人工智能

有监督(分类)算法

Supervised Learning I-ANN

中山大学 计算机学院

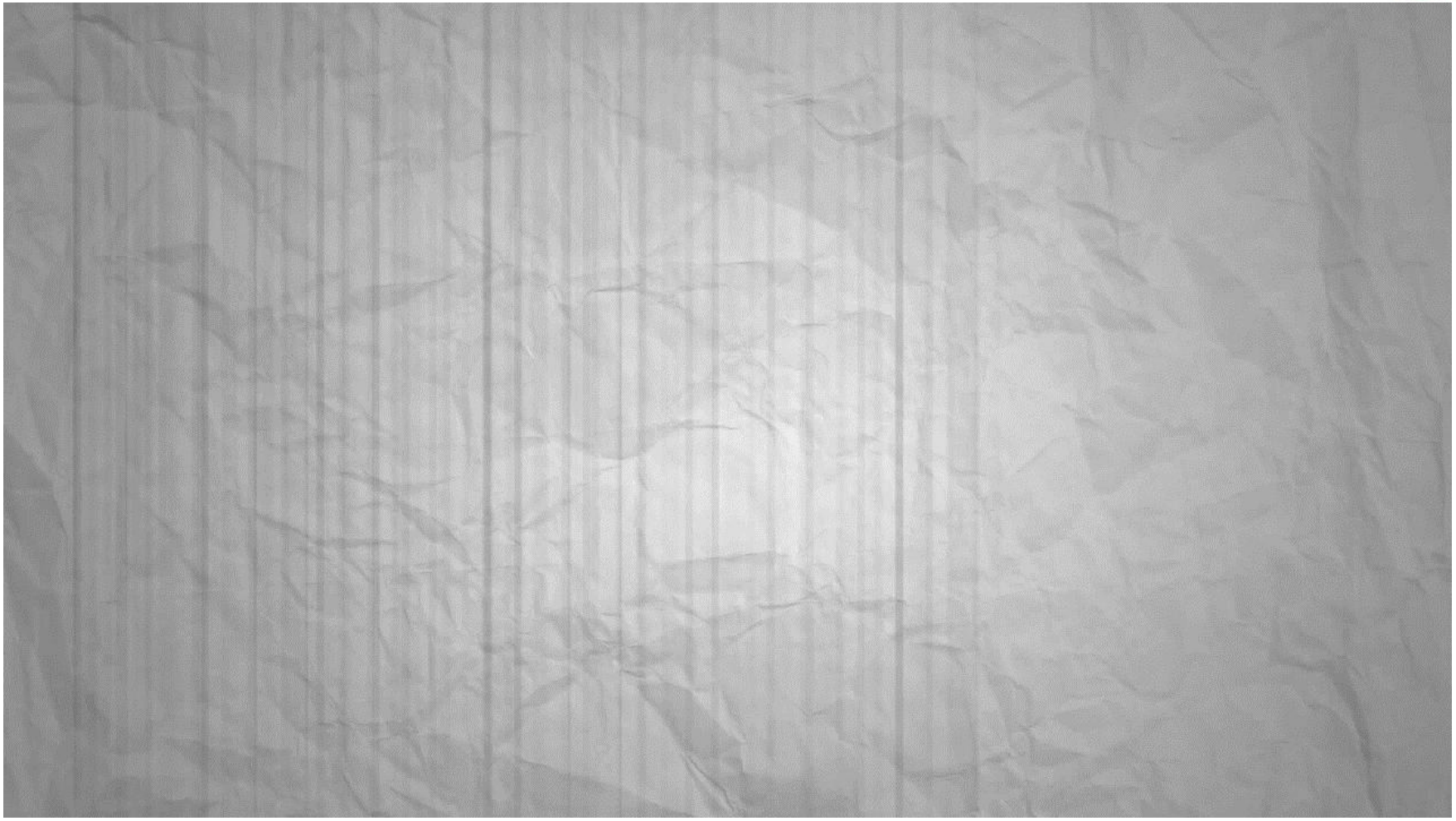
中山大學
SUN YAT-SEN UNIVERSITY

ANN history

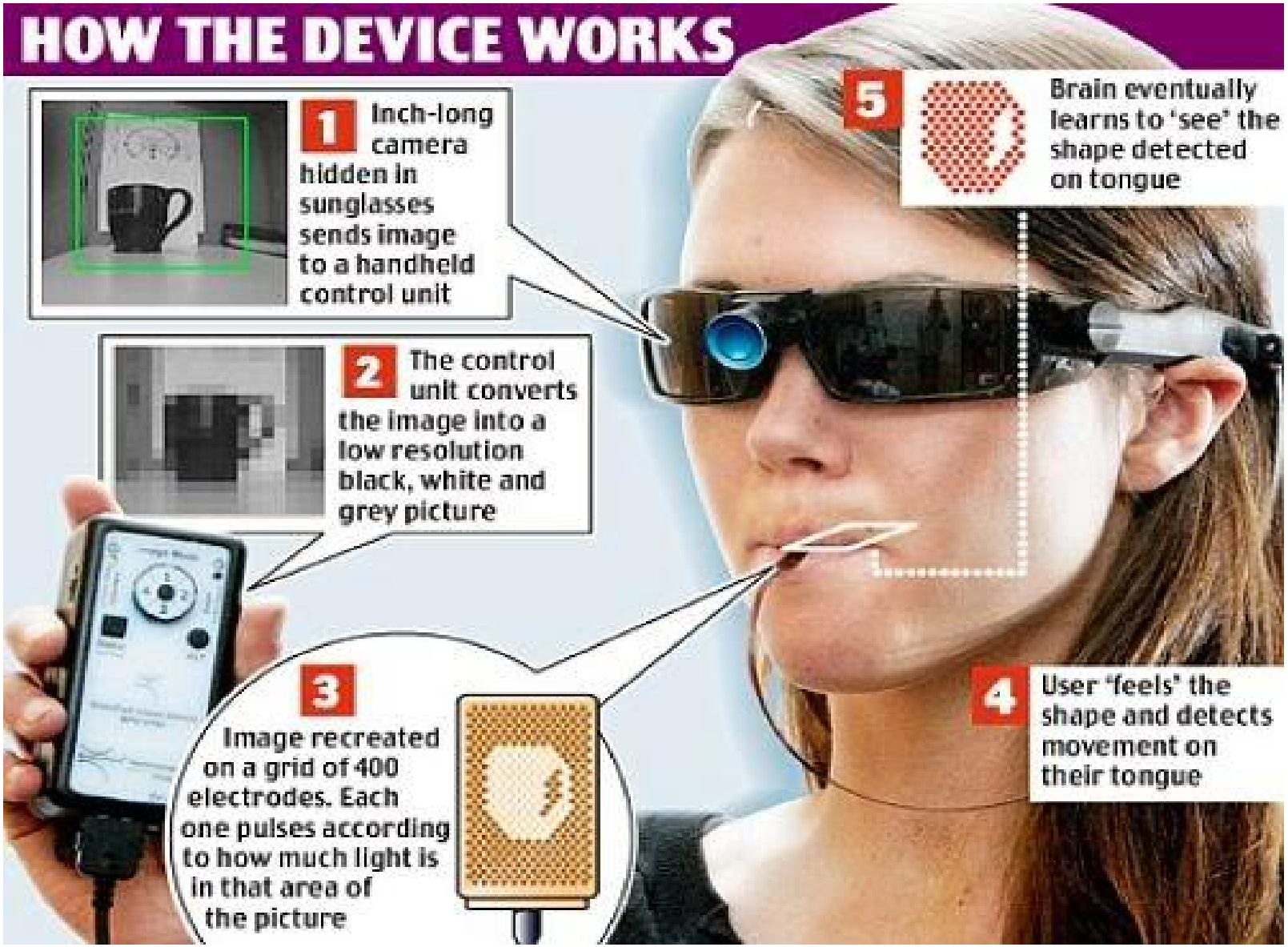
Neural Network History

- Origins: Algorithms that try to mimic the brain
- Widely used in 80s and early 90s; popularity diminished in late 90s
- Resurgence: State-of-the-art technique for many applications in recent years

Brain vs ANN



when you think of the brain…



when you think of the brain…

- 人脑中的神经网络是一个非常复杂的组织，由很多具有适应性的简单同构单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界所作出的交互反应

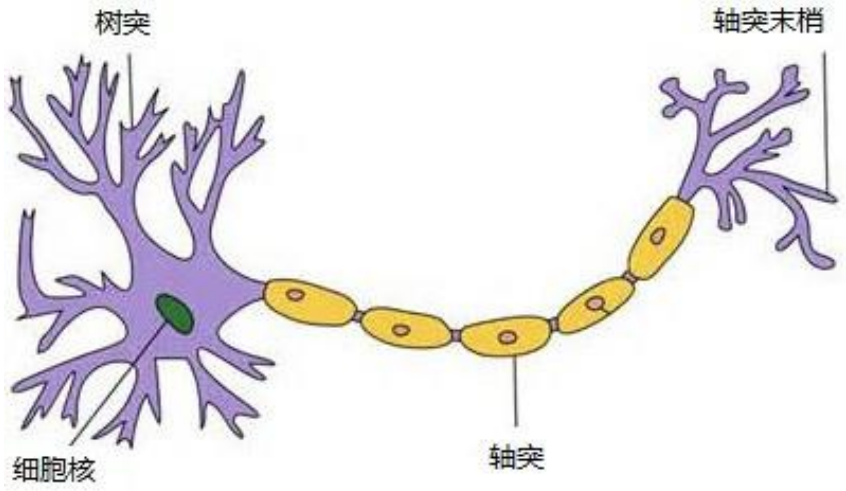
- 这一简单的结构就是**神经元**

人脑神经网络

- 成人的大脑中估计有 1000 亿个神经元

神经元 - 引入

- 1904年生物学家已经知晓**神经元**的组成结构
- 一个神经元通常具有多个**树突**，主要用来接受传入信息；而**轴突**只有一条，轴突尾端有许多轴突末梢可以给其他多个神经元传递信息。轴突末梢跟其他神经元的树突产生连接，从而传递信号。这个连接的位置在生物学上叫做“**突触**”。



神经元结构图

神经元 - 引入

- 1943年，心理学家McCulloch和数学家Pitts参考了生物神经元的结构，发表了抽象的神经元模型MP

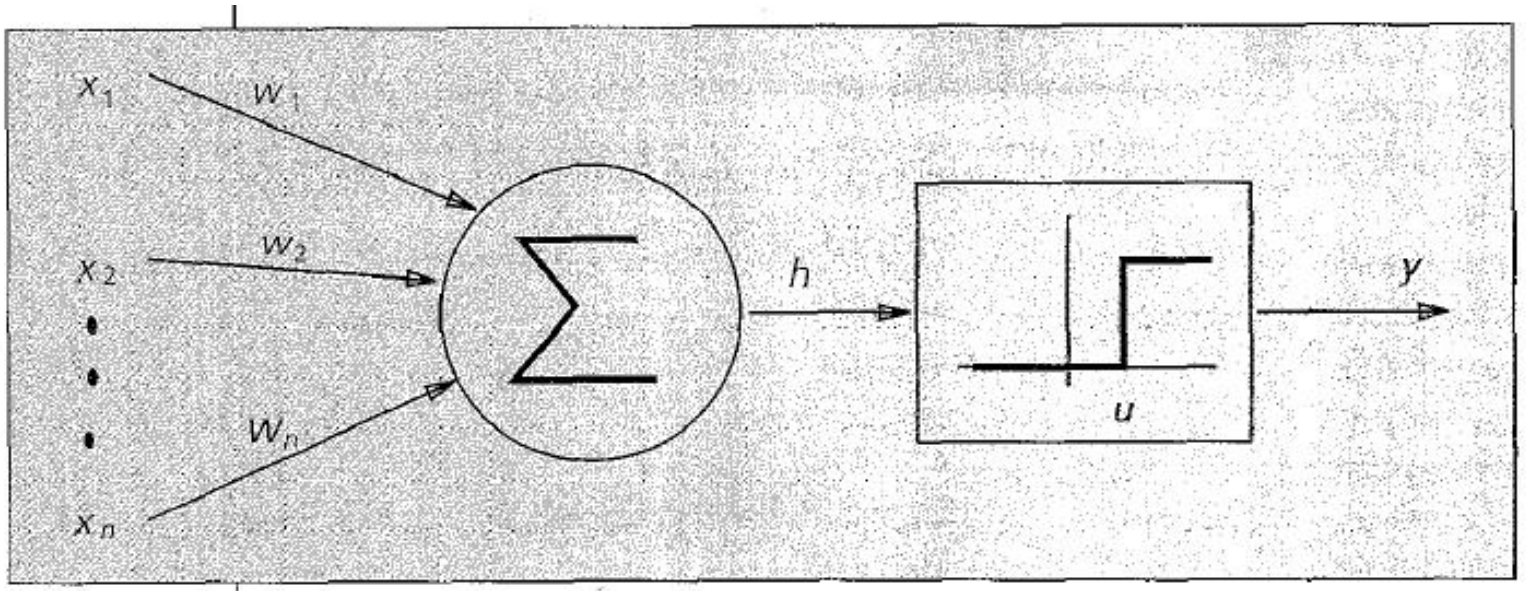
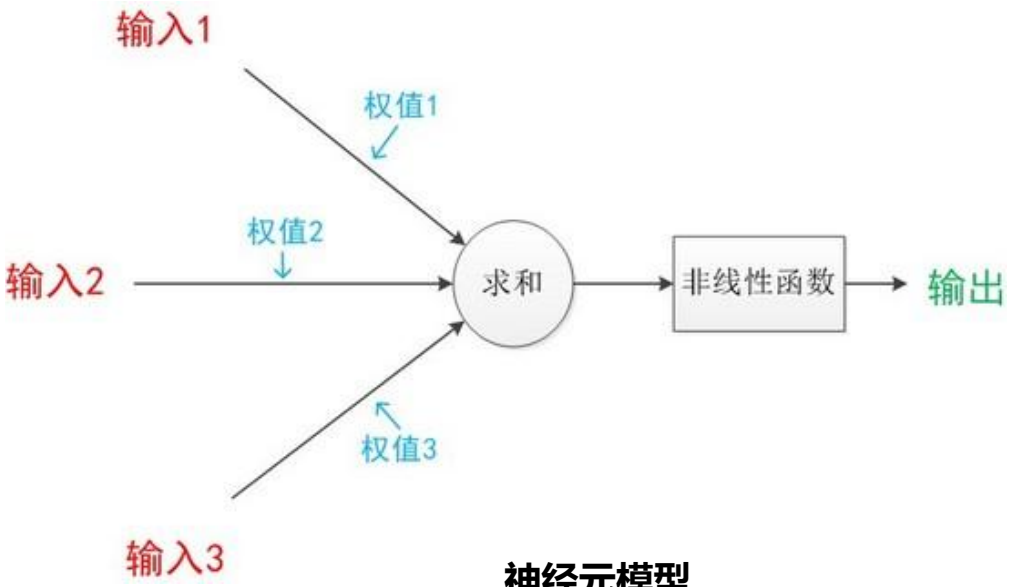


Figure 2. McCulloch-Pitts model of a neuron.

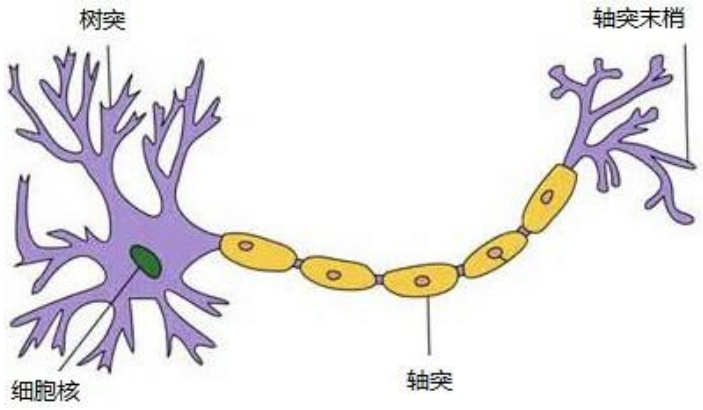
McCulloch, W.S. and W. Pitts. (1943), "A logical Calculus of the Ideas Immanent in Nervous Activity." Bulletin of Mathematical Biophysics, 5(4): 115-133.

神经元 - 结构

- 类比生物结构：输入可以类比为神经元的树突，而输出可以类比为神经元的轴突，计算则可以类比为细胞核。神经元模型是一个包含输入，输出与计算功能的模型。
- 下图是一个典型的神经元模型：包含有3个输入，1个输出，以及2个计算功能
- 注意中间的箭头线。这些线称为“连接”。每个上有一个“权值”

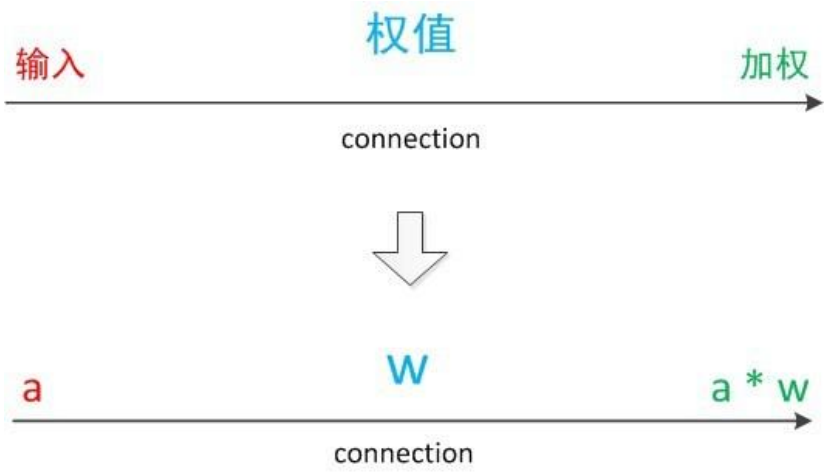


神经元模型



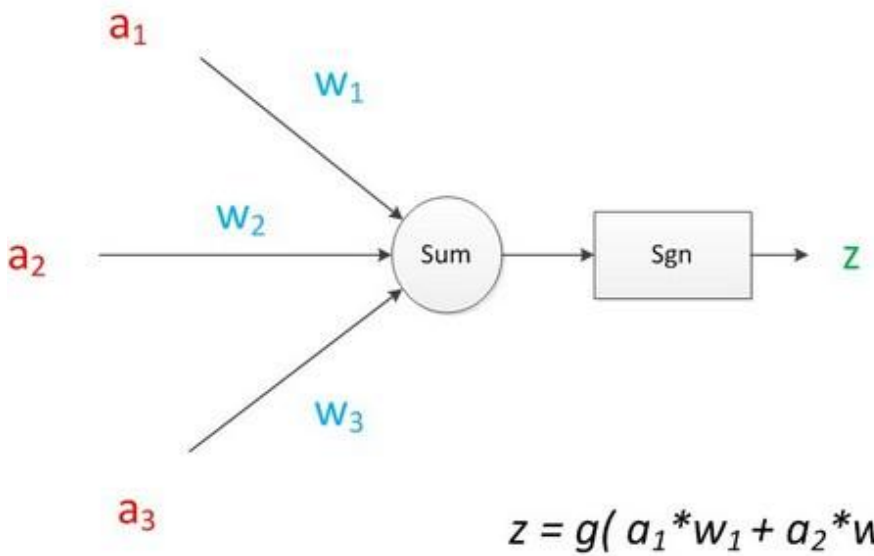
神经元 - 结构

- 连接是神经元中最重要的东西。
- 一个神经网络的训练算法就是让权重的值调整到最佳，以使得整个网络的预测效果最好
- 我们使用a来表示输入，用w来表示权值。一个表示连接的有向箭头可以这样理解：在初端，传递的信号大小仍然是a，端中间有加权参数w，经过这个加权后的信号会变成a*w，因此在连接的末端，信号的大小就变成了a*w
- 在其他绘图模型里，有向箭头可能表示的是值的不变传递。而在神经元模型里，每个有向箭头表示的是值的加权传递



