

Reactor

一起学人工智能系列
- 非线性分类

2021-10-20



Map



个人介绍



Kinfey Lo – (卢建晖)

Microsoft Cloud Advocate

前微软MVP、Xamarin MVP和微软RD，拥有超过10年的云原生、人工智能和移动应用经验，为教育、金融和医疗提供应用解决方案。 Microsoft Ignite, TechEd 会议讲师，Microsoft AI 黑客马拉松教练，目前在微软，为技术人员和不同行业宣讲技术和相关应用场景。



爱编程(Python , C# , TypeScript , Swift , Rust , Go)

专注于人工智能，云原生，跨平台移动开发

Github : <https://github.com/kinfey>

Email : kinfeylo@microsoft.com Blog : <https://blog.csdn.net/kinfey>

Twitter : @Ljh8304

回顾



多项式回归

一元一次线性模型

因为只有一项，所以不能称为多项式了。它可以解决单变量的线性回归，我们在前面学习过相关内容。其模型为：

$$z = xw + b \quad (1)$$

多元一次多项式

多变量的线性回归，我们在前面也学习过相关内容。其模型为：

$$z = x_1w_1 + x_2w_2 + \dots + x_mw_m + b \quad (2)$$

这里的多变量，是指样本数据的特征值为多个，上式中的 x_1, x_2, \dots, x_m 代表了m个特征值。

多项式回归

一元多次多项式

单变量的非线性回归，比如正弦曲线的拟合问题，很明显不是线性问题，但是只有一个x特征值，所以不满足前两种形式。如何解决这种问题呢？

有一个定理：任意一个函数在一个较小的范围内，都可以用多项式任意逼近。因此在实际工程实践中，有时候可以不管y值与x值的数学关系究竟是什么，而是强行用回归分析方法进行近似的拟合。

那么如何得到更多的特征值呢？对于只有一个特征值的问题，人们发明了一种聪明的办法，就是把特征值的高次方作为另外的特征值，加入到回归分析中，用公式描述：

$$z = xw_1 + x^2w_2 + \dots + x^m w_m + b \quad (3)$$

上式中x是原有的唯一特征值， x^m 是利用x的m次方作为额外的特征值，这样就把特征值的数量从1个变为m个。

换一种表达形式，令： $x_1 = x, x_2 = x^2, \dots, x_m = x^m$ ，则：

$$z = x_1 w_1 + x_2 w_2 + \dots + x_m w_m + b \quad (4)$$

可以看到公式4和上面的公式2是一样的，所以解决方案也一样。

多项式回归

多元多次多项式

多变量的非线性回归，其参数与特征组合繁复，但最终都可以归结为公式2和公式4的形式。

所以，不管是几元几次多项式，我们都可以使用线性回归学到的方法来解决。在用代码具体实现之前，我们先学习一些前人总结的经验。先看一个被经常拿出来讲解的例子，如图9-3所示。

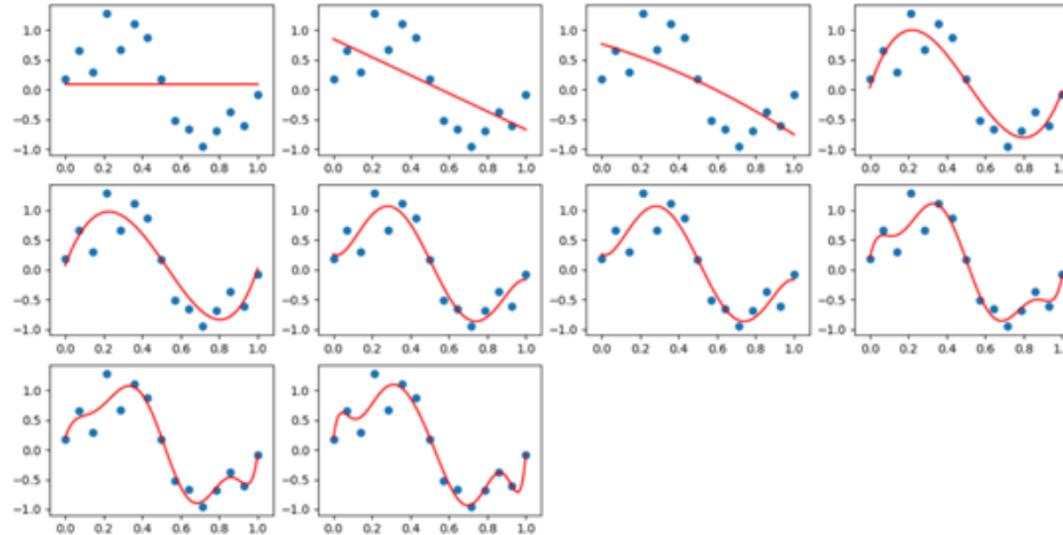
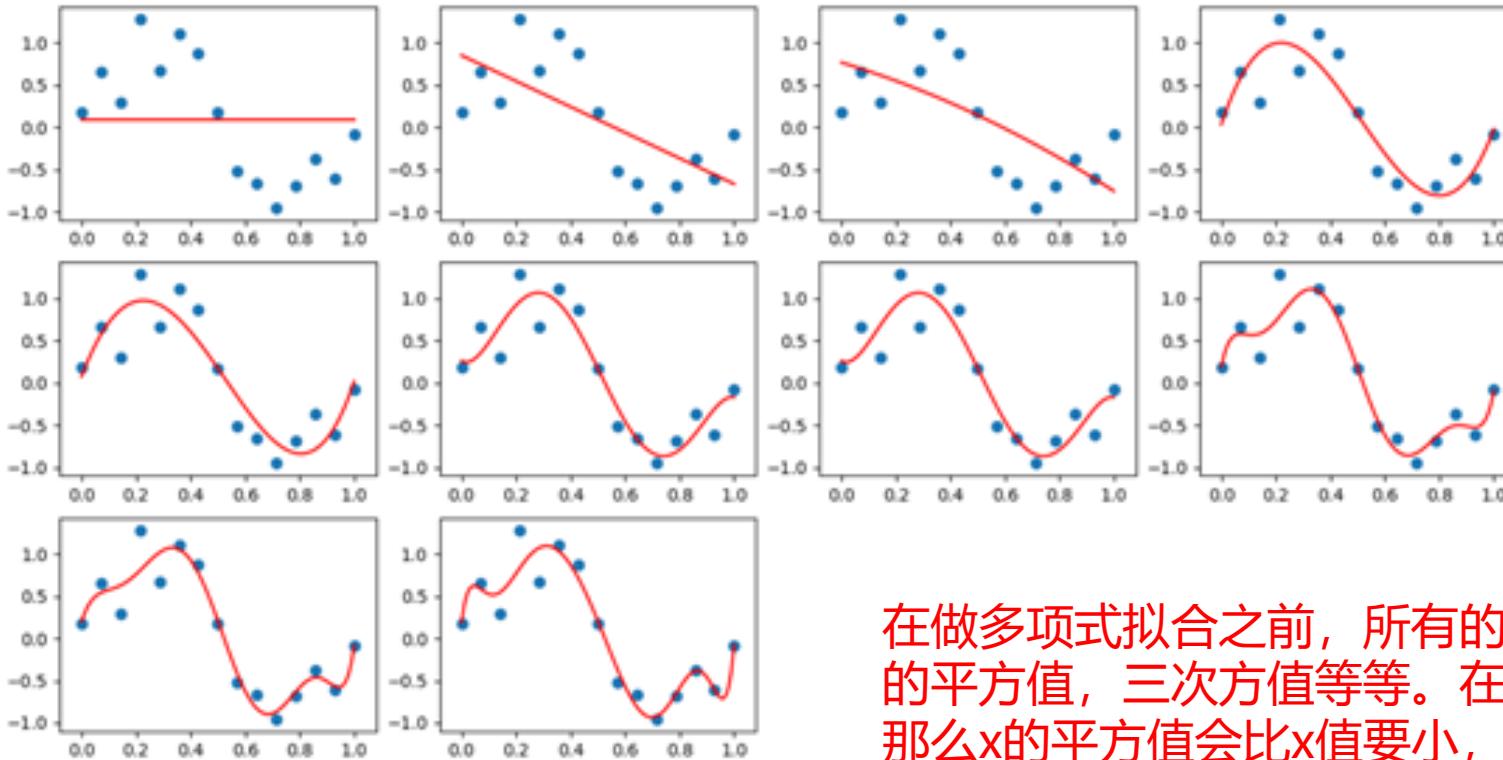


图9-3 对有噪音的正弦曲线的拟合

一堆散点，看上去像是一条带有很大噪音的正弦曲线，从左上到右下，分别是1次多项式、2次多项式……10次多项式，其中：

- 第4、5、6、7图是比较理想的拟合
- 第1、2、3图欠拟合，多项式的次数不够高
- 第8、9、10图，多项式次数过高，过拟合了

多项式回归



在做多项式拟合之前，所有的特征值都会先做归一化，然后再获得 x 的平方值，三次方值等等。在归一化之后， x 的值变成了 $[0,1]$ 之间，那么 x 的平方值会比 x 值要小， x 的三次方值会比 x 的平方值要小。假设 $x=0.5$, $x*x=0.25$, $x*x*x=0.125$, 所以次数越高，权重值会越大，特征值与权重值的乘积才会是一个不太小的数，以此来弥补特征值小的问题。

万能近似定理

万能近似定理(universal approximation theorem) 是深度学习最根本的理论依据。它证明了在给定网络具有足够多的隐藏单元的条件下，配备一个线性输出层和一个带有任何“挤压”性质的激活函数（如Sigmoid激活函数）的隐藏层的前馈神经网络，能够以任何想要的误差量近似任何从一个有限维度的空间映射到另一个有限维度空间的Borel可测的函数。

万能近似定理

前馈网络的导数也可以以任意好地程度近似函数的导数。

万能近似定理其实说明了理论上神经网络可以近似任何函数。但实践上我们不能保证学习算法一定能学习到目标函数。即使网络可以表示这个函数，学习也可能因为两个不同的原因而失败：

1. 用于训练的优化算法可能找不到用于期望函数的参数值；
2. 训练算法可能由于过拟合而选择了错误的函数。

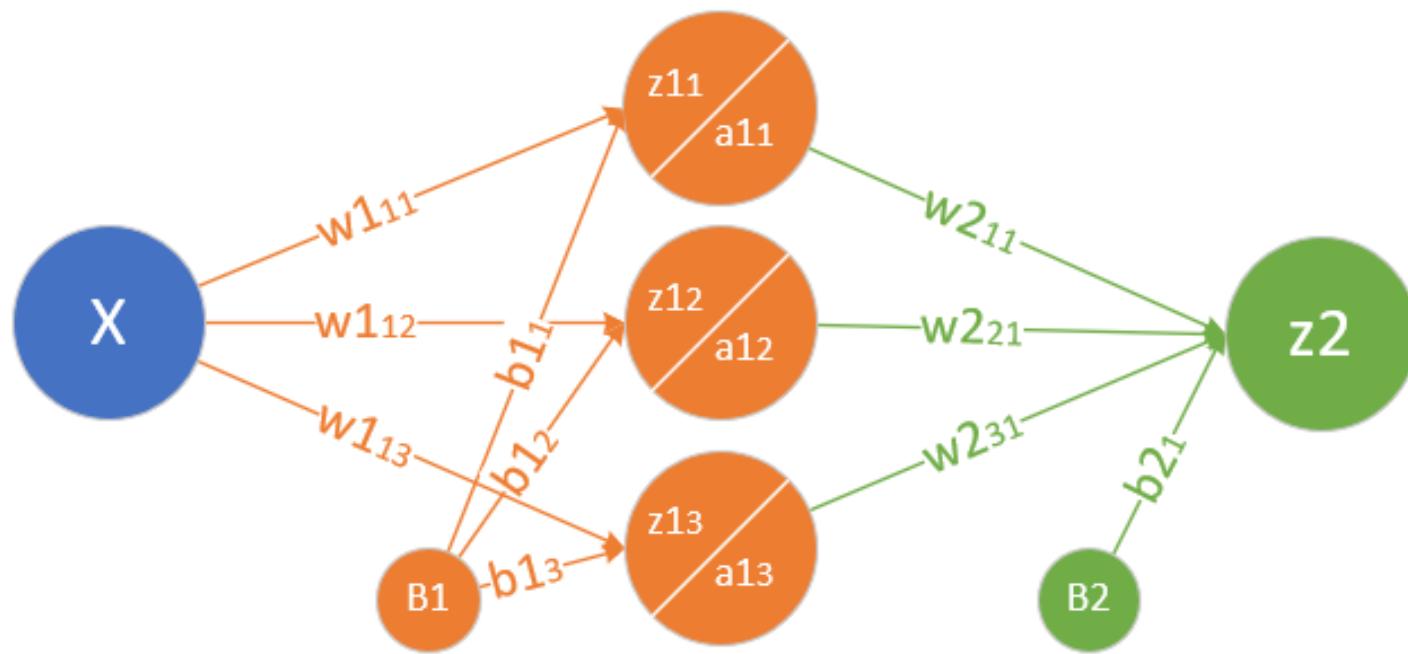
万能近似定理

根据“没有免费的午餐”定理，说明了没有普遍优越的机器学习算法。前馈网络提供了表示函数的万能系统，在这种意义上，给定一个函数，存在一个前馈网络能够近似该函数。但不存在万能的过程既能够验证训练集上的特殊样本，又能够选择一个函数来扩展到训练集上没有的点。

总之，具有单层的前馈网络足以表示任何函数，但是网络层可能大得不可实现，并且可能无法正确地学习和泛化。在很多情况下，使用更深的模型能够减少表示期望函数所需的单元的数量，并且可以减少泛化误差。

定义神经网络

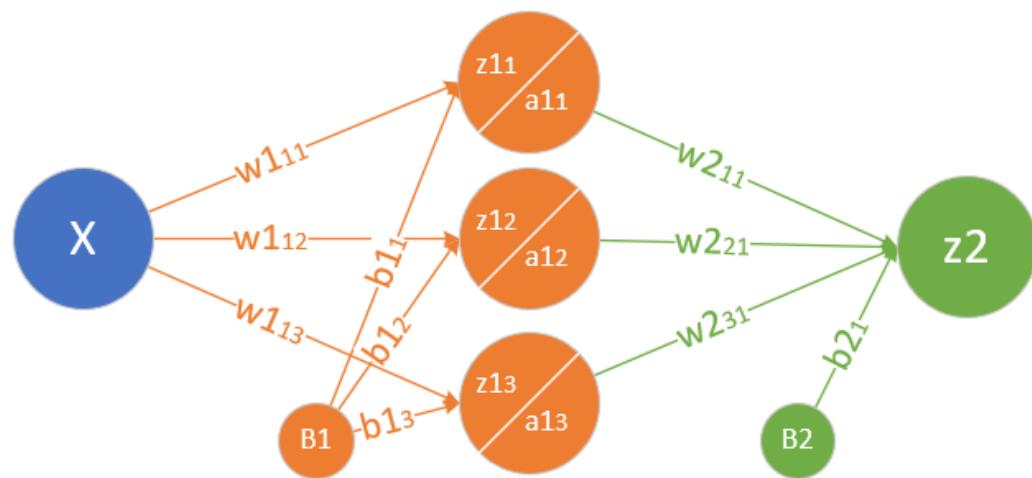
根据万能近似定理的要求，我们定义一个两层的神经网络，输入层不算，一个隐藏层，含3个神经元，一个输出层。



定义神经网络

权重矩阵W1/B1

$$W1 = (w_{11} \quad w_{12} \quad w_{13})$$



隐层

我们用3个神经元:

$$Z1 = (z_{11} \quad z_{12} \quad z_{13})$$

$$A1 = (a_{11} \quad a_{12} \quad a_{13})$$

权重矩阵W2/B2

W2的尺寸是3x1, B2的尺寸是1x1。

$$W2 = \begin{pmatrix} w_{11}^2 \\ w_{21}^2 \\ w_{31}^2 \end{pmatrix}$$

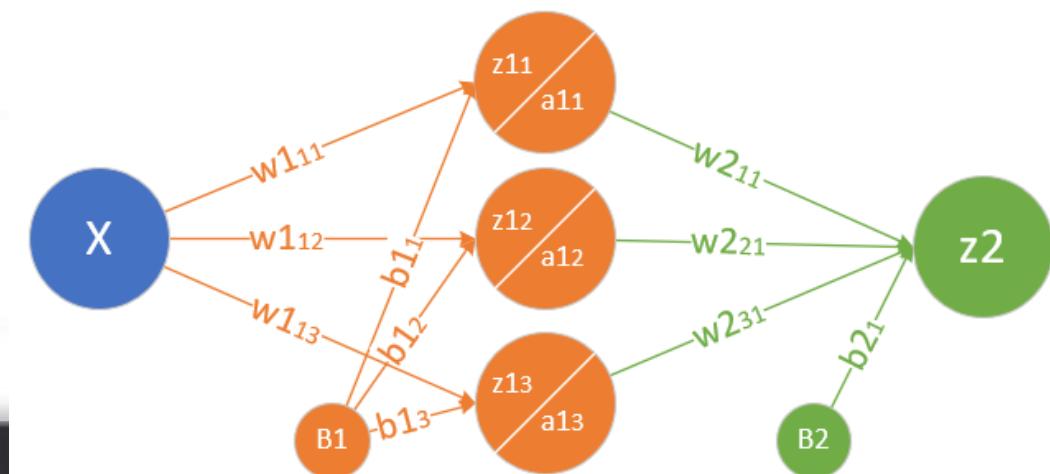
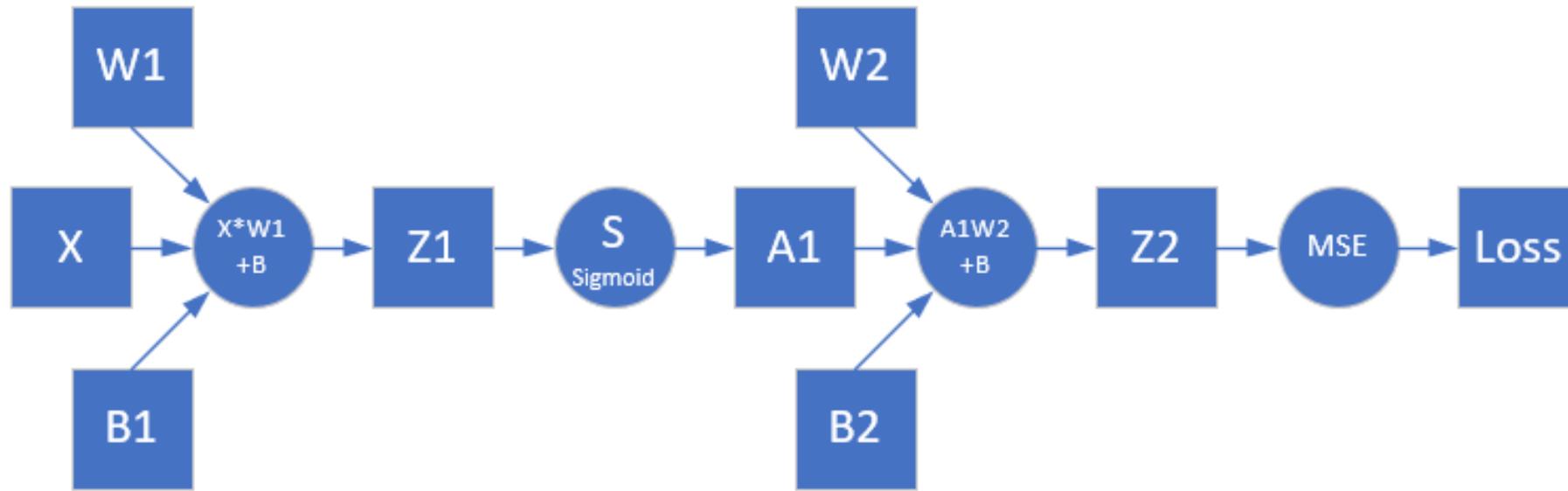
$$B2 = (b_1^2)$$

