

Reactor

一起学人工智能系列
- 非线性回归

2021-10-13



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>

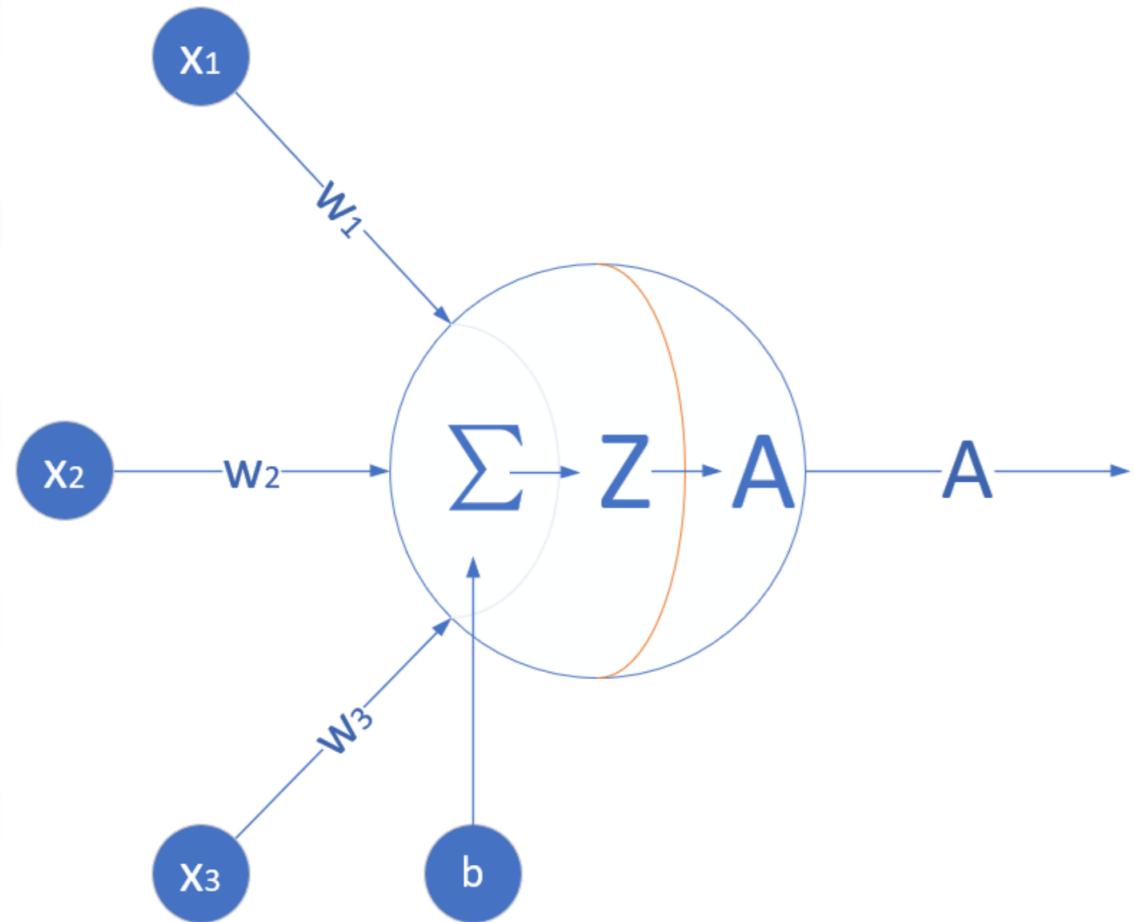
Twitter : @Ljh8304

回顾



激活函数的作用

假设该神经元有三个输入，分别为 x_1, x_2, x_3 ，那么：



$$z = x_1w_1 + x_2w_2 + x_3w_3 + b$$

$$a = \sigma(z)$$

给神经网络增加非线性因素

把公式1的计算结果压缩到[0,1]之间，便于后面的计算。

激活函数的基本性质

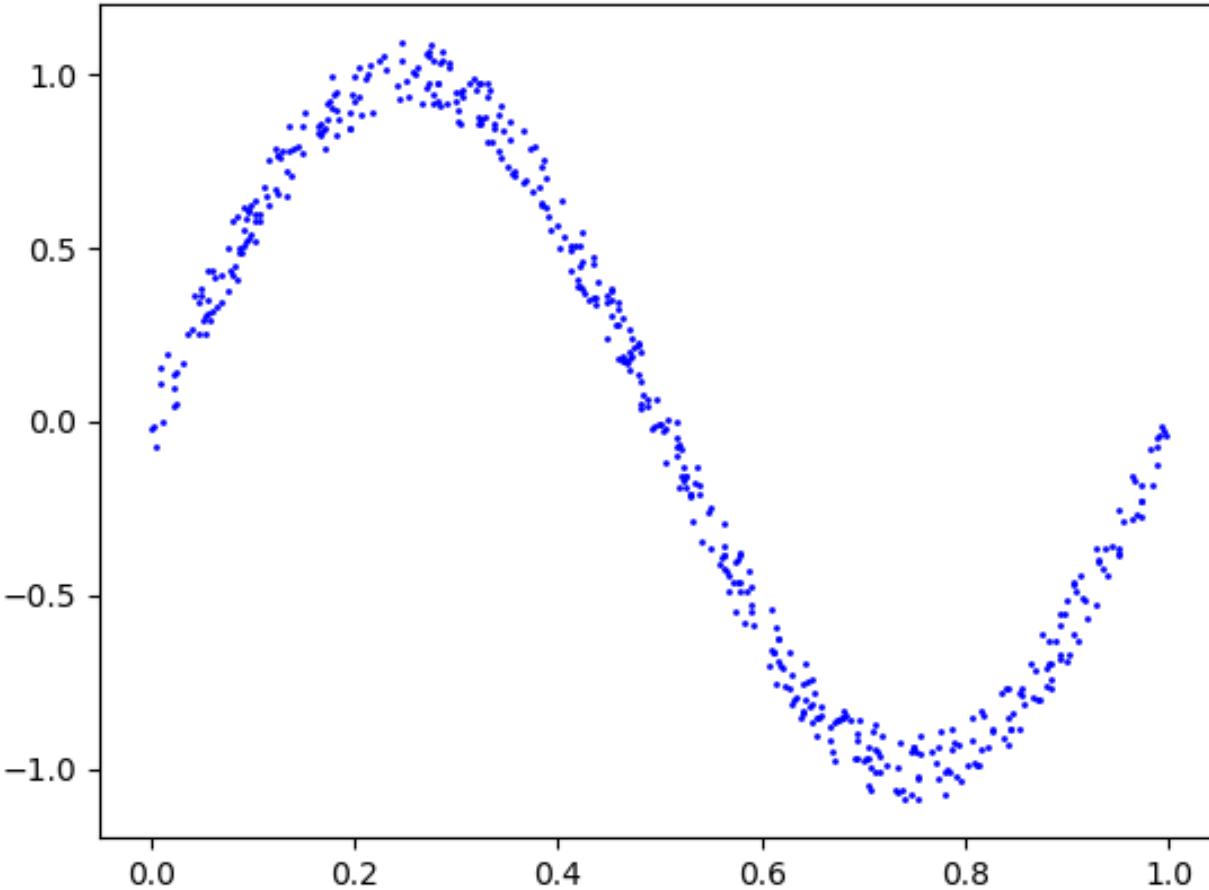
- 非线性：线性的激活函数和没有激活函数一样；
- 可导性：做误差反向传播和梯度下降，必须要保证激活函数的可导性；
- 单调性：单一的输入会得到单一的输出，较大值的输入得到较大值的输出。