神经元 - 结构

- 将神经元图中的所有变量用符号表示，并且写出输出的计算公式

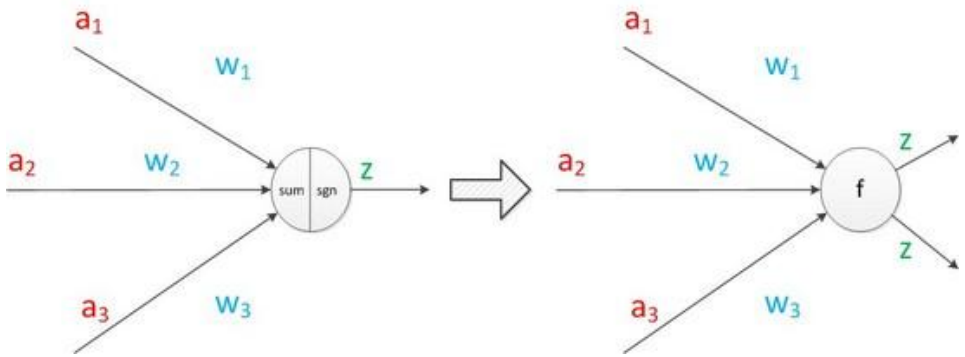


- z是在输入和权值的线性加权和叠加了一个**函数g**的值

- 在MP模型里，函数g是Sgn函数，也就是迁跃函数 binary-threshold function。当输入大于0时，输出1，否则输出0

神经元 - 结构

- 对神经元模型的图进行扩展
  - 将sum函数与sgn函数合并到一个圆圈里，代表神经元的内部计算
  - 把输入a与输出z写到连接线的左上方，便于后面画复杂的网络
- 神经元可以看作一个计算与存储单元。计算是神经元对其的输入进行计算功能。存储是神经元会暂存计算结果，并传递到下一层。



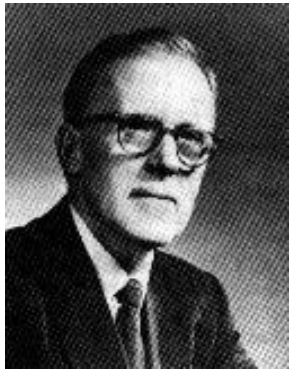
- 用“神经元”组成网络以后，描述网络中的某个“神经元”时，可以用“**单元**”（unit）来指代。由于神经网络的表现形式是一个有向图，也会用“**节点**”（node）来表达同样意思

神经元 - 效果

- 神经元模型的使用可以这样理解：
  - 数据样本有四个属性，其中3个属性已知，1个属性未知。通过3个已知属性 **预测** 未知属性
  - 使用神经元的公式进行计算。3个已知属性的值是a1，a2，a3，未知属性的值是z
  - 已知的属性称为 **特征** ，未知属性称为 **目标** 。假设特征与目标之间是线性关系，并且我们已经得到表示这个关系的权值w1，w2，w3。那么，可以通过神经元模型预测新样本的目标

神经元 - 影响

- 1943年发布的MP模型，虽然简单，但已经建立了神经网络大厦的地基。但是，MP模型中，权重的值都是预先设置的，因此不能学习。
- 1949年心理学家Hebb提出了Hebb学习率，认为人脑神经细胞的突触 （也就是连接）上的强度上可以变化的。于是计算科学家们开始考虑用调整权值的方法来让机器学习。这为后面的学习算法奠定了基础。



Donald Olding Hebb

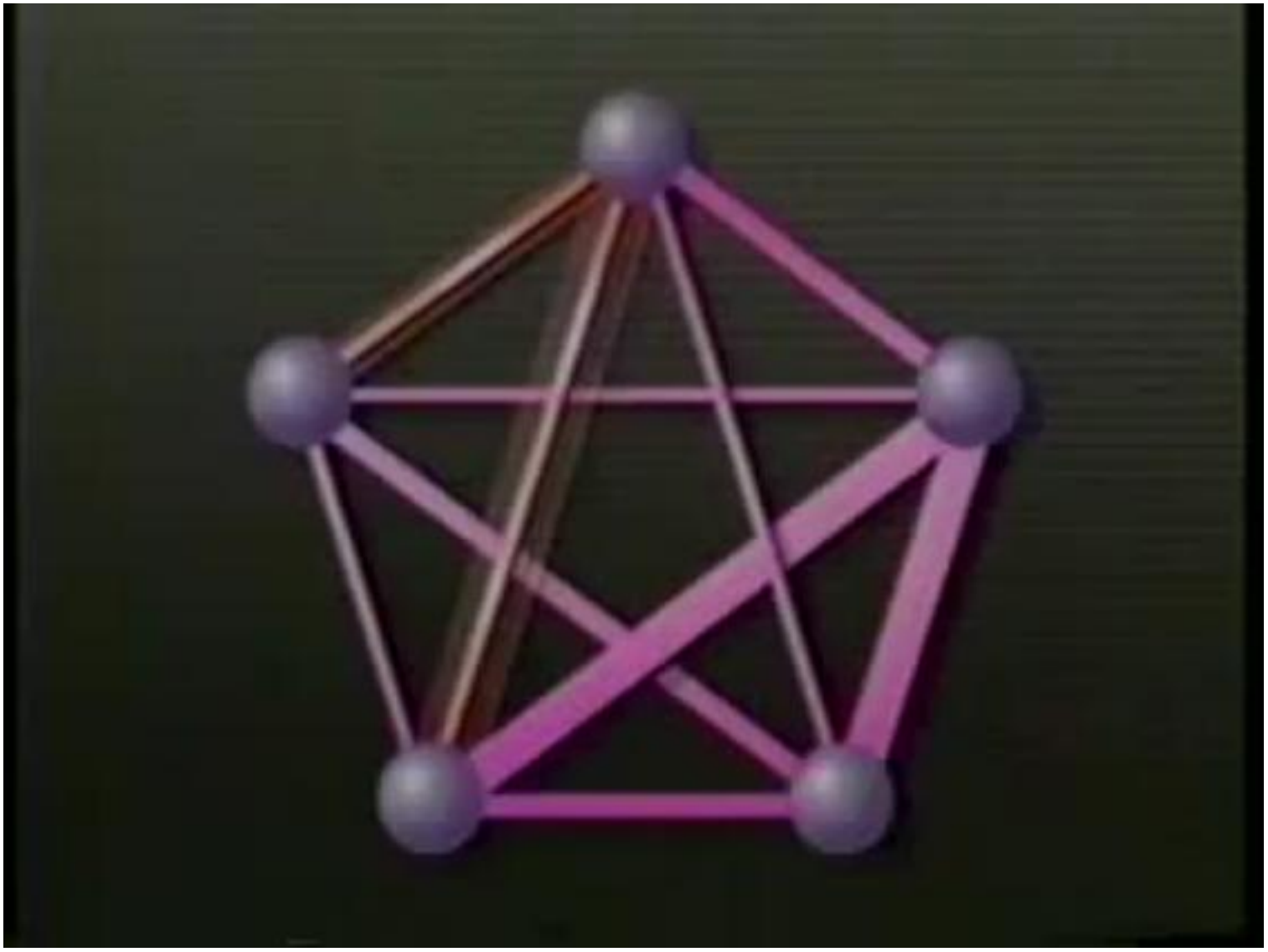
- 尽管神经元模型与Hebb学习律都已诞生，但限于当时的计算机能力，直到接近10年后，第一个真正意义的神经网络才诞生。

单层神经网络（感知器） - 引入

- 1958年，计算科学家Rosenblatt提出了由两层神经元组成的神经网络。他给它起了一个名字——“感知器”（Perceptron，有的文献翻译成“感知机”）
- 感知器是当时首个可以学习的人工神经网络。Rosenblatt现场演示了其学习识别简单图像的过程，在当时的社会引起了轰动
- 人们认为已经发现了智能的奥秘，许多学者和科研机构纷纷投入到神经网络的研究中。美国军方大力资助了神经网络的研究，并认为神经网络比 “原子弹工程” 更重要。这段时间直到1969年才结束，这个时期可以看作神经网络的第一次高潮。

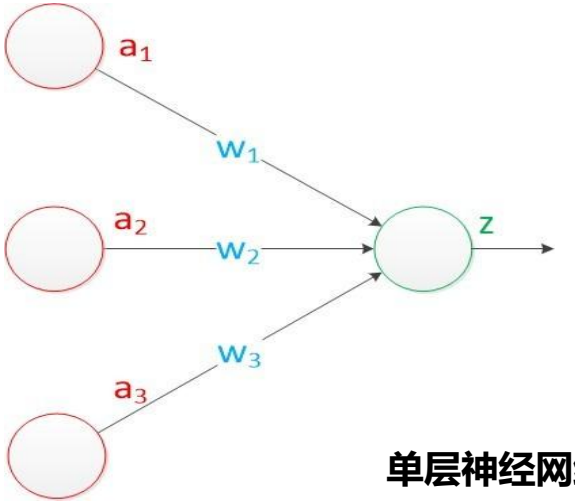
Rosenblat 与感知器

Perceptron in the 50's & 60's



感知器 - 结构

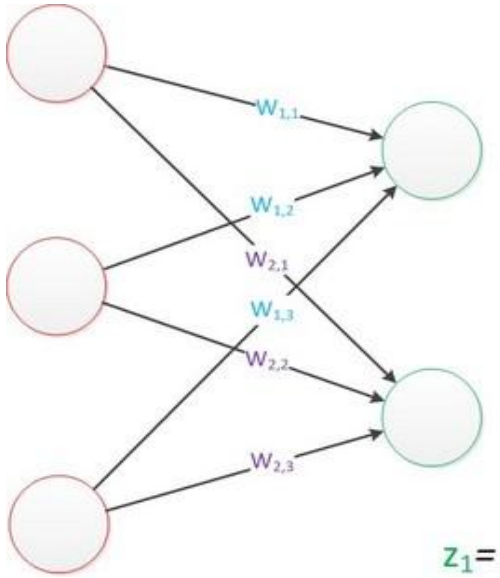
- 在原来MP模型的 “输入” 位置添加神经元节点，标志其为 “输入单元”
  - 在“感知器”中，有两个层次，分别是输入层和输出层。 输入层里的“输入单元”只负责传输数据， 不做计算。 输出层里的“输出单元”则需要对前面一层的输入进行计算
  - 把需要计算的层次称之为“计算层”，并把拥有一个计算层的网络称之为“单层神经网络”。有一些文献会按照网络拥有的层数来命名，例如把“感知器”称为两层神经网络
  - 假如预测目标不是一个值，而是一个向量。可以在输出层再增加一个“输出单元”



单层神经网络

感知器 - 结构

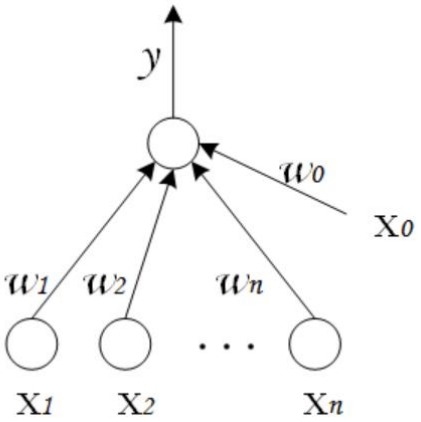
- $w_{1,2}$ 代表后一层的第1个神经元与前一层的第2个神经元的连接的权值



$$z_1 = g(a_1 * w_{1,1} + a_2 * w_{1,2} + a_3 * w_{1,3})$$
$$z_2 = g(a_1 * w_{2,1} + a_2 * w_{2,2} + a_3 * w_{2,3})$$

- 这两个公式是线性代数方程组，可以用矩阵乘法来表达这两个公式
  - 输入的变量是 $[a_1, a_2, a_3]^T$（代表由 a_1，a_2，a_3 组成的列向量），用向量 **a** 来表示
  - 方程的左边是 $[z_1, z_2]^T$，用向量 **z** 来表示
  - 系数则是矩阵 **W**（2行3列的矩阵，排列形式与公式中的一样）
  - 输出公式可以改写成：**g(W * a) = z**
  - 这个公式就是神经网络中从前一层计算后一层的运算过程

感知器 - 训练法则



与神经元模型不同，感知器中的权值是通过训练得到

$$y = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x})$$

$$\mathrm{sgn}(y) = \begin{cases} 1, & \text{if } y > 0 \\ 0, & \text{otherwise} \end{cases}$$

感知机训练法则：

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

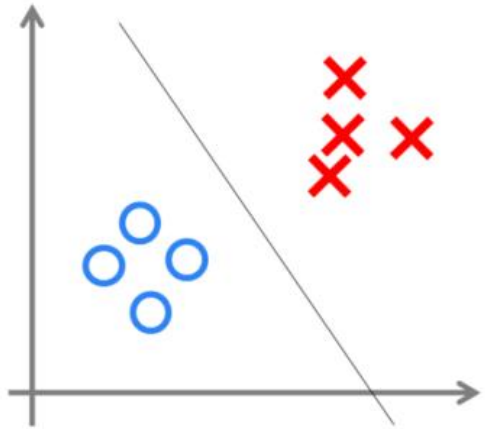
t：是当前训练样例的目标输出
o：是感知机的输出
η：是一个正的常数称为学习速率（**0.1**）
xi：训练样例

感知器 - 效果

$$\boldsymbol{w} \cdot \boldsymbol{x} = 0$$

- 根据以前的知识我们知道，感知器类似一个 **逻辑回归** 模型，可以做线性分类任务。可以把感知机当做n维空间中的决策超平面，对于超平面一侧的实例感知器输出 1，对于另一侧的实例输出 0

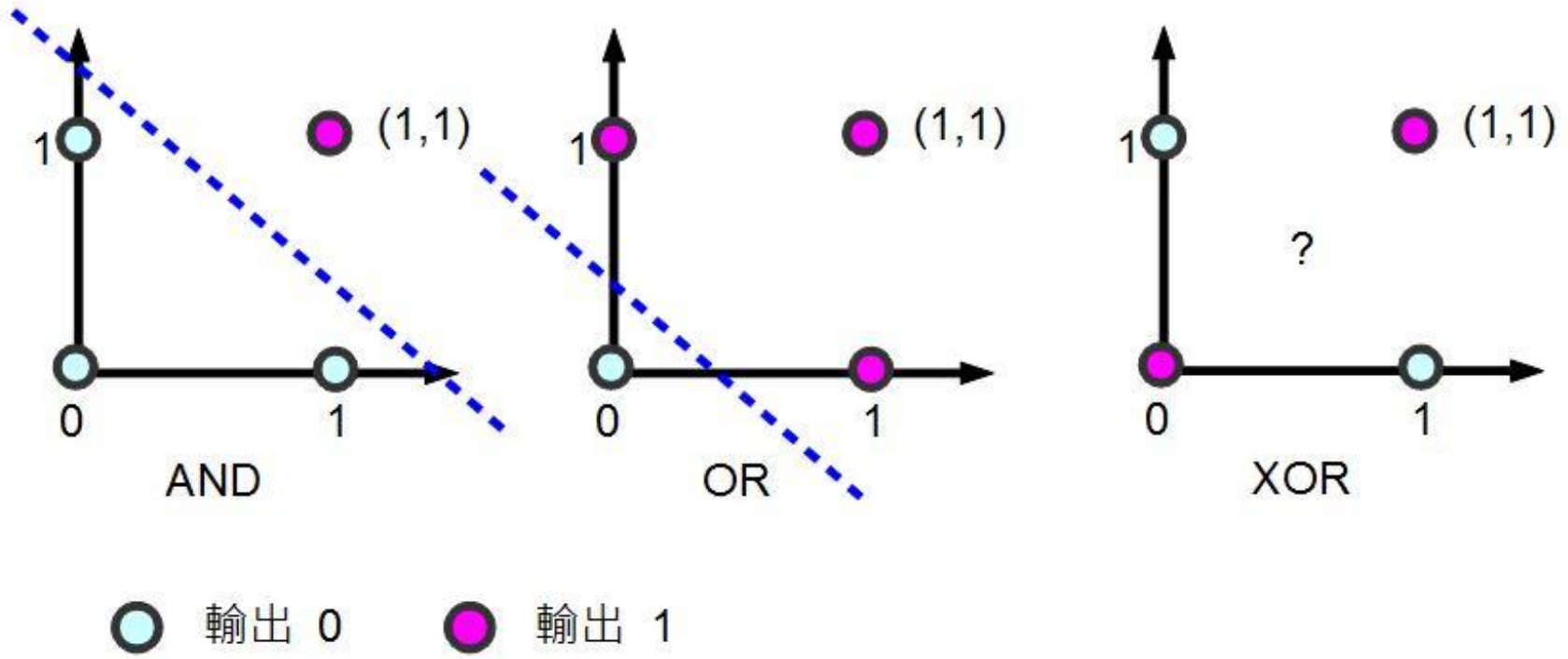
- 我们可以用 **决策分界** 来形象的表达分类的效果。决策分界就是在二维的数据平面中划出一条直线，当数据的维度是3维的时候，就是划出一个平面，当数据的维度是n维时，就是划出一个n-1维的超平面。下图显示了在二维平面中划出决策分界的效果，也就是感知器的分类效果。



单层神经网络（决策分界）

感知器 - 影响

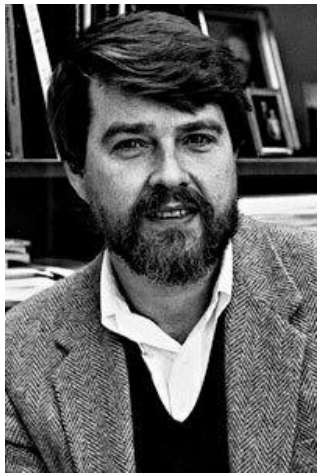
- 感知器只能做简单的线性分类任务。但是当时的人们热情太过于高涨，并没有人清醒的认识到这点
- Minsky在1969年出版了一本叫《Perceptron》的书，里面用详细的数学证明了感知器的弱点，尤其是单层感知器对XOR（异或）这样的简单分类任务都无法解决



Minsky - The problem with percetrons

两层神经网络（多层感知器） - 引入

- 单层神经网络无法解决异或问题。但是当增加一个计算层以后，两层神经网络不仅可以解决异或问题，而且具有非常好的非线性分类效果。不过两层神经网络的计算是一个问题，没有一个较好的解法
- 1986 年， Rumelhar 和 Hinton 等人提出了反向传播（ Backpropagation， BP ）算法，解决了两层神经网络所需要的复杂计算量问题，从而带动了业界使用两层神经网络研究的热潮
- 这时候的Hinton还很年轻，30年以后，正是他重新定义了神经网络，带来了神经网络复苏