输出层

由于我们只想完成一个拟合任务, 所以输出层只有一个神经元, 尺寸为1x1:

$$Z2 = (z_1^2)$$

定义神经网络 - 前向计算



隐层

- 线性计算

$$z_1^1 = x \cdot w_{11}^1 + b_1^1$$

$$z_2^1 = x \cdot w_{12}^1 + b_2^1$$

$$z_3^1 = x \cdot w_{13}^1 + b_3^1$$

矩阵形式:

$$Z1 = X \cdot W1 + B1 \quad (1)$$

- 激活函数

$$a_1^1 = Sigmoid(z_1^1)$$

$$a_2^1 = Sigmoid(z_2^1)$$

$$a_3^1 = Sigmoid(z_3^1)$$

矩阵形式:

$$A1 = Sigmoid(Z1) \quad (2)$$

输出层

由于我们只想完成一个拟合任务，所以输出层只有一个神经元：

$$z2 = a_1^1 w_{11}^2 + a_2^1 w_{21}^2 + a_3^1 w_{31}^2 + b_1^2$$

矩阵形式

$$Z2 = A1 \cdot W2 + B2 \quad (3)$$

损失函数

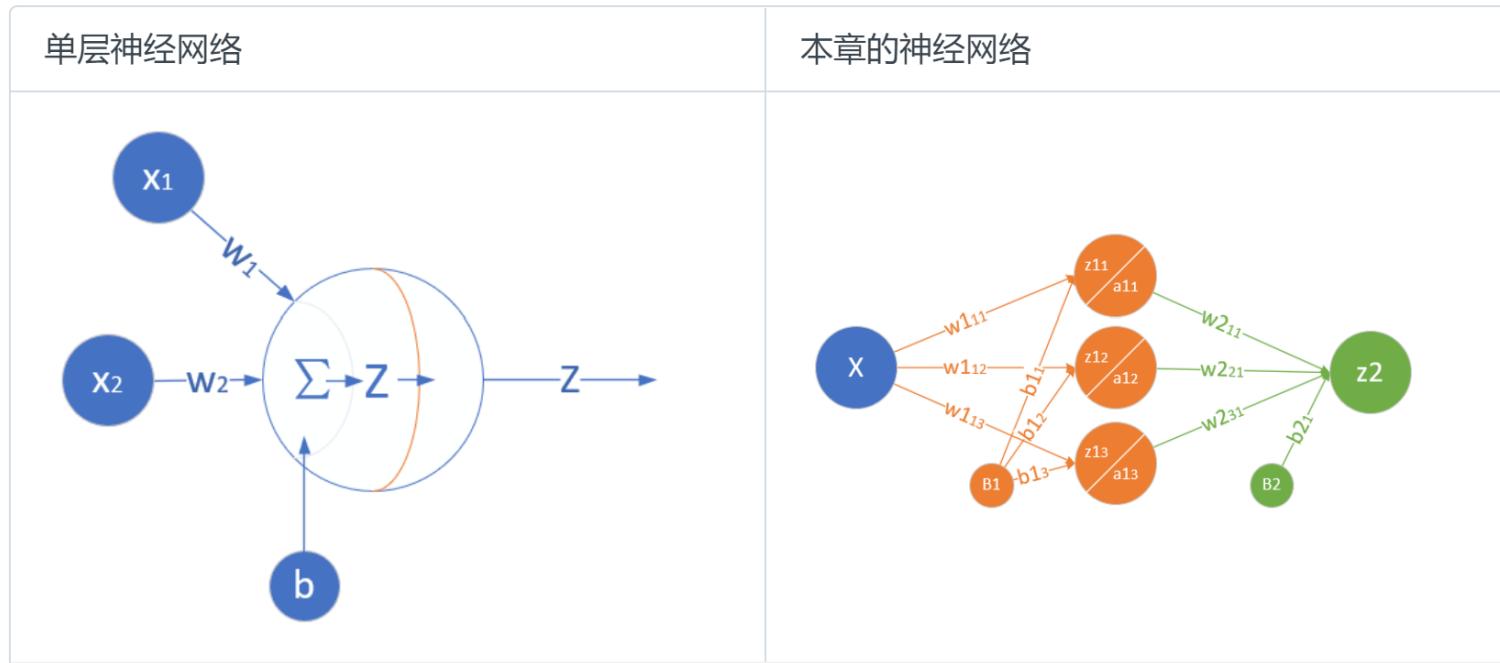
均方差损失函数：

$$loss(w, b) = \frac{1}{2}(z2 - y)^2 \quad (4)$$

其中， $z2$ 是预测值， y 是样本的标签值。



定义神经网络 - 反向计算



本章使用了真正的“网络”，而单层神经网络其实只是一个神经元而已。再看本章的网络的右半部分，从隐层到输出层的结构，和单层的神经元结构一模一样，只是输入为3个特征，而左图中输入为两个特征。比较正向计算公式的话，也可以得到相同的结论。这就意味着反向传播的公式应该也是一样的。

定义神经网络 - 反向计算

求损失函数对输出层的反向误差

根据公式4：

$$\frac{\partial \text{loss}}{\partial z_2} = z_2 - y \rightarrow dZ2 \quad (5)$$

求W2的梯度

根据公式3和W2的矩阵形状，把标量对矩阵的求导分解到矩阵中的每一元素：

$$dW2 = \frac{\partial \text{loss}}{\partial W2} = \begin{pmatrix} \frac{\partial \text{loss}}{\partial z_2} \frac{\partial z_2}{\partial w_{11}^2} \\ \frac{\partial \text{loss}}{\partial z_2} \frac{\partial z_2}{\partial w_{21}^2} \\ \frac{\partial \text{loss}}{\partial z_2} \frac{\partial z_2}{\partial w_{31}^2} \end{pmatrix} = \begin{pmatrix} dZ2 \cdot a_1^1 \\ dZ2 \cdot a_2^1 \\ dZ2 \cdot a_3^1 \end{pmatrix}$$

$$= (a_1^1 \quad a_2^1 \quad a_3^1)^T \cdot dZ2 = A1^T \cdot dZ2 \rightarrow dW2 \quad (6)$$

与单层神经网络相比，除了把X换成A以外，其它的都一样。对于输出层来说，A就是它的输入，也就相当于X。

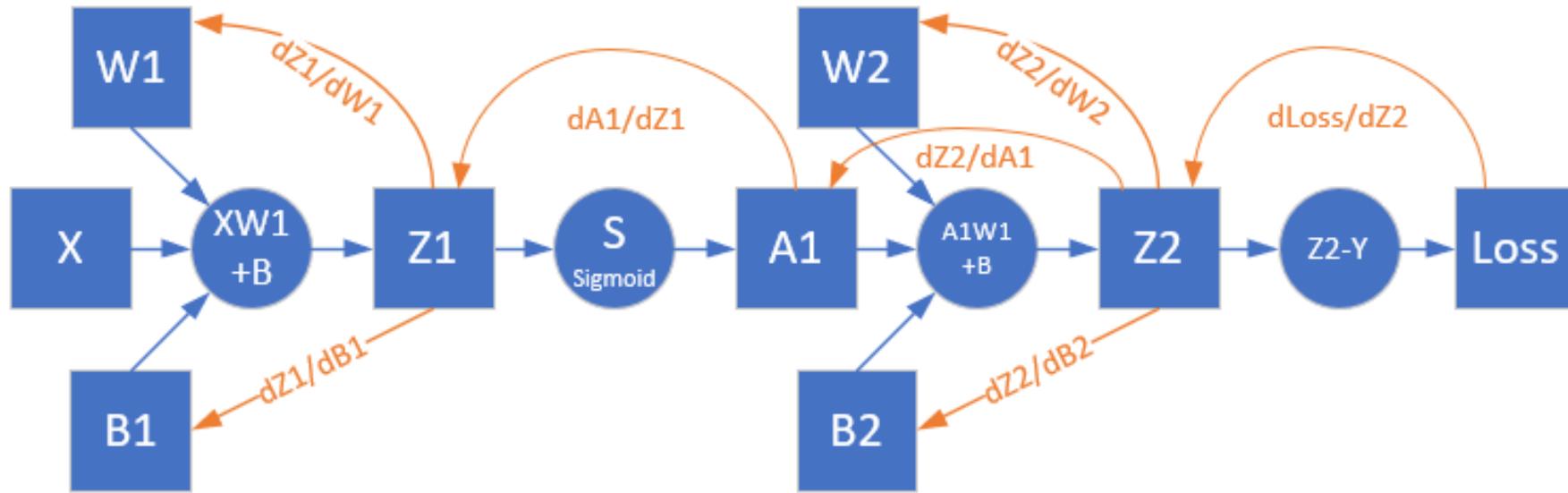
求B2的梯度

$$\frac{\partial \text{loss}}{\partial B2} = dZ2 \rightarrow dB2 \quad (7)$$

与单层神经网络相比，除了把X换成A以外，其它的都一样。对于输出层来说，A就是它的输入，也就相当于X。



定义神经网络 – 求损失函数对隐层的反向误差



蓝色矩形表示数值或矩阵；
蓝色圆形表示计算单元；
蓝色的箭头表示正向计算过程；
红色的箭头表示反向计算过程。

如果想计算 W_1 和 B_1 的反向误差，必须先得到 Z_1 的反向误差，再向上追溯，可以看到 $Z_1 \rightarrow A_1 \rightarrow Z_2 \rightarrow Loss$ 这条线， $Z_1 \rightarrow A_1$ 是一个激活函数的运算，比较特殊，所以我们先看 $Loss \rightarrow Z_2 \rightarrow A_1$ 如何解决。

定义神经网络 – 求损失函数对隐层的反向误差

根据公式3和A1矩阵的形状：

$$\begin{aligned}\frac{\partial \text{loss}}{\partial A1} &= \left(\frac{\partial \text{loss}}{\partial Z2} \frac{\partial Z2}{\partial a_{11}} \quad \frac{\partial \text{loss}}{\partial Z2} \frac{\partial Z2}{\partial a_{12}} \quad \frac{\partial \text{loss}}{\partial Z2} \frac{\partial Z2}{\partial a_{13}} \right) \\ &= \left(dZ2 \cdot w_{11}^2 \quad dZ2 \cdot w_{12}^2 \quad dZ2 \cdot w_{13}^2 \right) \\ &= \left(dZ2 \right) \left(w_{11}^2 \quad w_{21}^2 \quad w_{31}^2 \right)^T \\ &= dZ2 \begin{pmatrix} w_{11}^2 \\ w_{21}^2 \\ w_{31}^2 \end{pmatrix} = dZ2 \cdot W2^T\end{aligned}\tag{8}$$

现在来看激活函数的误差传播问题，由于公式2在计算时，并没有改变矩阵的形状，相当于做了一个矩阵内逐元素的计算，所以它的导数也应该是逐元素的计算，不改变误差矩阵的形状。根据Sigmoid激活函数的导数公式，有：

$$\frac{\partial A1}{\partial Z1} = \text{Sigmoid}'(A1) = A1 \odot (1 - A1)\tag{9}$$

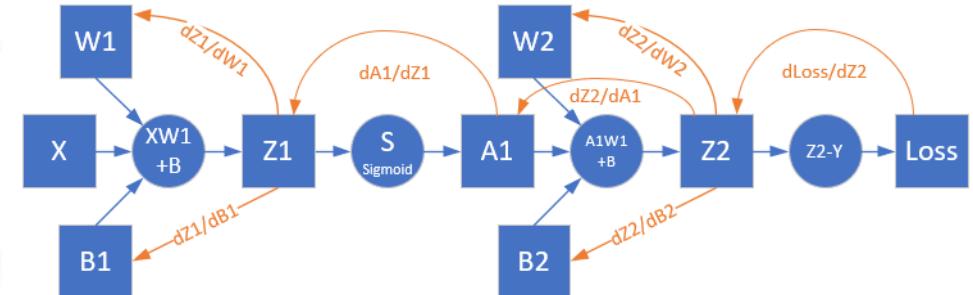
所以最后到达Z1的误差矩阵是：

$$\frac{\partial \text{loss}}{\partial Z1} = \frac{\partial \text{loss}}{\partial A1} \frac{\partial A1}{\partial Z1} = dZ2 \cdot W2^T \odot \text{Sigmoid}'(A1) \rightarrow dZ1\tag{10}$$

有了dZ1后，再向前求W1和B1的误差，就和单层神经网络一样了，我们直接列在下面：

$$dW1 = X^T \cdot dZ1\tag{11}$$

$$dB1 = dZ1\tag{12}$$



多项式为何能拟合曲线？

多项式回归法，它成功地用于正弦曲线和复合函数曲线的拟合，其基本工作原理是把单一特征值的高次方做为额外的特征值加入，使得神经网络可以得到附加的信息用于训练。实践证明其方法有效，但是当问题比较复杂时，需要高达8次方的附加信息，且训练时间也很长。

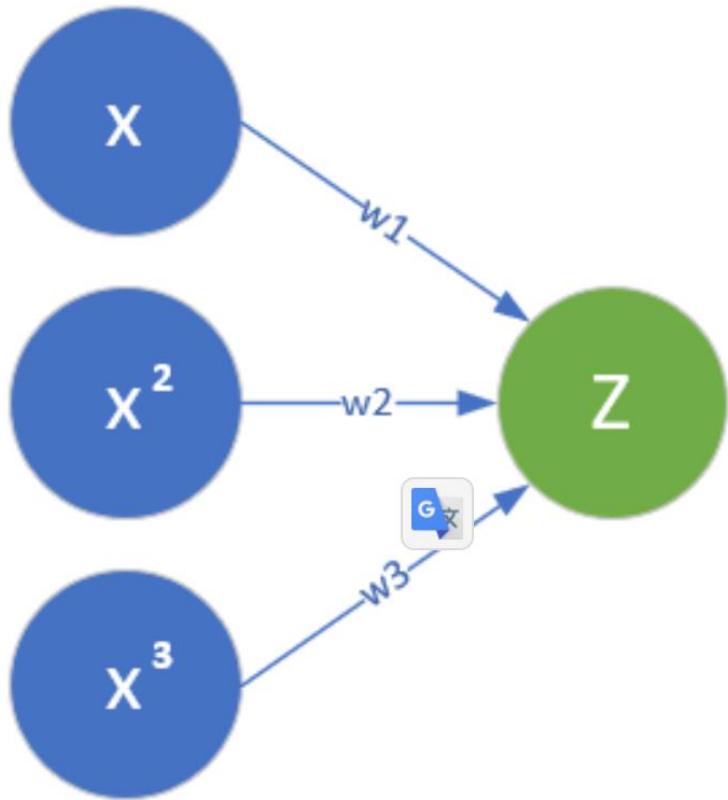
当我们使用双层神经网络时，在隐层只放置了三个神经元，就轻松解决了复合函数拟合的问题，效率高出十几倍，复杂度却降低了几倍。

多项式为何能拟合曲线？

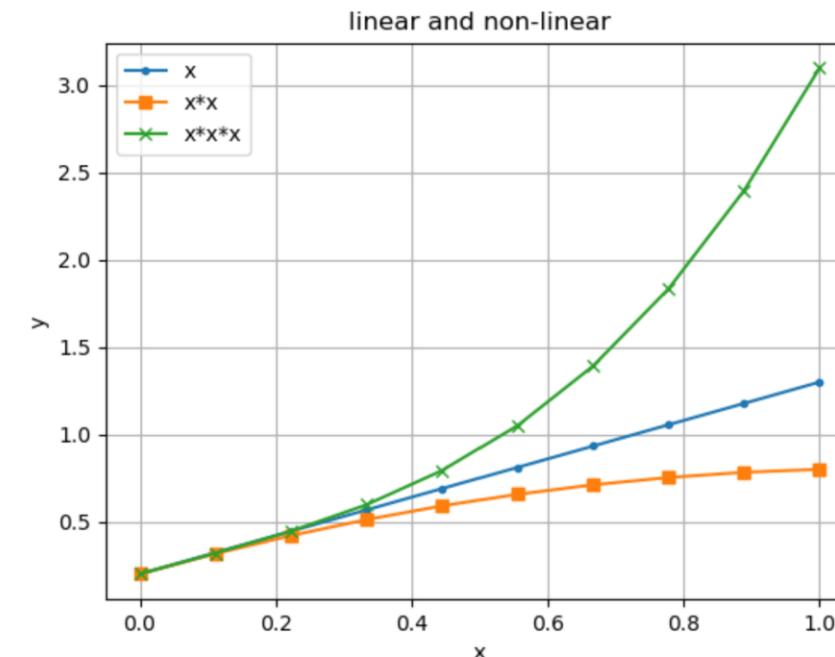
以正弦曲线拟合为例

单层神经网络的多项式回归法，需要 x , x^2 , x^3 三个特征值，组成如下公式来得到拟合结果：

$$z = x \cdot w_1 + x^2 \cdot w_2 + x^3 \cdot w_3 + b \quad (1)$$



蓝色直线是线性回归数值序列；
红色曲线是二项式回归数值序列；
绿色曲线是三项式回归数值序列。

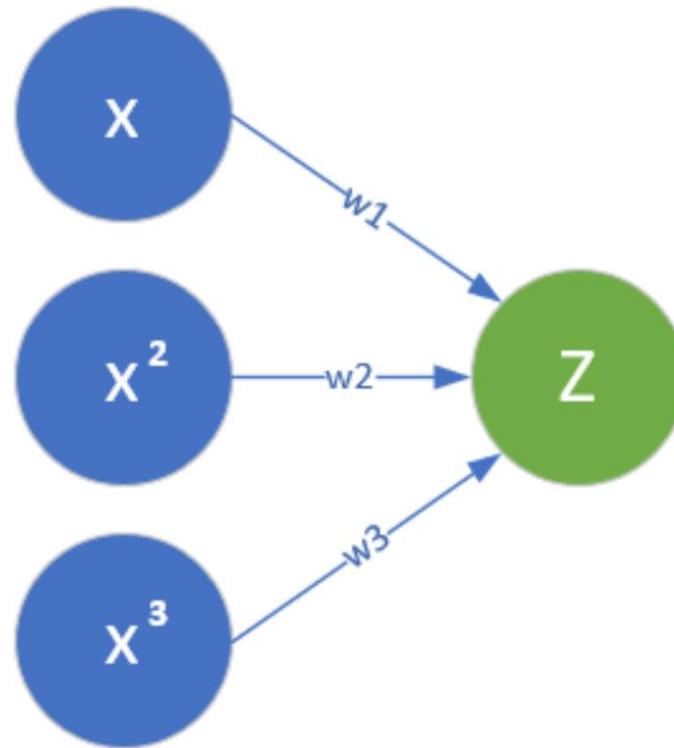


我们只使用了同一个x序列的原始值，却可以得到三种不同数值序列，这就是多项式拟合的原理。当多项式次数很高、数据样本充裕、训练足够多的时候，甚至可以拟合出非单调的曲线。

