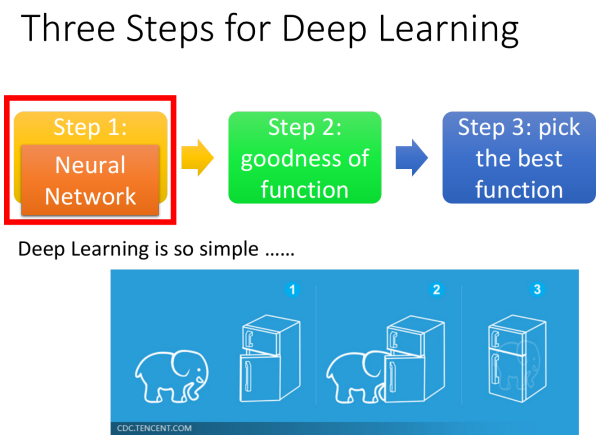
Deep Learning

Ups and downs of Deep Learning

- 1958：Perceptron(linear model)，感知机的提出
  - 和Logistic Regression类似，只是少了sigmoid的部分
- 1969：Perceptron has limitation，from MIT
- 1980s：Multi-layer Perceptron，多层感知机
  - 和今天的DNN很像
- 1986：Backpropagation，反向传播
  - Hinton propose的Backpropagation
  - 存在problem：通常超过3个layer的neural network，就train不出好的结果
- 1989: 1 hidden layer is “good enough”，why deep?
  - 有人提出一个理论：只要neural network有一个hidden layer，它就可以model出任何的function，所以根本没有必要叠加很多个hidden layer，所以Multi-layer Perceptron的方法又坏掉了，这段时间Multi-layer Perceptron这个东西是受到抵制的
- 2006：RBM initialization(breakthrough)：Restricted Boltzmann Machine，受限玻尔兹曼机
  - Deep learning -> another Multi-layer Perceptron ？在当时看来，它们的不同之处在于在做gradient descent的时候选取初始值的方法如果是用RBM，那就是Deep learning；如果没有用RBM，就是传统的Multi-layer Perceptron
  - 那实际上呢，RBM用的不是neural network base的方法，而是graphical model，后来大家试验得多了发现RBM并没有什么太大的帮助，因此现在基本上没有人使用RBM做initialization了
  - RBM最大的贡献是，它让大家重新对Deep learning这个model有了兴趣(石头汤的故事)
- 2009：GPU加速的发现
- 2011：start to be popular in speech recognition，语音识别领域
- 2012：win ILSVRC image competition，Deep learning开始在图像领域流行开来

实际上，Deep learning跟machine learning一样，也是“大象放进冰箱”的三个步骤：

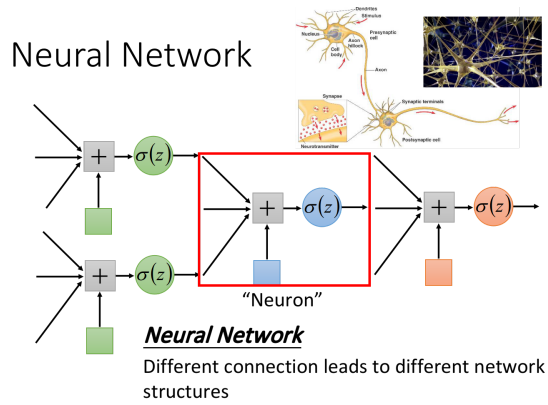
在Deep learning的step1里define的那个function，就是neural network



Neural Network

concept

把多个Logistic Regression前后connect在一起，然后把一个Logistic Regression称之为neuron，整个称之为neural network