单入单出的双层神经网络



问题一

正弦曲线

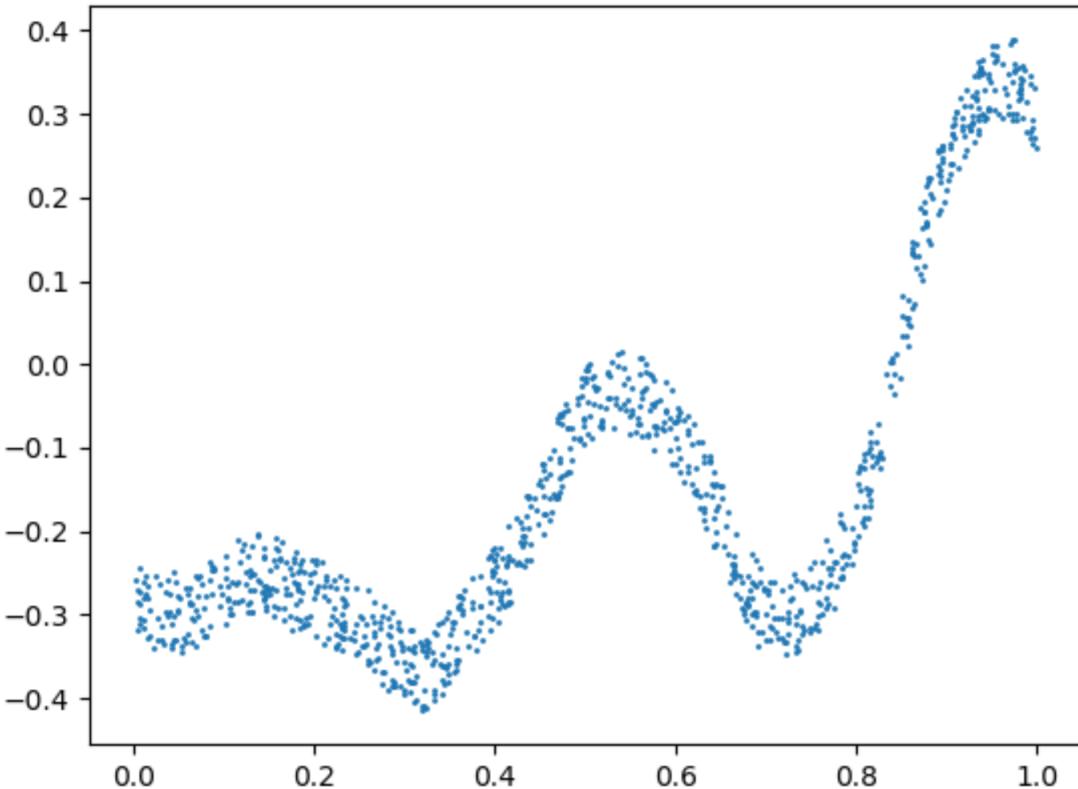


如何使用神经网络拟合一条有很
强规律的曲线，比如正弦曲线？

样本	x	y
1	0.1199	0.6108
2	0.0535	0.3832
3	0.6978	0.9496
...

问题二

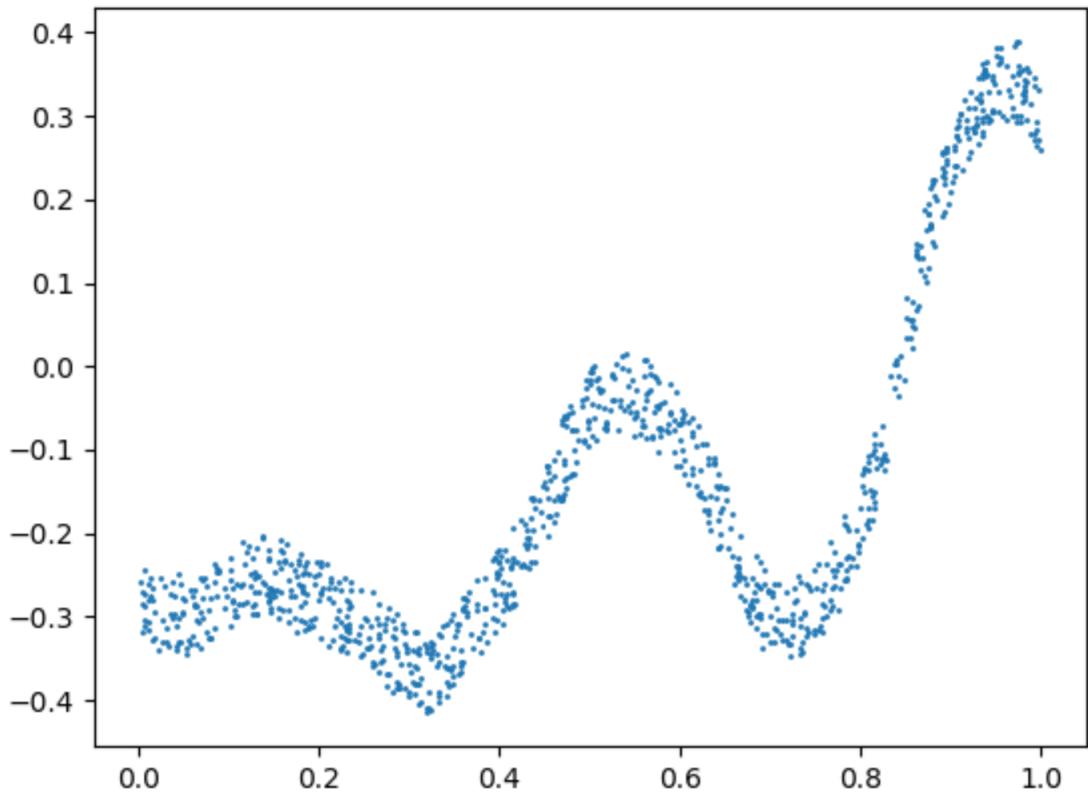
更复杂的曲线



如果是更复杂的曲线，单层神经网络还能轻易做到吗？

样本	x	y
1	0.606	-0.113
2	0.129	-0.269
3	0.582	0.027
...
1000	0.199	-0.281

问题二



$$y = 0.4x^2 + 0.3x\sin(15x) + 0.01\cos(50x) - 0.3$$

我们特意把数据限制在[0,1]之间，避免做归一化的麻烦。要是觉得这个公式还不够复杂，大家可以用更复杂的公式去自己做试验。

以上问题可以叫做非线性回归，即自变量X和因变量Y之间不是线性关系。常用的传统的处理方法有线性迭代法、分段回归法、迭代最小二乘法等。在神经网络中，解决这类问题的思路非常简单，就是使用带有一个隐层的两层神经网络。