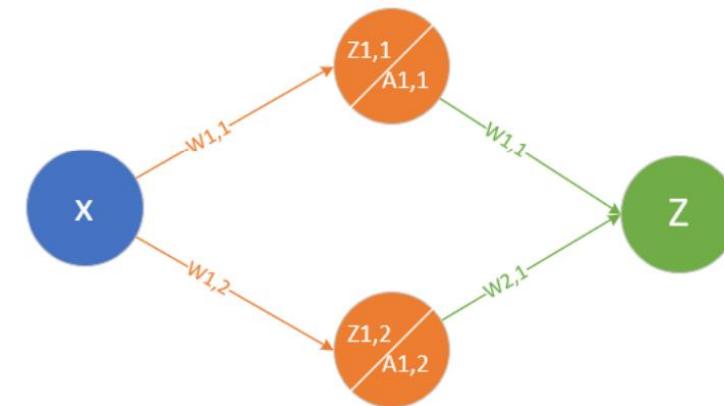
神经网络的非线性拟合工作原理

以正弦曲线的例子来讲解神经网络非线性回归的工作过程和原理

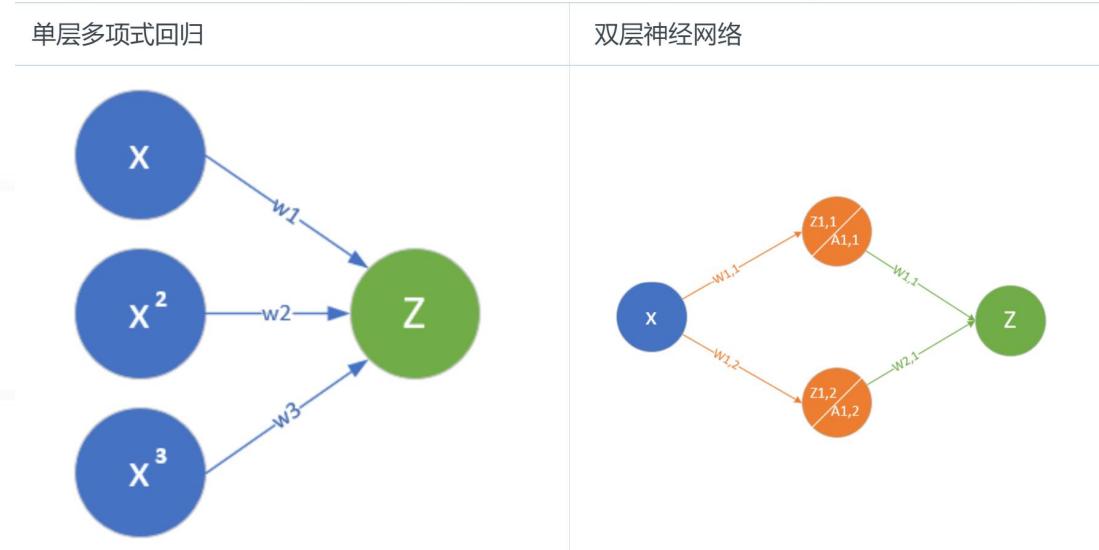
单层多项式回归



双层神经网络



神经网络的非线性拟合工作原理



左图中，通过人为的方式，给 Z 的输入增加了 x^2 和 x^3 项。

右图中，通过线性变换的方式，把 x 变成了两部分： z_{11}/a_{11} , z_{12}/a_{12} ，然后再通过一次线性变换把两者组合成为 Z ，这种方式和多项式回归非常类似：

1. 隐层把 x 拆成不同的特征，根据问题复杂度决定神经元数量，神经元的数量相当于特征值的数量；
2. 隐层通过激活函数做一次非线性变换；
3. 输出层使用多变量线性回归，把隐层的输出当作输入特征值，再做一次线性变换，得出拟合结果。

与多项式回归不同的是，不需要指定变换参数，而是从训练中学习到参数，这样的话权重值不会大得离谱。

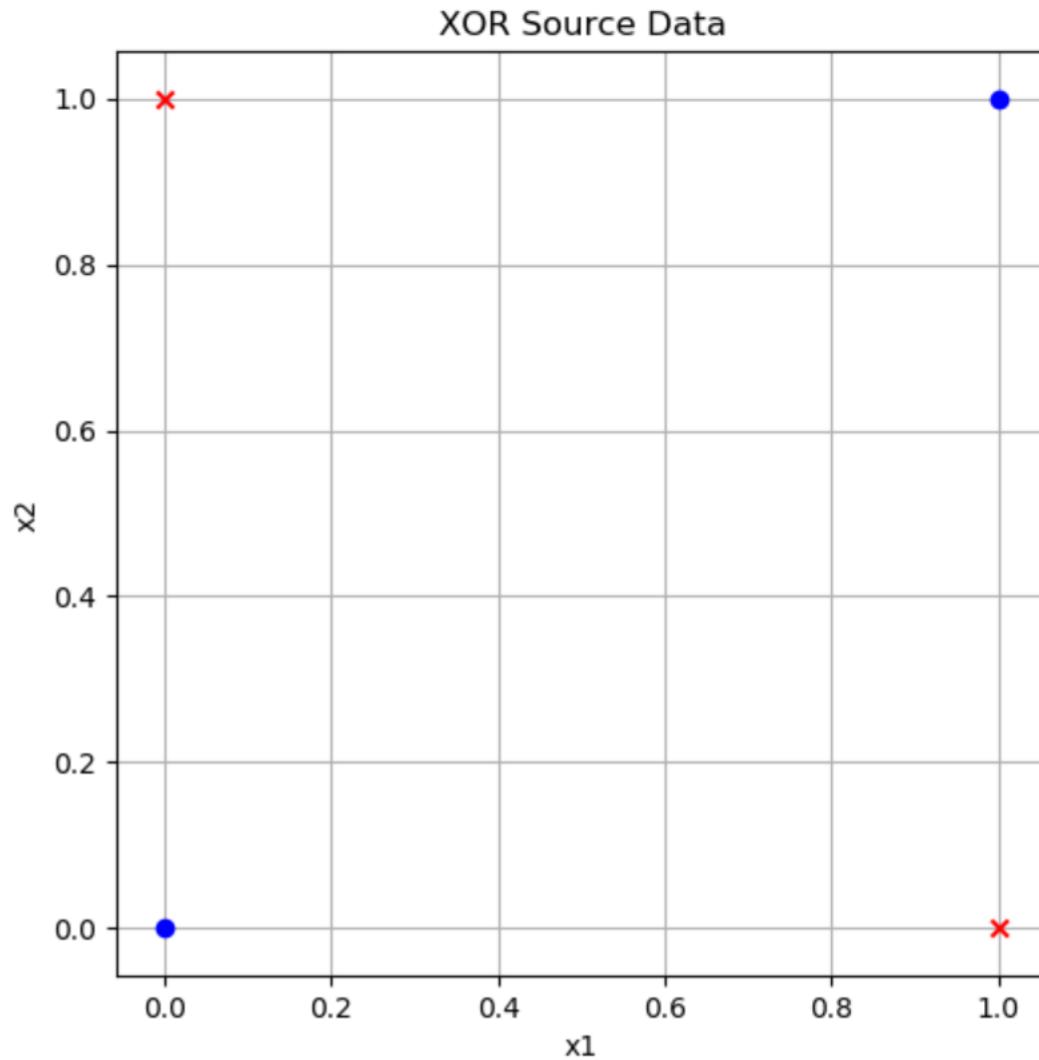
比较多项式回归和双层神经网络解法

	多项式回归	双层神经网络
特征提取方式	特征值的高次方	线性变换拆分
特征值数量级	高几倍的数量级	数量级与原特征值相同
训练效率	低，需要迭代次数多	高，比前者少好几个数量级