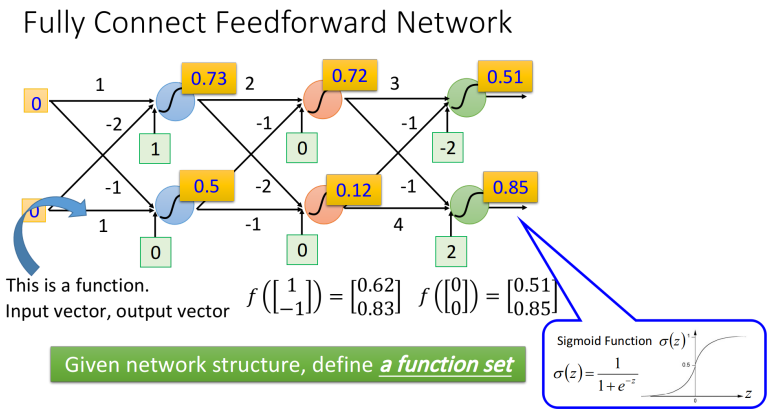
Network parameter θ: all the weights and biases in the "neurons"

我们可以用不同的方法连接这些neuron，就可以得到不同的structure，neural network里的每一个Logistic Regression都有自己的weight和bias，这些weight和bias集合起来，就是这个network的parameter，我们用θ来描述

Fully Connect Feedforward Network

那该怎么把它们连接起来呢？这是需要你手动去设计的，最常见的连接方式叫做**Fully Connect Feedforward Network(全连接前馈网络)**

如果一个neural network的参数weight和bias已知的话，它就是一个function，它的input是一个vector，output是另一个vector，这个vector里面放的是样本点的feature，vector的dimension就是feature的个数



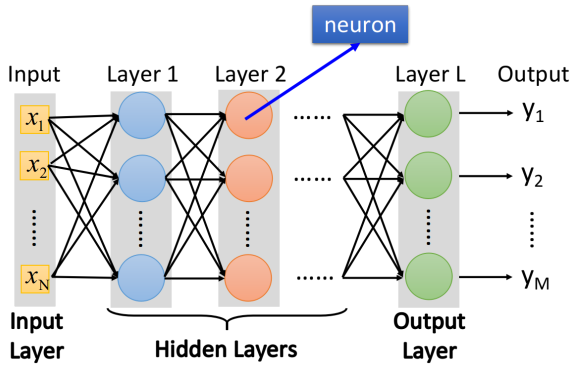
如果今天我们还不知道参数，只是定出了这个network的structure，只是决定好这些neuron该怎么连接在一起，这样的一个network structure其实是define了一个function set(model)，我们给这个network设不同的参数，它就变成了不同的function，把这些可能的function集合起来，我们就得到了一个function set

只不过我们用neural network决定function set的时候，这个function set是比较大的，它包含了很多原来你做Logistic Regression、做linear Regression所没有办法包含的function

下图中，每一排表示一个layer，每个layer里面的每一个球都代表一个neuron

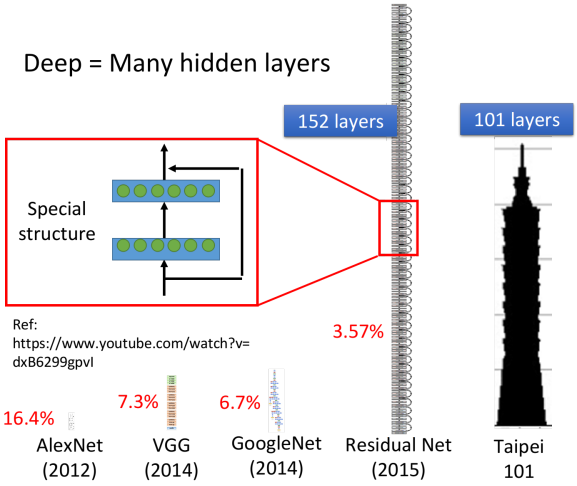
- layer和layer之间neuron是两两互相连接的，layer 1的neuron output会连接给layer 2的每一个neuron作为input
- 对整个neural network来说，它需要一个input，这个input就是一个feature的vector，而对layer 1的每一个neuron来说，它的input就是input layer的每一个dimension
- 最后那个layer L，由于它后面没有接其它东西了，所以它的output就是整个network的output
- 这里每一个layer都是有名字的