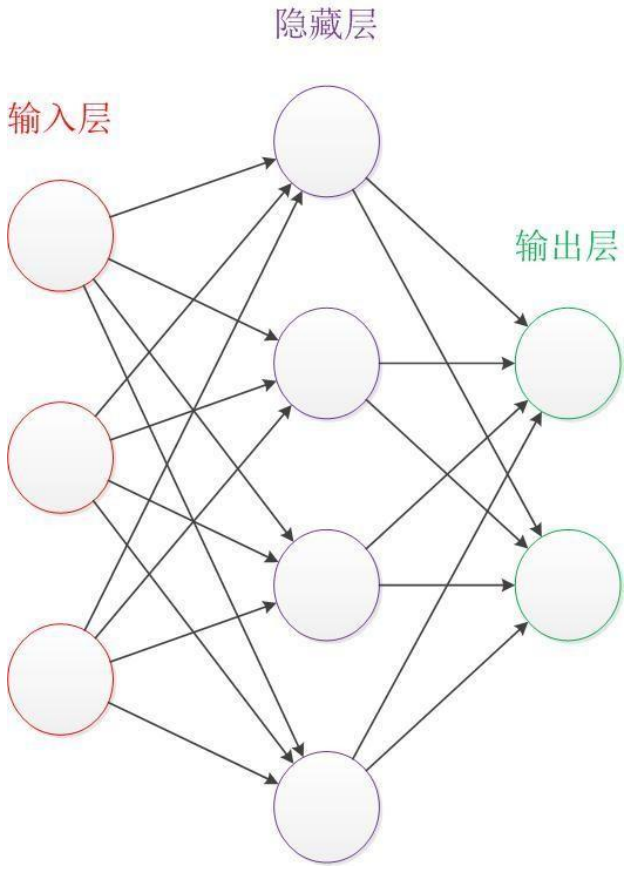


David Rumelhart

Geoffery Hinton

一个简单的神经网络：多层感知机

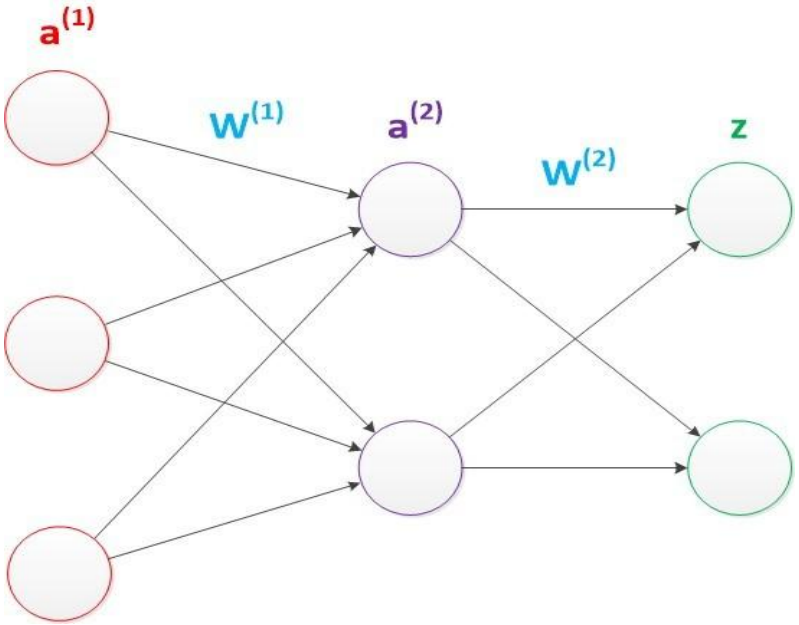
- 一个经典的神经网络包含三个层次：红色的是 **输入层**，绿色的是 **输出层**，紫色的是 **中间层**（也叫 **隐藏层**）
  - 输入层与输出层的节点数往往是固定的，中间层则可以自由指定
  - 结构图里的关键不是圆圈（代表“神经元”），而是连接线。每个连接线对应一个不同的**权重**



神经网络结构图

多层感知器 - 结构

- 假设预测目标是一个向量，只需要在“输出层”再增加节点
- 使用向量和矩阵来表示层次中的变量。 $\mathbf{a}^{(1)}$， $\mathbf{a}^{(2)}$， $\mathbf{z}$ 是网络中传输的向量数据。 $\mathbf{W}^{(1)}$ 和 $\mathbf{W}^{(2)}$ 是网络的矩阵参数
- 矩阵运算的简洁，也不会受节点数增多影响



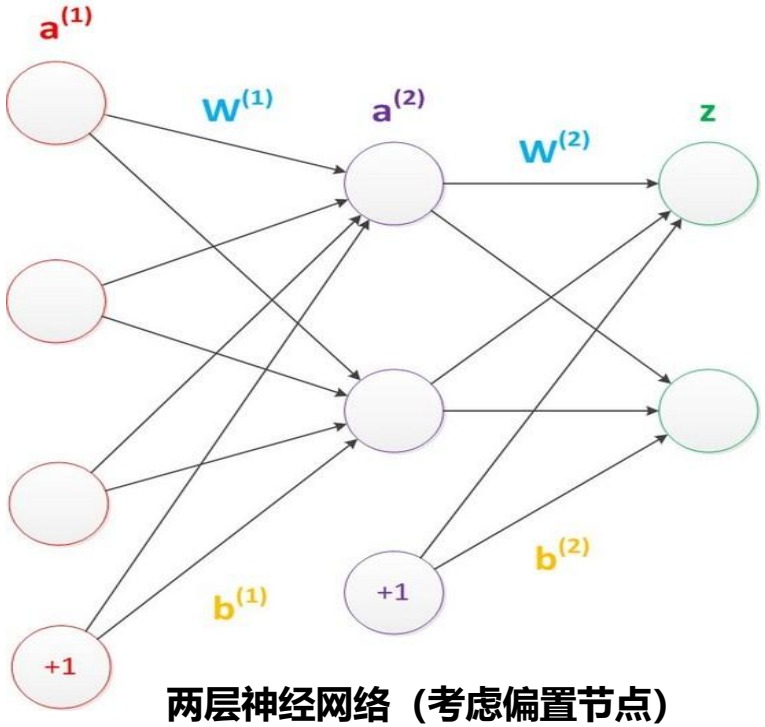
两层神经网络（向量形式）

$$g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)}) = \mathbf{a}^{(2)}$$

$$g(\mathbf{W}^{(2)} * \mathbf{a}^{(2)}) = \mathbf{z};$$

多层感知器 - 结构

- 一般还会引入偏置单元，他与后一层的所有节点都有连接，我们设这些参数值为向量 **b** ，称之为偏置。如下图。



两层神经网络（考虑偏置节点）

- 偏置节点很好认，因为其没有输入（前一层中没有箭头指向它）。有些神经网络的结构图中会把偏置节点明显画出来，有些不会。一般情况下，都不会明确画出偏置节点

多层感知器 - 结构

- 考虑了偏置后的神经网络的矩阵运算：

$$g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)} + \mathbf{b}^{(1)}) = \mathbf{a}^{(2)};$$

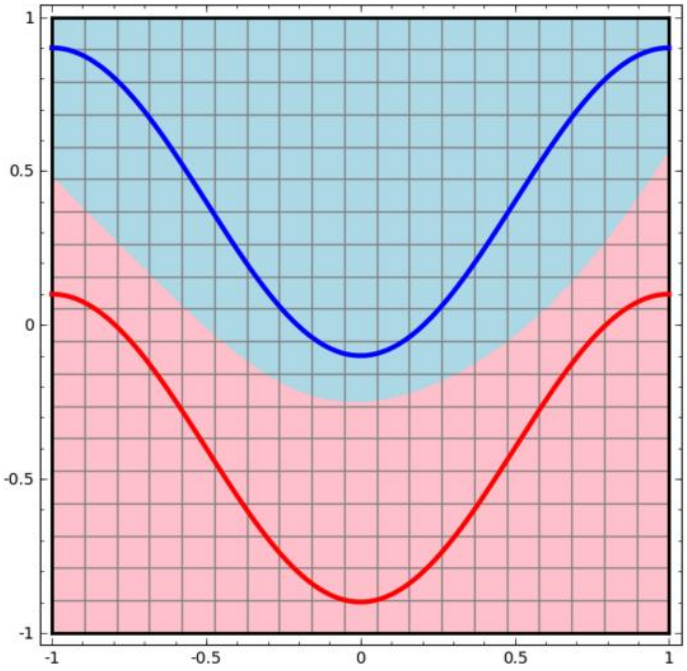
$$g(\mathbf{W}^{(2)} * \mathbf{a}^{(2)} + \mathbf{b}^{(2)}) = \mathbf{z};$$

- 两层神经网络不再使用Sgn函数作为函数**g**，而是使用平滑函数sigmoid作为函数**g**。函数**g**也称作激活函数（active function）。
- 神经网络的本质就是通过矩阵参数**W**与激活函数**g**来拟合特征与目标之间的真实函数关系

多层感知器 - 效果

- 理论证明，两层神经网络可以无限逼近任意连续函数。面对复杂的非线性分类任务，两层（带一个隐藏层）神经网络可以分类的很好
- 例子
  - 红色的线与蓝色的线代表数据
  - 而红色区域和蓝色区域代表由神经网络划开的区域，两者的分界线就是决策分界

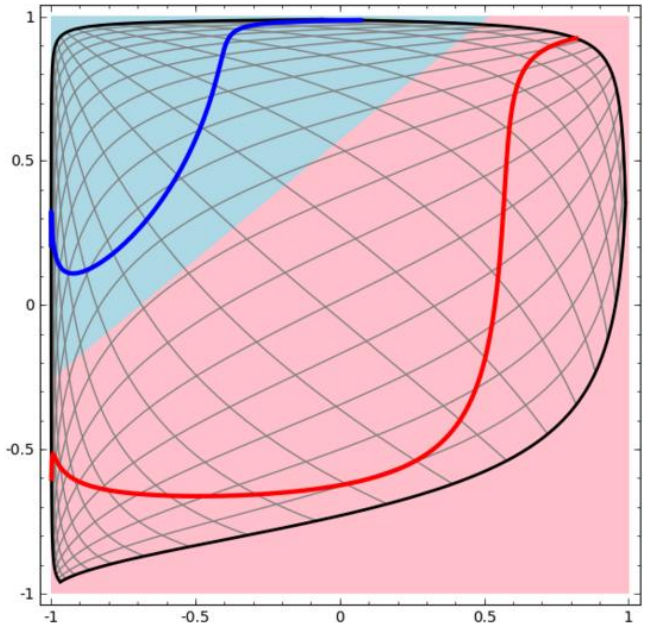
- 这个两层神经网络的决策分界是非常平滑的曲线，而且分类的很好。
- **Q：单层网络只能做线性分类任务。而两层神经网络中的后一层也是线性分类层，应该只能做线性分类任务。为什么两个线性分类任务结合就可以做非线性分类任务?**



两层神经网络（决策分界）

多层感知器 - 效果

- 我们可以把输出层的决策分界单独拿出来看一下。就是下图。

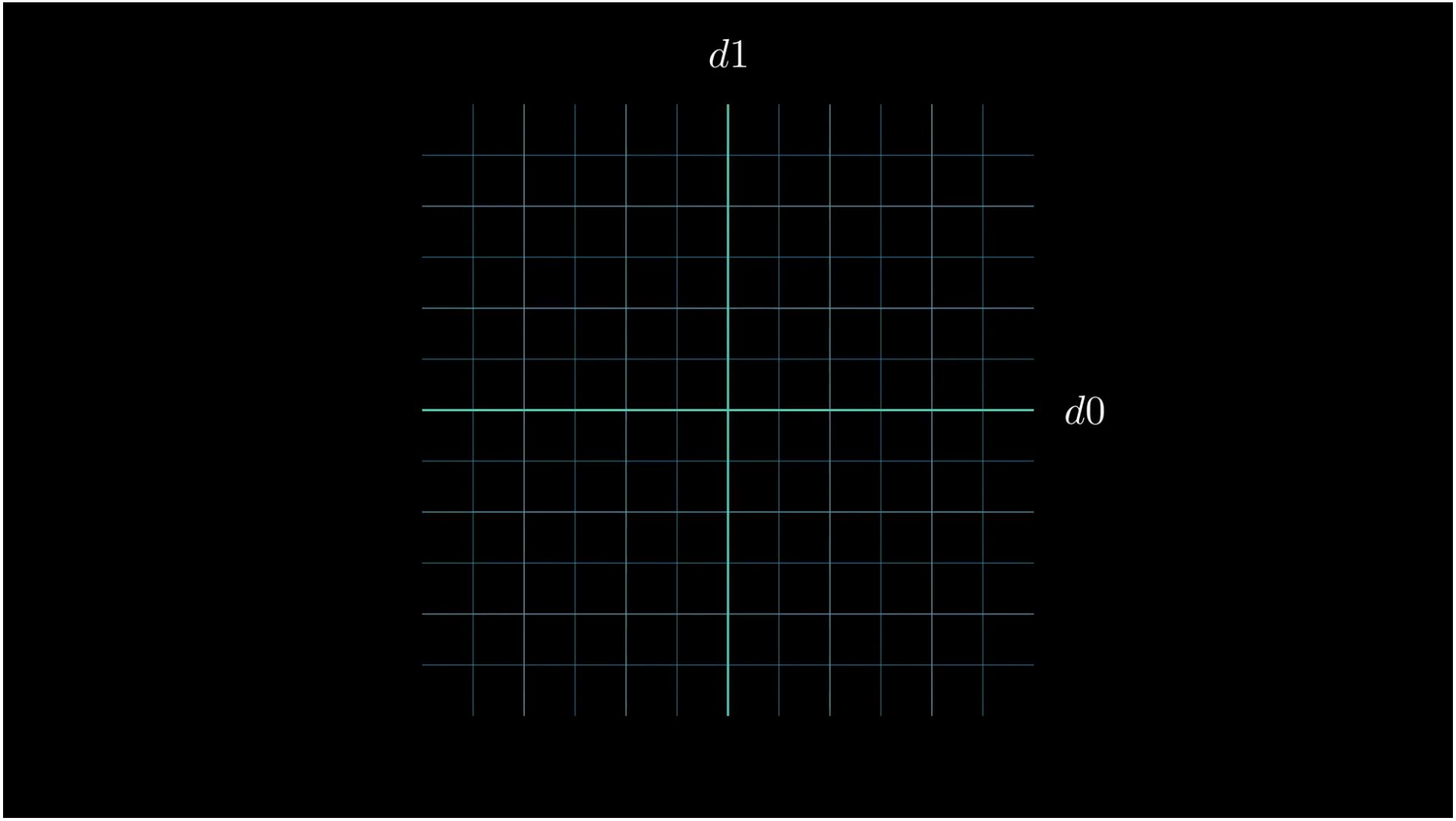


两层神经网络（空间变换）

- 输出层的决策分界仍然是直线。关键是，从输入层到隐藏层时，数据发生了空间变换。也就是说，两层神经网络中，隐藏层对原始的数据进行了一个空间变换，使其可以被线性分类，然后输出层的决策分界划出了一个线性分类分界线，对其进行分类