多入单出的双层神经网络



问题一：异或问题

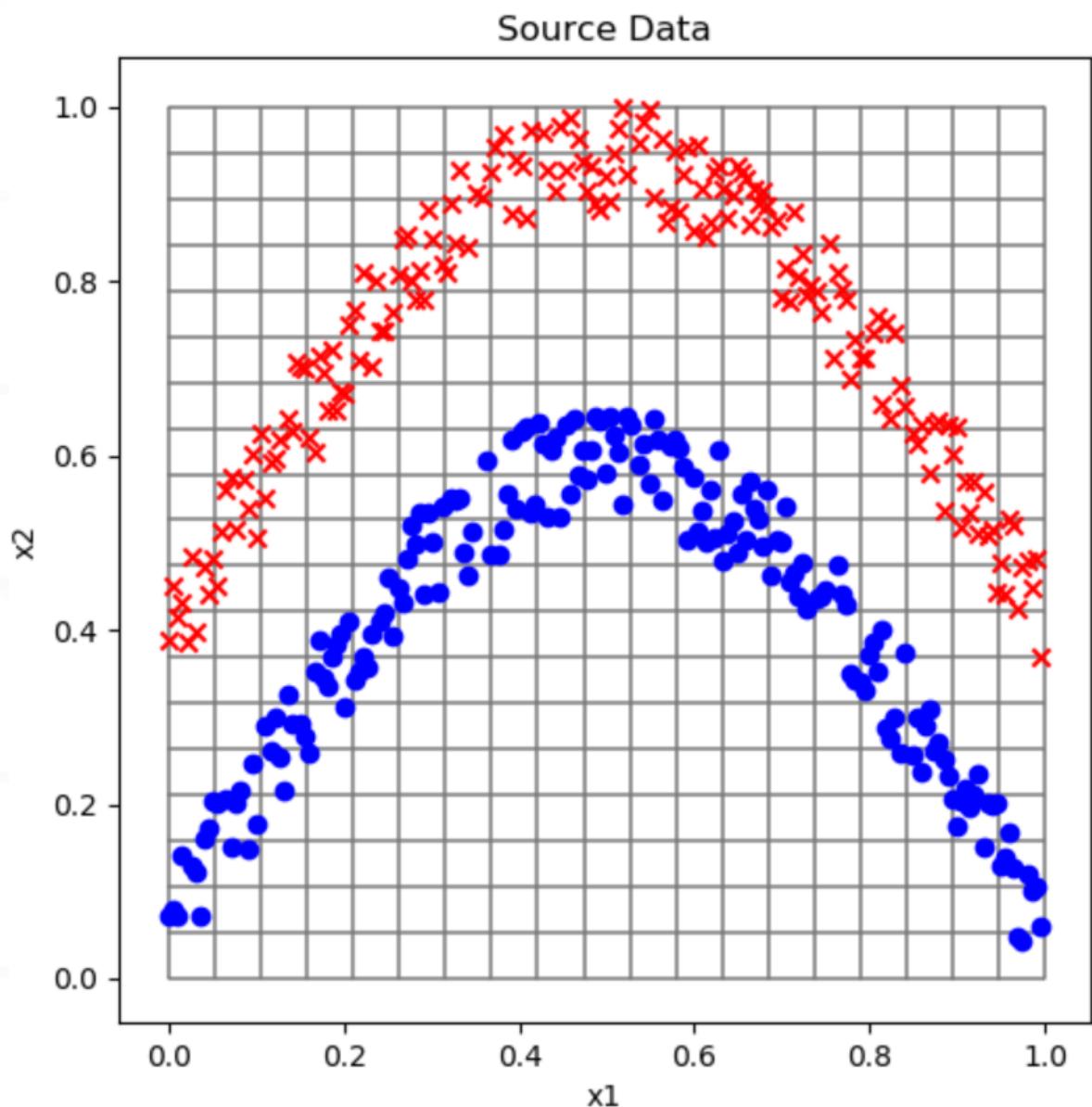


两类样本点（红色叉子和蓝色圆点）交叉分布在[0,1]空间的四个角上，用一条直线无法分割开两类样本。

神经网络是建立在感知器的基础上的，那么我们用神经网络如何解决异或问题呢？

问题二：双弧形问题

平面上有两类样本数据，都成弧形分布，由于弧度的存在，使得我们无法使用一根直线来分开红蓝两种样本点，那么神经网络能用一条曲线来分开它们吗？



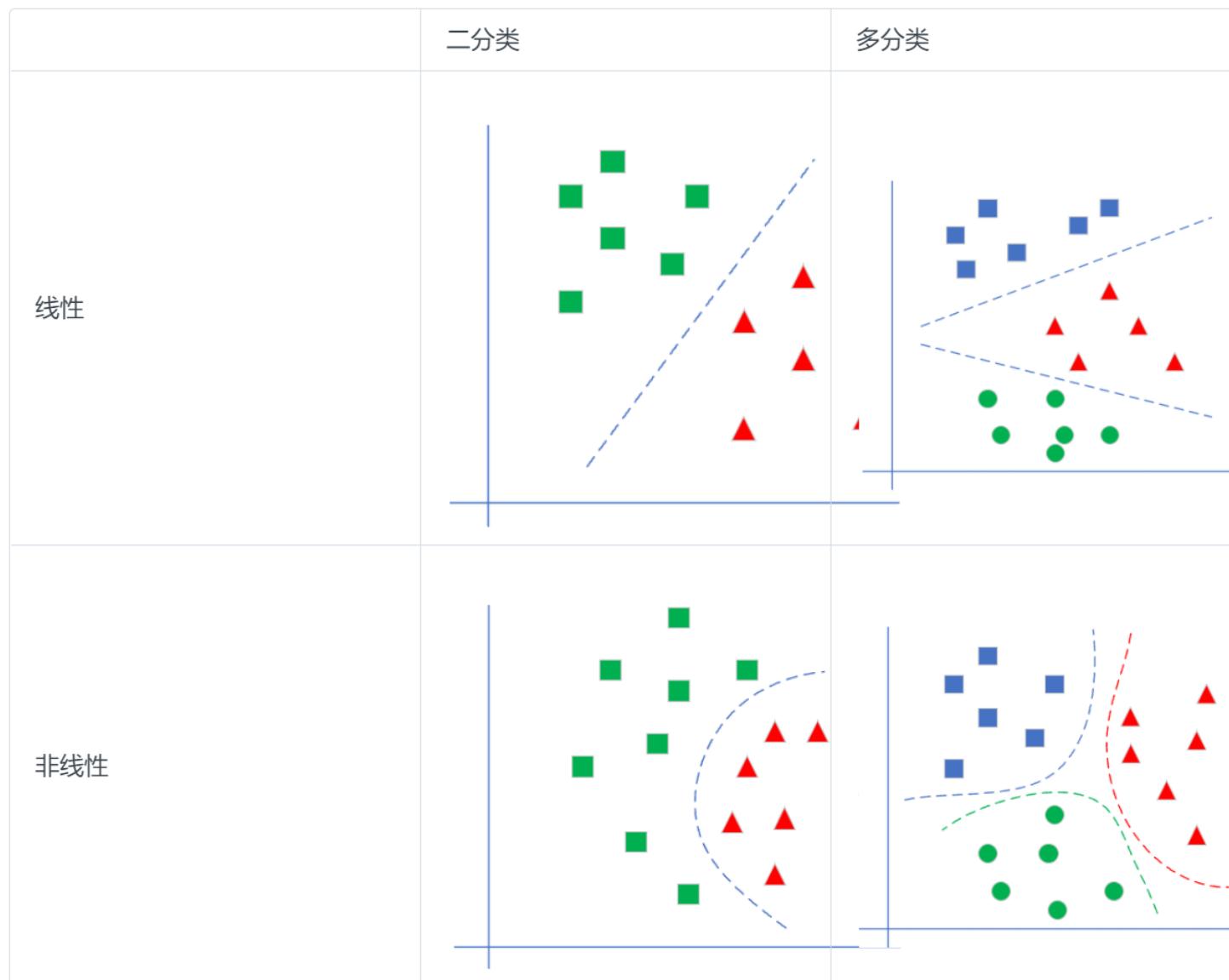
双层神经网络



分类

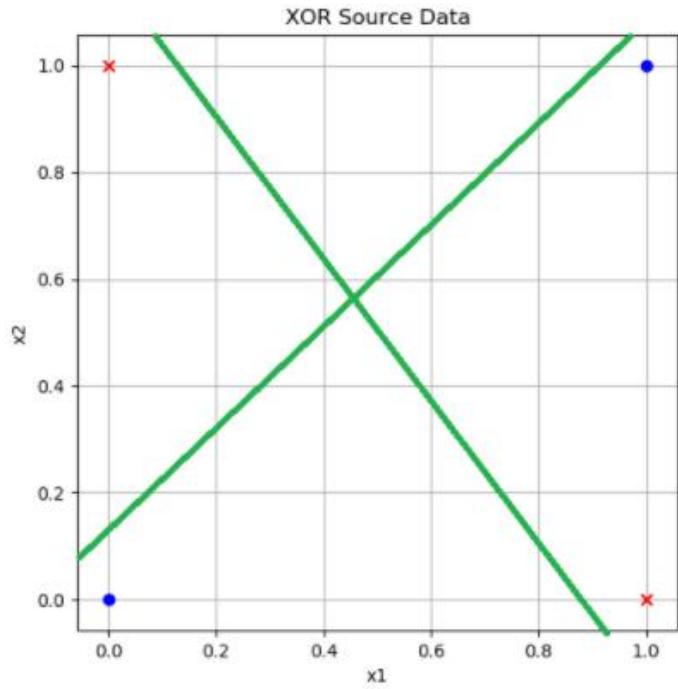
分类的含义：

从复杂程度上分，有线性/非线性之分；
从样本类别上分，有二分类/多分类之分。

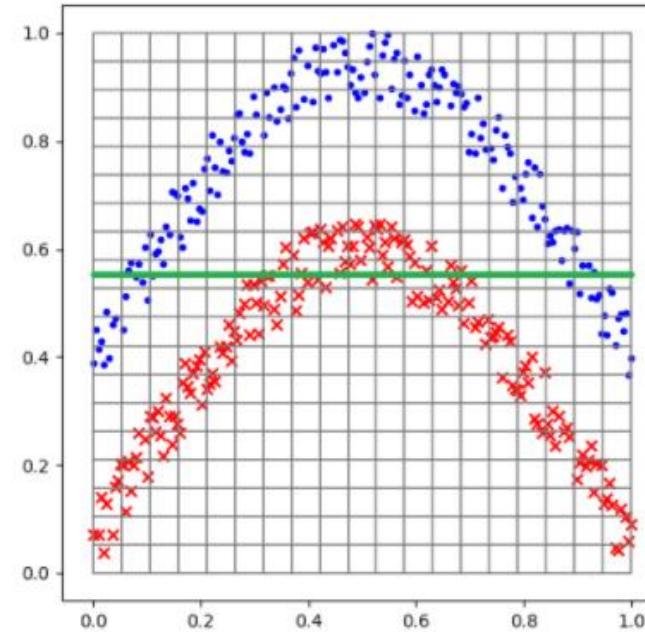


分类

XOR问题



弧形问题



图中两根直线中的任何一根，都不可能把蓝色点分到一侧，同时红色点在另一侧

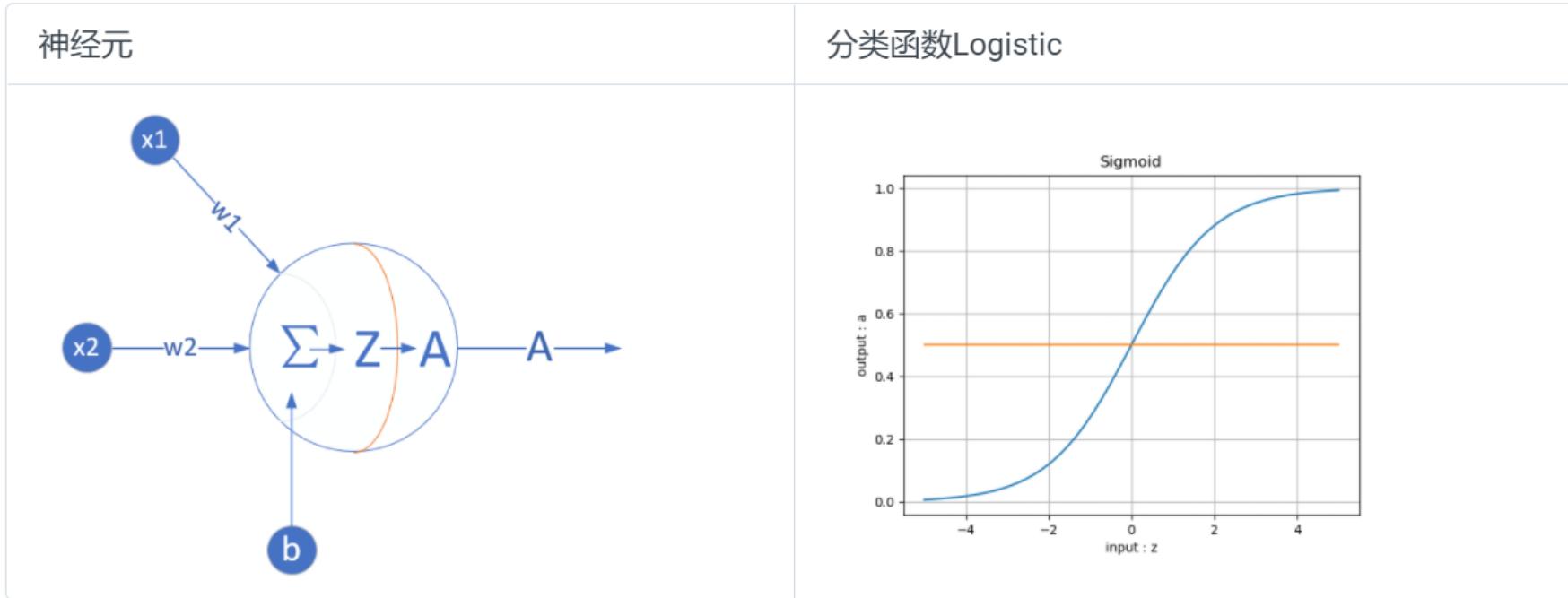
对于线性技术来说，它已经尽力了，使得两类样本尽可能地分布在直线的两侧

分类 - 异或问题的不可能性

用单个感知机或者单层神经网络的异或样本数据

样本	x1	x2	y
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

分类



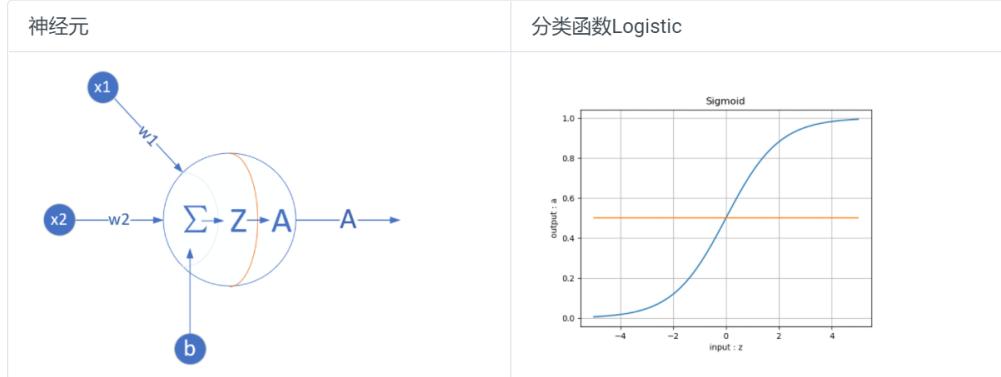
前向计算公式:

$$z = x_1w_1 + x_2w_2 + b \quad (1)$$

$$a = Logistic(z) \quad (2)$$

单个神经元（感知机）的神经元结构与二分类函数

分类



前向计算公式:

$$z = x_1 w_1 + x_2 w_2 + b \quad (1)$$

$$a = \text{Logistic}(z) \quad (2)$$

- 对于第一个样本数据

$x_1=0, x_2=0, y=0$ 。如果需要 $a=y$ 的话，从Logistic函数曲线看，需要 $z<0$ ，于是有：

$$\$ \$ x_1 w_1 + x_2 w_2 + b < 0 \$ \$$$

因为 $x_1=0, x_2=0$ ，所以只剩下 b 项：

$$b < 0 \quad (3)$$

- 对于第二个样本数据

$x_1=0, x_2=1, y=1$ 。如果需要 $a=y$ ，则要求 z 值大于0，不等式为：

$$x_1 w_1 + x_2 w_2 + b = w_2 + b > 0 \quad (4)$$

- 对于第三个样本数据

$x_1=1, x_2=0, y=1$ 。如果需要 $a=y$ ，则要求 z 值大于0，不等式为：

$$x_1 w_1 + x_2 w_2 + b = w_1 + b > 0 \quad (5)$$

- 对于第四个样本

$x_1=1, x_2=1, y=0$ 。如果需要 $a=y$ ，则要求 z 值小于0，不等式为：

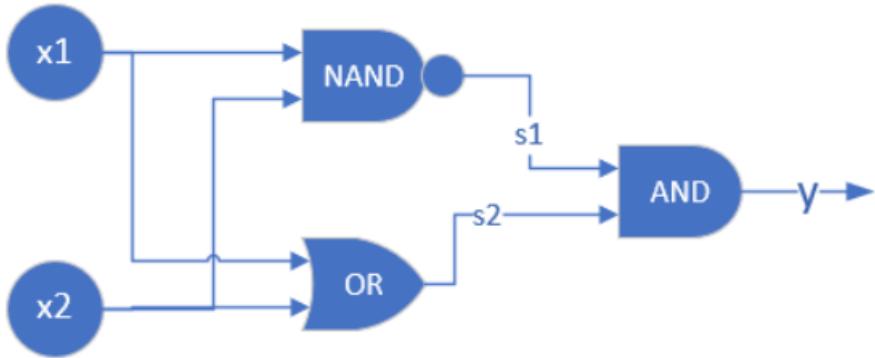
$$x_1 w_1 + x_2 w_2 + b = w_1 + w_2 + b < 0 \quad (6)$$

把公式6两边都加 b ，并把公式3接续：

$$(w_1 + b) + (w_2 + b) < b < 0 \quad (7)$$

再看公式4、5，不等式左侧括号内的两个因子都大于0，其和必然也大于0，不可能小于 b 。因此公式7不成立，无论如何也不能满足所有的4个样本的条件，所以单个神经元做异或运算是不可能的。

非线性的可能性



与、与非、或、或非

样本与计算

$$\begin{array}{l} x_1 \\ x_2 \end{array}$$

$$s_1 = x_1 \text{ NAND } x_2$$

$$s_2 = x_1 \text{ OR } x_2$$

$$y = s_1 \text{ AND } s_2$$

可以看到y的输出与x1,x2的输入相比，就是异或逻辑了。所以，实践证明两层逻辑电路可以解决问题。另外，我们在第四步中学习了非线性回归，使用双层神经网络可以完成一些神奇的事情，比如复杂曲线的拟合，只需要6、7个参数就搞定了。

	1	2	3	4
x1	0	0	1	1
x2	0	1	0	1
s1=x1 NAND x2	1	1	1	0
s2=x1 OR x2	0	1	1	1
y=s1 AND s2	0	1	1	0

非线性二分类



定义神经网络结构

- 输入层两个特征值 x_1, x_2

$$X = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$$

- 隐层 2×2 的权重矩阵 $W1$

$$W1 = \begin{pmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{pmatrix}$$

- 隐层 1×2 的偏移矩阵 $B1$

$$B1 = \begin{pmatrix} b_1^1 & b_2^1 \end{pmatrix}$$

- 隐层由两个神经元构成

$$Z1 = \begin{pmatrix} z_{11} & z_{12} \end{pmatrix}$$

$$A1 = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix}$$

- 输出层 2×1 的权重矩阵 $W2$

$$W2 = \begin{pmatrix} w_{11}^2 \\ w_{21}^2 \end{pmatrix}$$

- 输出层 1×1 的偏移矩阵 $B2$

$$B2 = \begin{pmatrix} b_1^2 \end{pmatrix}$$

- 输出层有一个神经元使用Logistic函数进行分类

$$Z2 = \begin{pmatrix} z_1^2 \end{pmatrix}$$

$$A2 = \begin{pmatrix} a_1^2 \end{pmatrix}$$

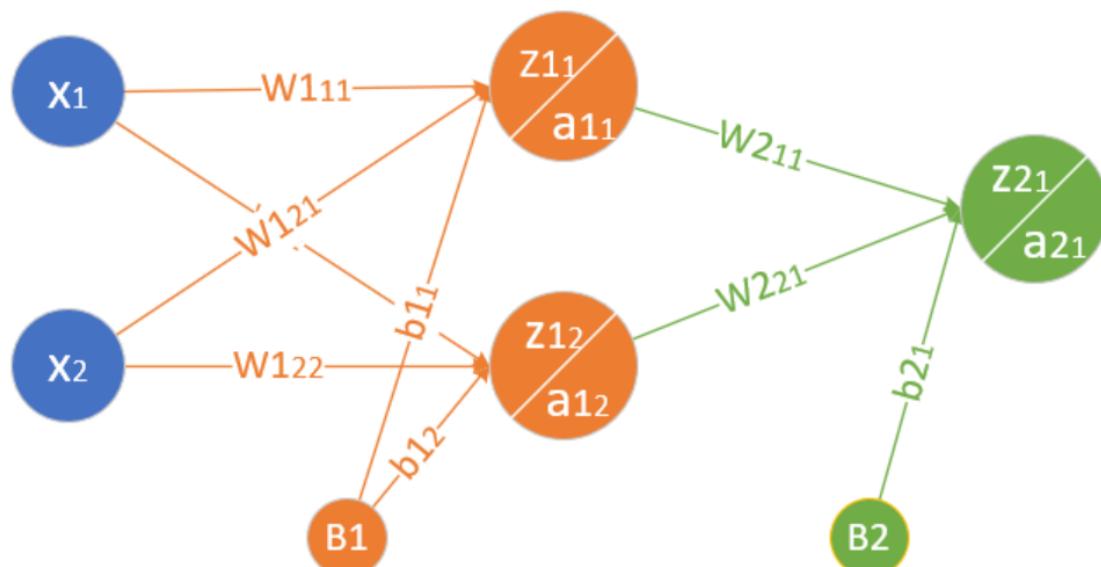
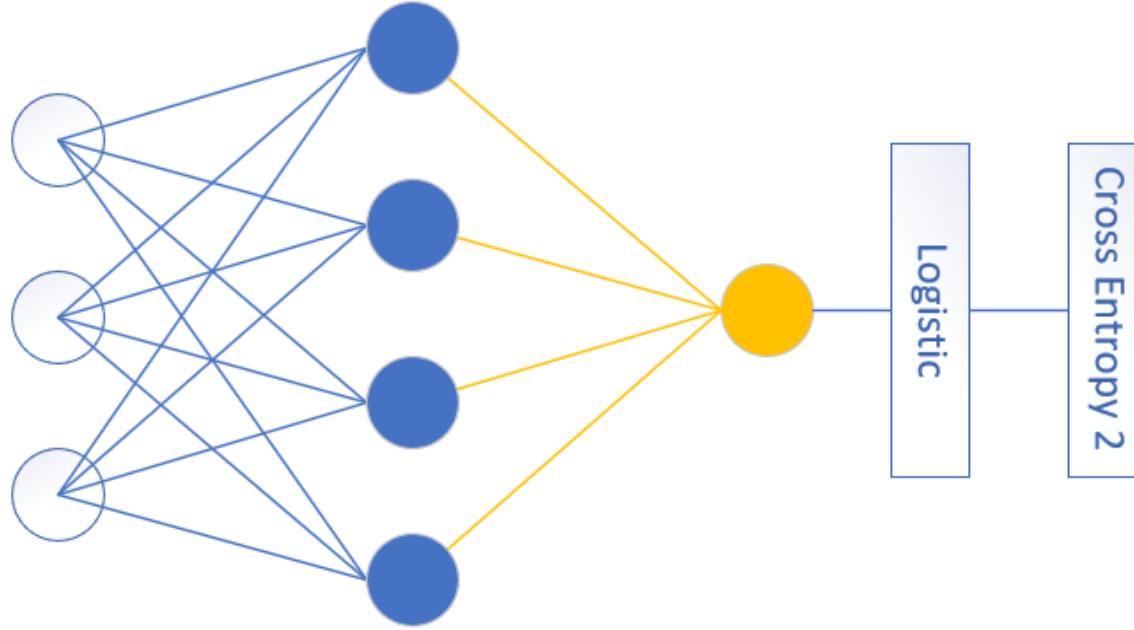