回归模型评估



回归模型的评估标准

回归问题主要是求值，评价标准主要是看求得值与实际结果的偏差有多大，所以，回归问题主要以下方法来评价模型。

平均绝对误差

MAE (Mean Absolute Error) 。

$$MAE = \frac{1}{m} \sum_{i=1}^m |a_i - y_i| \quad (1)$$

对异常值不如均方差敏感，类似中位数。

绝对平均值率误差

MAPE (Mean Absolute Percentage Error) 。

$$MAPE = \frac{100}{m} \sum_{i=1}^m \left| \frac{a_i - y_i}{y_i} \right| \quad (2)$$

和方差

SSE (Sum Squared Error) 。

$$SSE = \sum_{i=1}^m (a_i - y_i)^2 \quad (3)$$

得出的值与样本数量有关系，假设有1000个测试样本，得到的值是120；如果只有100个测试样本，得到的值可能是11，我们不能说11就比120要好。

回归模型的评估标准

均方差

MSE (Mean Squared Error) .



$$MSE = \frac{1}{m} \sum_{i=1}^m (a_i - y_i)^2 \quad (4)$$

就是实际值减去预测值的平方再求期望，没错，就是线性回归的代价函数。由于MSE计算的是误差的平方，所以它对异常值是非常敏感的，因为一旦出现异常值，MSE指标会变得非常大。MSE越小，证明误差越小。

均方根误差

RMSE (Root Mean Squared Error) .

$$RMSE = \sqrt{\frac{1}{m} \sum_{i=1}^m (a_i - y_i)^2} \quad (5)$$

是均方差开根号的结果，其实质是一样的，只不过对结果有更好的解释。

例如：要做房价预测，每平方是万元，我们预测结果也是万元，那么MSE差值的平方单位应该是千万级别的。假设我们的模型预测结果与真实值相差1000元，则用MSE的计算结果是1000,000，这个值没有单位，如何描述这个差距？于是就求个平方根就好了，这样误差可以是标签值是同一个数量级的，在描述模型的时候就说，我们模型的误差是多少元。

回归模型的评估标准

R平方

R-Squared。

上面的几种衡量标准针对不同的模型会有不同的值。比如说预测房价，那么误差单位就是元，比如3000元、11000元等。如果预测身高就可能是0.1、0.2米之类的。也就是说，对于不同的场景，会有不同量纲，因而也会有不同的数值，无法用一句话说得很清楚，必须啰嗦一大堆条件才能表达完整。

我们通常用概率来表达一个准确率，比如89%的准确率。那么线性回归有没有这样的衡量标准呢？答案就是R-Squared。

$$R^2 = 1 - \frac{\sum(a_i - \bar{y}_i)^2}{\sum(\bar{y}_i - y_i)^2} = 1 - \frac{MSE(a, y)}{Var(y)} \quad (6)$$

R平方是多元回归中的回归平方和（分子）占总平方和（分母）的比例，它是度量多元回归方程中拟合程度的一个统计量。R平方值越接近1，表明回归平方和占总平方和的比例越大，回归线与各观测点越接近，回归的拟合程度就越好。

- 如果结果是0，说明模型跟瞎猜差不多；
- 如果结果是1，说明模型无错误；
- 如果结果是0-1之间的数，就是模型的好坏程度；
- 如果结果是负数，说明模型还不如瞎猜。

用多项式回归法拟合正弦曲线



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。

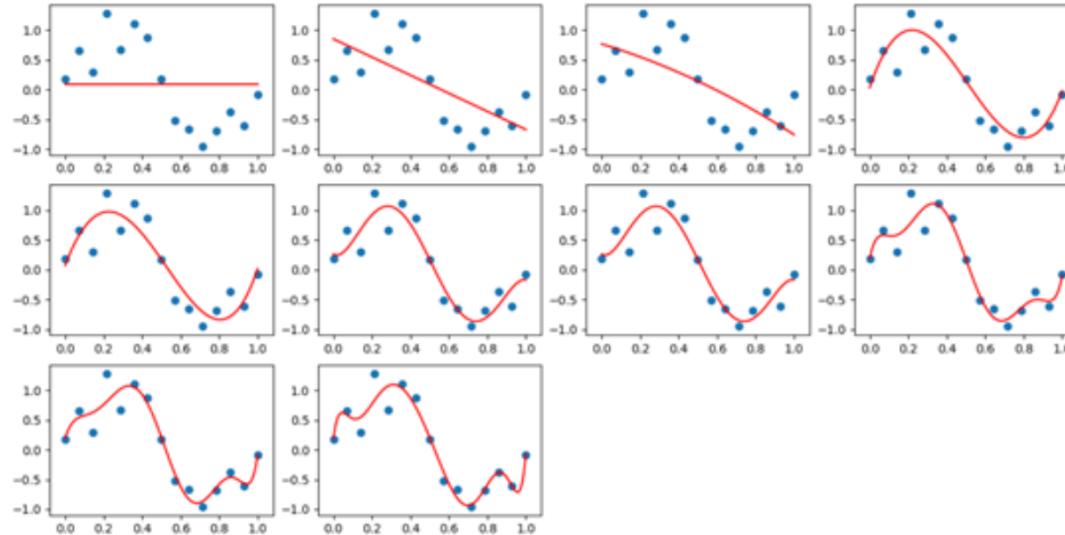
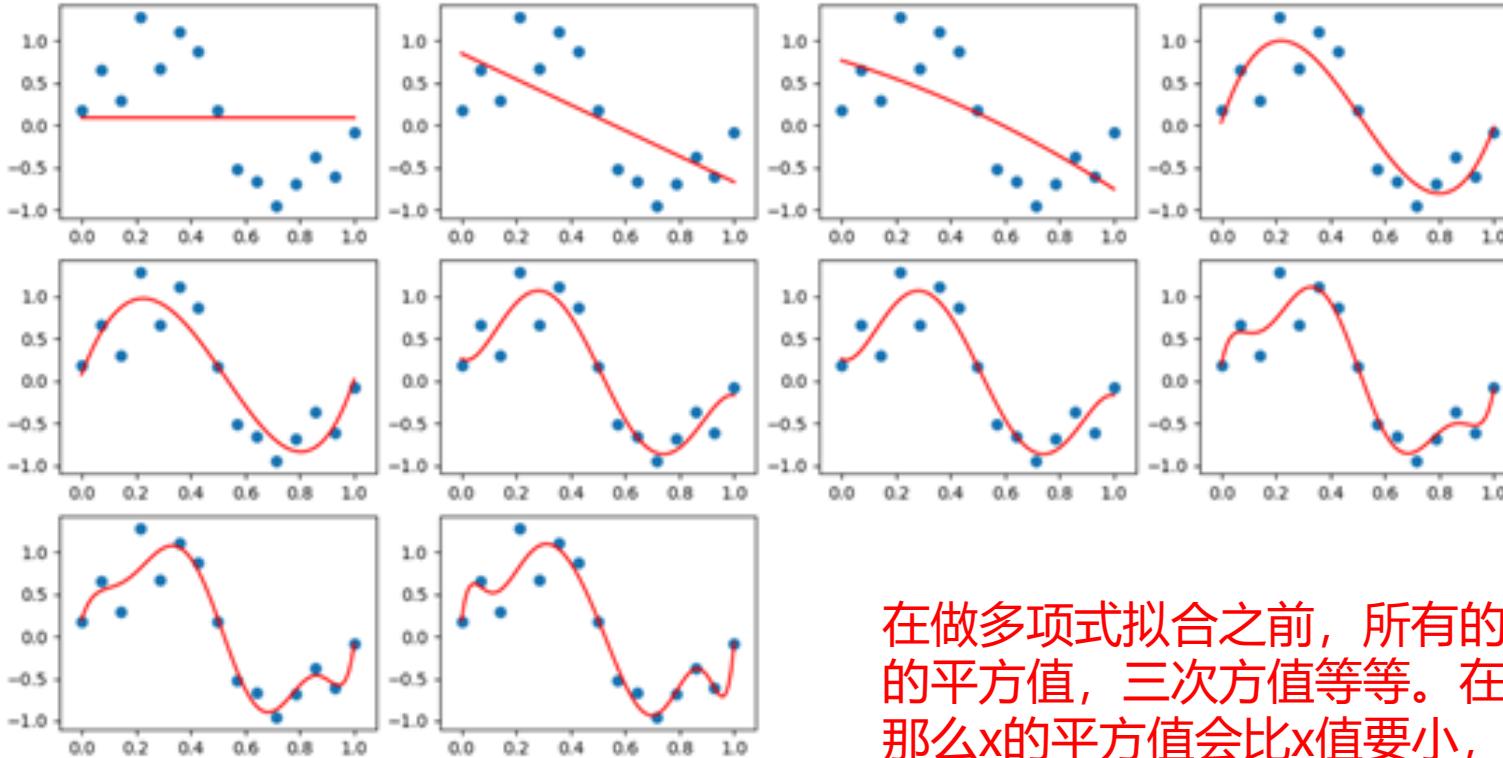