    - input的地方，叫做**input layer**，输入层(严格来说input layer其实不是一个layer，它跟其他layer不一样，不是由neuron所组成的)
    - output的地方，叫做**output layer**，输出层
    - 其余的地方，叫做**hidden layer**，隐藏层
- 每一个neuron里面的sigmoid function，在Deep Learning中被称为**activation function**(激励函数)，事实上它不见得一定是sigmoid function，还可以是其他function(sigmoid function是从Logistic Regression迁移过来的，现在已经较少在Deep learning里使用了)
- 有很多层layers的neural network，被称为**DNN(Deep Neural Network)**



因为layer和layer之间，所有的neuron都是两两连接，所以它叫Fully connected的network；因为现在传递的方向是从layer 1->2->3，由后往前传，所以它叫做Feedforward network

那所谓的deep，是什么意思呢？有很多层hidden layer，就叫做deep，具体的层数并没有规定，现在只要是neural network base的方法，都被称为Deep Learning，下图是一些model使用的hidden layers层数举例

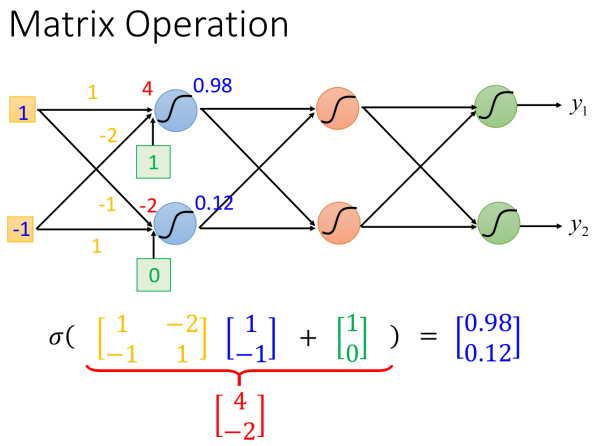


你会发现使用了152个hidden layers的Residual Net，它识别图像的准确率比人类还要高当然它不是使用一般的Fully Connected Feedforward Network，它需要设计特殊的special structure才能训练这么深的network

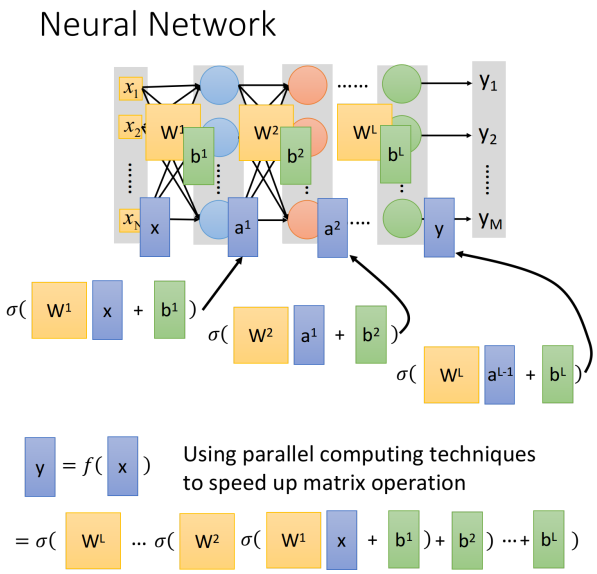
Matrix Operation

network的运作过程，我们通常会用Matrix Operation来表示，以下图为例，假设第一层hidden layers的两个neuron，它们的weight分别是$w_1 = 1, w_2 = -2, w_1' = -1, w_2' = 1$，那就可以把它们排成一个matrix：$\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}$，而我们的input又是一个2*1的vector：$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$，将w和x相乘，再加上bias的vector：$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$，就可以得到这一层的vector z，再经过activation function得到这一层的output：
(activation function可以是很多类型的function，这里还是用Logistic Regression迁移过来的sigmoid function作为运算)

$$\sigma\left(\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}\right)=\sigma\left(\begin{bmatrix}4\\ -2\end{bmatrix}\right)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$