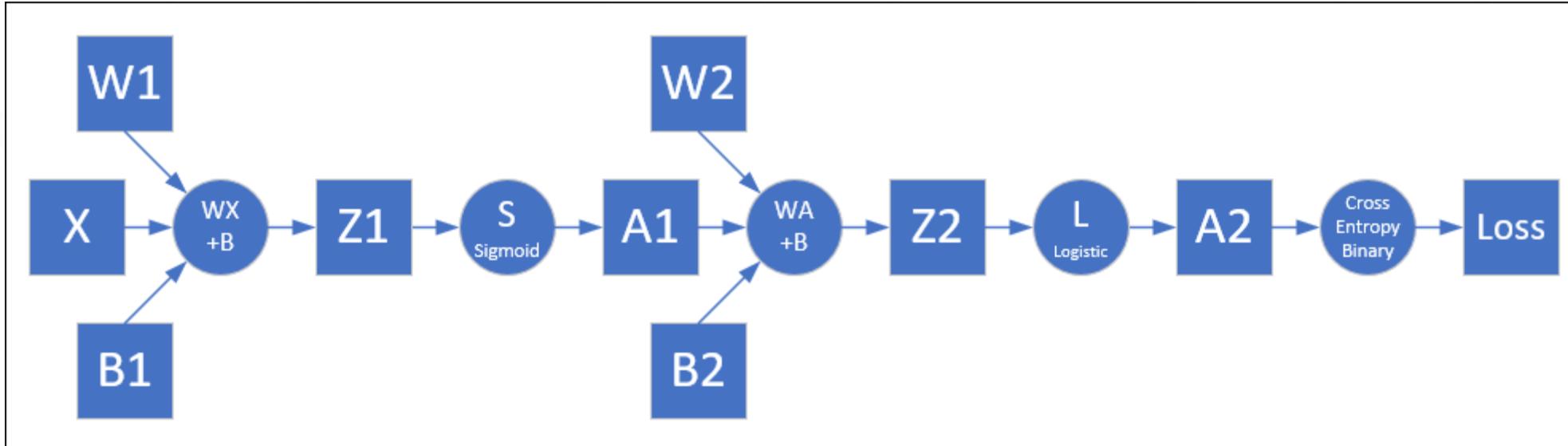
图10-6 非线性二分类神经网络结构图

定义神经网络结构 - 二分类的双层神经网络



输入特征值可以有很多，隐层单元也可以有很多，输出单元只有一个，且后面要接Logistic分类函数和二分类交叉熵损失函数。

定义神经网络结构 - 前向计算



第一层

- 线性计算

$$z_1^1 = x_1 w_{11}^1 + x_2 w_{21}^1 + b_1^1$$

$$z_2^1 = x_1 w_{12}^1 + x_2 w_{22}^1 + b_2^1$$

$$Z1 = X \cdot W1 + B1$$

- 激活函数

$$a_1^1 = Sigmoid(z_1^1)$$

$$a_2^1 = Sigmoid(z_2^1)$$

$$A1 = (a_1^1 \ a_2^1)$$

第二层

- 线性计算

$$z_1^2 = a_1^1 w_{11}^2 + a_2^1 w_{21}^2 + b_1^2$$

$$Z2 = A1 \cdot W2 + B2$$

- 分类函数

$$a_1^2 = Logistic(z_1^2)$$

$$A2 = Logistic(Z2)$$

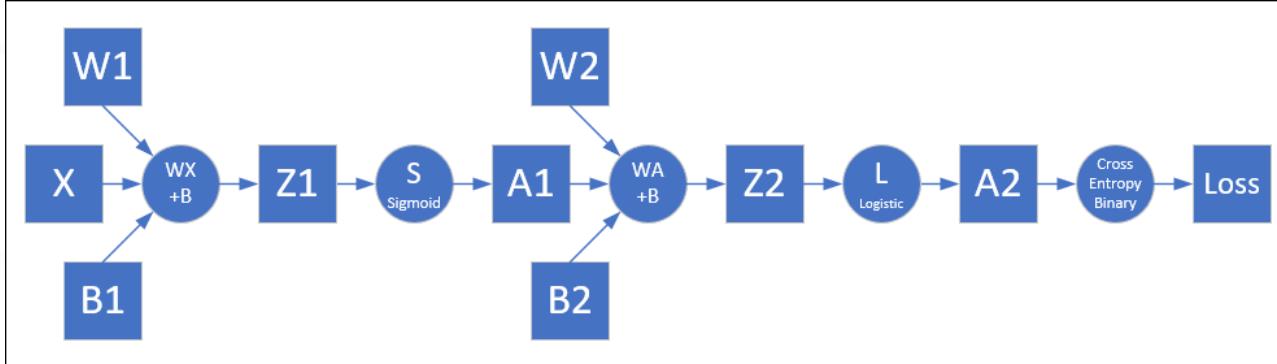
损失函数

我们把异或问题归类成二分类问题，所以使用二分类交叉熵损失函数：

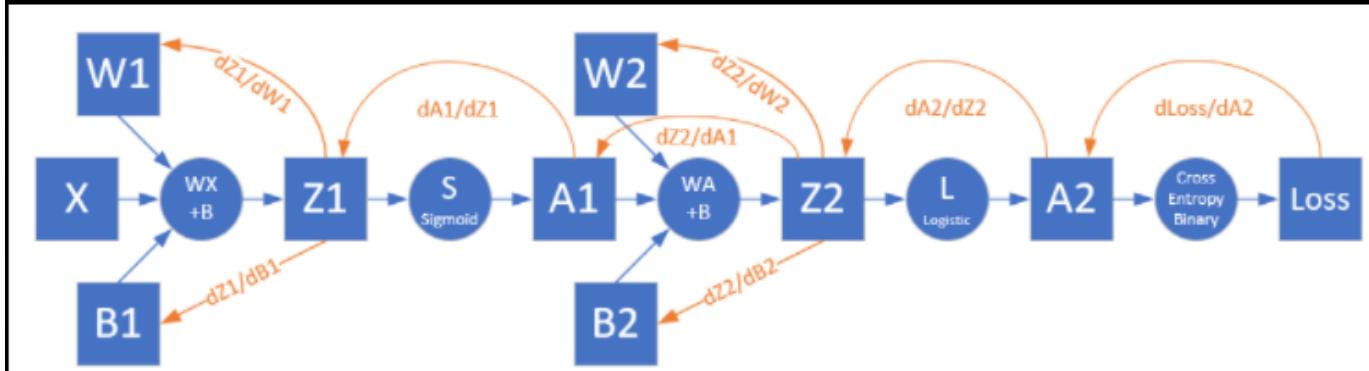
$$loss = -Y \ln A2 + (1 - Y) \ln(1 - A2) \quad (12)$$

在二分类问题中，Y、A2都是一个单一的数值，而非矩阵，但是为了前后统一，我们可以把它们看作是一个1x1的矩阵。

定义神经网络结构 - 前向计算



定义神经网络结构 – 反向计算



求损失函数对输出层的反向误差

对损失函数求导，可以得到损失函数对输出层的梯度值，即图10-9中的Z2部分。

根据公式12，求A2和Z2的导数（此处A2、Z2、Y可以看作是标量，以方便求导）：

$$\frac{\partial \text{loss}}{\partial Z2} = \frac{\partial \text{loss}}{\partial A2} \frac{\partial A2}{\partial Z2} = \frac{A2 - Y}{A2(1 - A2)} \cdot A2(1 - A2) = A2 - Y \rightarrow dZ2 \quad (13)$$

求W2和B2的梯度

$$\frac{\partial \text{loss}}{\partial W2} = \begin{pmatrix} \frac{\partial \text{loss}}{\partial w_{11}} \\ \frac{\partial \text{loss}}{\partial w_{21}} \end{pmatrix} = \begin{pmatrix} \frac{\partial \text{loss}}{\partial Z2} \frac{\partial Z2}{\partial w_{11}} \\ \frac{\partial \text{loss}}{\partial Z2} \frac{\partial Z2}{\partial w_{21}} \end{pmatrix}$$

$$= \begin{pmatrix} dZ2 \cdot a_1^1 \\ dZ2 \cdot a_2^1 \end{pmatrix} = \begin{pmatrix} a_1^1 \\ a_2^1 \end{pmatrix} dZ2$$

$$= A1^T \cdot dZ2 \Rightarrow dW2 \quad (14)$$

$$\frac{\partial \text{loss}}{\partial B2} = dZ2 \Rightarrow dB2 \quad (15)$$

求损失函数对隐层的反向误差

$$\begin{aligned} \frac{\partial \text{loss}}{\partial A1} &= \left(\frac{\partial \text{loss}}{\partial a_1^1}, \frac{\partial \text{loss}}{\partial a_2^1} \right) \\ &= \left(\frac{\partial \text{loss}}{\partial Z2} \frac{\partial Z2}{\partial a_1^1}, \frac{\partial \text{loss}}{\partial Z2} \frac{\partial Z2}{\partial a_2^1} \right) \\ &= (dZ2 \cdot w_{11}^0, dZ2 \cdot w_{21}^0) \end{aligned}$$

$$= dZ2 \cdot (w_{11}^0 \quad w_{21}^0)$$

$$\begin{aligned} &= dZ2 \cdot W2^T \\ &= dZ2 \cdot W2^T \quad (16) \end{aligned}$$

$$\frac{\partial A1}{\partial Z1} = A1 \odot (1 - A1) \Rightarrow dA1 \quad (17)$$

所以最后到达z1的误差矩阵是：

$$\frac{\partial \text{loss}}{\partial Z1} = \frac{\partial \text{loss}}{\partial A1} \frac{\partial A1}{\partial Z1} = dZ2 \cdot W2^T \odot dA1 \rightarrow dZ1 \quad (18)$$

有了dZ1后，再向前求W1和B1的误差，我们直接列在下面了：

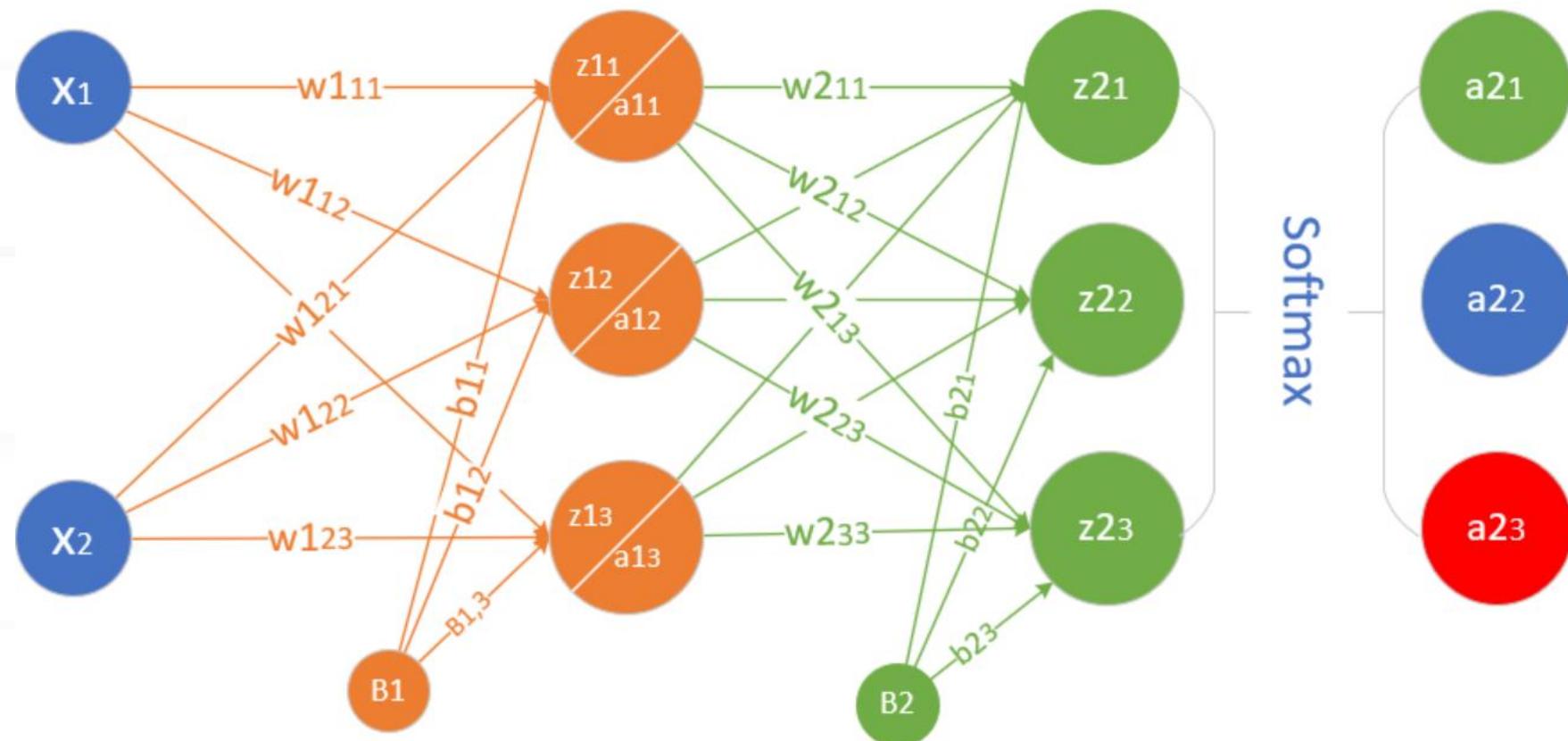
$$dW1 = X^T \cdot dZ1 \quad (19)$$

$$dB1 = dZ1 \quad (20)$$

非线性多分类



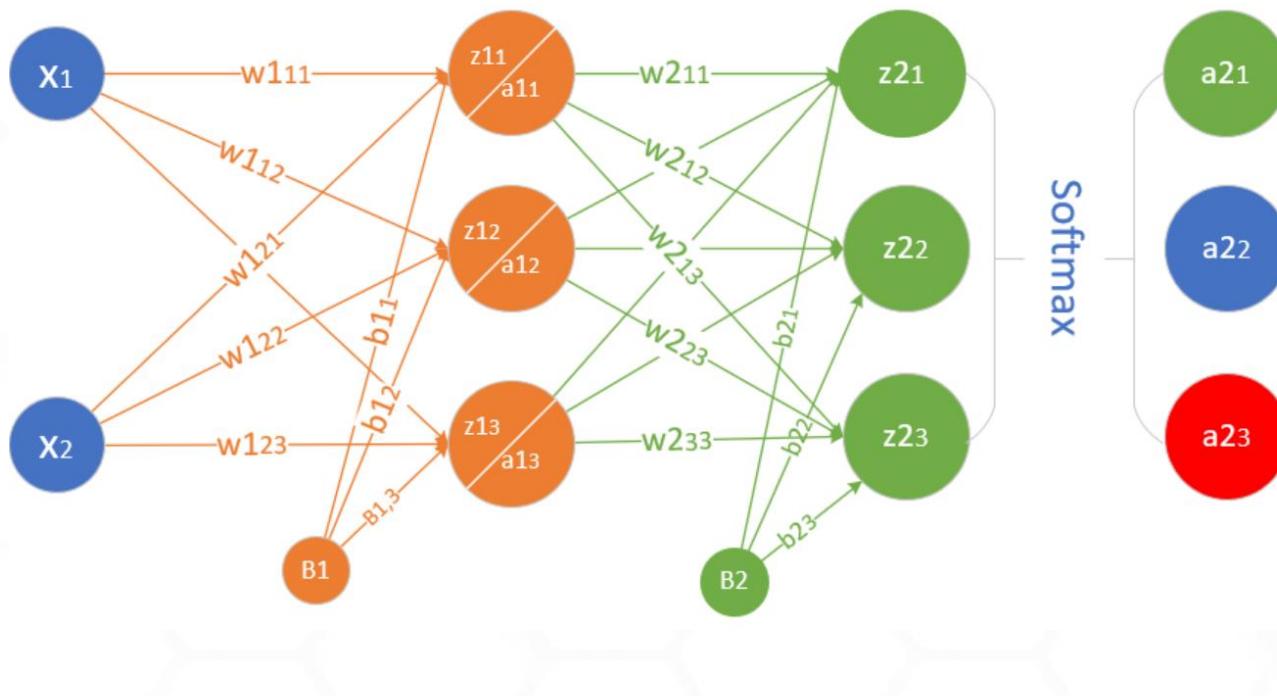
神经网络结构



神经网络结构

- 输入层两个特征值 x_1, x_2

$$x = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$$



- 隐层2x3的权重矩阵 $W1$

$$W1 = \begin{pmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \end{pmatrix}$$

- 隐层1x3的偏移矩阵 $B1$

$$B1 = \begin{pmatrix} b_1^1 & b_2^1 & b_3^1 \end{pmatrix}$$

- 隐层由3个神经元构成
- 输出层3x3的权重矩阵 $W2$

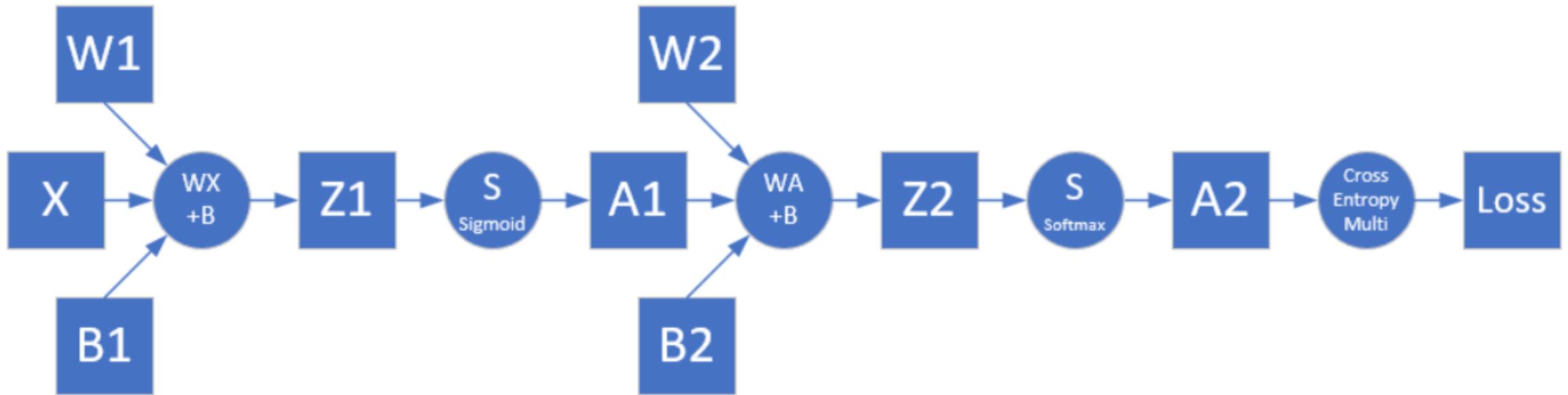
$$W2 = \begin{pmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \end{pmatrix}$$

- 输出层1x1的偏移矩阵 $B2$

$$B2 = \begin{pmatrix} b_1^2 & b_2^2 & b_3^2 \end{pmatrix}$$

- 输出层有3个神经元使用Softmax函数进行分类

非线性多分类 - 前向计算



非线性多分类 - 前向计算

第一层

- 线性计算

$$z_1^1 = x_1 w_{11}^1 + x_2 w_{21}^1 + b_1^1$$

$$z_2^1 = x_1 w_{12}^1 + x_2 w_{22}^1 + b_2^1$$

$$z_3^1 = x_1 w_{13}^1 + x_2 w_{23}^1 + b_3^1$$

$$Z1 = X \cdot W1 + B1$$

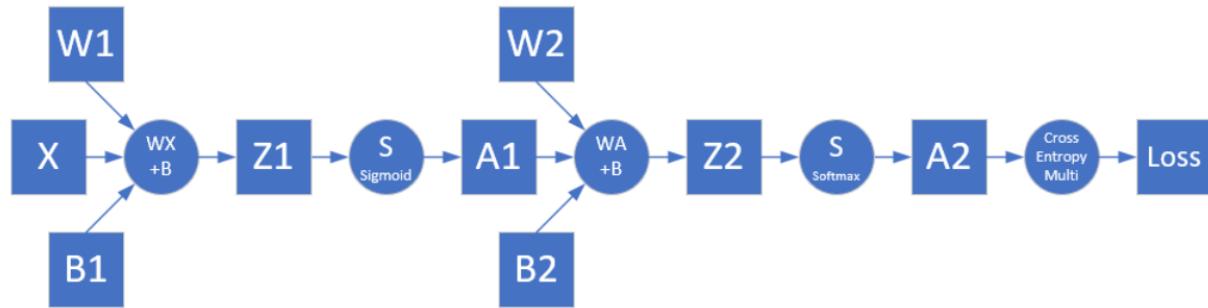
- 激活函数

$$a_1^1 = Sigmoid(z_1^1)$$

$$a_2^1 = Sigmoid(z_2^1)$$

$$a_3^1 = Sigmoid(z_3^1)$$

$$A1 = Sigmoid(Z1)$$



非线性多分类 - 前向计算

- 线性计算

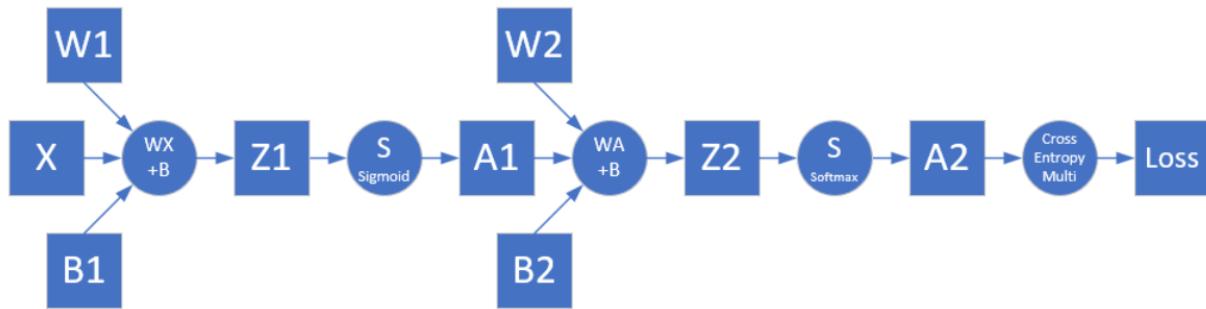
$$z_1^2 = a_1^1 w_{11}^2 + a_2^1 w_{21}^2 + a_3^1 w_{31}^2 + b_1^2$$