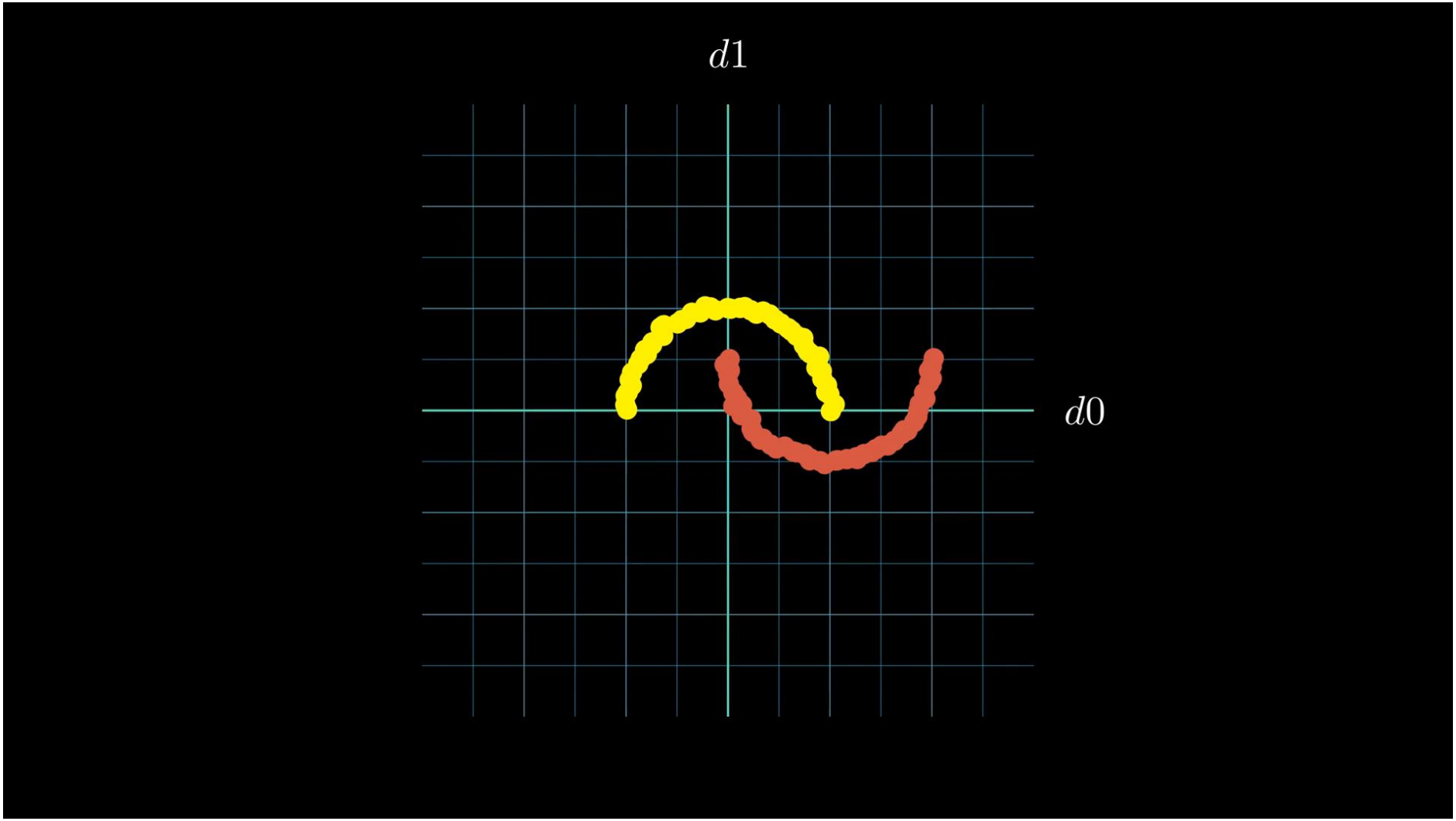
多层感知器 - 效果

Neural Network Transformation Visualization

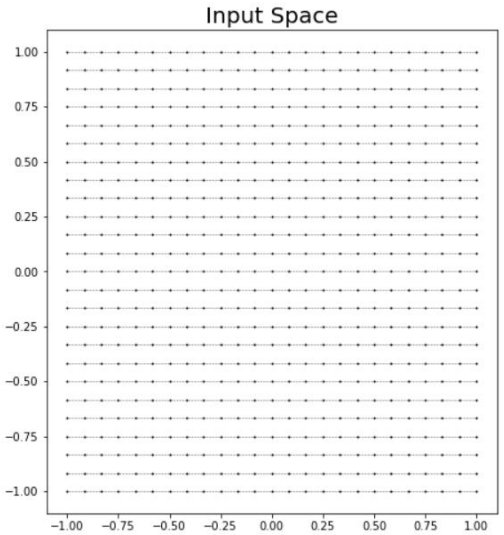


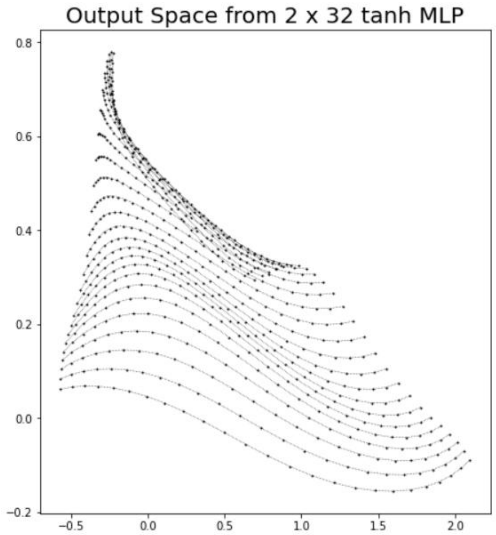
多层感知器 - 效果

Neural Network Decision Boundary Visualization



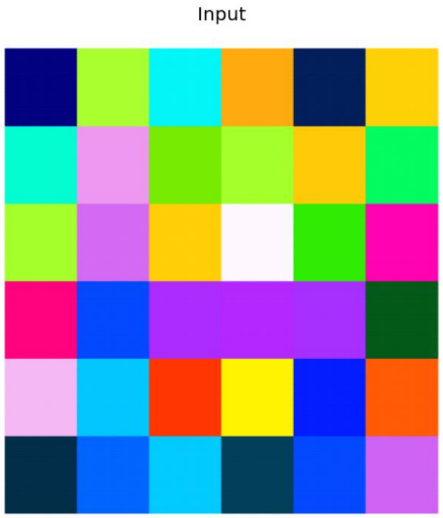
多层感知器 - 效果





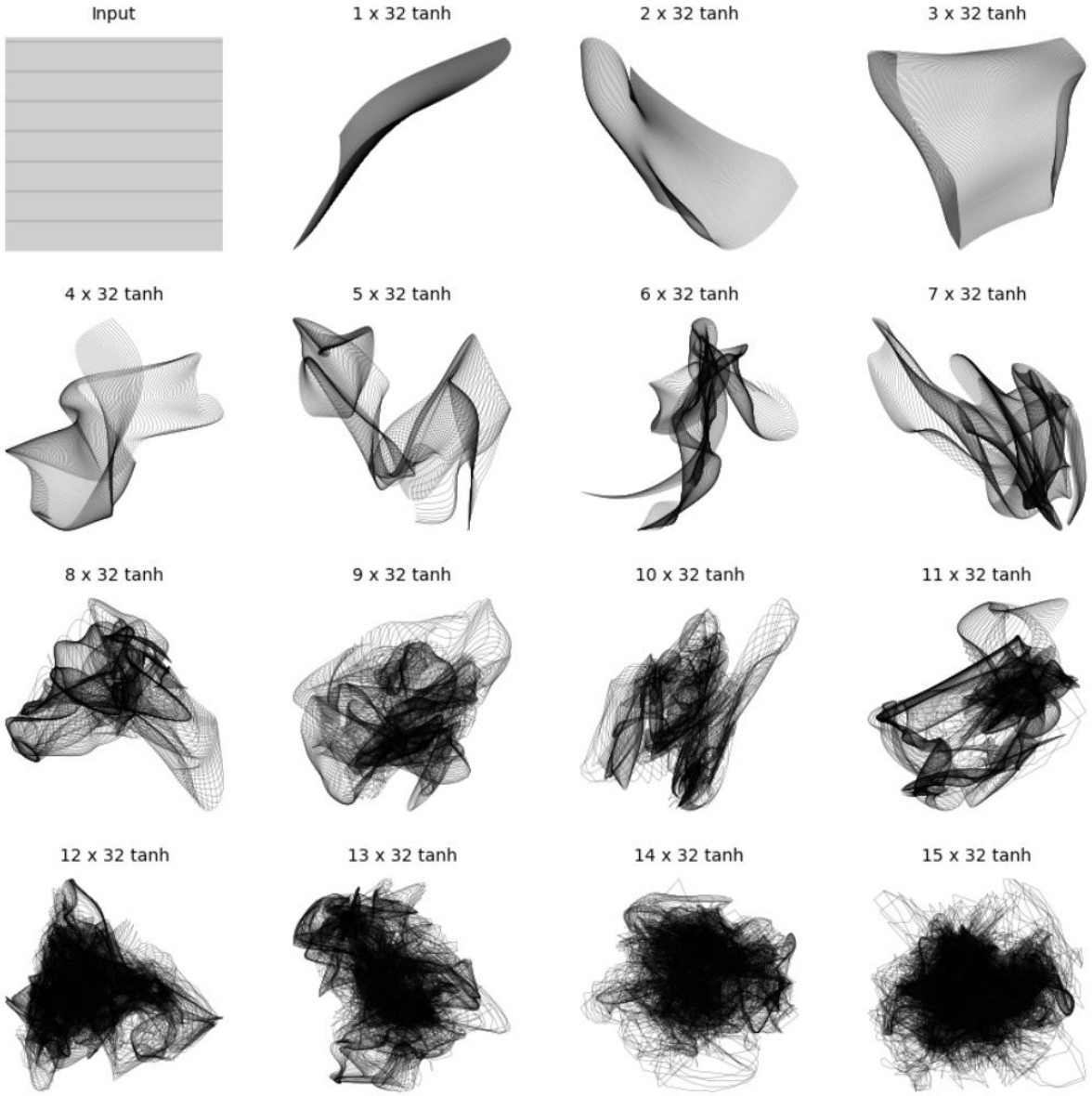

```
units_per_layer = 64
num_layers = 2
activation_fn = 'tanh'
```

```
units_per_layer = 32
num_layers = 2
activation_fn = 'tanh'
```



https://colab.research.google.com/github/murphyka/ml_colabs/blob/main/Simple_MLP_Visualization.ipynb#scrollTo=HpMdcCE7NoB0

多层感知器 - 效果


Input
1 x 32 tanh
2 x 32 tanh
3 x 32 tanh
4 x 32 tanh
5 x 32 tanh
6 x 32 tanh
7 x 32 tanh
8 x 32 tanh
9 x 32 tanh
10 x 32 tanh
11 x 32 tanh
12 x 32 tanh
13 x 32 tanh
14 x 32 tanh
15 x 32 tanh

多层感知器 - 效果

- 两层神经网络可以做非线性分类的关键--隐藏层
- 矩阵公式中矩阵和向量相乘以及非线性函数sigmod/tanh/...，本质上就是对向量的坐标空间进行一个变换。因此，隐藏层的作用就是使得数据的原始坐标空间**从线性不可分，转换成了线性可分**
- 两层神经网络通过两层的线性模型模拟了数据内真实的非线性函数。因此，多层的神经网络的**本质就是复杂函数拟合**
- 设计一个神经网络时，输入层的节点数需要与特征的维度匹配，输出层的节点数要与目标的维度匹配。如何决定这个自由层的节点数呢？一般根据经验来设置
- Grid Search （网格搜索）：预先设定几个可选值，通过切换这几个值来看整个模型的预测效果，选择效果最好的值作为最终选择

多层感知器 - 训练

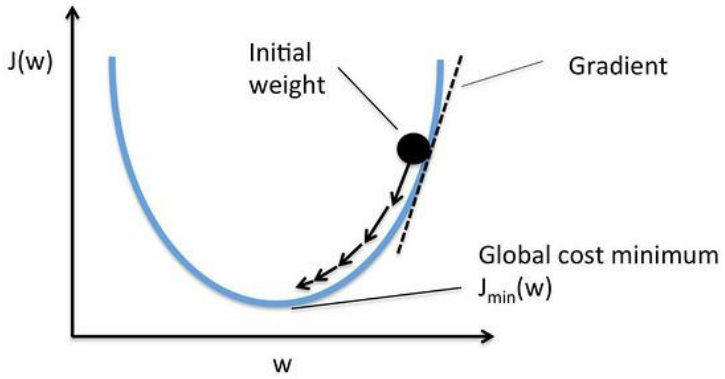
- Rosenblat提出的感知器模型中，模型中的参数可以被训练，但是使用的方法较为简单，并没有使用目前机器学习中通用的方法，这导致其扩展性与适用性非常有限
- 从两层神经网络开始，神经网络的研究人员开始使用机器学习相关的技术进行神经网络的训练。例如用大量的数据（1000-10000左右），使用算法进行优化等等，从而使得模型训练可以获得性能与数据利用上的双重优势。
- 机器学习模型训练的目的，就是使得参数尽可能的与真实的模型逼近。具体做法是这样的：首先给所有参数赋上随机值，然后使用这些随机生成的参数值，来预测训练数据中的样本，不断循环迭代更新参数。
- 样本的预测目标为 out ，真实目标为 target。**损失** loss的计算公式如下，目标是使对所有训练数据的损失和尽可能的小

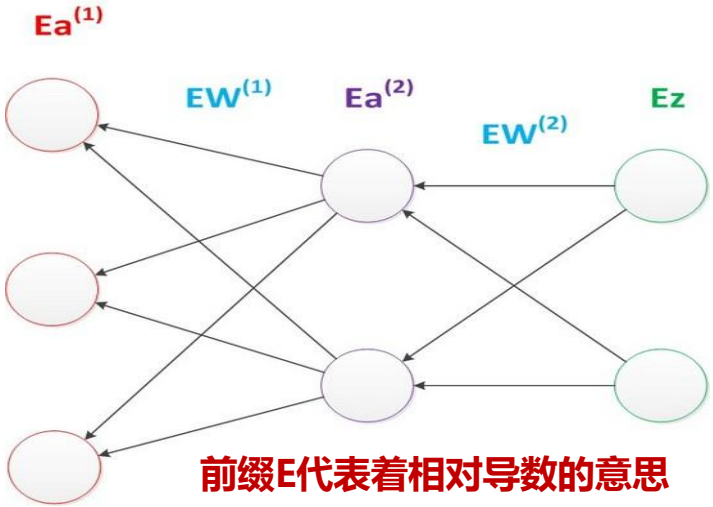
$$loss = (target-out)^2$$

多层感知器 - 训练

- 如何优化参数，能够让损失函数的值最小？
- **梯度下降** 算法：每次计算参数在当前的梯度，然后让参数向着梯度的反方向前进一段距离，不断重复，直到梯度接近零时截止。一般这个时候，所有的参数恰好达到使损失函数达到一个最低值的状态

- **反向传播** 算法：反向传播算法不一次计算所有参数的梯度，而是从后往前。首先计算输出层的梯度，然后是第二个参数矩阵的梯度，接着是中间层的梯度，再然后是第一个参数矩阵的梯度，最后是输入层的梯度。计算结束以后，所要的两个参数矩阵的梯度就都有了

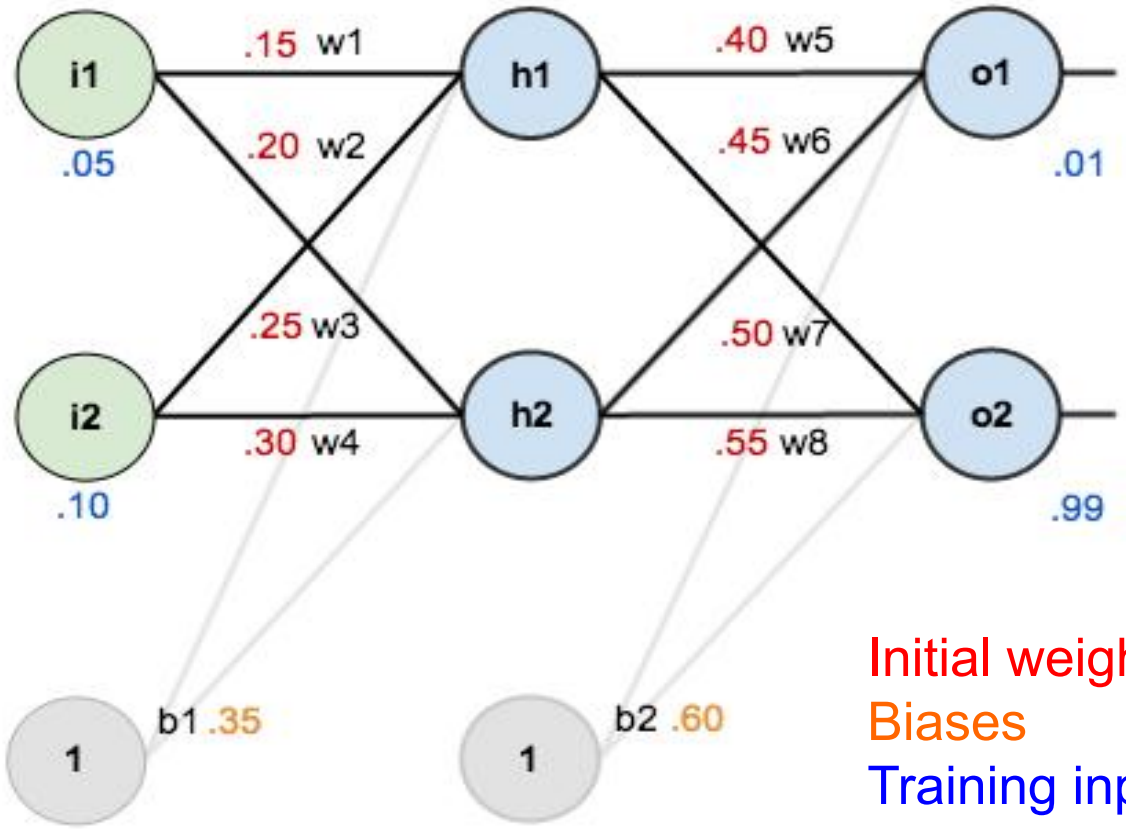




多层感知器 - 训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- Basic structure and initializing:

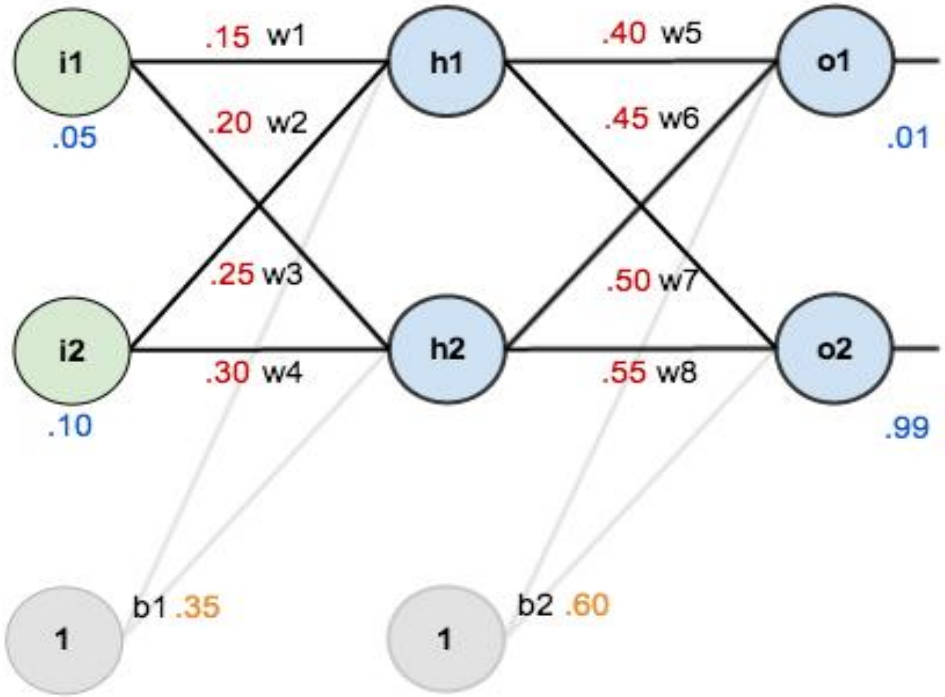


Initial weights
Biases
Training inputs/outputs

多层感知器 - 训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

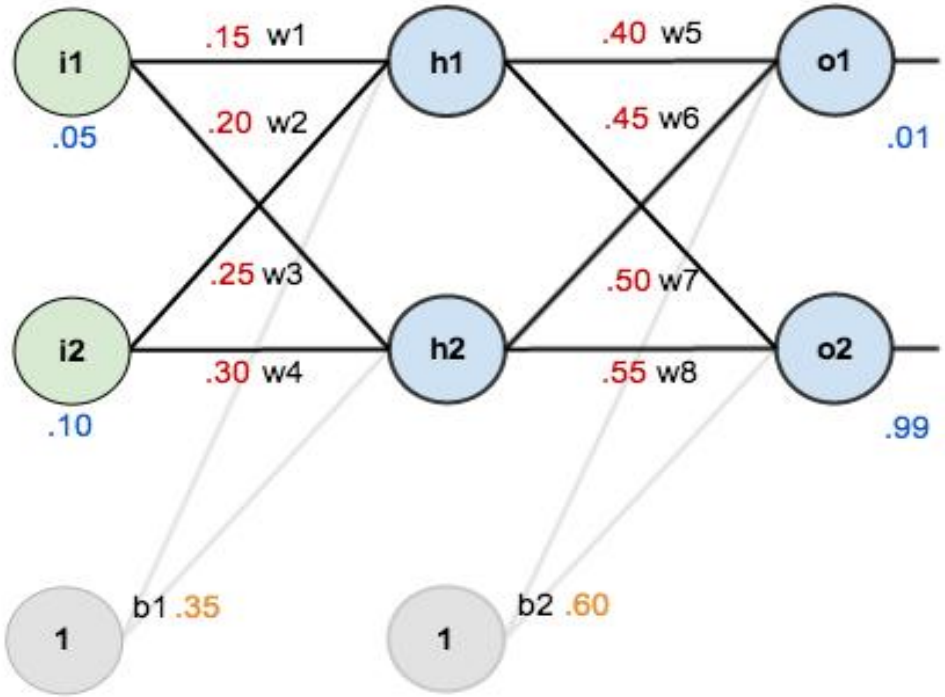
- Goal?
  - To optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.



多层感知器 - 训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

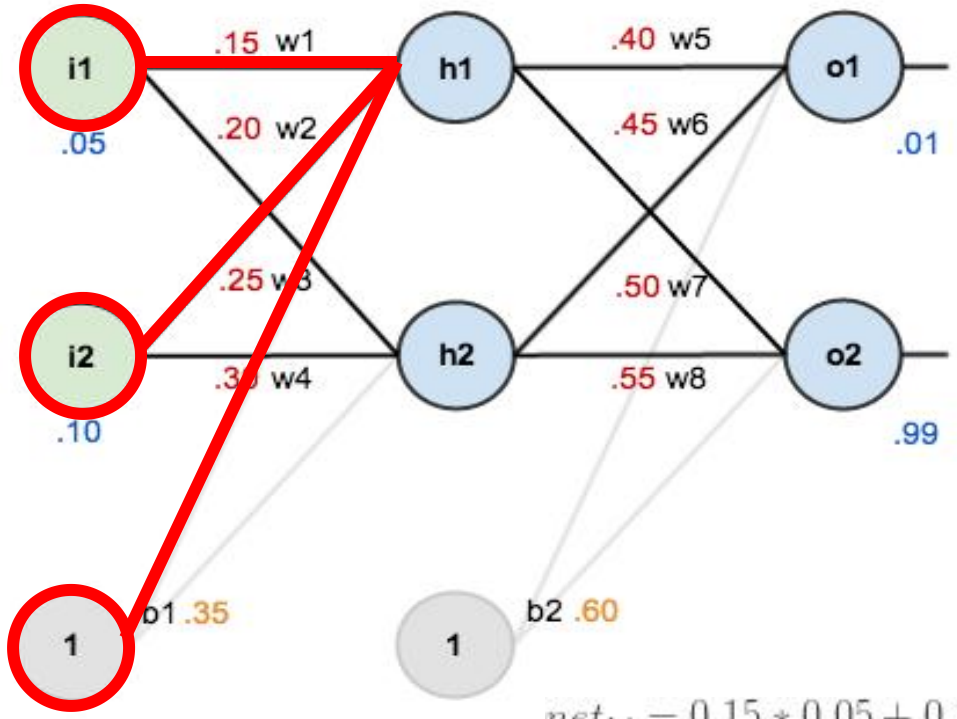
- A single training set:
  - Given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.



多层感知器 - 正向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Forward Pass:
  - Calculate (total) net input.