图9-3 对有噪音的正弦曲线的拟合

一堆散点，看上去像是一条带有很多噪音的正弦曲线，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

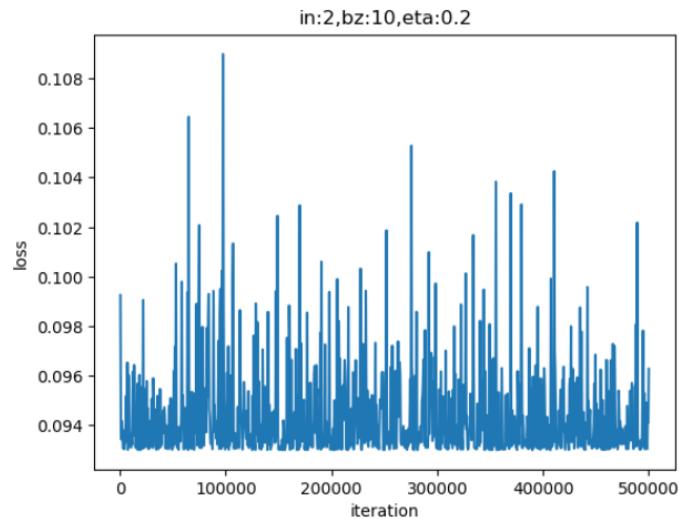


在做多项式拟合之前，所有的特征值都会先做归一化，然后再获得 x 的平方值，三次方值等等。在归一化之后， x 的值变成了 $[0,1]$ 之间，那么 x 的平方值会比 x 值要小， x 的三次方值会比 x 的平方值要小。假设 $x=0.5$, $x*x=0.25$, $x*x*x=0.125$, 所以次数越高，权重值会越大，特征值与权重值的乘积才会是一个不太小的数，以此来弥补特征值小的问题。