$$z_2^2 = a_1^1 w_{12}^2 + a_2^1 w_{22}^2 + a_3^1 w_{32}^2 + b_2^2$$

$$z_3^2 = a_1^1 w_{13}^2 + a_2^1 w_{23}^2 + a_3^1 w_{33}^2 + b_3^2$$

$$Z2 = A1 \cdot W2 + B2$$

- 分类函数



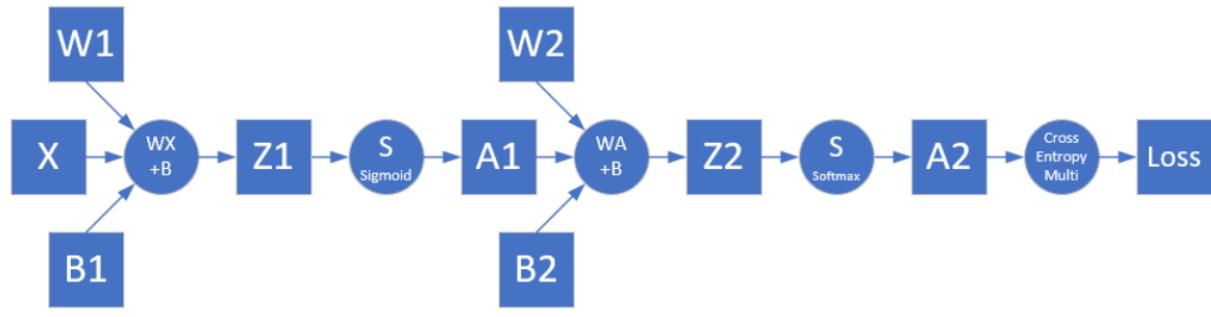
$$a_1^2 = \frac{e^{z_1^2}}{e^{z_1^2} + e^{z_2^2} + e^{z_3^2}}$$

$$a_2^2 = \frac{e^{z_2^2}}{e^{z_1^2} + e^{z_2^2} + e^{z_3^2}}$$

$$a_3^2 = \frac{e^{z_3^2}}{e^{z_1^2} + e^{z_2^2} + e^{z_3^2}}$$

$$A2 = Softmax(Z2)$$

非线性多分类 - 前向计算



损失函数

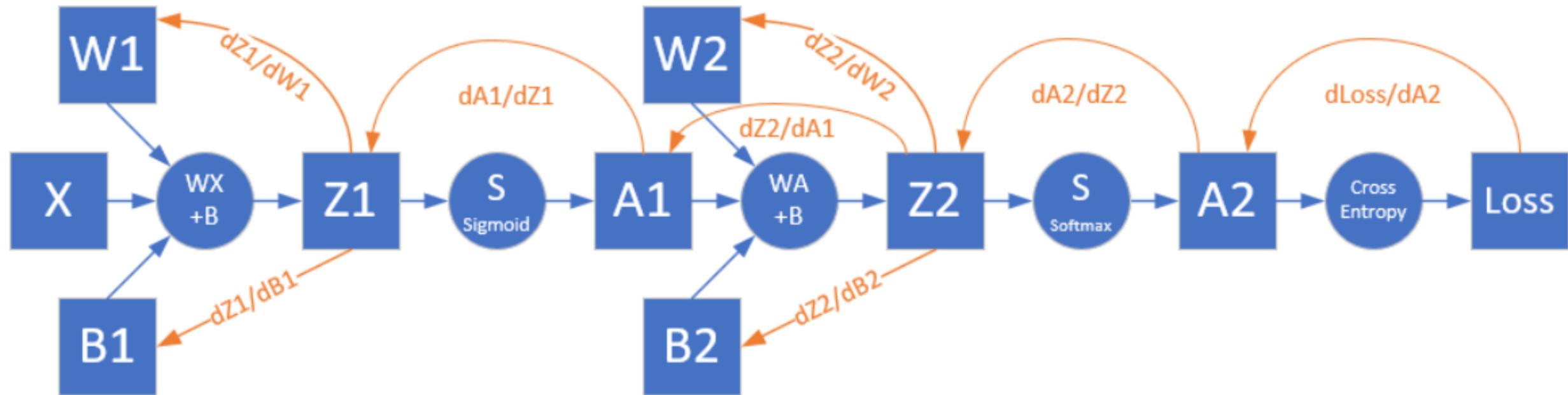
使用多分类交叉熵损失函数:

$$loss = -(y_1 \ln a_1^2 + y_2 \ln a_2^2 + y_3 \ln a_3^2)$$

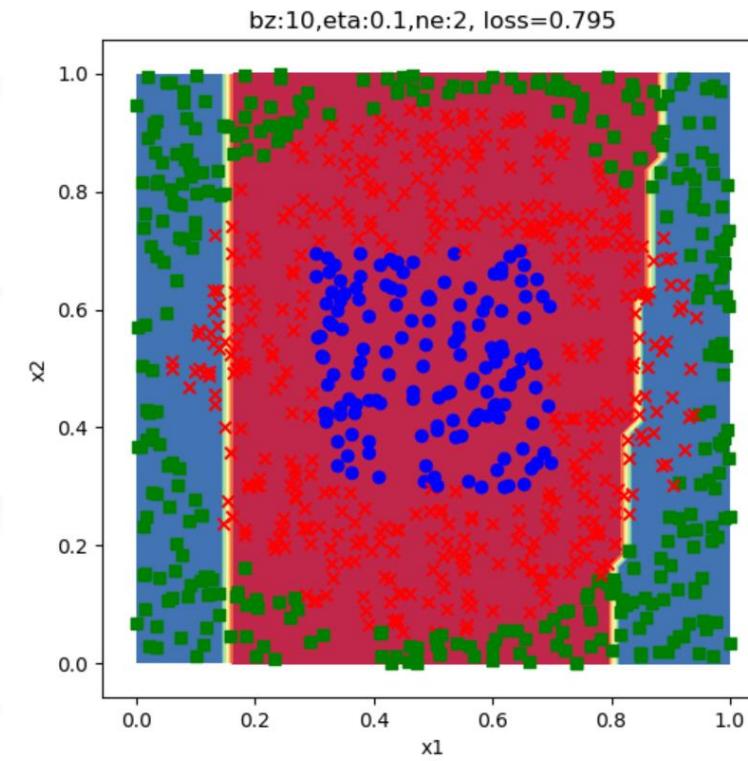
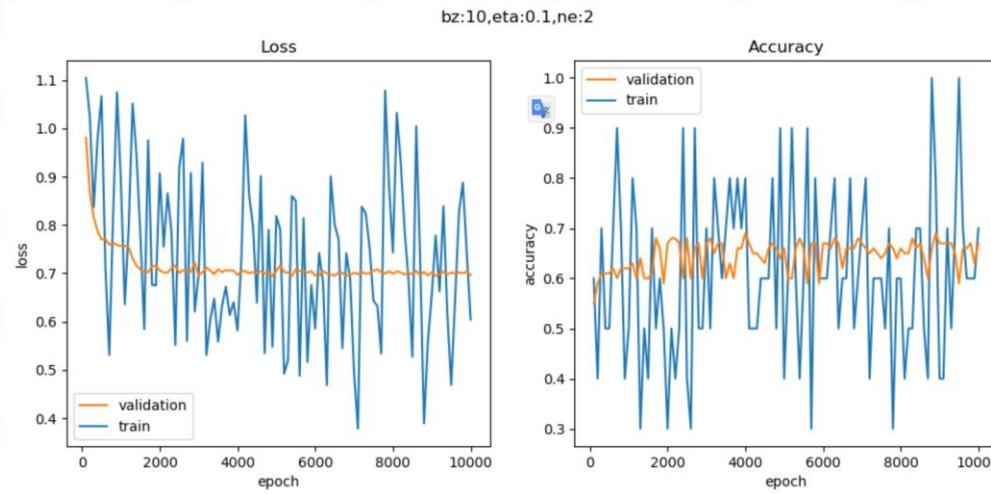
$$J(w, b) = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n y_{ij} \ln(a_{ij}^2)$$

m为样本数，n为类别数。

非线性多分类 – 反向计算



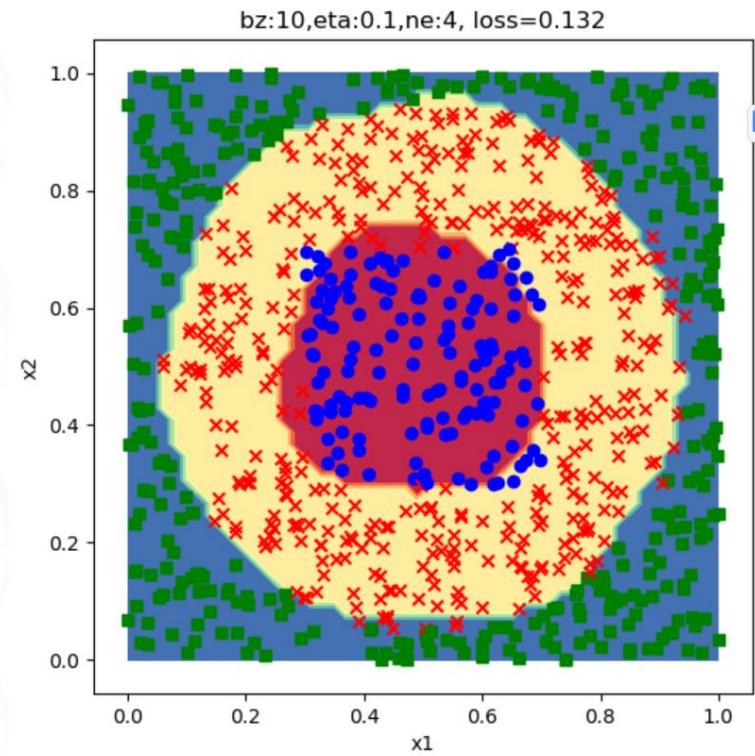
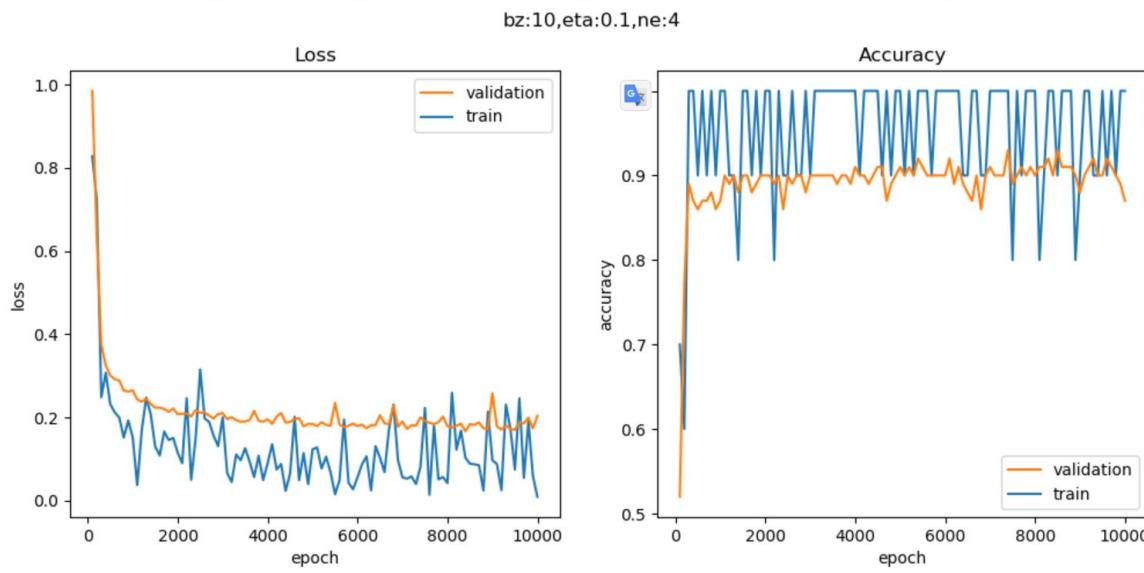
非线性多分类的工作原理 – 隐层神经元数量 2



测试集准确度0.618，耗时49秒，损失函数值0.795。类似这种曲线的情况，损失函数值降不下去，准确度值升不上去，主要原因
是网络能力不够。

没有完成分类任务

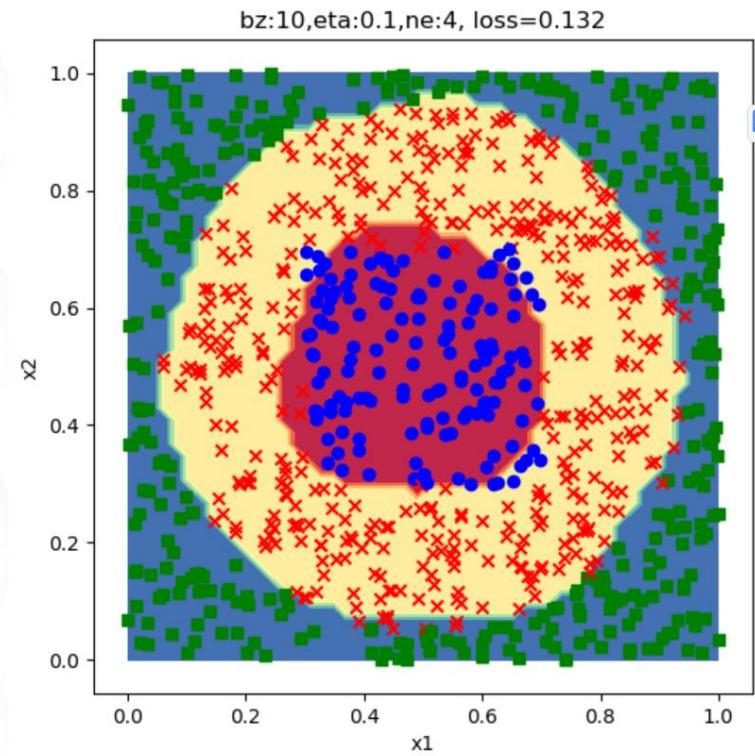
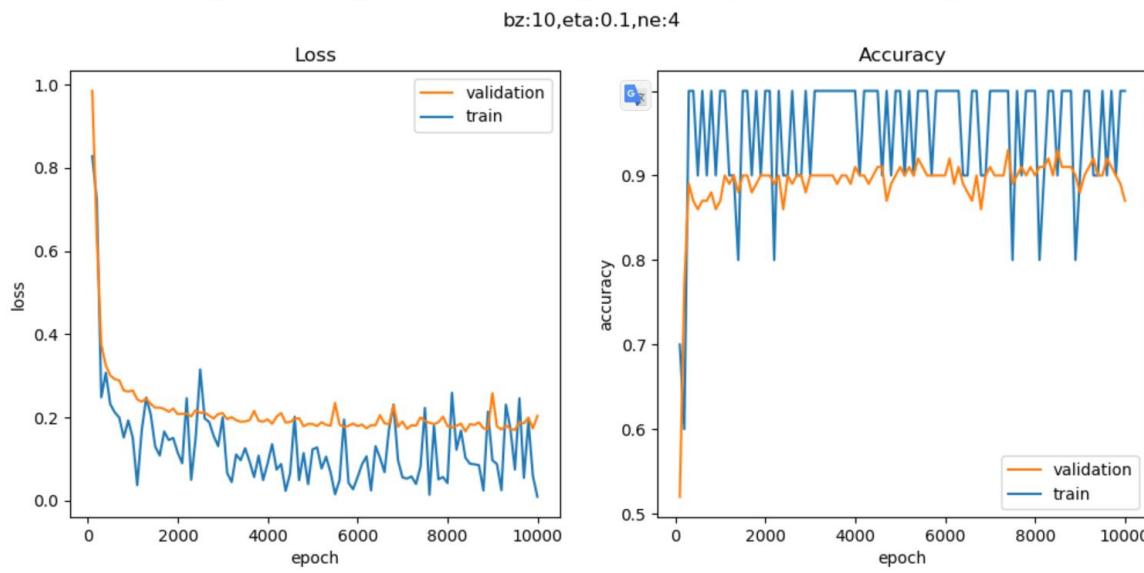
非线性多分类的工作原理 – 隐层神经元数量 4



测试准确度0.954，耗时51秒，损失函数值0.132。
虽然可以基本完成分类任务，网络能力仍然不够。

基本完成，但是边缘不够清晰

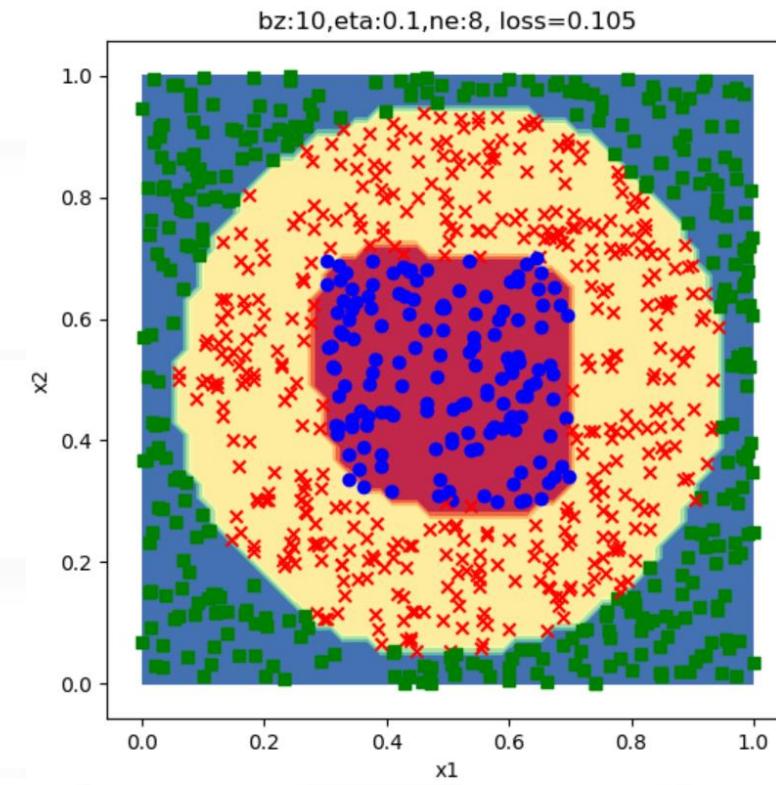
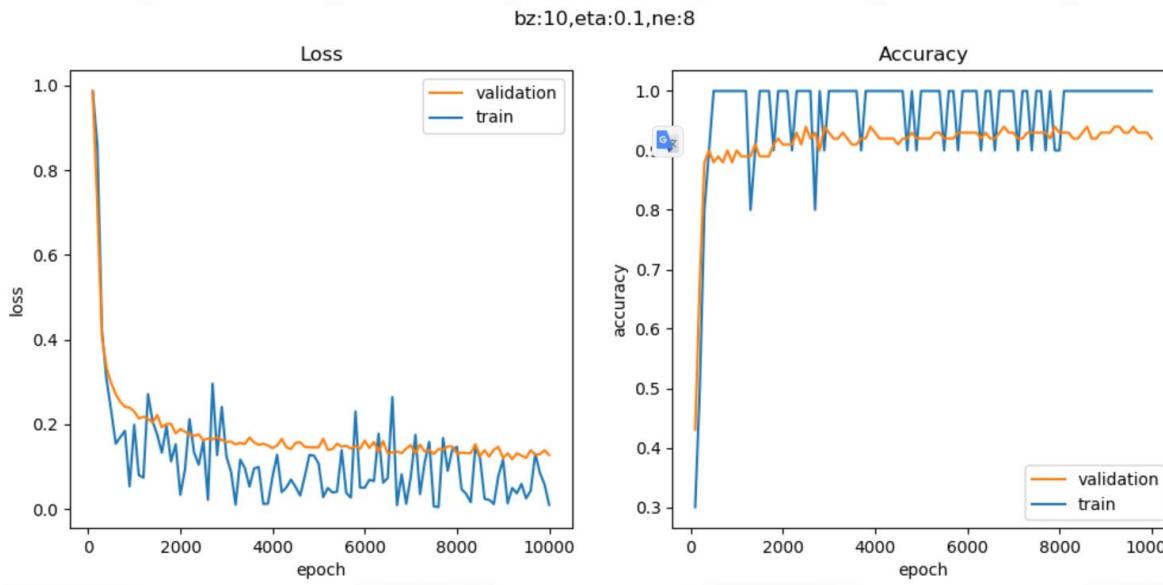
非线性多分类的工作原理 – 隐层神经元数量 4