这里我们把所有的变量都以matrix的形式表示出来，注意W^i的matrix，每一行对应的是一个neuron的weight，行数就是neuron的个数，而input x，bias b和output y都是一个列向量，行数就是feature的个数(也是neuron的个数，neuron的本质就是把feature transform到另一个space)

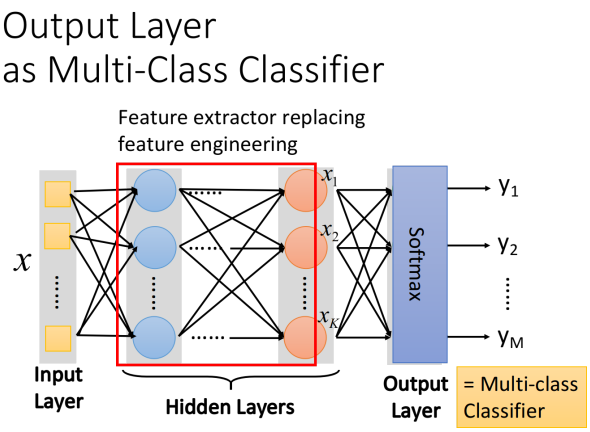


把这件事情写成矩阵运算的好处是，可以用GPU加速，GPU对matrix的运算是比CPU要来的快的，所以我们写neural network的时候，习惯把它写成matrix operation，然后call GPU来加速它

Output Layer

我们可以把hidden layers这部分，看做是一个**feature extractor(特征提取器)**，这个feature extractor就replace了我们之前手动做feature engineering，feature transformation这些事情，经过这个feature extractor得到的$x_1, x_2, \ldots, x_k$就可以被当作一组新的feature

output layer做的事情，其实就是把它当做一个**Multi-class classifier**，它是拿经过feature extractor转换后的那一组比较好的feature(能够被很好地separate)进行分类的，由于我们把output layer看做是一个Multi-class classifier，所以我们会在最后一个layer加上**softmax**

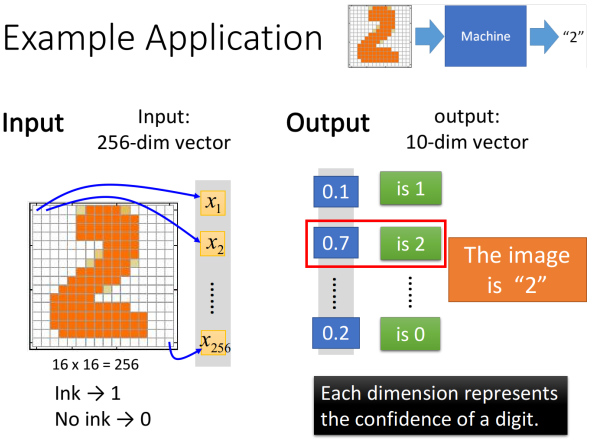


Example Application

Handwriting Digit Recognition

这里举一个手写数字识别的例子，input是一张image，对机器来说一张image实际上就是一个vector，假设这是一张16*16的image，那它有256个pixel，对machine来说，它是一个256维的vector，image中的每一个都对应到vector中的一个dimension，简单来说，我们把黑色的pixel的值设为1，白色的pixel的值设为0

而neural network的output，如果在output layer使用了softmax，那它的output就是一个突出极大值的Probability distribution，假设我们的output是10维的话(10个数字，0~9)，这个output的每一维都对应到它可能是某一个数字的几率，实际上这个neural network的作用就是计算这张image成为10个数字的几率各自有多少，几率最大(softmax突出极大值的意义所在)的那个数字，就是机器的预测值



在这个手写字体识别的demo里，我们唯一需要的就是一个function，这个function的input是一个256的vector，output是一个10维的vector，这个function就是neural network(这里我们用简单的Feedforward network)