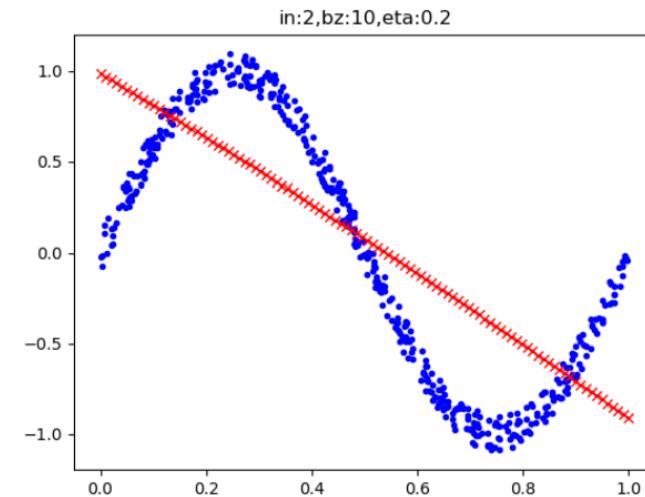
场景-用二次多项式拟合

$$z = xw_1 + x^2w_2 + b$$

损失函数值



拟合结果

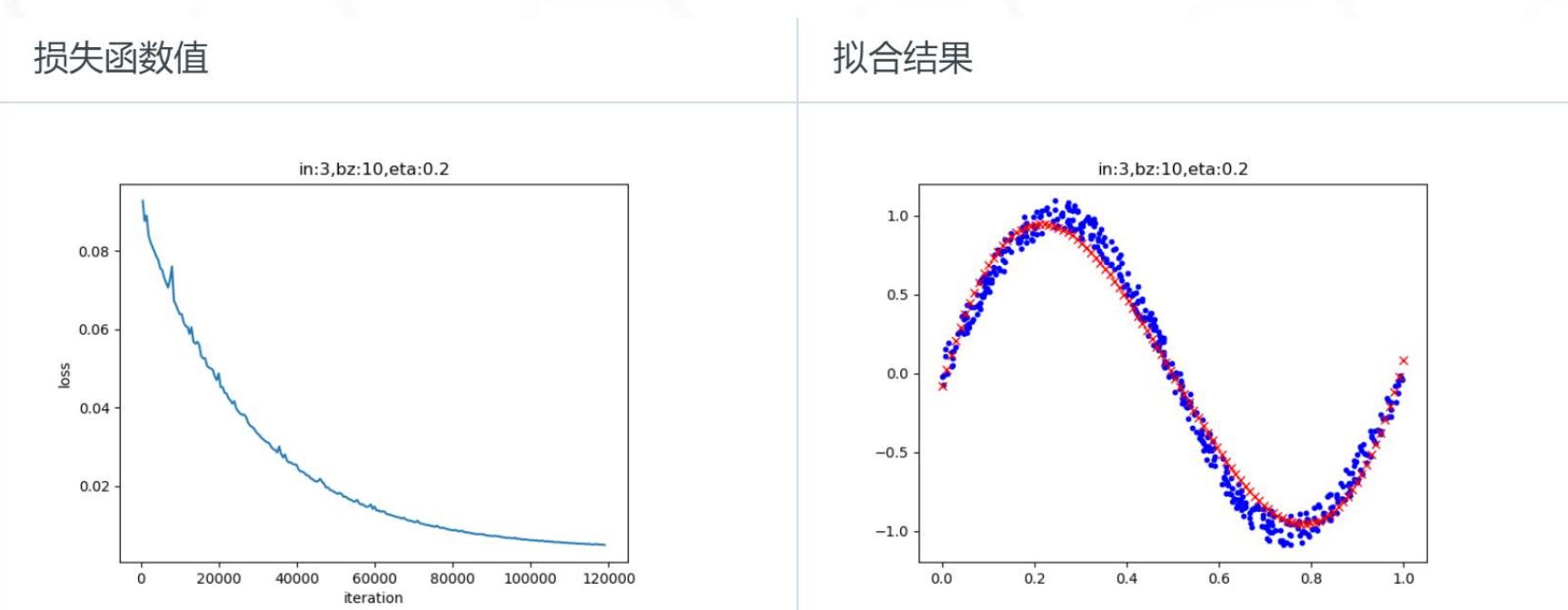


没有任何损失值下降的趋势

拟合情况，只拟合成了一条直线

场景-用三次多项式拟合

$$z = xw_1 + x^2w_2 + x^3w_3 + b$$

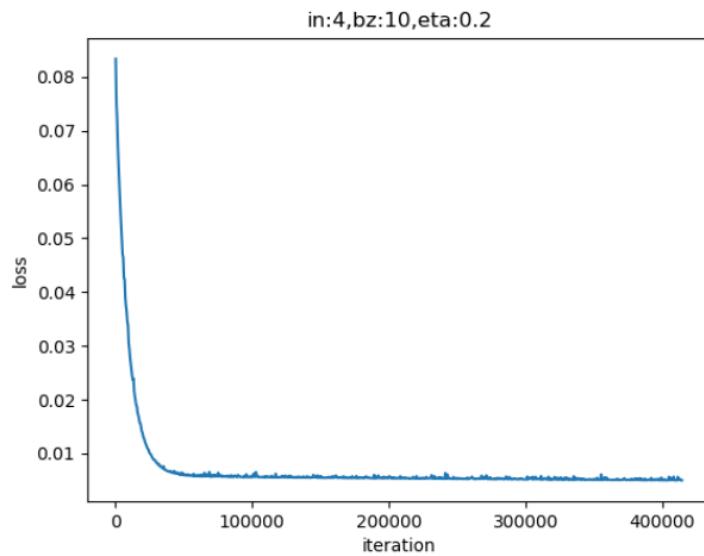


损失函数值下降得很平稳

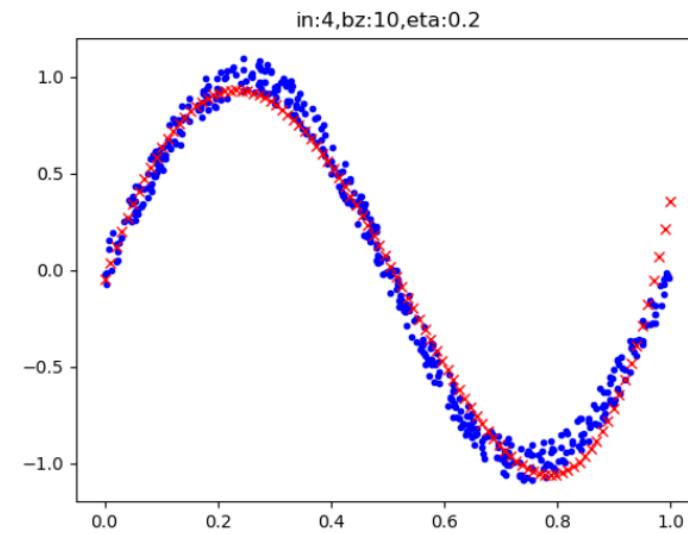
拟合的结果也很令人满意，虽然红色线没有严丝合缝地落在蓝色样本点内

场景-用四次多项式拟合

损失函数值



拟合结果



场景-结果比较

多项式次数	迭代数	损失函数值
2	10000	0.095
3	2380	0.005
4	8290	0.005

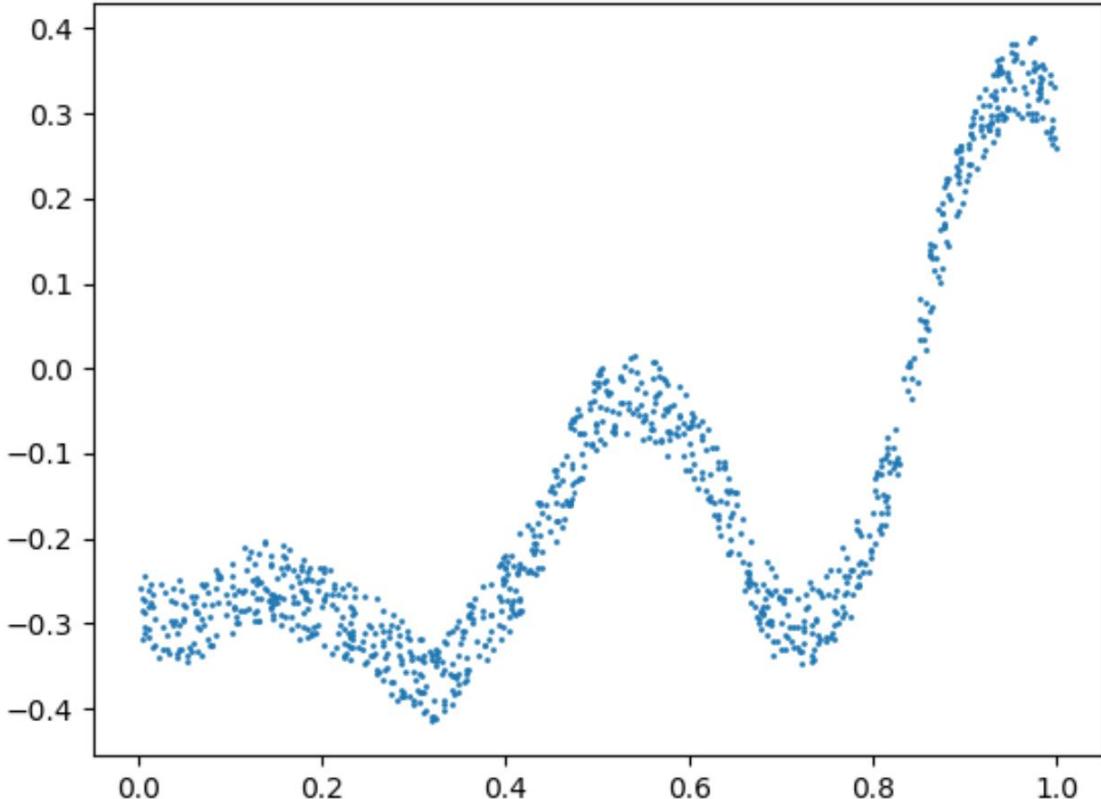
1. 二次多项式的损失值在下降了一定程度后，一直处于平缓期，不再下降，说明网络能力到了一定的限制，直到10000次迭代也没有达到目的；
2. 损失值达到0.005时，四项式迭代了8290次，比三次多项式的2380次要多很多，说明四次多项式多出的一个特征值，没有给我们带来什么好处，反而是增加了网络训练的复杂度。