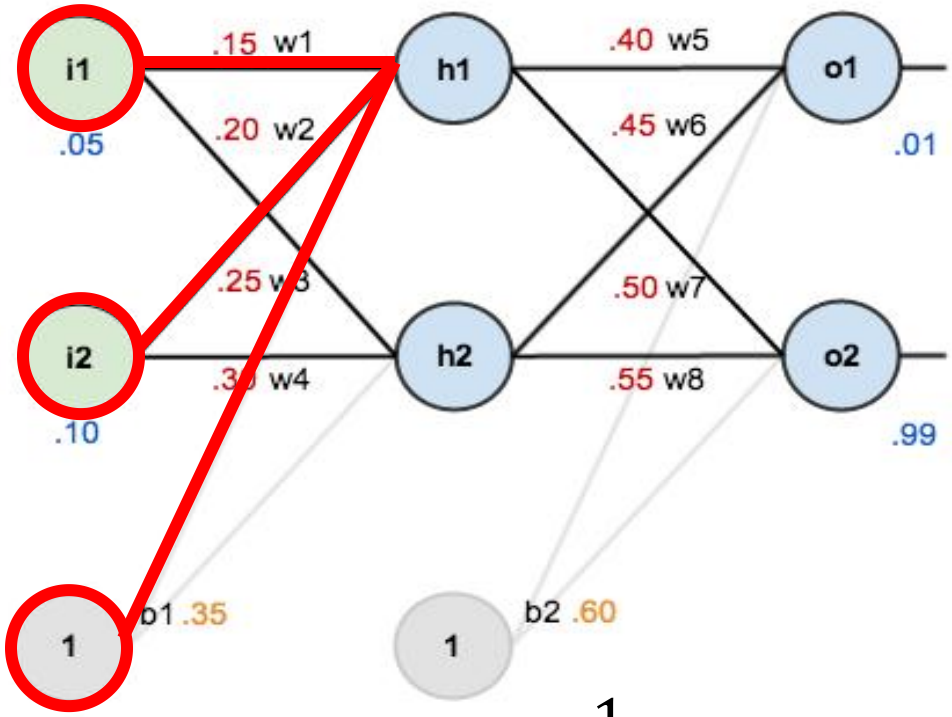
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1 = 0.3775$$

多层感知器 - 正向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Forward Pass:
  - Squash using an activation function (sigmoid function here).

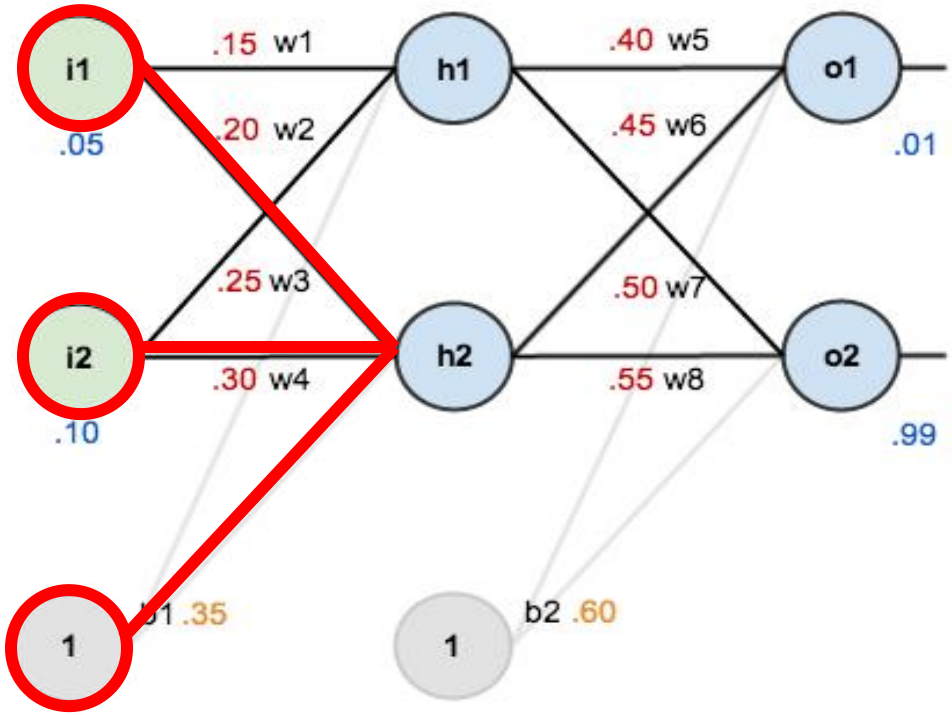


$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = 0.59326992$$

多层感知器 - 正向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Forward Pass:
  - Same for h2.

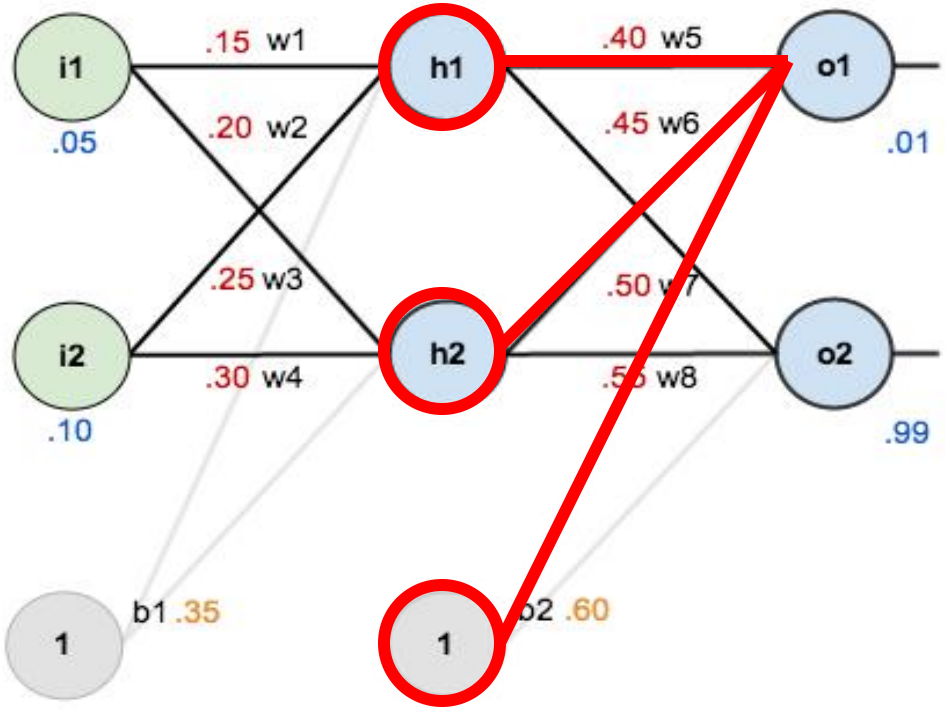


$$out_{h2} = \frac{1}{1+e^{-net_{h2}}} = 0.596884378$$

多层感知器 - 正向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Forward Pass:
  - Repeat for o1.



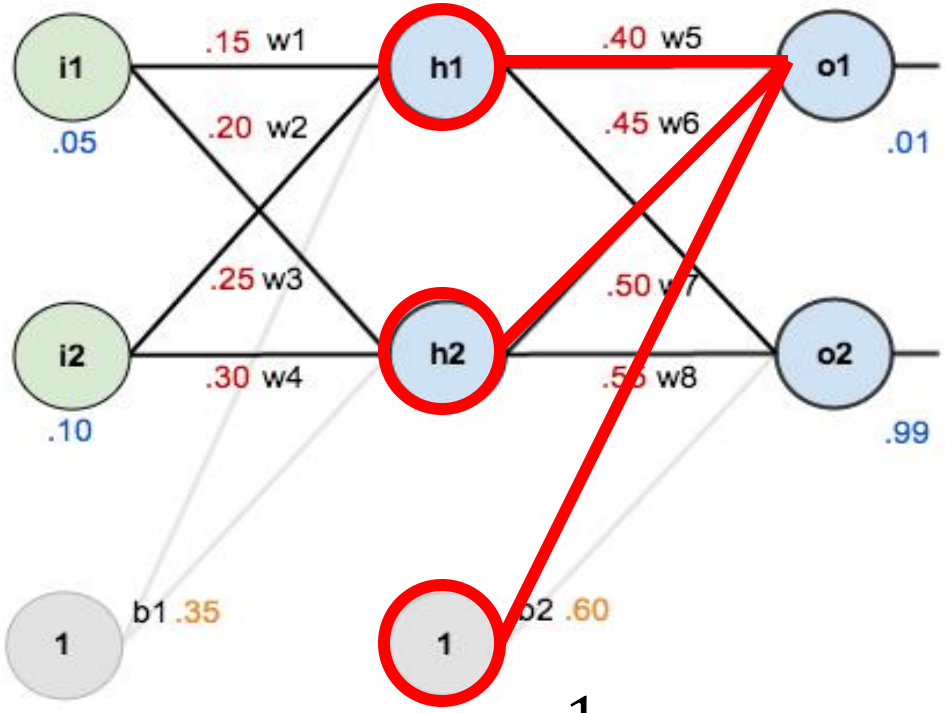
$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$net_{o1} = w_5 * h_1 + w_6 * h_2 + b_2 * 1 = 1.105905967$$

多层感知器 - 正向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Forward Pass:
  - Repeat for o1.

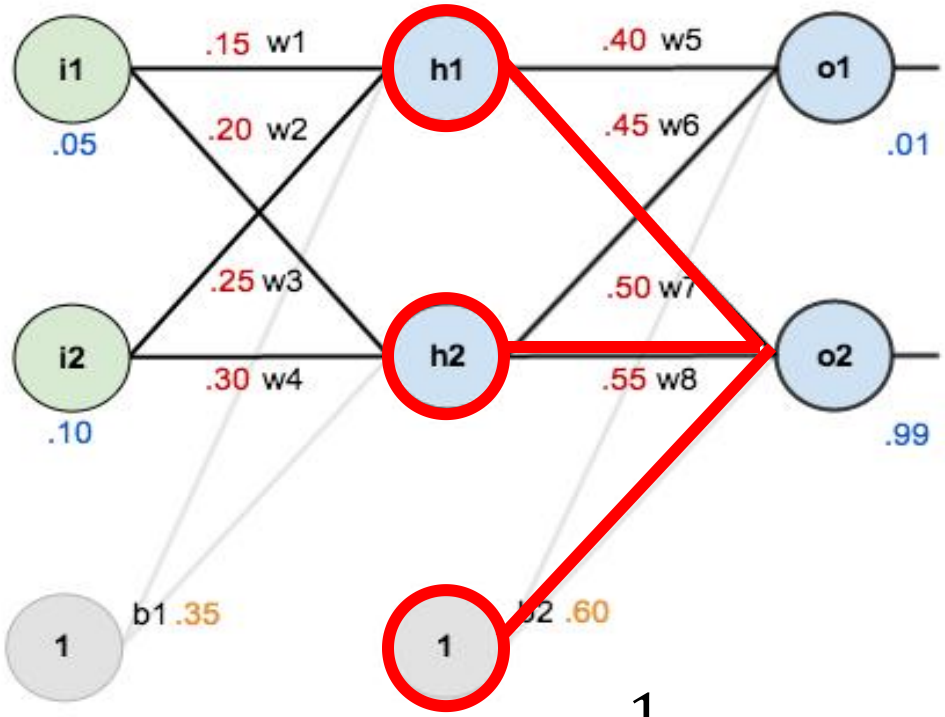


$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = 0.75136507$$

多层感知器 - 正向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Forward Pass:
  - Repeat for o2.



$$out_{o2} = \frac{1}{1 + e^{-net_{o2}}} = 0.772928465$$

多层感知器 - 正向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- Calculating the Total Error:
  - Using the squared error function.
  - Target: $target_{o1}$ =0.01， $target_{o2}$ =0.99
  - Output: out_{o2} = 0.75136507， out_{o1} = 0.772928465

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

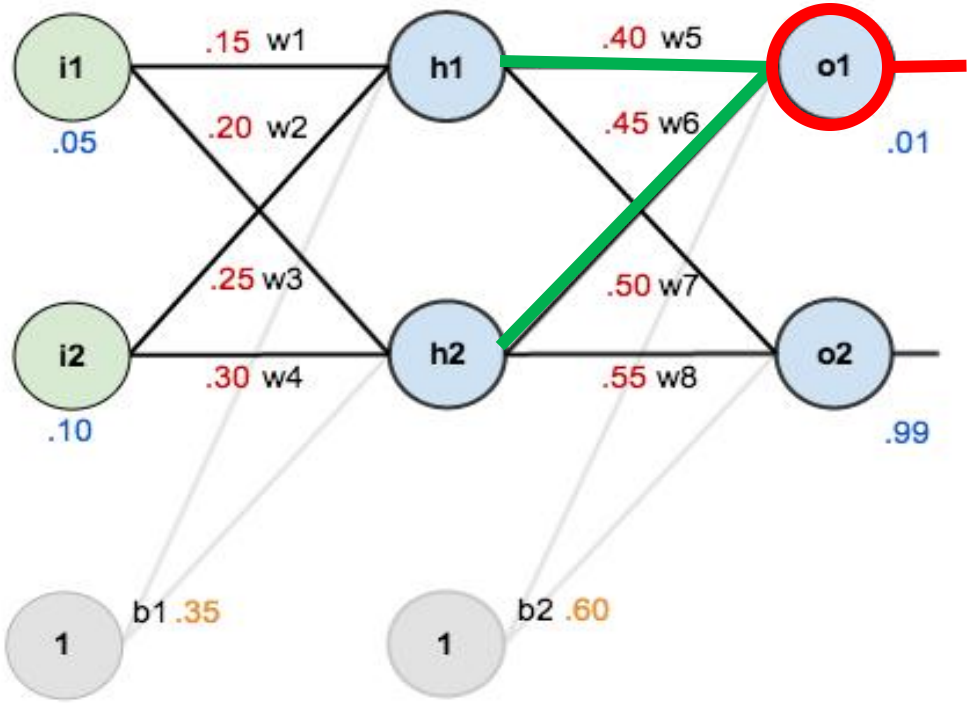
$$\boldsymbol{E_{total} = \sum \frac{1}{2}(target - output)^2}$$

$$\boldsymbol{E_{total} = E_{o1} + E_{o2}} = 0.298371109$$

多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass:
  - Output Layer:



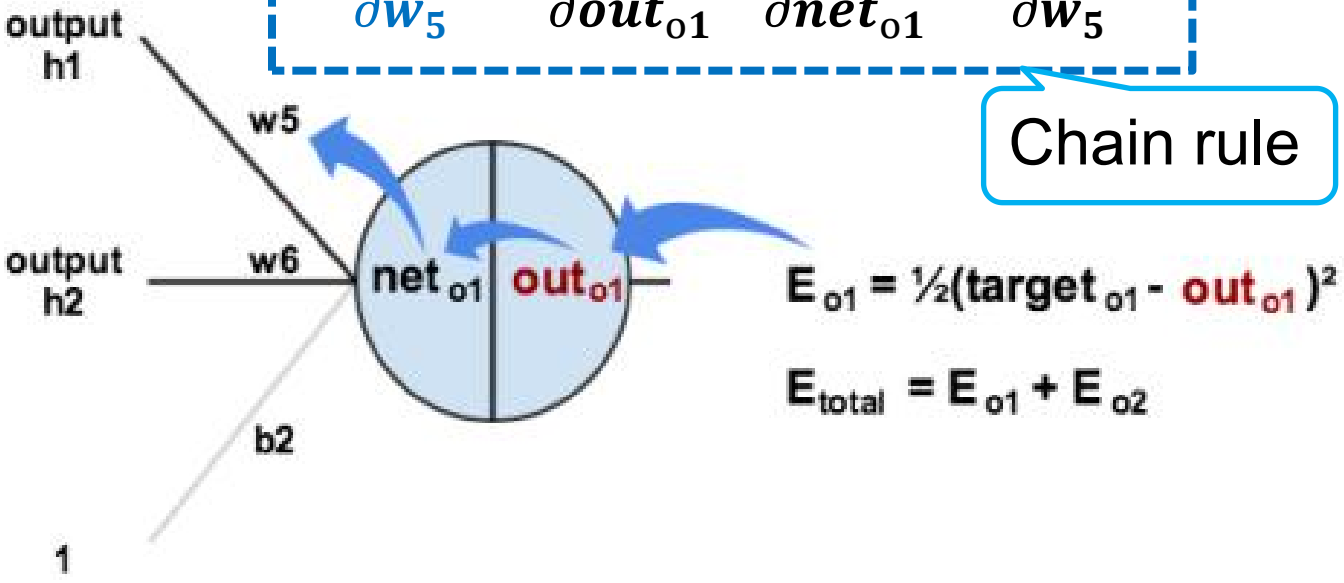
多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Output Layer):
  - Consider w_5, how much a change in w_5 affects the total error?

$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Chain rule



多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Output Layer):
  - Consider w_5, how much a change in w_5 affects the total error?

$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \sum \frac{1}{2}(target_i - output_i)^2$$

$$\frac{\partial E_{\text{total}}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * (-1) + 0$$
$$= -(target_{o1} - out_{o1}) = 0.74136507$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

多层感知器 - 训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Output Layer):
  - Consider w_5, how much a change in w_5 affects the total error?

$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.186815602$$

多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Output Layer):
  - Consider w_5, how much a change in w_5 affects the total error?