测试准确度0.954，耗时51秒，损失函数值0.132。
虽然可以基本完成分类任务，网络能力仍然不够。

基本完成，但是边缘不够清晰

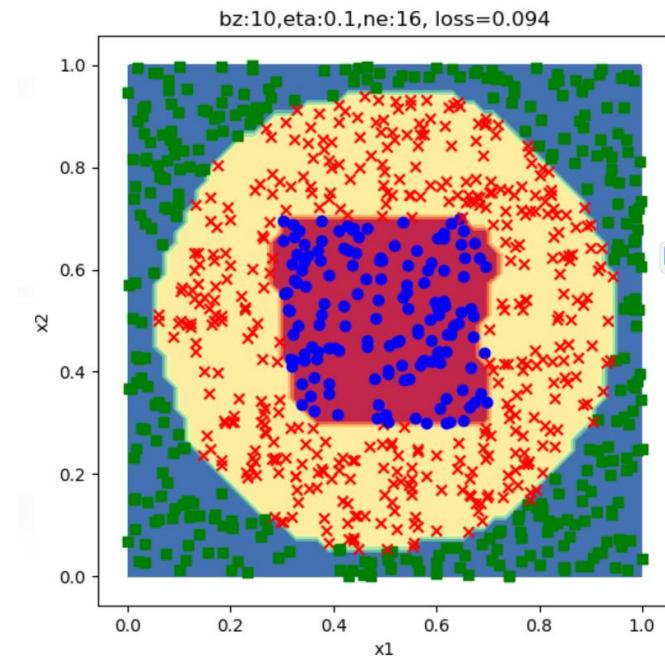
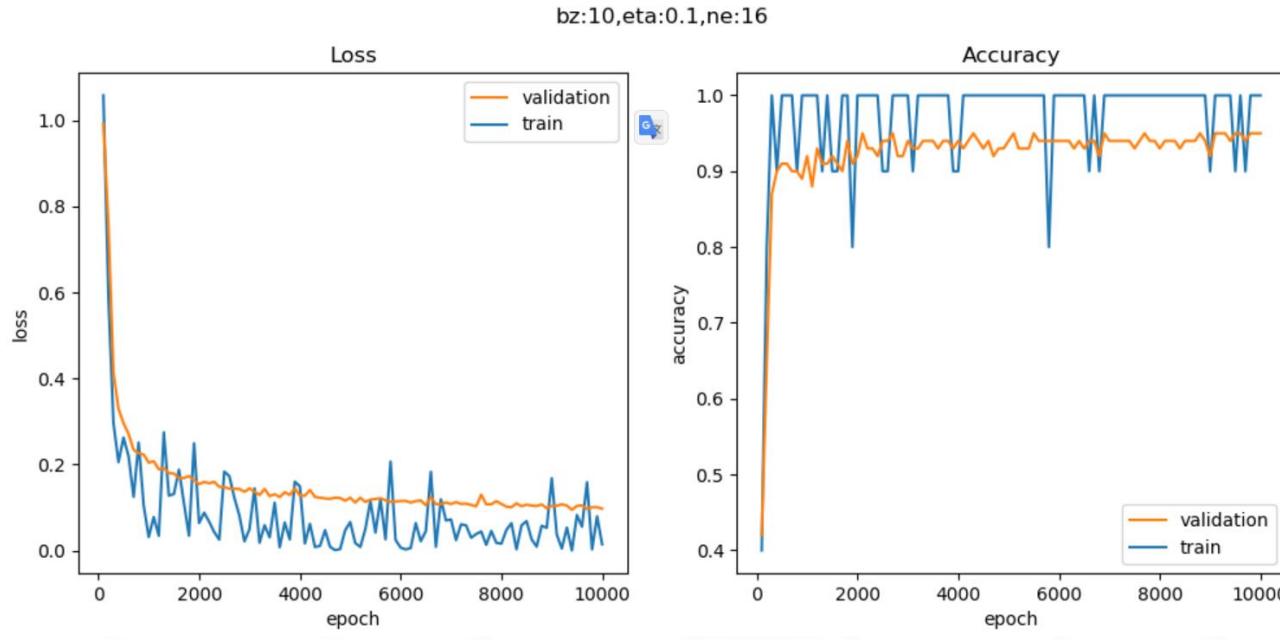
非线性多分类的工作原理 – 隐层神经元数量 8



测试准确度0.97，耗时52秒，损失函数值0.105。可以先试试在后期衰减学习率，如果再训练5000轮没有改善的话，可以考虑增加网络能力。

基本完成，但是边缘不够清晰

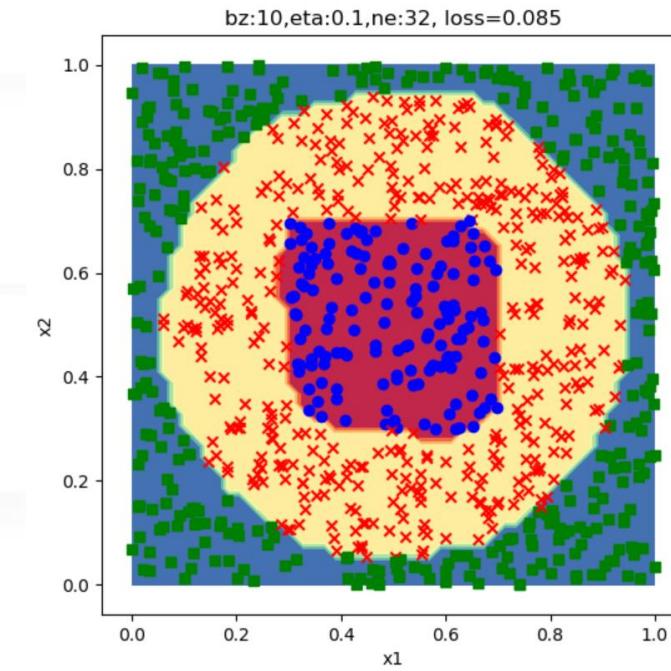
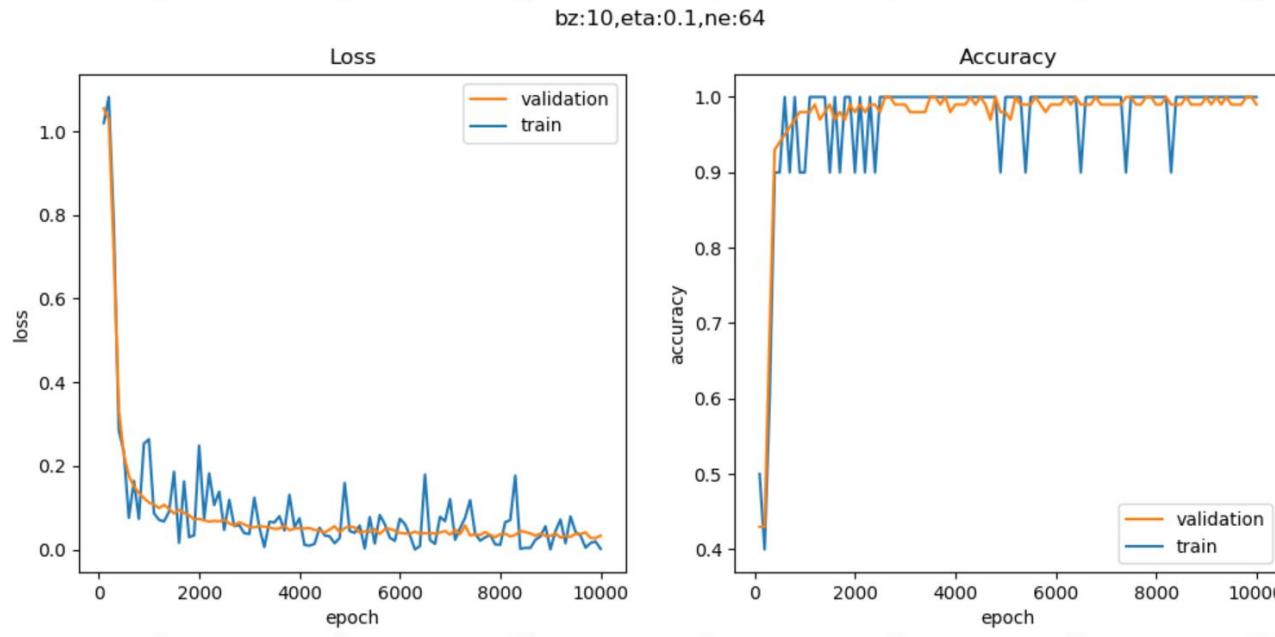
非线性多分类的工作原理 – 隐层神经元数量 16



测试准确度0.978，耗时53秒，损失函数值0.094。同上，
可以同时试着使用优化算法，看看是否能收敛更快。

较好地完成了分类任务

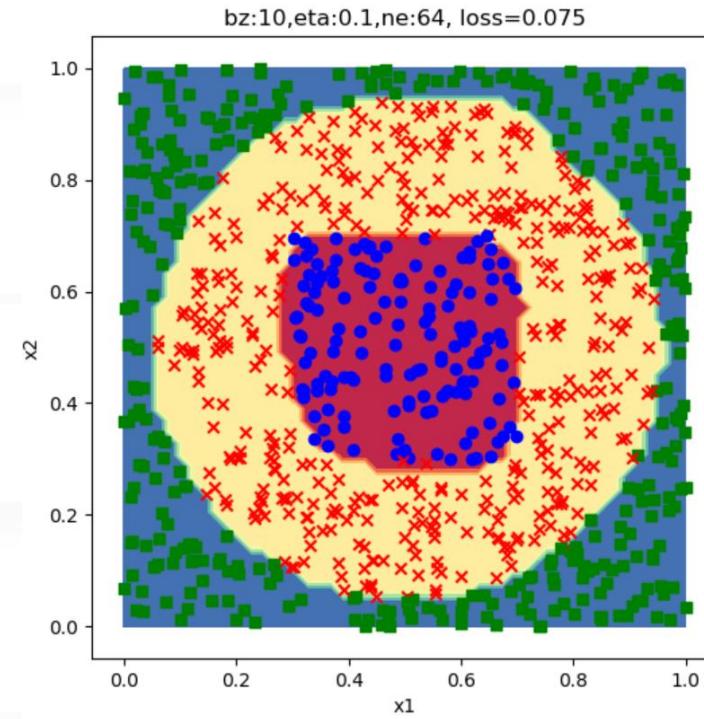
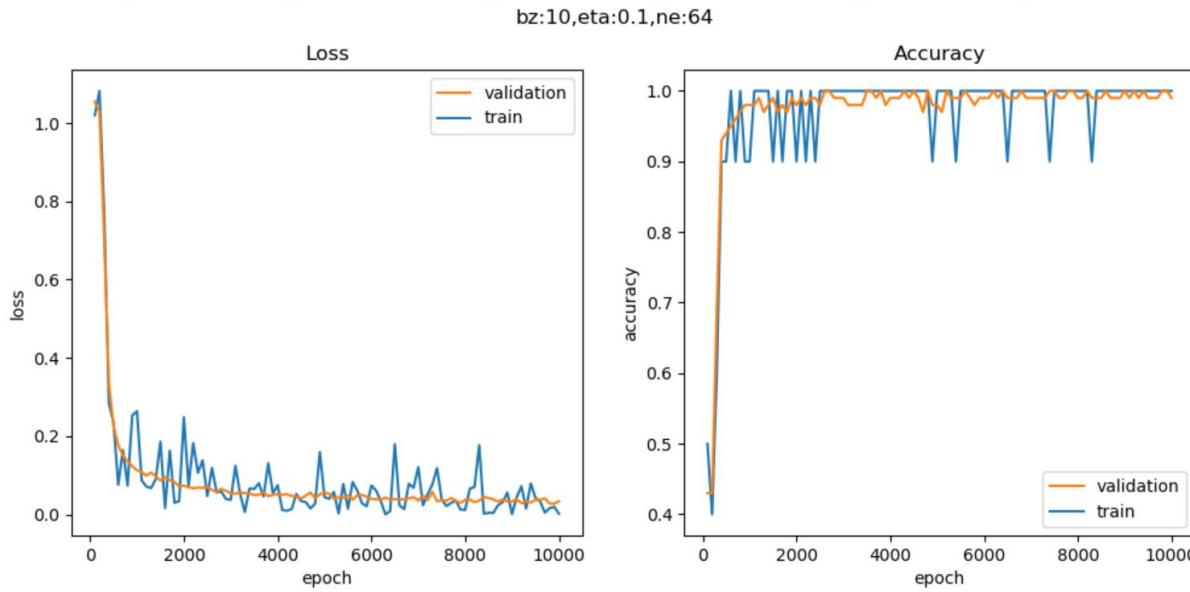
非线性多分类的工作原理 – 隐层神经元数量 32



测试准确度0.974，耗时53秒，损失函数值0.085。网络能力够了，从损失值下降趋势和准确度值上升趋势来看，可能需要更多的迭代次数。

较好地完成了分类任务

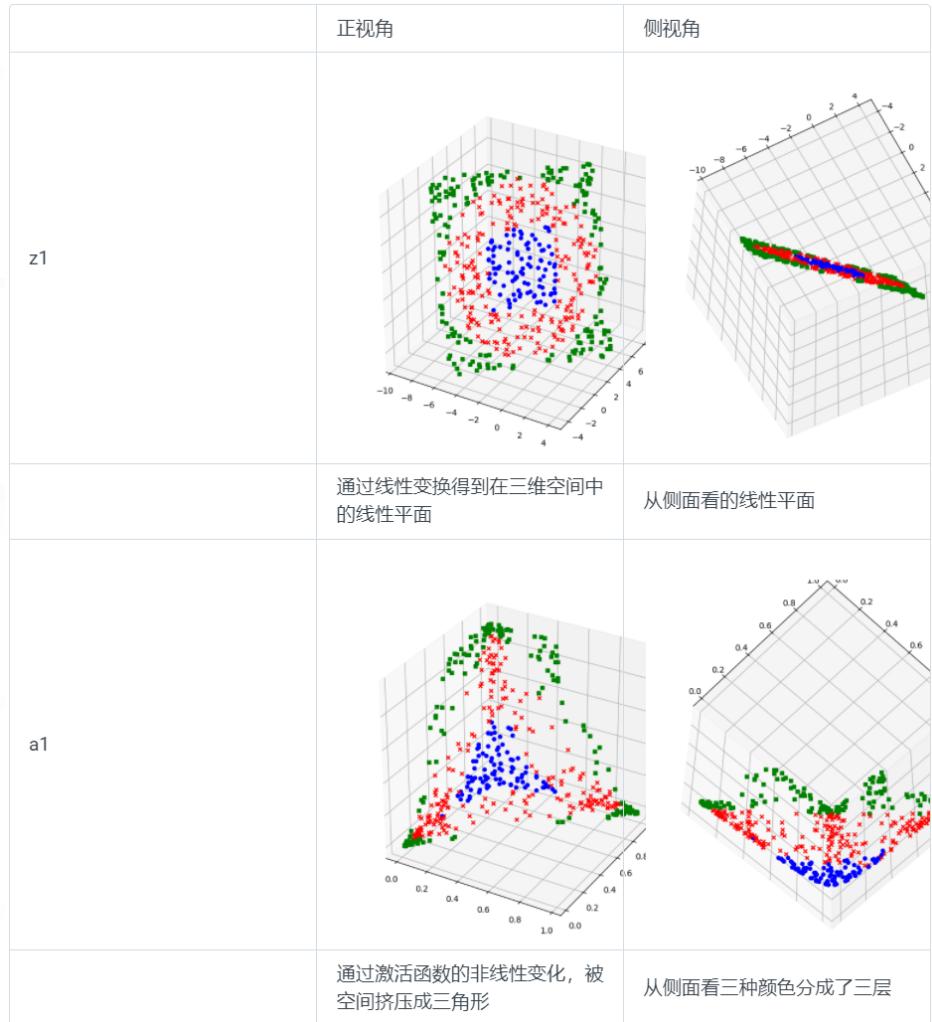
非线性多分类的工作原理 – 隐层神经元数量 64



测试准确度0.972，耗时64秒，损失函数值0.075。网络能力足够。

较好地完成了分类任务

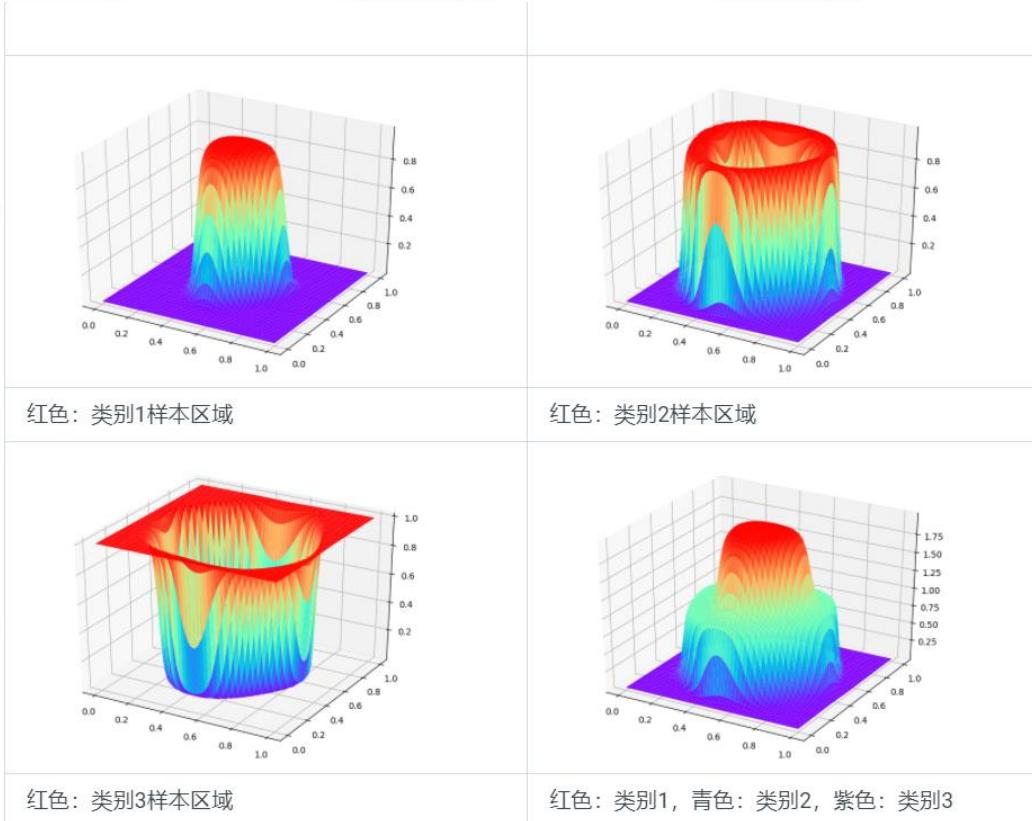
三维空间内的变换过程



net.Z1的点图的含义是，输入数据经过线性变换后的结果，可以看到由于只是线性变换，所以从侧视角看还只是一个二维平面的样子。

net.A1的点图含义是，经过激活函数做非线性变换后的图。由于绿色点比较靠近边缘，所以三维坐标中的每个值在经过Sigmoid激活函数计算后，都有至少一维坐标会是向1靠近的值，所以分散的比较开，形成外围的三角区域；蓝色点正好相反，三维坐标值都趋近于0，所以最后都集中在三维坐标原点的三角区域内；红色点处于前两者之间，因为有很多中间值。再观察net.A1的侧视图，似乎是已经分层了，蓝点沉积下去，绿点浮上来，红点在中间，像鸡尾酒一样分成了三层，这就给第二层神经网络创造了做一个线性三分类的条件，只需要两个平面，就可以把三者轻松分开了