多项式次数并不是越高越好，对不同的问题，有特定的限制，需要在实践中摸索，并无理论指导。

用多项式回归法拟合正弦曲线



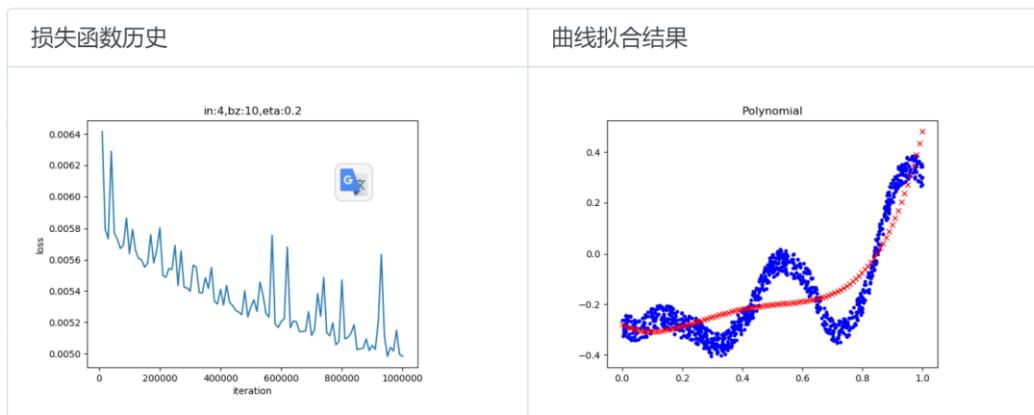
场景-复合函数曲线



不但有正弦式的波浪，还有线性的爬升，转折处也不是很平滑，所以难度很大。从正弦曲线的拟合经验来看，三次多项式以下肯定无法解决，所以我们可以从四次多项式开始试验。

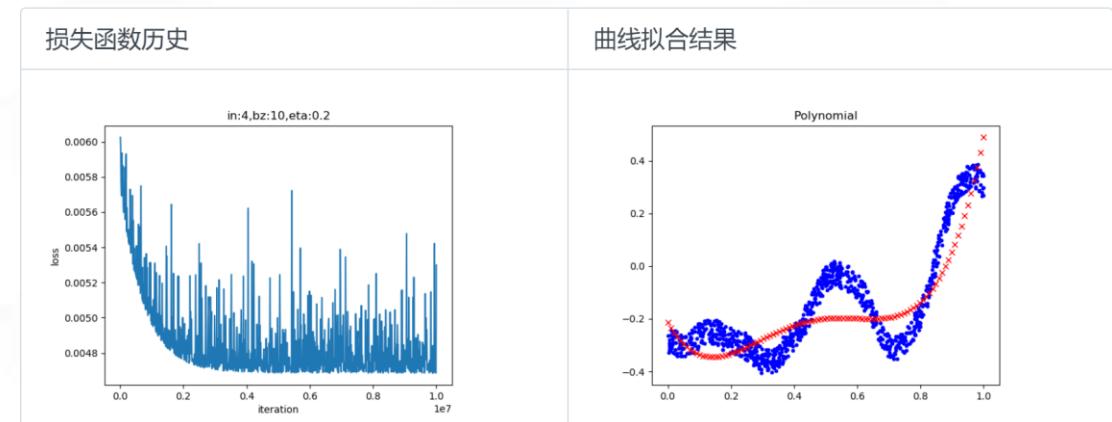
场景-用四次多项式拟合

四次多项式1万次迭代的训练结果



看到损失函数值还有下降的空间，拟合情况很糟糕

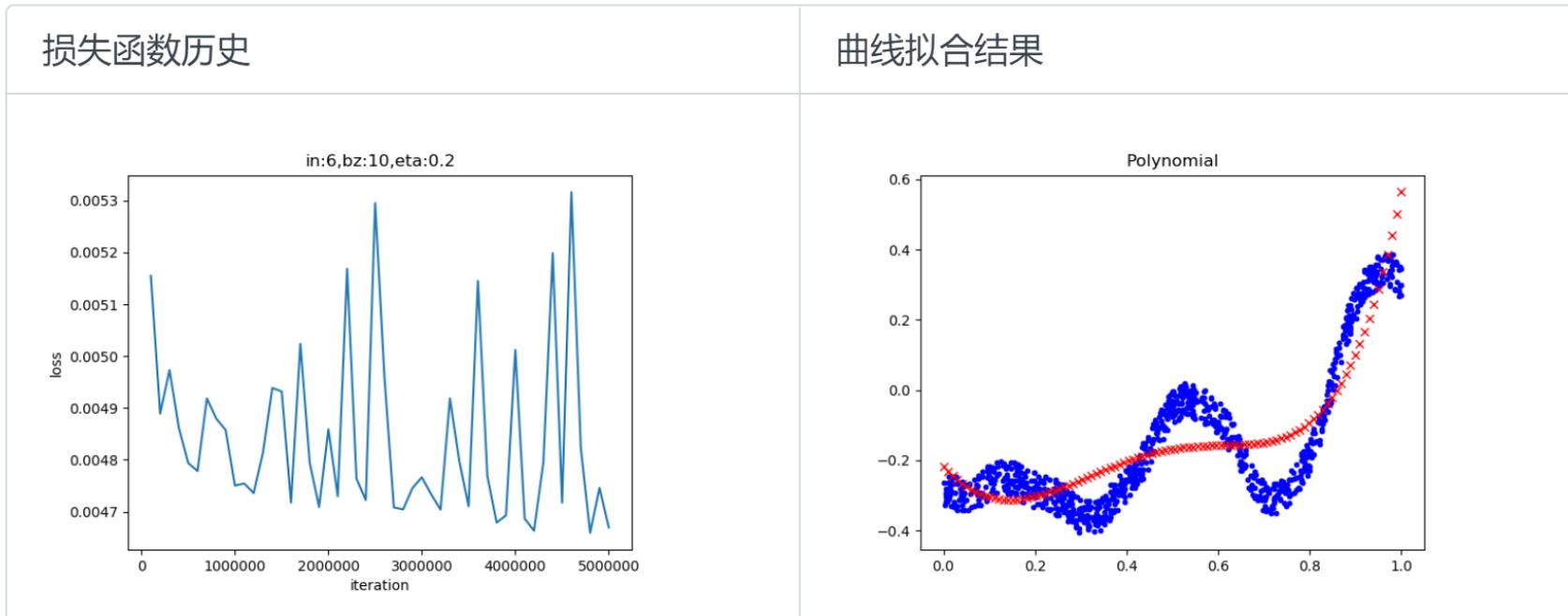
四次多项式10万次迭代的训练结果



损失函数值到了一定程度后就不再下降了，说明网络能力有限。

场景-用六次多项式拟合

