

深度学习的发展过程：

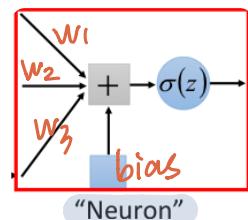
- 感知机 (Perceptron) (线性模型)
 - 多层感知机 (Multi-layer perceptron) 和现在的DNN已经很像了.
 - 反向传播 (Backpropagation) 多于3个隐藏层的NN就train不出好结果.
 - 增加更多隐藏层到NN中 但其实只有3个隐藏层已经可以model所有function.
 - 受限玻尔兹曼机 (Restricted Boltzmann Machine) 它很复杂, 但效果并没有很好
 - GPU发展 为矩阵计算加速
 - DL开始在语音识别 (Speech Recognition) 领域流行.
 - 在 ILSVRC image competition 赛能. 在图像领域流行.
- 当时如何区分 Deep Learning 和 Multi-layer Perception?
gradient descent 时初值选取 | RBM = DL
不用 RBM: Multi-layer P.

DL的3个步骤:

- ① 定义一个函数集合
= 网络结构 (goodness of function)
② 看哪个函数适合的程度
↳ 有几层 layer?
每层有几个 neurons?
- ③ 选最好的那个函数.

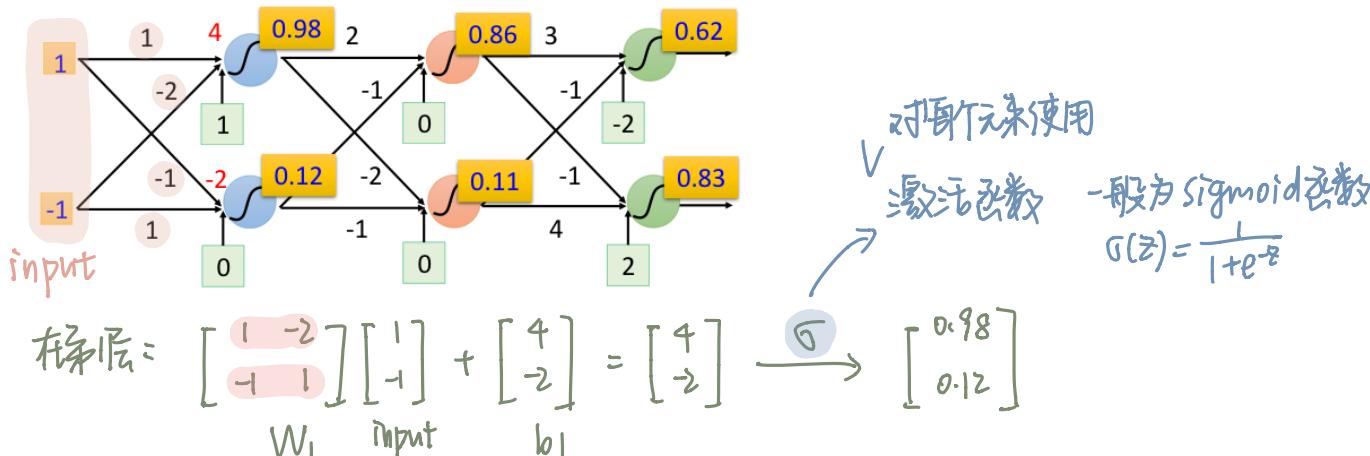
神经网络相当于一个 input-output 均为 vector 的 function.

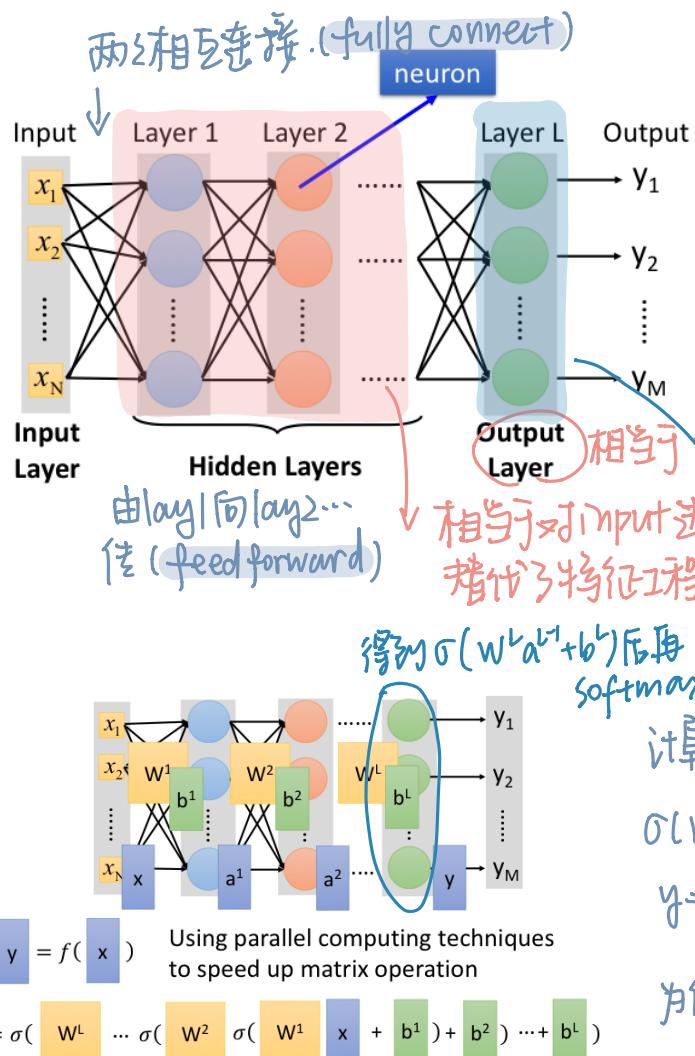
里面有 many neurons (每一个 neuron 为一个 logistic regression), 用不同方法连接.
参数即为所有 neurons 中的 weights 和 biases.
即网络结构.