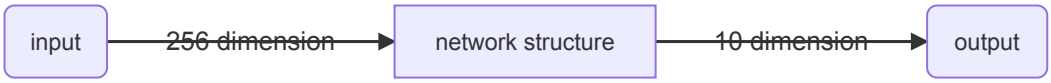
input固定为256维，output固定为10维的feedforward neural network，实际上这个network structure就已经确定了一个function set(model)的形状，在这个function set里的每一个function都可以拿来做手写数字识别，接下来我们要做的事情是用gradient descent去计算出一组参数，挑一个最适合拿来做手写数字识别的function

注：input、output的dimension，加上network structure，就可以确定一个model的形状，前两个是容易知道的，而决定这个network的structure则是整个Deep Learning中最为关键的步骤

所以这里很重要的一件事情是，我们要对network structure进行design，之前在做Logistic Regression或者是linear Regression的时候，我们对model的structure是没有什么好设计的，但是对neural network来说，我们现在已知的constraint只有input是256维，output是10维，而中间要有几个hidden layer，每个layer要有几个neuron，都是需要我们自己去设计的，它们近乎是决定了function set长什么样子

如果你的network structure设计的很差，这个function set里面根本就没有好的function，那就会像大海捞针一样，结果针并不在海里(滑稽

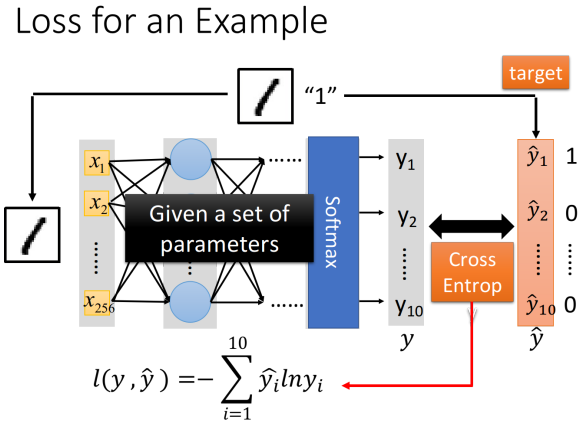
Step 1：Neural Network



input 256维，output 10维，以及自己design的network structure =》function set(model)

Step 2：Goodness of function

定义一个function的好坏，由于现在我们做的是一个Multi-class classification，所以image为数字1的label “1”告诉我们，现在的target是一个10维的vector，只有在第一维对应数字1的地方，它的值是1，其他都是0

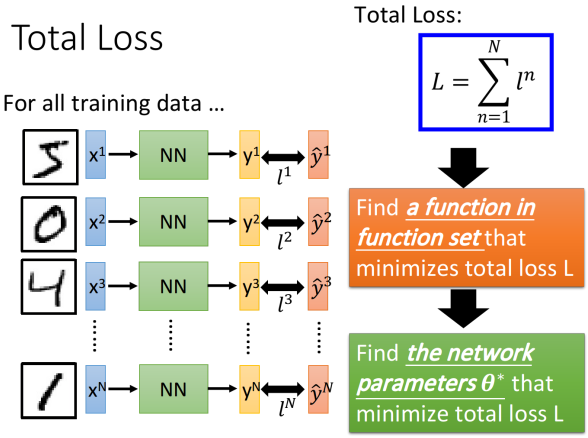


input这张image的256个pixel，通过这个neural network之后，会得到一个output，称之为y；而从这张image的label中转化而来的target，称之为$\hat{y}$，有了output y和target $\hat{y}$之后，要做的事情是计算它们之间的cross entropy(交叉熵)，这个做法跟我们之前做Multi-class classification的时候是一模一样的