$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$net_{o1} = w_5 * h_1 + w_6 * h_2 + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * h_1 * w_5^{(1-1)} + 0 + 0$$

$$= h_1 = 0.593269992$$

多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Output Layer):
  - Consider w_5, how much a change in w_5 affects the total error?

$$\frac{\partial \boldsymbol{E}_{\text{total}}}{\partial \boldsymbol{w_5}} = \frac{\partial \boldsymbol{E}_{\text{total}}}{\partial \boldsymbol{out}_{\boldsymbol{o1}}} * \frac{\partial \boldsymbol{out}_{\boldsymbol{o1}}}{\partial \boldsymbol{net}_{\boldsymbol{o1}}} * \frac{\partial \boldsymbol{net}_{\boldsymbol{o1}}}{\partial \boldsymbol{w_5}}$$

$$= -(\boldsymbol{target}_{\boldsymbol{o1}} - \boldsymbol{out}_{\boldsymbol{o1}}) * \boldsymbol{out}_{\boldsymbol{o1}}(1 - \boldsymbol{out}_{\boldsymbol{o1}}) * \boldsymbol{h_1}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

  - To decrease the error, set learning rate $\boldsymbol{\eta}$ = 0.5

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Output Layer):
  - Repeat for w_6, w_7, w_8:

$$\boldsymbol{w_6}^+ = 0.408666186$$

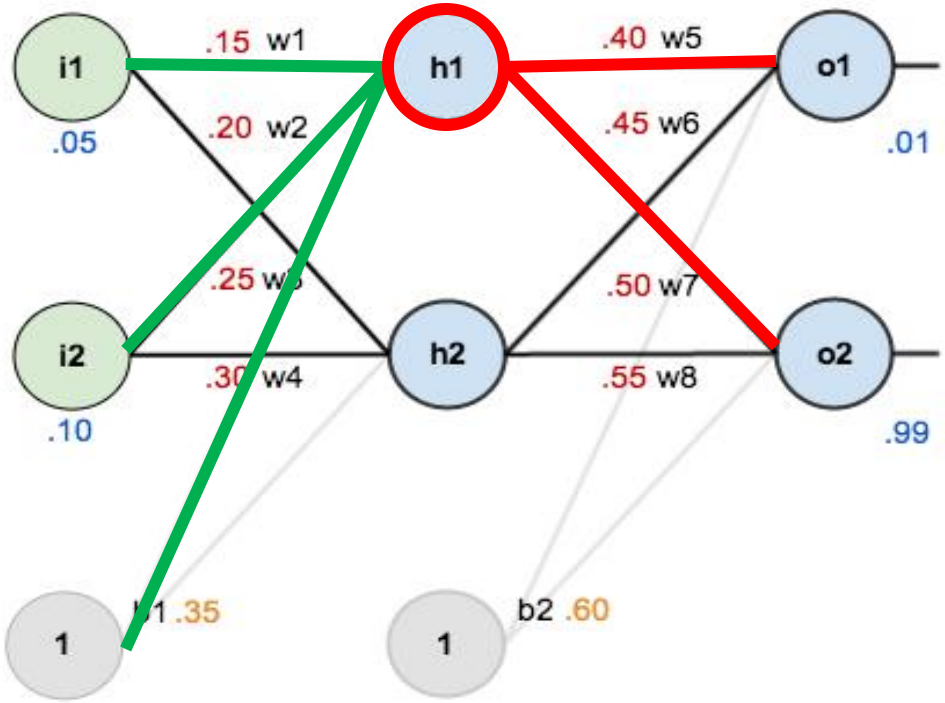
$$\boldsymbol{w_7}^+ = 0.511301270$$

$$\boldsymbol{w_8}^+ = 0.561370121$$

多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

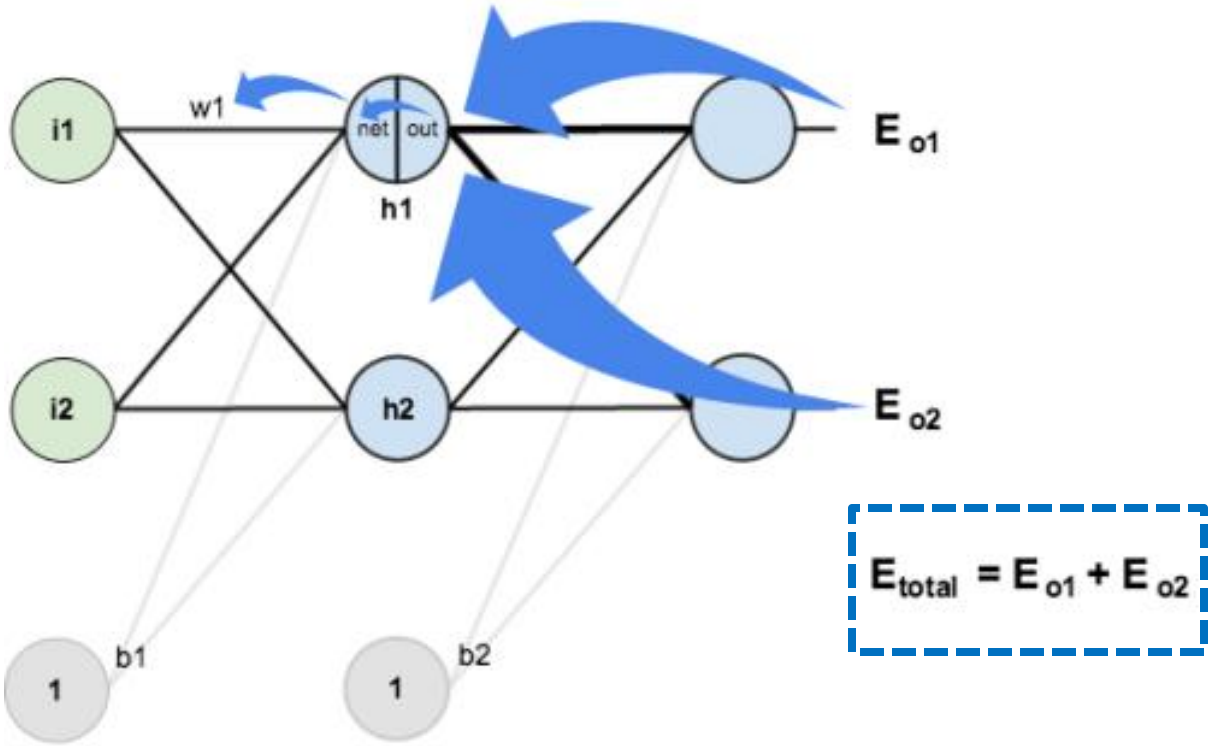
- The Backward Pass:
  - Hidden Layer:



多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Hidden Layer):
  - Continue the backwards pass for w_1:



多层感知器 - 反向传播训练

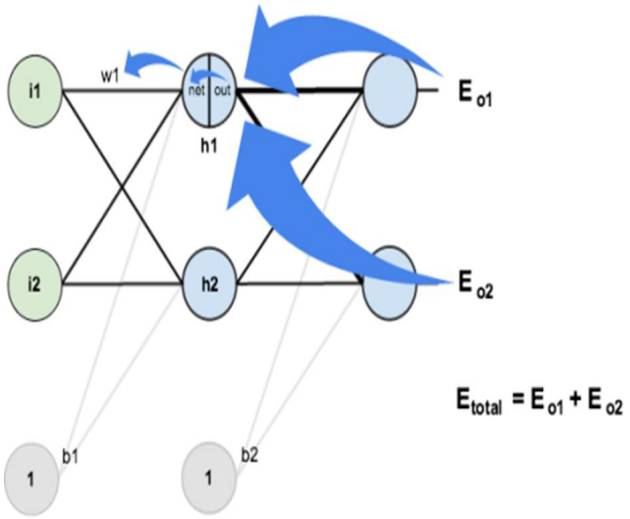
Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Hidden Layer):
  - Continue the backwards pass for w_1:

$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{\text{total}}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{\text{h1}}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$



多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- The Backward Pass (Hidden Layer):
  - Continue the backwards pass for w1:
  - Calculate as before and update w1.

$$\boldsymbol{w_1}^{+} = \boldsymbol{w_1} - \boldsymbol{\eta} * \frac{\partial \boldsymbol{E}_{\text{total}}}{\partial \boldsymbol{w_1}} = 0.149780716$$

  - Repeat for w2, w3, w4:

$$\boldsymbol{w_2}^{+} = 0.19956143$$

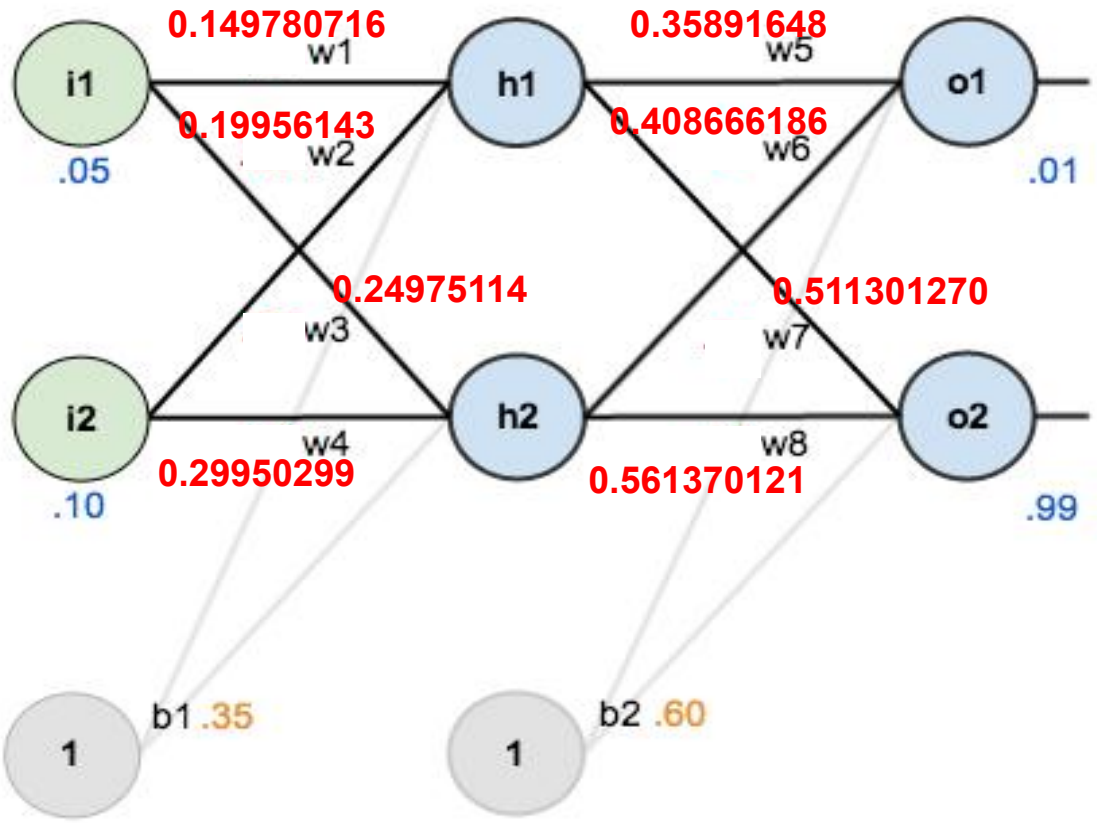
$$\boldsymbol{w_3}^{+} = 0.24985114$$

$$\boldsymbol{w_4}^{+} = 0.29950229$$

多层感知器 - 反向传播训练

Example: A neural network with two inputs, two hidden neurons, two output neurons.

- Update weights and repeat training.



多层感知器 - 反向传播训练

- After first round of backpropagation, total error: 0.298371109 → 0.291027924

- After repeating this process 10,000 times, the error plummets to 0.000035085.

- At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

多层感知器 - 训练

- 反向传播算法的启示是数学中的**链式法则**
- 从BP算法开始，研究者们更多地从数学上寻求问题的最优解。不再盲目模拟人脑网络（神经网络研究走向成熟的标志）。正如科学家们可以从鸟类的飞行中得到启发，但没有必要一定要完全模拟鸟类的飞行方式，也能制造可以飞天的飞机
- 机器学习问题之所以称为学习问题，而不是优化问题，就是因为它不仅要求数据在训练集上求得一个较小的误差，在测试集上也要表现好。模型需要具有**泛化** （generalization）能力，相关方法被称作正则化（regularization）。 神经网络中常用的泛化技术有**权重衰减** 等

多层感知器 - 影响

- 神经网络的学者们再次登上了《纽约时报》的专访。人们认为神经网络可以解决许多问题。就连娱乐界都开始受到了影响，《终结者》电影中的阿诺都赶时髦地说：“我的CPU是一个神经网络处理器，一个会学习的计算机”。

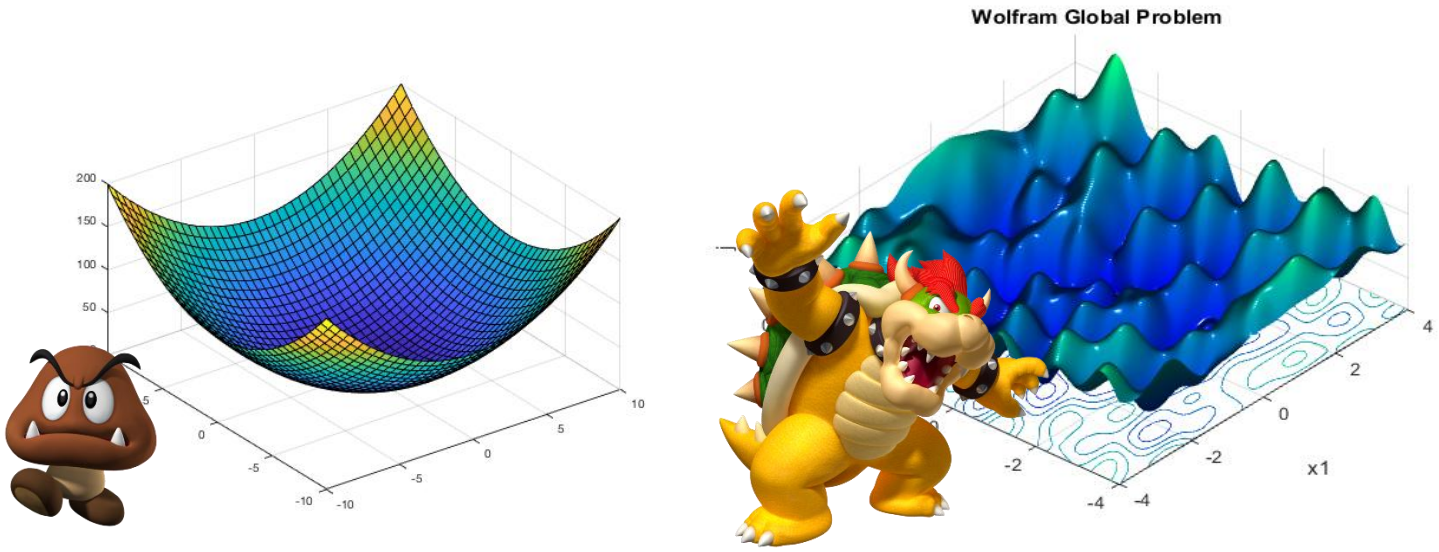
多层感知器 - Flappy Bird

多层感知器 - 存在问题

- 存在问题：尽管使用了BP算法，一次多层神经网络的训练仍然耗时太久，而且困扰训练优化的一个问题就是局部最优解问题，这使得神经网络的优化较为困难。同时，隐藏层的节点数需要调参，这使得使用不太方便，工程和研究人员对此多有抱怨

多层感知器 - 非凸模型优化问题

- 问题 a：非凸模型优化的一个问题就是局部最优解问题，这使得神经网络的优化较为困难。(Hmm... 真的非常难....)



- 非凸模型： $g(....g(\mathbf{W}(g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)}) + g(\mathbf{W}^{(2)} * \mathbf{a}^{(2)}))...)$
- 优化效率慢限制了感知器规模，节点数也需要精确调配。

多层感知器 - 梯度消失问题

- 问题 b：在深度网络中，不同的层的学习速度差异很大。使用sigmoid作为神经元的输入输出函数，在BP反向传播梯度时，每传递一层梯度衰减为原来的0.25。层数一多，梯度指数衰减后低层基本上接受不到有效的训练信号。

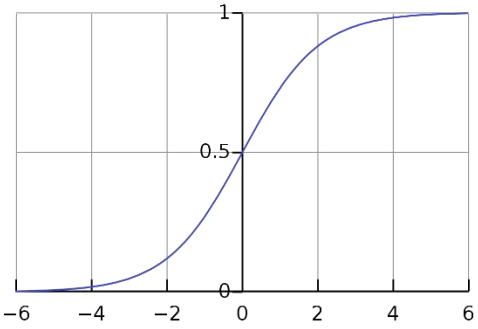
$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{\text{total}}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

the sigmoid function as $\sigma(x) = \frac{1}{1+e^{-x}}$.

0<= $\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$. <=0.25



- 梯度消失，使得多层网络等价于浅层网络的学习

多层感知器 - 冰河期

- 90年代中期，由Vapnik等人发明的SVM（Support Vector Machines，支持向量机）算法诞生，很快就在若干个方面体现出了对比神经网络的优势：无需调参；高效；全局最优解。基于以上种种理由，SVM迅速打败了神经网络算法成为主流。

Vladimir Vapnik

- 神经网络的研究再次陷入了冰河期。当时，只要你的论文中包含神经网络相关的字眼，非常容易被会议和期刊拒收，研究界那时对神经网络的不待见可想而知。

多层神经网络（深度学习） - 引入

- 在被人摒弃的10年中，有几个学者仍然在坚持研究，包括加拿大多伦多大学的Geoffery Hinton教授
- 2006年，Hinton在《Science》和相关期刊上发表了论文，首次提出了“深度信念网络”的概念
- 与传统的训练方式不同，“深度信念网络”有一个“**预训练**”（pre-training）的过程，这可以方便的让神经网络中的权值找到一个接近最优解的值，之后再使用“**微调**”（fine-tuning)技术来对整个网络进行优化训练。这两个技术的运用大幅度减少了训练多层神经网络的时间。他给多层神经网络相关的学习方法赋予了一个新名词--“**深度学习**”（深度学习是多个算法的总称）

多层神经网络（深度学习） - 引入

- 参数越多的模型复杂度越高
  - 优点： “容量”(capacity)越大，能完成更复杂的任务
  - 缺点： 训练效率低，数据量少时易陷入过拟合
- 缺点解决方案
  - 云计算，大数据时代下，计算能力大幅提升，可缓解训练低效性
  - 训练数据大幅增加，降低过拟合风险
- 以深度学习为代表的复杂模型开始受人关注

多层神经网络（深度学习） - 引入

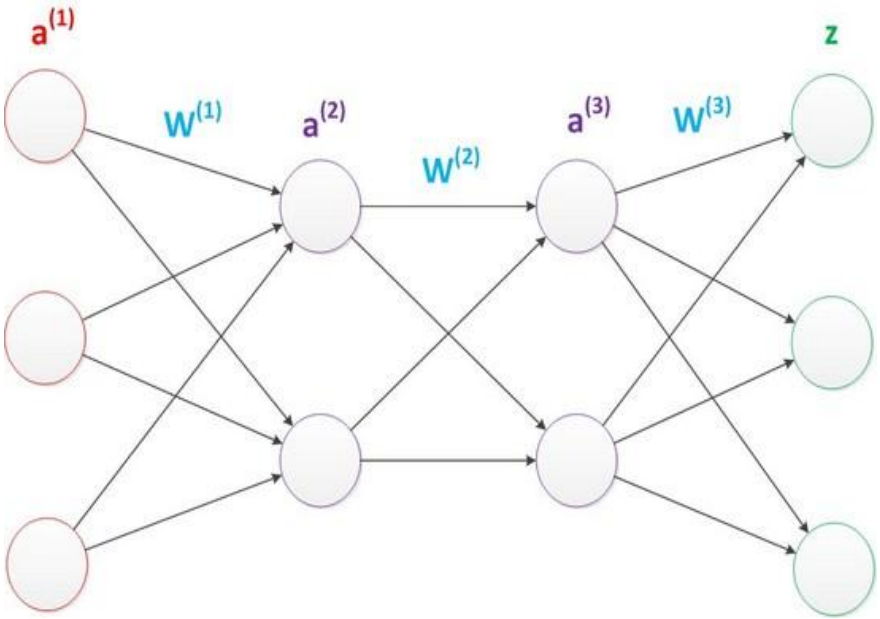
- 很快，深度学习在语音识别领域暂露头角。接着，2012年，Hinton与他的学生在 ImageNet 竞赛中，用多层的卷积神经网络成功地对包含一千类别的一百万张图片进行了训练，取得了分类错误率 15 % 的好成绩，这个成绩比第二名高了近 11 个百分点，充分证明了多层神经网络识别效果的优越性
- 在这之后，关于深度神经网络的研究与应用不断涌现。

Geoffery Hinton
1986，BP算法

Geoffery Hinton

多层神经网络（深度学习） - 结构

- 延续两层神经网络的方式来设计一个多层神经网络
- 在两层神经网络的输出层后面，继续添加层次。原来的输出层变成中间层，新加的层次成为新的输出层。所以可以得到下图。