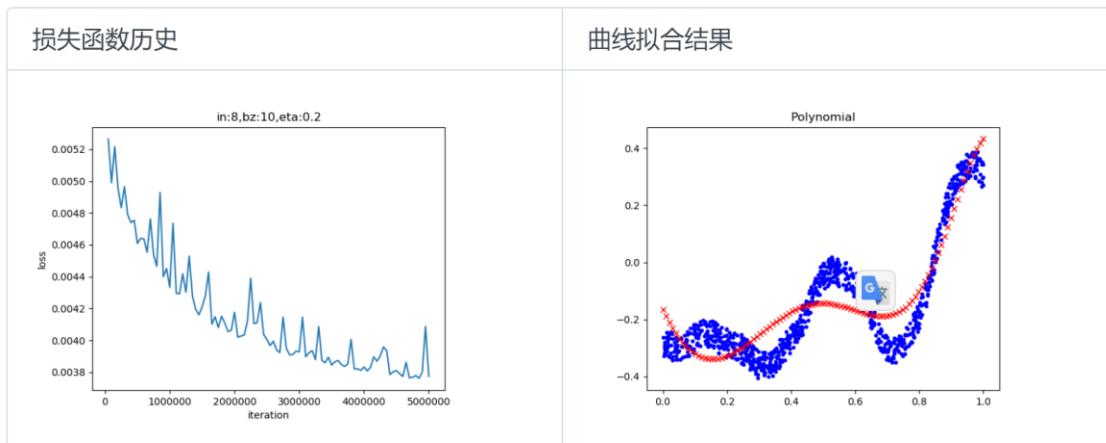
六次多项式5万次迭代的训练结果



损失值最开始几轮就已经是0.0047了，到了最后一轮，是0.0046，并不理想，说明网络能力还是不够

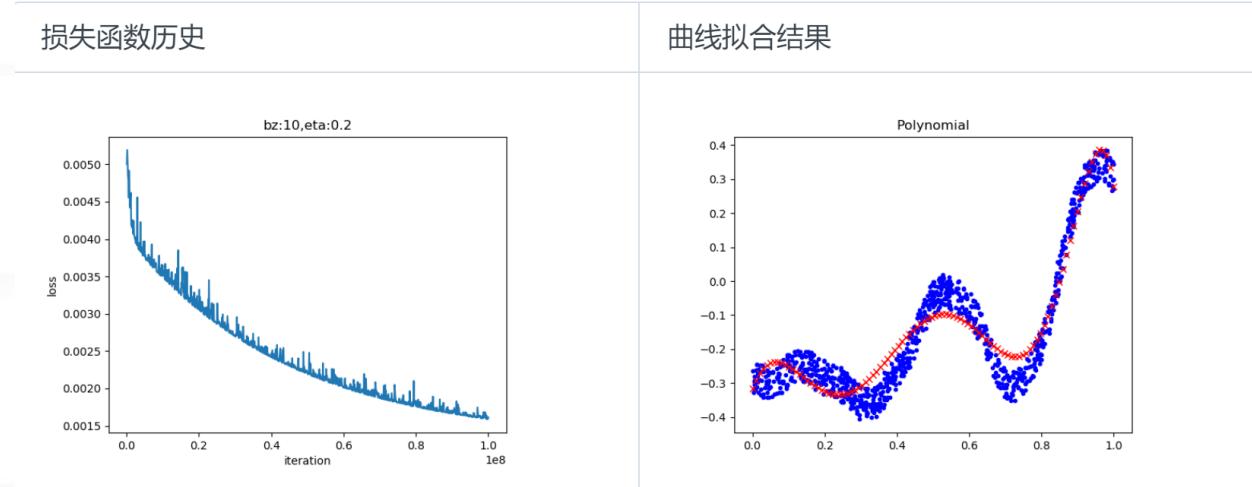
场景-用八次多项式拟合

八项式5万次迭代的训练结果



损失函数值下降的趋势非常可喜，似乎还没有遇到什么瓶颈，仍有下降的空间，并且拟合的效果也已经初步显现出来了

八项式100万次迭代的训练结果

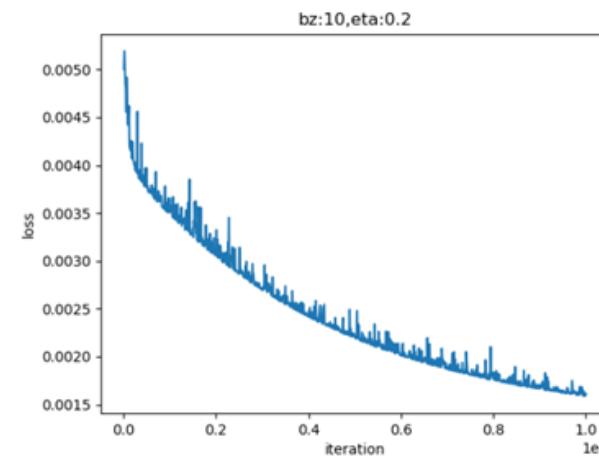


损失函数值还有下降的空间和可能性，已经到了0.0016的水平（从后面的章节中可以知道，0.001的水平可以得到比较好的拟合效果），拟合效果也已经初步呈现出来了，所有转折的地方都可以复现，只是精度不够，相信更多的训练次数可以达到更好的效果。