3D分类结果图



更高维的空间无法展示，所以当隐层神经元数量为4或8或更多时，基本无法理解空间变换的样子了。但是有一个方法可以近似地解释高维情况：在三维空间时，蓝色点会被推挤到一个角落形成一个三角形，那么在N ($N > 3$) 维空间中，蓝色点也会被推挤到一个角落。由于N很大，所以一个多边形会近似成一个圆形，也就是我们下面要生成的这些立体图的样子

多入多出的三层神经网络

多变量非线性多分类

提出问题

手写识别是人工智能的重要课题之一。MNIST数字手写体识别图片集，大家一定不陌生，下面就是一些样本。



图12-1 MNIST数据集样本示例

由于这是从欧美国家和地区收集的数据，从图中可以看出有几点和中国人的手写习惯不一样：

- 数字2，下面多一个圈
- 数字4，很多横线不出头
- 数字6，上面是直的
- 数字7，中间有个横杠

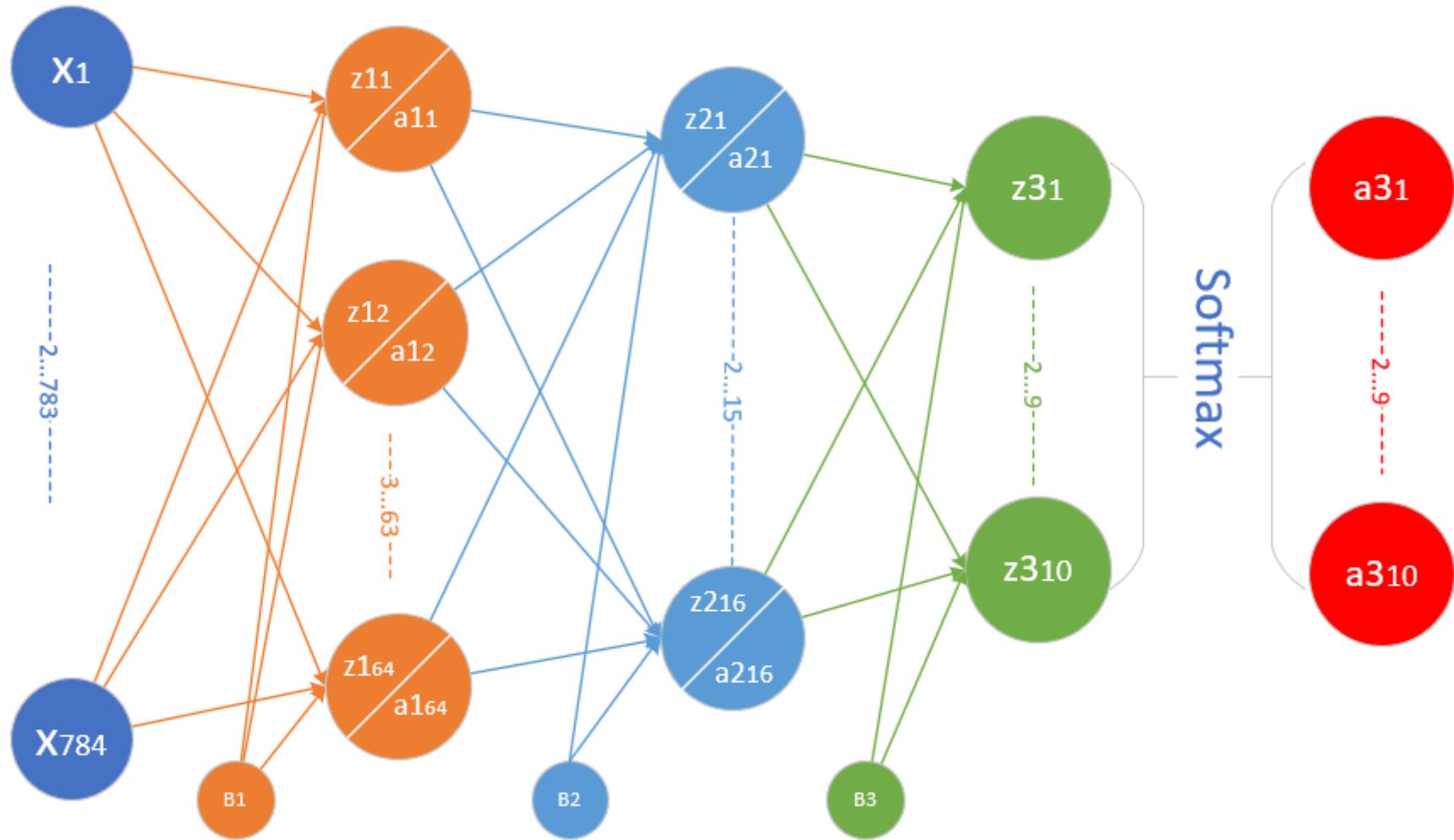
不过这些细节不影响咱们学习课程，正好还可以验证一下中国人的手写习惯是否能够被正确识别。

由于不是让我们识别26个英文字母或者3500多个常用汉字，所以问题还算是比较简单，不需要图像处理知识，也暂时不需要卷积神经网络的参与。咱们可以试试用一个三层的神经网络解决此问题，把每个图片的像素都当作一个向量来看，而不是作为点阵。

图片数据归一化

数据归一化，是针对数据的特征值做的处理，也就是针对样本数据的列做的处理。

三层神经网络的实现



三层神经网络的实现

输入层

28x28=784个特征值:

$$X = (x_1 \quad x_2 \quad \dots \quad x_{784})$$

隐层1

- 权重矩阵w1形状为784x64

$$W1 = \begin{pmatrix} w_{1,1}^1 & w_{1,2}^1 & \dots & w_{1,64}^1 \\ \dots & \dots & \dots & \dots \\ w_{784,1}^1 & w_{784,2}^1 & \dots & w_{784,64}^1 \end{pmatrix}$$

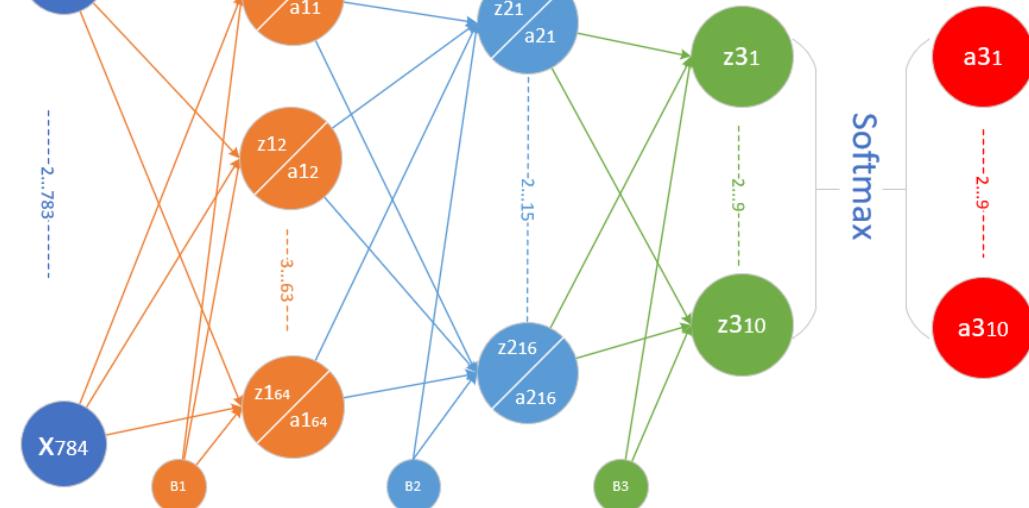
- 偏移矩阵b1的形状为1x64

$$B1 = (b_1^1 \quad b_2^1 \quad \dots \quad b_{64}^1)$$

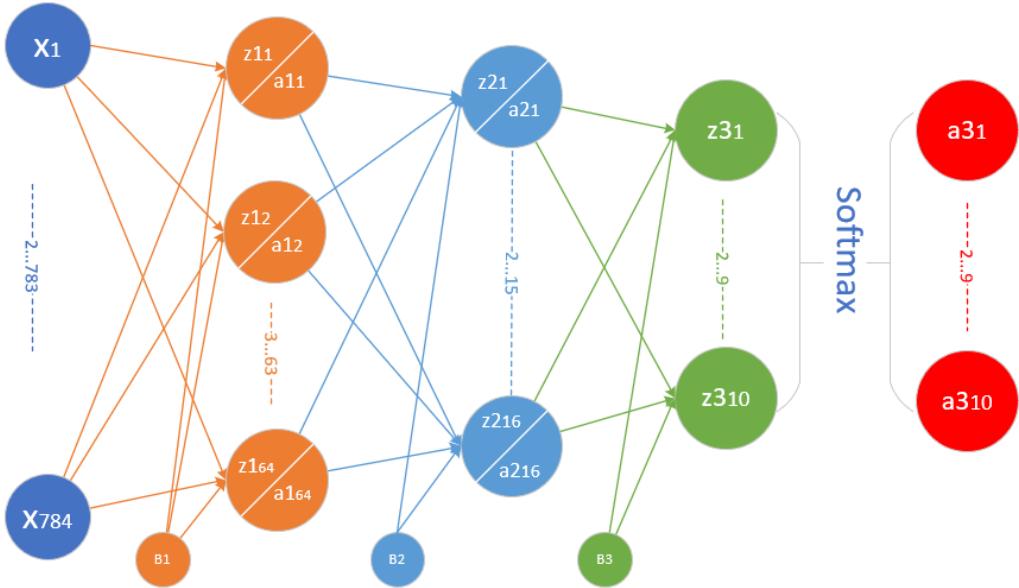
- 隐层1由64个神经元构成，其结果为1x64的矩阵

$$Z1 = (z_1^1 \quad z_2^1 \quad \dots \quad z_{64}^1)$$

$$A1 = (a_1^1 \quad a_2^1 \quad \dots \quad a_{64}^1)$$



三层神经网络的实现



隐层2

- 权重矩阵w2形状为64x16

$$W2 = \begin{pmatrix} w_{1,1}^2 & w_{1,2}^2 & \dots & w_{1,16}^2 \\ \dots & \dots & \dots & \dots \\ w_{64,1}^2 & w_{64,2}^2 & \dots & w_{64,16}^2 \end{pmatrix}$$

- 偏移矩阵b2的形状是1x16

$$B2 = (b_1^2 \ b_2^2 \ \dots \ b_{16}^2)$$

- 隐层2由16个神经元构成

$$Z2 = (z_1^2 \ z_2^2 \ \dots \ z_{16}^2)$$

$$A2 = (a_1^2 \ a_2^2 \ \dots \ a_{16}^2)$$

输出层②

- 权重矩阵w3的形状为16x10

$$W3 = \begin{pmatrix} w_{1,1}^3 & w_{1,2}^3 & \dots & w_{1,10}^3 \\ \dots & \dots & \dots & \dots \\ w_{16,1}^3 & w_{16,2}^3 & \dots & w_{16,10}^3 \end{pmatrix}$$

- 输出层的偏移矩阵b3的形状是1x10

$$B3 = (b_1^3 \ b_2^3 \ \dots \ b_{10}^3)$$

- 输出层有10个神经元使用Softmax函数进行分类

三层神经网络的实现-前向计算

隐层1

$$Z1 = X \cdot W1 + B1 \quad (1)$$

$$A1 = Sigmoid(Z1) \quad (2)$$

隐层2

$$Z2 = A1 \cdot W2 + B2 \quad (3)$$

$$A2 = Tanh(Z2) \quad (4)$$

输出层

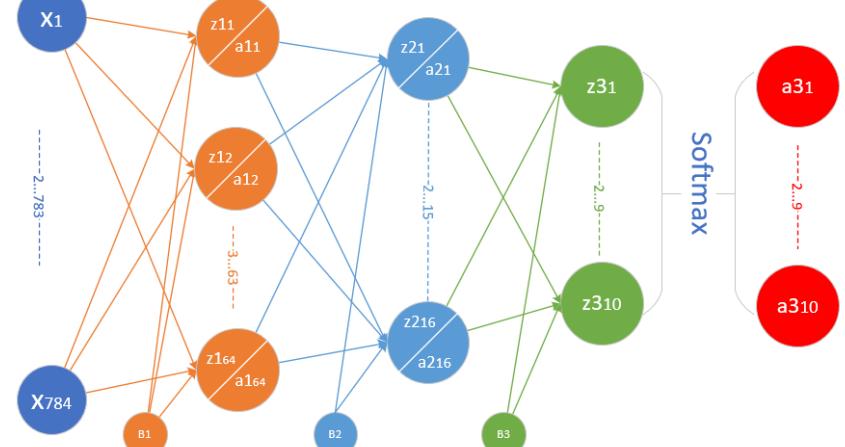
$$Z3 = A2 \cdot W3 + B3 \quad (5)$$

$$A3 = Softmax(Z3) \quad (6)$$

我们的约定是行为样本，列为一个样本的所有特征，这里是784个特征，因为图片高和宽是28x28，总共784个点，把每一个点的值做为特征向量。

两个隐层，分别定义64个神经元和16个神经元。第一个隐层用Sigmoid激活函数，第二个隐层用Tanh激活函数。

输出层10个神经元，再加上一个Softmax计算，最后有a1,a2,...a10十个输出，分别代表0-9的10个数字。



三层神经网络的实现-反向传播

输出层

$$dZ3 = A3 - Y \quad (7)$$

$$dW3 = A2^T \cdot dZ3 \quad (8)$$

$$dB3 = dZ3 \quad (9)$$

隐层2

$$dA2 = dZ3 \cdot W3^T \quad (10)$$

$$dZ2 = dA2 \odot (1 - A2 \odot A2) \quad (11)$$

$$dW2 = A1^T \cdot dZ2 \quad (12)$$

$$dB2 = dZ2 \quad (13)$$

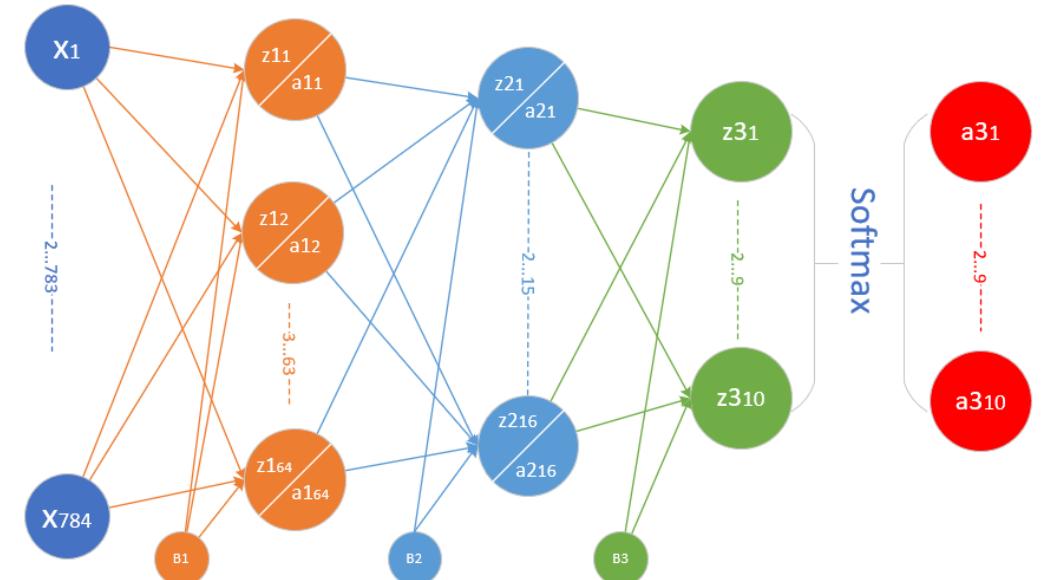
隐层1 ⊙

$$dA1 = dZ2 \cdot W2^T \quad (14)$$

$$dZ1 = dA1 \odot A1 \odot (1 - A1) \quad (15)$$

$$dW1 = X^T \cdot dZ1 \quad (16)$$

$$dB1 = dZ1 \quad (17)$$





Reactor

Thank You!