多层神经网络

正向传播

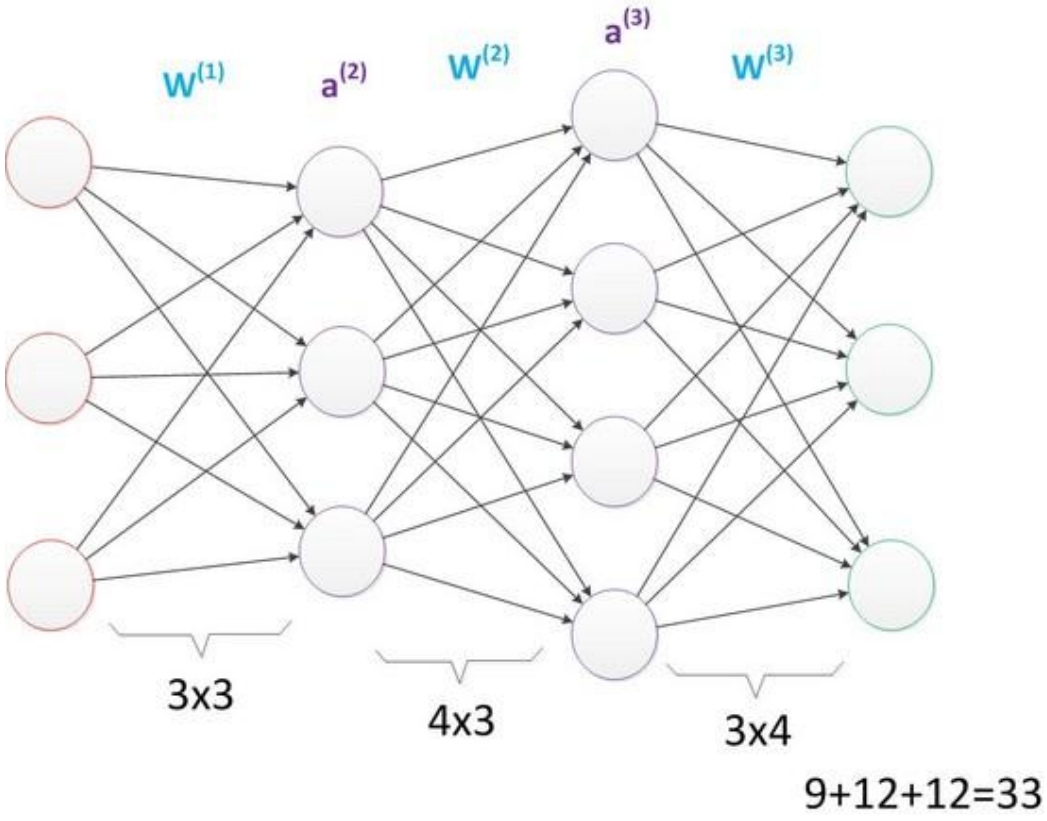
$$g(W^{(1)} * a^{(1)}) = a^{(2)};$$

$$g(W^{(2)} * a^{(2)}) = a^{(3)};$$

$$g(W^{(3)} * a^{(3)}) = z;$$

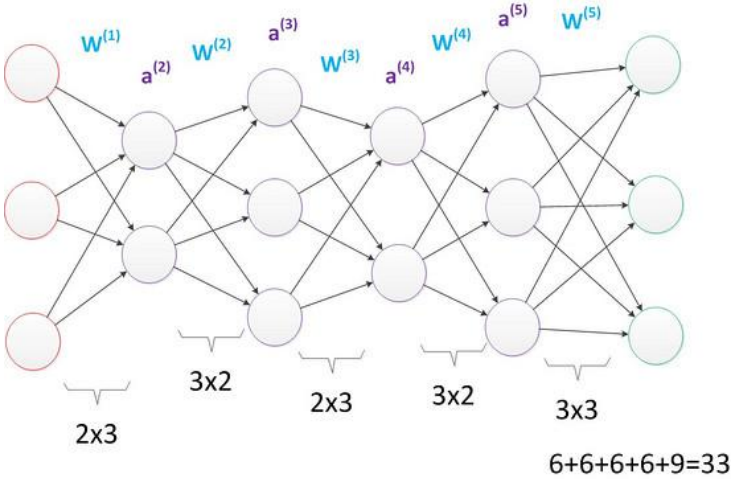
多层神经网络（深度学习） - 结构

- 网络的参数为33个。



多层神经网络（深度学习） - 结构

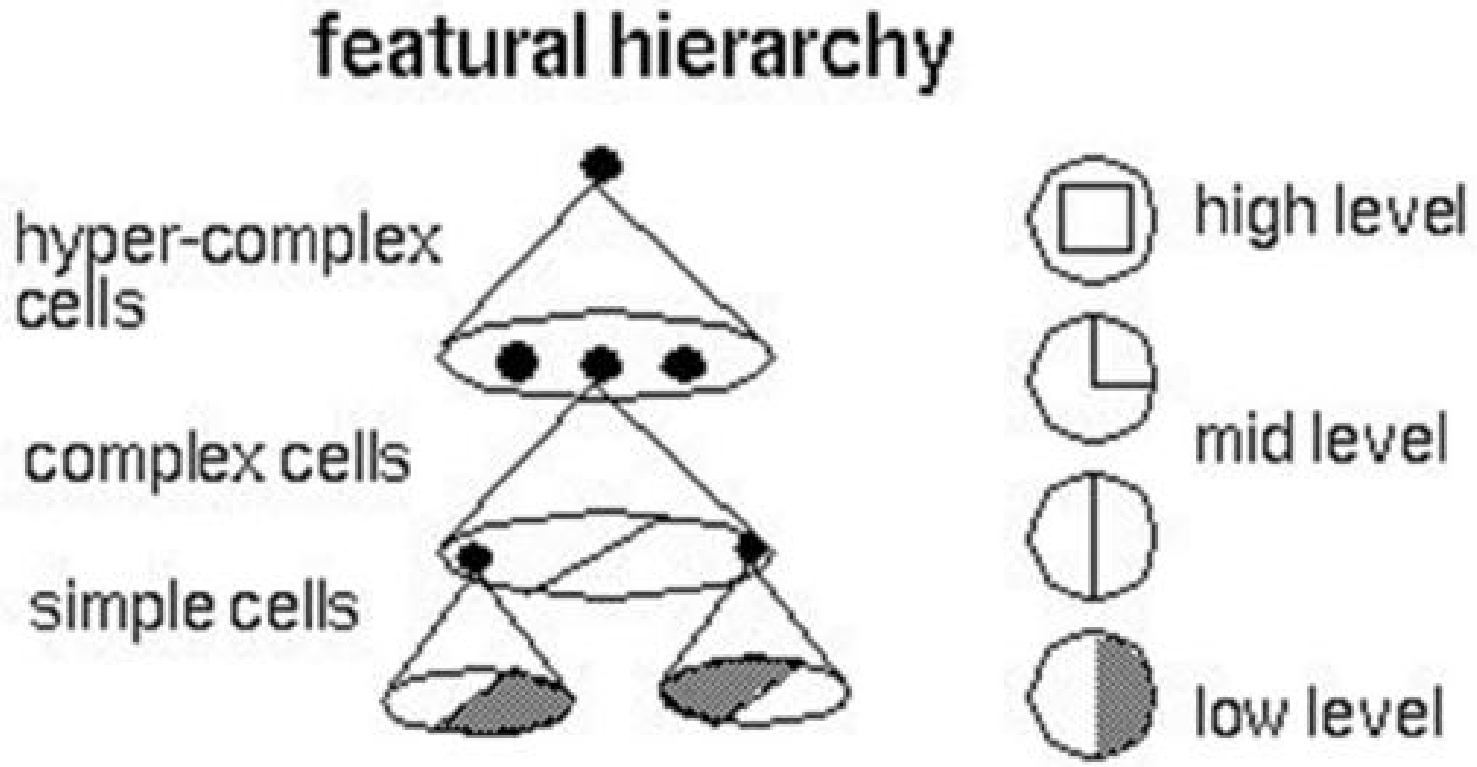
- 在参数一致的情况下，我们也可以获得一个“更深”的网络



- 参数数量仍然是33，但却有4个中间层，更多的层次带来更深入的特征表示能力，以及更强的函数模拟能力
- 随着层数增加，每一层对于前一层次的抽象表示更深入。每一层神经元学习到的是前一层神经元值的更抽象的表示。例如第一个隐藏层学习到的是 “边缘” 的特征，第二个隐藏层学习到由“边缘”组成的“形状”，第三个隐藏层学习到的是由“形状”组成的“图案”的特征，最后的隐藏层学习到的是由“图案”组成的“目标”的特征
- 通过抽取更抽象的特征来对事物进行区分，从而获得更好的区分与分类能力

多层神经网络（深度学习） - 效果

- 逐层特征学习例子：



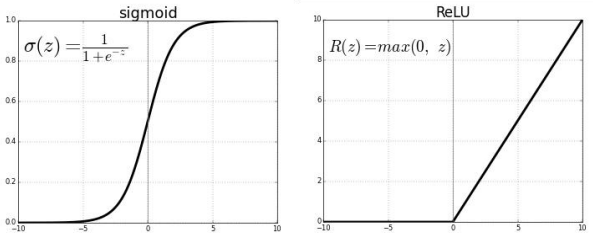
多层神经网络（特征学习）

多层神经网络（深度学习） - 效果

- 更强的函数模拟能力是由于随着层数的增加，整个网络的参数就越多。而神经网络其实本质就是模拟特征与目标之间的真实关系函数的方法，更多的参数意味着其模拟的函数可以更加的复杂，可以有更多的 **容量** （capcity）去拟合真正的关系（**强悍的非线性函数拟合**）
- 在参数数量一样的情况下，更深的网络往往具有比浅层的网络更好的识别效率
  - 2012年起，每年获得ImageNet冠军的深度神经网络的层数逐年增加
  - 2015年最好的方法GoogleNet是一个多达22层的神经网络
  - 2016拿到最好成绩的MSRA团队的方法使用的更是一个152层的网络！
  - 2016商汤团队1207层的网络

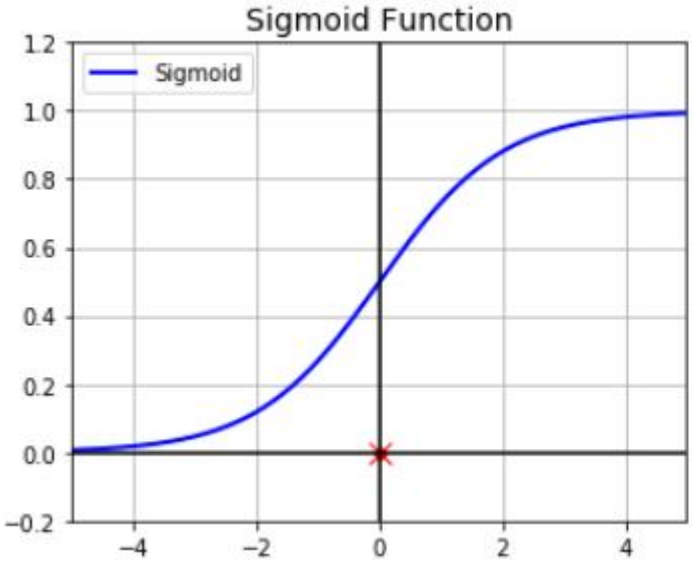
多层训练解决手段

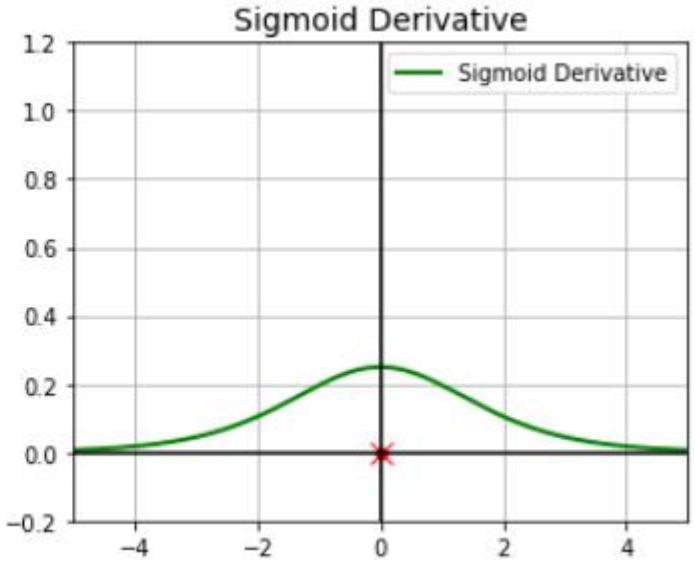
- **2006年Hinton提出的逐层预训练方法**
  - pre-training + fine-tunning…
- **使用Rectified Linear Unit(Relu)激活函数**

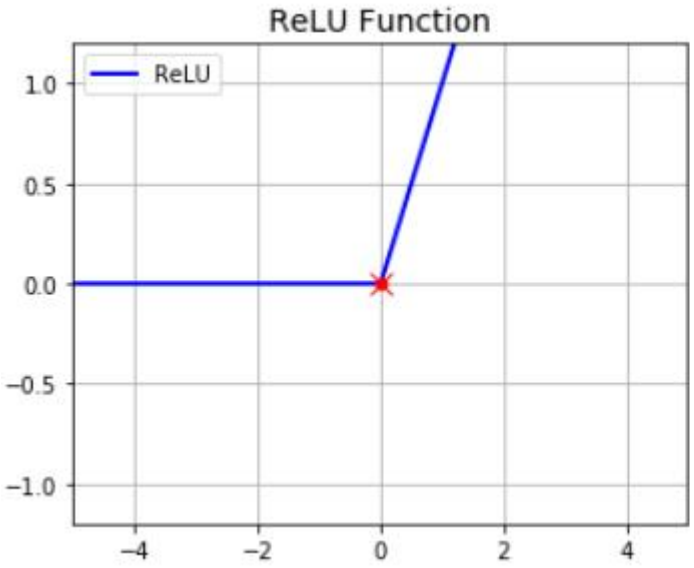


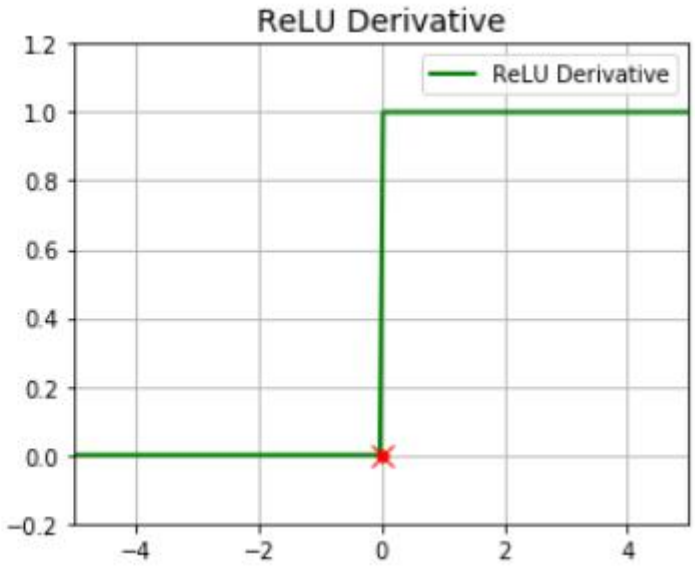
  - sigmoid等函数，算激活函数时（指数运算），计算量大，反向传播求误差梯度时，求导涉及除法，计算量相对大，而采用Relu激活函数，整个过程的计算量节省很多
  - sigmoid函数反向传播时，容易出现梯度消失
  - Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生
- **Dropout**
  - 模型训练时随机让网络某些隐含层节点的权重不工作，防止过拟合

不同激活函数求导

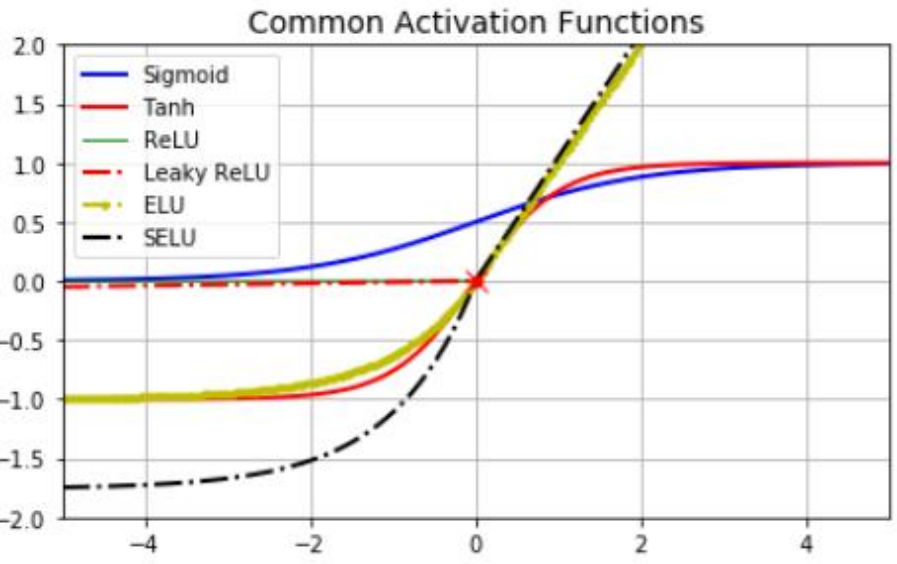

Sigmoid Function
Sigmoid
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−4
−2
0
2
4

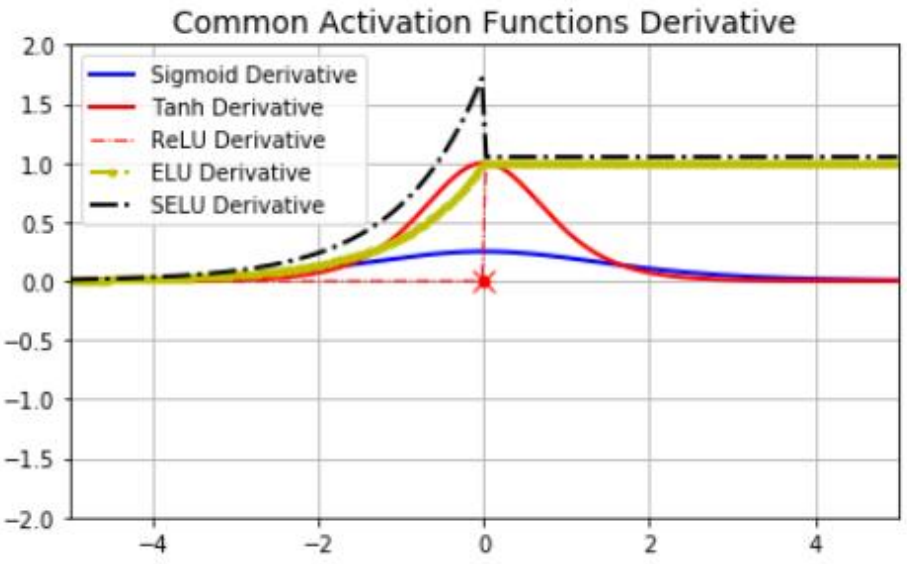

Sigmoid Derivative
Sigmoid Derivative
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−4
−2
0
2
4


ReLU Function
ReLU
1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4


ReLU Derivative
ReLU Derivative
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−4
−2
0
2
4

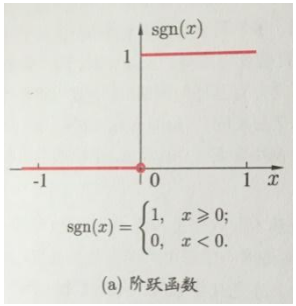
不同激活函数求导

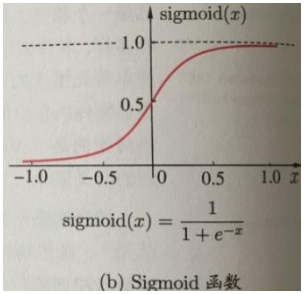

Common Activation Functions
Sigmoid
Tanh
ReLU
Leaky ReLU
ELU
SELU
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−4
−2
0
2
4


Common Activation Functions Derivative
Sigmoid Derivative
Tanh Derivative
ReLU Derivative
ELU Derivative
SELU Derivative
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−4
−2
0
2
4