① 定义一个函数集合

全连接前馈神经网络 (Fully Connected Feedforward Network)





这种有很多层的 NN 被称为 DNN
隐藏

现在只要是 NN base 的方法都被称为 DL.

输入为提取好的特征 (隐藏层的结果)
↓ 输出为分类结果 (几类就有几个)

相当于一个分类器 (multi-class classifier)

相当于对 input 进行特征提取 (feature extractor)

为了满足分类器结果和为1, 会将 output layer 结果加上一个 softmax 再输出 (y1~yM)

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

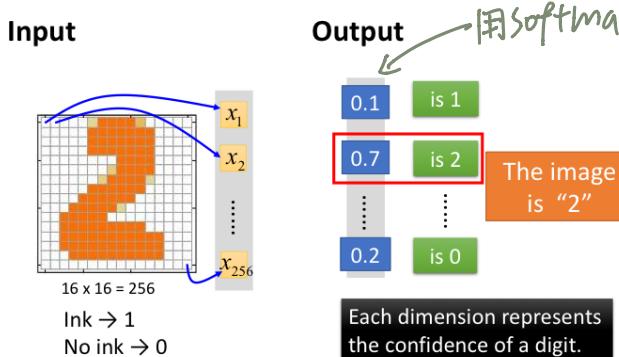
计算过程:

$$\sigma(w^L x + b^L) \rightarrow \sigma(w^2 a^1 + b^2) \rightarrow \dots \rightarrow \sigma(w^L a^{L-1} + b^L) = y$$

$$y = \sigma(w^L \dots \sigma(w^2 \sigma(w^1 x + b^1) + b^2) \dots + b^L)$$

为何写成矩阵运算? 可用 GPU 加速.

例子: 数字识别



这里 input 为 256 维 vector, output 为 10 维 vector,
定好维度和结构 (fully connect feedforward) 相当于
确定好了参数集合, 接下来用 gradient descent
算出参数即得到一确定的参数.

由 ML → DL, 难点由如何特征工程 → 如何设计网络的结构

在语音识别中 ML
表现不好因为人类
不清楚哪些特征好.

(试错+直觉)

在 ML 中实现没有想象中
好, 因为规则易设计, ML 已
可取得较好结果.

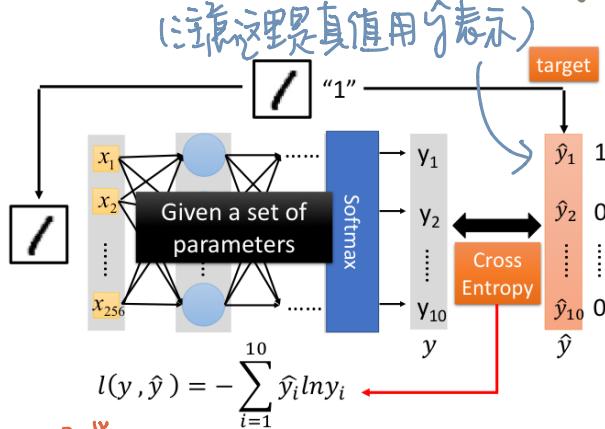
(现在也有自动找结构的 NN: Evolutionary Artificial Neural Networks)

② 看参数的好坏 (等于看一组参数的好坏)

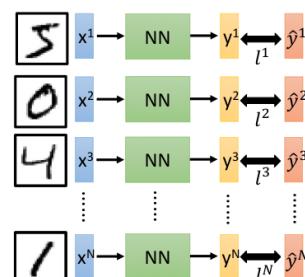
$$\text{Cross Entropy} : C(y, \hat{y}) = -\sum_{i=1}^{10} \hat{y}_i \ln y_i$$

$$\text{Total loss: } L(\theta) = \sum_{n=1}^N C_n(\theta) \quad \begin{array}{l} \text{将所有 data 用 \theta 得到的交叉熵加和.} \\ \theta \text{ 为所有 neurons 的参数.} \end{array}$$

e.g.



For all training data ...



③ 选最好的参数

找 θ : $\theta^* = \arg \min_{\theta} L(\theta)$ 用 gradient descent 找.

Universality Theorem: 任何一个连续函数都可以用一个 (neuron 足够多的) 一层网络表示.