$$Cross\ Entropy: l(y,\hat{y}) = -\sum_{i=1}^{10} \hat{y}_i ln y_i$$

Step 3：Pick the best function

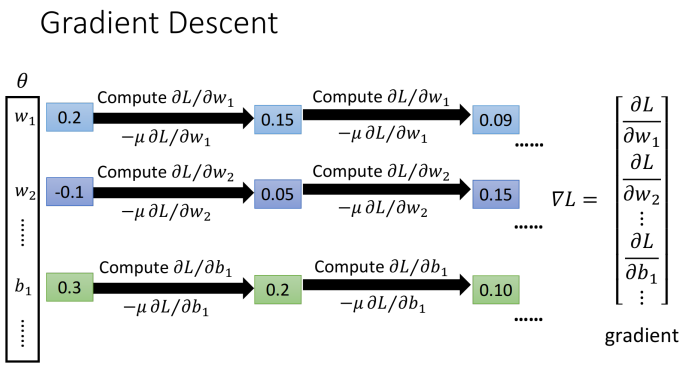
接下来就去调整参数，让这个cross entropy越小越好，当然整个training data里面不会只有一笔data，你需要把所有data的cross entropy都sum起来，得到一个total loss $L = \sum_{n=1}^{N} l^n$，得到loss function之后你要做的事情是找一组network的parameters：θ^*，它可以minimize这个total loss，这组parameter对应的function就是我们最终训练好的model



那怎么去找这个使total loss minimize的θ^*呢？使用的方法就是我们的老朋友——**Gradient Descent**

实际上在deep learning里面用gradient descent，跟在linear regression里面使用完全没有什么差别，只是function和parameter变得更复杂了而已，其他事情都是一模一样的

现在你的θ里面是一大堆的weight、bias参数，先random找一个初始值，接下来去计算每一个参数对total loss的偏微分，把这些偏微分全部集合起来，就叫做gradient，有了这些偏微分以后，你就可以更新所有的参数，都减掉learning rate乘上偏微分的值，这个process反复进行下去，最终找到一组好的参数，就做完deep learning的training了



toolkit

你可能会问，这个gradient descent的function式子到底是长什么样子呢？之前我们都是一步一步地把那个算式推导出来的，但是在neural network里面，有成百上千个参数，如果要一步一步地人工推导并求微分的话是比较困难的，甚至是不可行的

其实，在现在这个时代，我们不需要像以前一样自己去implement Backpropagation(反向传播)，因为有太多太多的toolkit可以帮你计算Backpropagation，比如**tensorflow、pytorch**

注：Backpropagation就是算微分的一个比较有效的方式

something else

所以，其实deep learning就是这样子了，就算是alpha go，也是用gradient descent train出来的，可能在你的想象中它有多么得高大上，实际上就是在用gradient descent这样朴素的方法

有一些常见的问题：

Q：有人可能会问，机器能不能自动地学习network的structure?

- 其实是可以的，基因演算法领域是有很多的technique是可以让machine自动地去找出network structure，只不过这些方法目前没有非常普及

Q：我们可不可以自己去design一个新的network structure，比如说可不可以不要Fully connected layers(全连接层)，自己去DIY不同layers的neuron之间的连接?

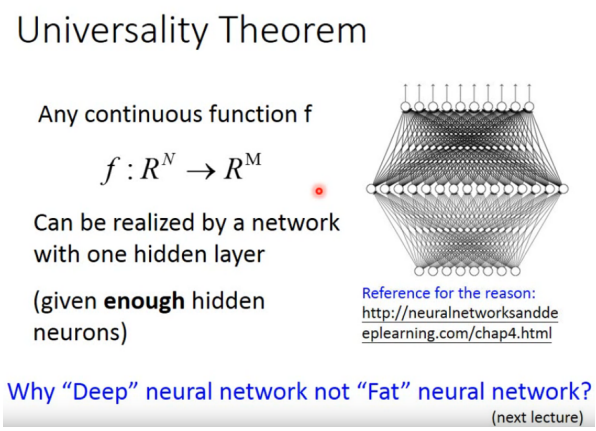
- 当然可以，一个特殊的接法就是CNN(Convolutional Neural Network)，即卷积神经网络，这个下一章节会介绍

Why Deep?