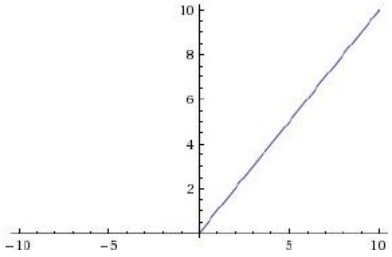
多层神经网络（深度学习） - 训练

- 单层神经网络：sgn函数
- 两层神经网络：sigmoid函数





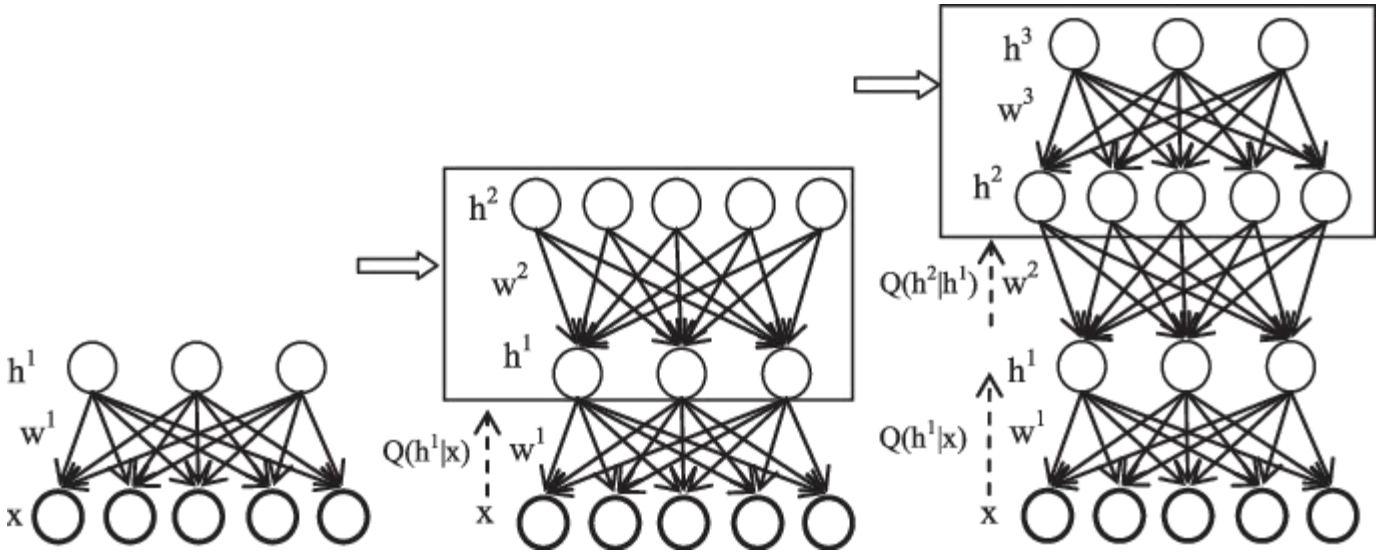
- 多层神经网络：ReLU函数，y=max(x,0)，在x大于0，输出就是输入，而在x小于0时，输出就保持为0。函数设计启发来自于生物神经元对于激励的线性响应，以及当低于某个阈值后就不再响应的模拟



- 神经网络的层数增加了，参数也增加了，表示能力大幅度增强，很容易出现**过拟合现象**。 因此正则化技术就显得十分重要。Dropout技术是目前使用的最多的正则化技术

深度学习 - 无监督逐层训练

- 多隐层网络训练的方法：无监督逐层训练 (unsupervised layer-wise training)
  - 预训练：每次训练一层隐结点，训练时将上一层隐结点的输出作为输入，而本层隐结点的输出作为下一层隐结点的输入。
  - 在预训练全部完成后， 对整个网络进行“微调” (fine-tunning)训练。



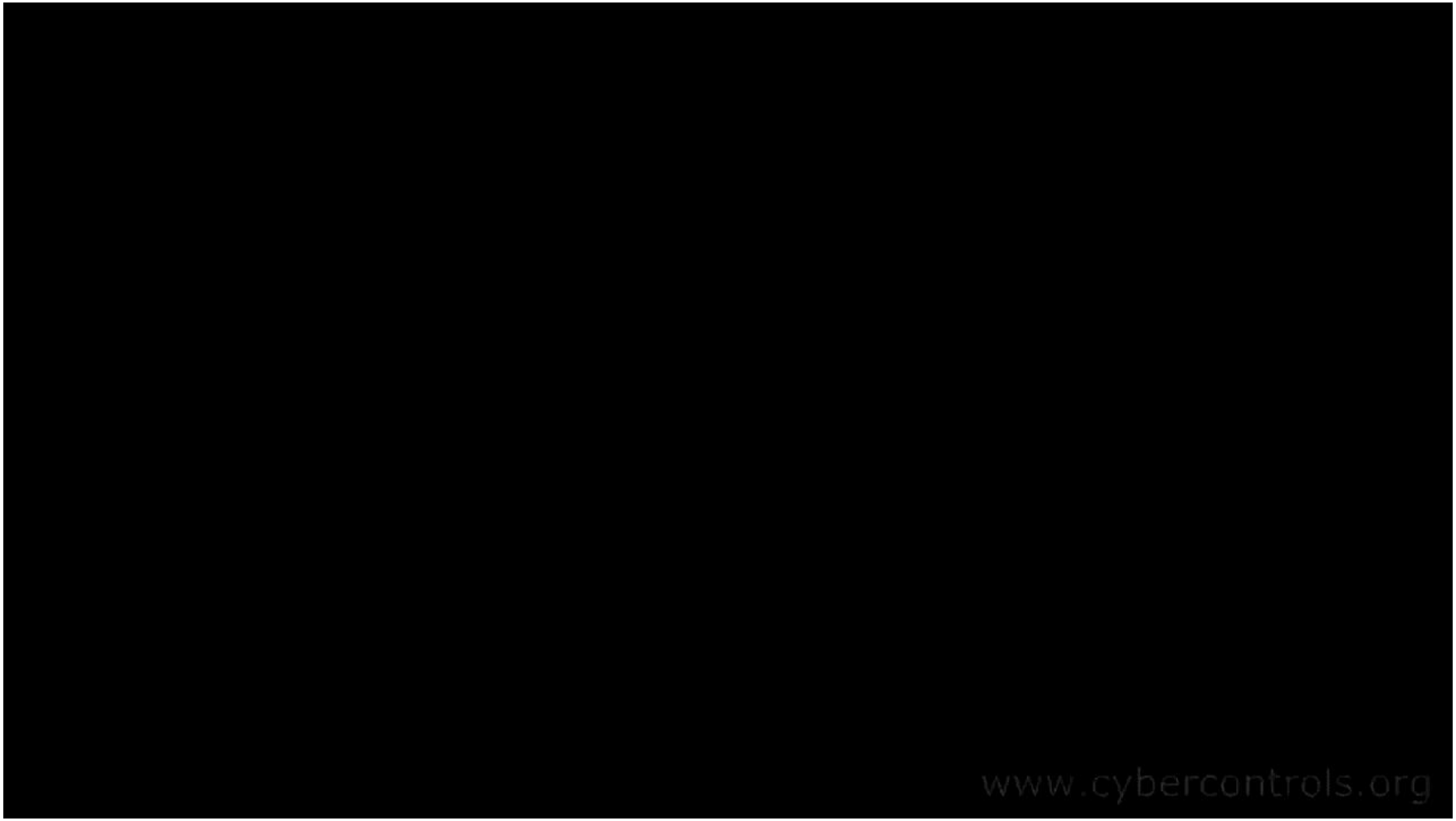
无监督逐层训练

- 举例 - - 深度信念网络 (Deep belief network, DBN):
  - 每层都是一个受限Boltzmann机，即整个网络可视为若干个受限玻尔兹曼机 (RBM) 堆叠而成
  - 首先训练第一层（是关于训练样本的RBM模型，可按标准的RBM训练)
  - 将第一层预训练好的隐结点视为第二层的输入结点；对第二层进行预训练；... ...
  - 各层预训练完成后，利用BP算法对整个网络进行训练

深度学习 - 无监督逐层训练

- “预训练” + “微调”
  - 可视为将大量参数分组，对每组先找到局部看来比较好的设置
  - 再基于这些局部较优的结果联合起来进行全局寻优
  - 这样利用了模型大量参数所提供的自由度的同时，有效地节省了训练开销

Neural Network 3D Visualization



深度学习 - 影响

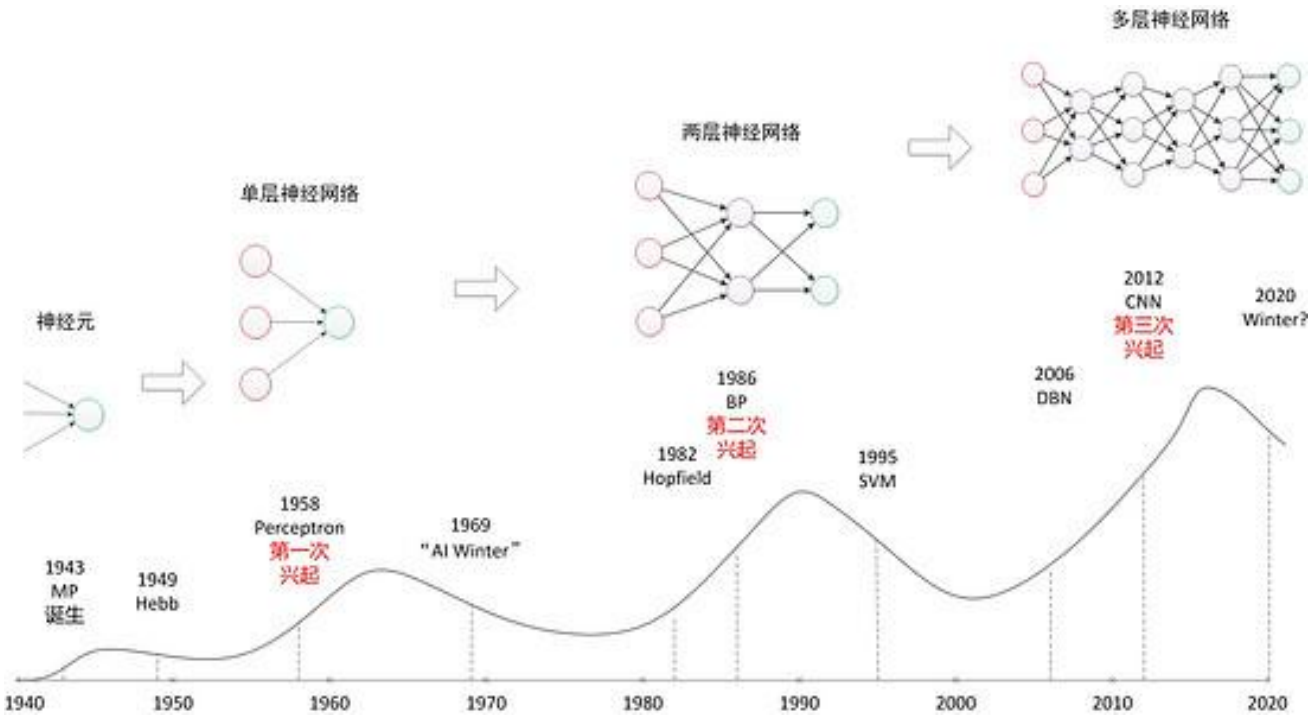
- 目前，深度神经网络在人工智能界占据统治地位。神经网络界当下的四位引领者除了Ng，Hinton以外，还有CNN的发明人Yann Lecun，以及《Deep Learning》的作者Bengio
- 马斯克的OpenAI项目，邀请Bengio作为高级顾问。马斯克认为，人工智能技术不应该掌握在大公司如Google，Facebook的手里，更应该作为一种开放技术，让所有人都可以参与研究。

Yann LeCun

Yoshua Bengio

历史回顾 - 影响

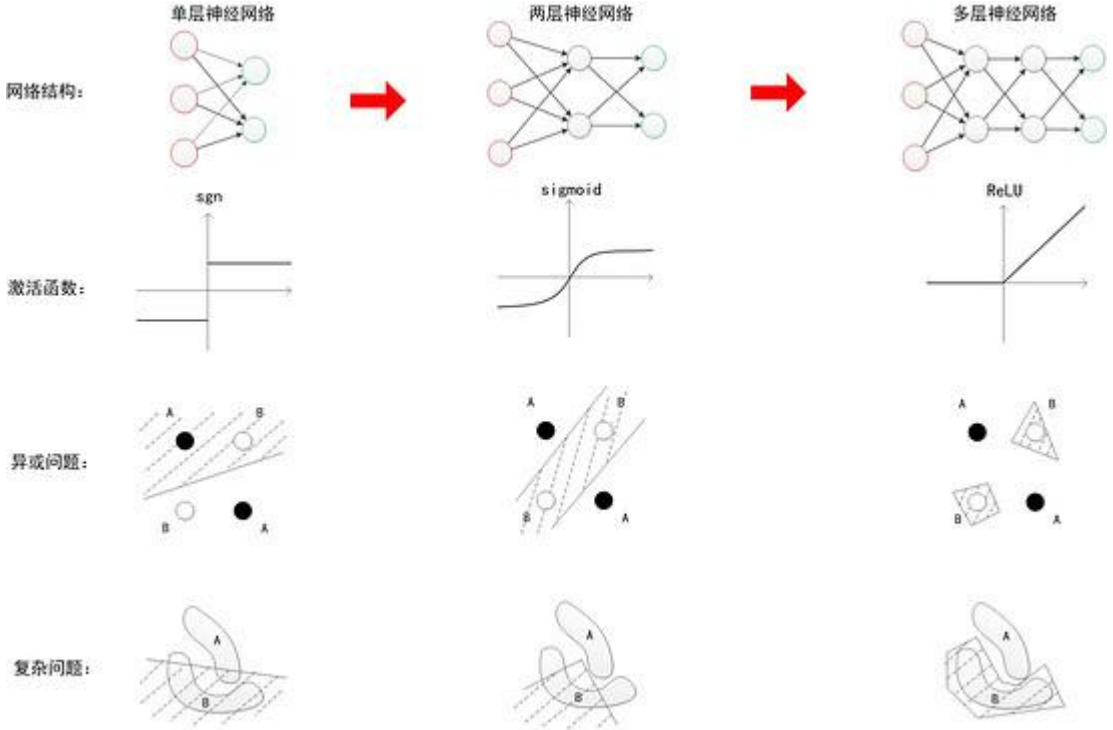
- 神经网络发展历程：曲折荡漾，既有被人捧上天的时刻，也有摔落在街头无人问津的时段，中间经历了数次大起大落
- 从单层神经网络（感知器）开始，到包含一个隐藏层的两层神经网络，再到多层的深度神经网络，一共有三次兴起过程



三起三落的神经网络

历史回顾 - 效果

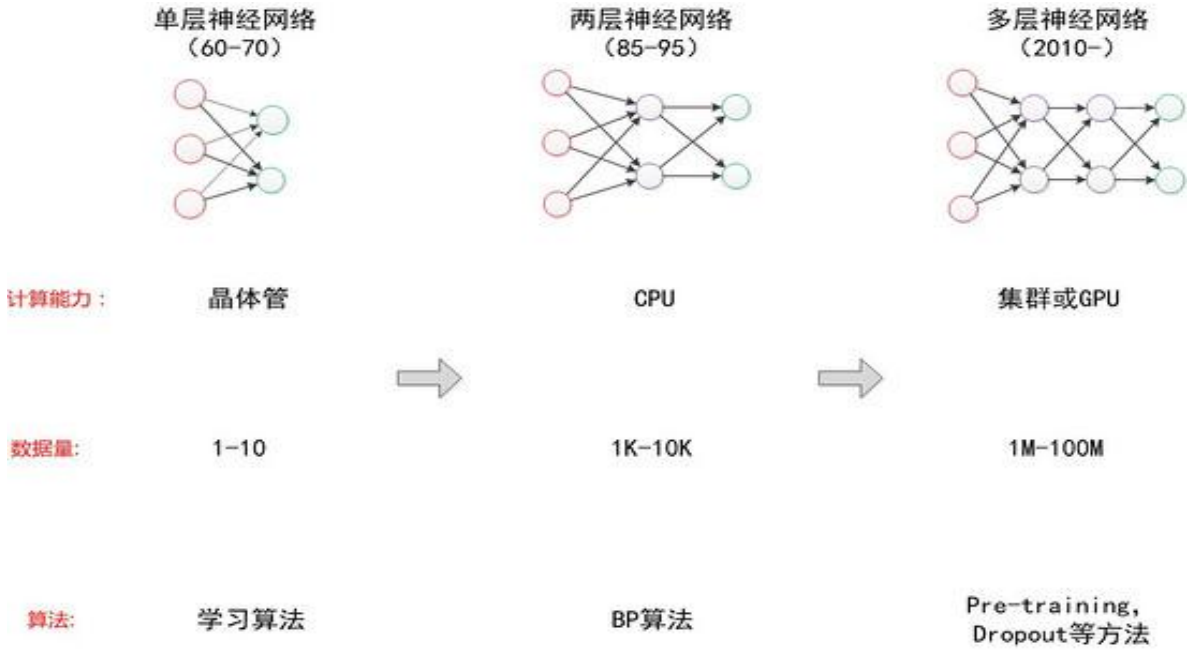
- 神经网络为什么能这么火热？简而言之，就是其学习效果的强大。随着神经网络的发展，其表示性能越来越强
- 从单层神经网络，到两层神经网络，再到多层神经网络，下图说明了，随着网络层数的增加，以及激活函数的调整，神经网络所能拟合的决策分界平面的能力



表示能力不断增强

历史回顾 - 因素

- 神经网络发展外在原因：更强的计算性能，更多的数据，以及更好的训练方法。只有满足这些条件时，神经网络的函数拟合能力才能得已体现
- Hinton 在 2006年 的论文里说道的
  “... provided that computers were fast enough, data sets were big enough, and the initial weights were close enough to a good solution. All three conditions are now satisfied. ”



神经网络类别

- 神经网络中的不同类别:

前馈神经网络：有向图是没有回路的

反馈神经网络：有向图是有回路的，Hopfield网络是反馈神经网络，RNN也属于一种反馈神经网络

展望

- 神经网络以及深度学习会不会像以往一样再次陷入谷底?
  - 可能取决于量子计算机的发展
  - 目前已知的最大神经网络跟人脑的神经元数量相比，仅不及1%左右。想实现人脑神经网络的模拟，需要更强大计算能力
  - 谷歌开展量子计算机D-wave的研究，希望用量子计算来进行机器学习
- 距离真正的人工智能还有很大的距离。拿计算机视觉方向来说，面对稍微复杂一些的场景，以及易于混淆的图像，计算机就可能难以识别
  - 虽然计算机需要很大的运算量才能完成一个普通人简单能完成的识图工作，但计算机最大的优势在于并行化与批量推广能力。可以轻易地将以前需要人眼去判断的工作交给计算机做，而且几乎没有任何的推广成本。正如火车刚诞生的时候，有人嘲笑它又笨又重，速度还没有马快。但是很快规模化推广的火车就替代了马车的使用

Reference

- 神经网络浅讲：从神经元到深度学习 http://www.cnblogs.com/subconscious/p/5058741.html
- http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
- 一文读懂卷积神经网络，http://www.36dsj.com/archives/24006
- Deep Learning（深度学习）学习笔记整理系列之（七） http://blog.csdn.net/zdy0_2004/article/details/49683313
- Neural Networks and Learning Machines(神经网络与机器学习). Simon Haykin
- Pattern Recognition And Machine Learning(模式识别与机器学习). Christopher M. Bishop ----Chap 5.
- http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial
- Wang, H., Raj, B., & Xing, E. P. (2017). On the origin of deep learning. arXiv preprint arXiv:1702.07800.