最后还有一个问题，为什么我们要deep learning？一个很直觉的答案是，越deep，performance就越好，一般来说，随着deep learning中的layers数量增加，error率不断降低

但是，稍微有一点machine learning常识的人都不会觉得太surprise，因为本来model的parameter越多，它cover的function set就越大，它的bias就越小，如果今天你有足够多的training data去控制它的variance，一个比较复杂、参数比较多的model，它performance比较好，是很正常的，那变deep有什么特别了不起的地方？

甚至有一个理论是这样说的，任何连续的function，它input是一个N维的vector，output是一个M维的vector，它都可以用一个hidden layer的neural network来表示，只要你这个hidden layer的neuron够多，它可以表示成任何的function，既然一个hidden layer的neural network可以表示成任何的function，而我们在做machine learning的时候，需要的东西就只是一个function而已，那做deep有什么特殊的意义呢？



所以有人说，deep learning就只是一个噱头而已，因为做deep感觉比较潮，如果你只是增加neuron把它变宽，变成fat neural network，那就感觉太"虚弱"了，所以我们要做deep learning，给它增加layers而不是增加neuron：DNN(deep) is better than FNN(fat)

真的是这样吗？后面的章节会解释这件事情

Design network structure V.s. Feature Engineering

> 下面聊一些经验之谈

其实network structure的design是一件蛮难的事情，我们到底要怎么决定layer的数目和每一个layer的neuron的数目呢？其实这个只能够凭着经验和直觉、多方面的尝试，有时候甚至会需要一些domain knowledge(专业领域的知识)，从非deep learning的方法到deep learning的方法，并不是说machine learning比较简单，而是我们把一个问题转化成了另一个问题

本来不是deep learning的model，要得到一个好的结果，往往需要做feature engineering(特征工程)，也就是做feature transform，然后找一组好的feature；一开始学习deep learning的时候，好像会觉得deep learning的layers之间也是在做feature transform，但实际上在做deep learning的时候，往往不需要一个好的feature ，比如说在做影像辨识的时候，你可以把所有的pixel直接丢进去，但是在过去做图像识别，你是需要对图像抽取出一些人定的feature出来的，这件事情就是feature transform，但是有了deep learning之后，你完全可以直接丢pixel进去硬做

但是，今天deep learning制造了一个新的问题，它所制造的问题就是，你需要去design network的structure，所以**你的问题从本来的如何抽取feature转化成怎么design network structure**，所以deep learning是不是真的好用，取决于你觉得哪一个问题比较容易

如果是影响辨识或者是语音辨识的话，design network structure可能比feature engineering要来的容易，因为，虽然我们人都会看、会听，但是这件事情，它太过潜意识了，它离我们意识的层次太远，我们无法意识到，我们到底是怎么做语音辨识这件事情，所以对人来说，你要抽一组好的feature，让机器可以很方便地用linear的方法做语音辨识，其实是很难的，因为人根本就不知道好的feature到底长什么样子；所以还不如design一个network structure，或者是尝试各种network structure，让machine自己去找出好的feature，这件事情反而变得比较容易，对影像来说也是一样的

有这么一个说法：deep learning在NLP上面的performance并没有那么好。语音辨识和影像辨识这两个领域是最早开始用deep learning的，一用下去进步量就非常地惊人，比如错误率一下子就降低了20%这样，但是在NLP上，它的进步量似乎并没有那么惊人，甚至有很多做NLP的人，现在认为说deep learning不见得那么work，这个原因可能是，人在做NLP这件事情的时候，由于人在文字处理上是比较强的，比如叫你设计一个rule去detect一篇document是正面的情绪还是负面的情绪，你完全可以列

表，列出一些正面情绪和负面情绪的词汇，然后看这个document里面正面情绪的词汇出现的百分比是多少，你可能就可以得到一个不错的结果。所以NLP这个task，对人来说是比较容易设计rule的，你设计的那些ad-hoc(特别的)的rule，往往可以得到一个还不错的结果，这就是为什么deep learning相较于NLP传统的方法，觉得没有像其他领域一样进步得那么显著(但还是有一些进步的)

长久而言，可能文字处理中会有一些隐藏的资讯是人自己也不知道的，所以让机器自己去学这件事情，还是可以占到一些优势，只是眼下它跟传统方法的差异看起来并没有那么的惊人，但还是有进步的