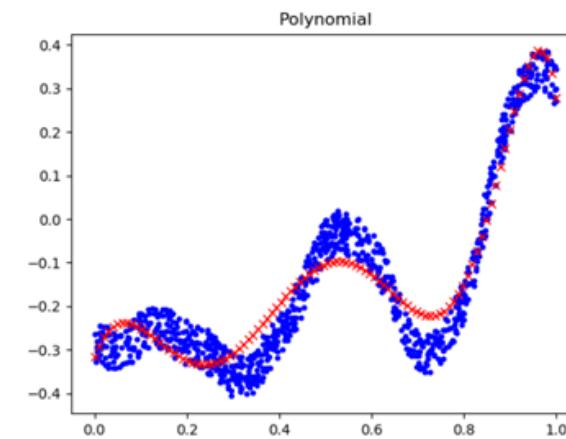
结论

多项式回归确实可以解决复杂曲线拟合问题，但是代价有些高，我们训练了一百万次，才得到初步满意的结果

损失函数历史



曲线拟合结果



双层神经网络实现非线性回归



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。

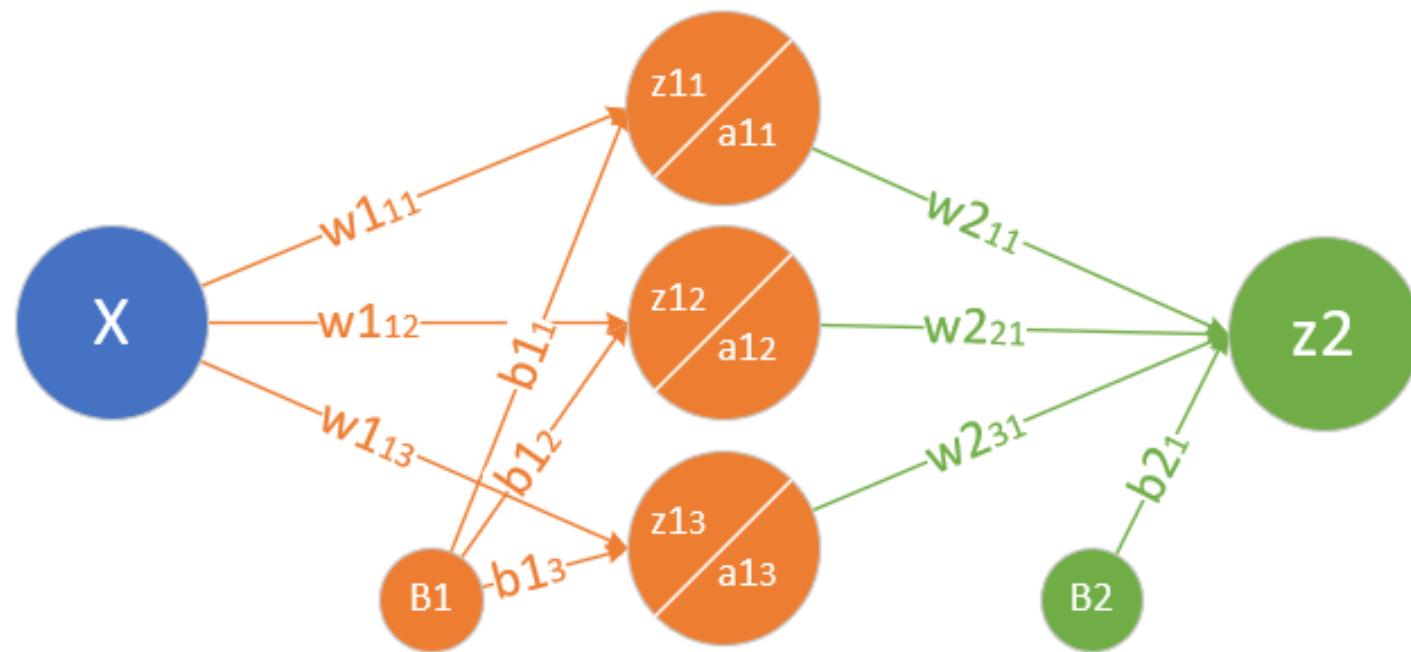
万能近似定理

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

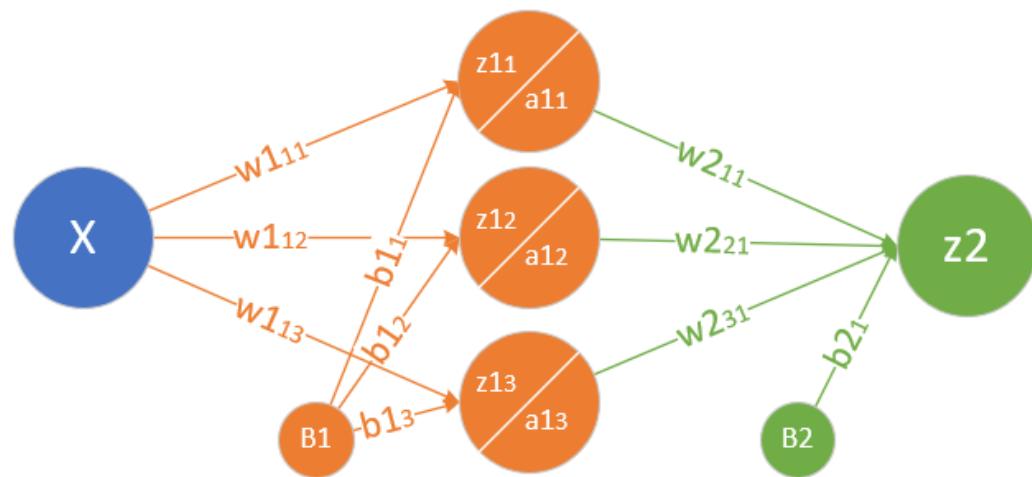
根据万能近似定理的要求，我们定义一个两层的神经网络，输入层不算，一个隐藏层，含3个神经元，一个输出层。



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。

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

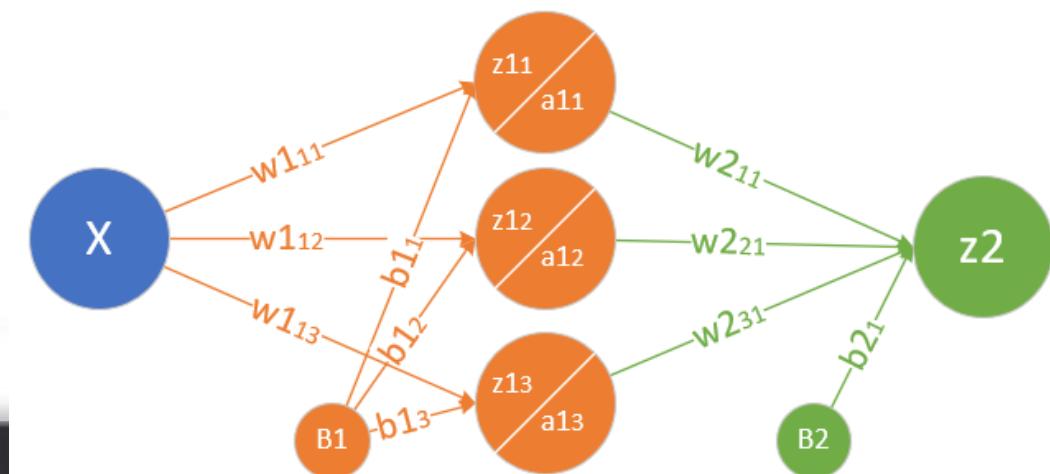
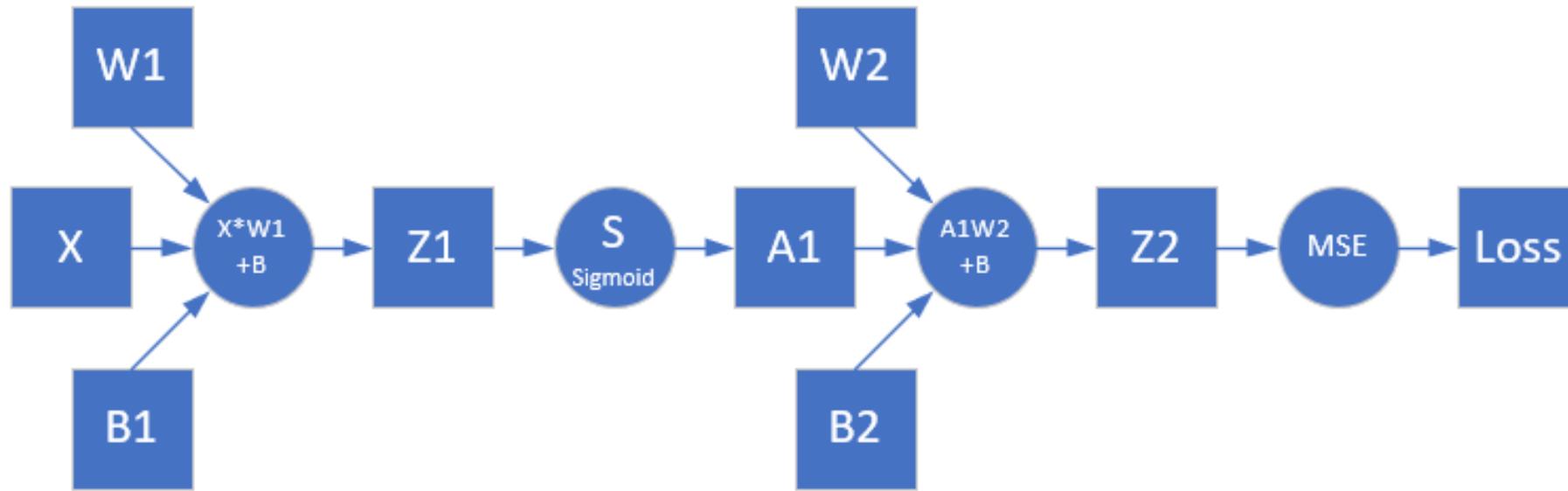
$$B2 = (b_1^2)$$

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

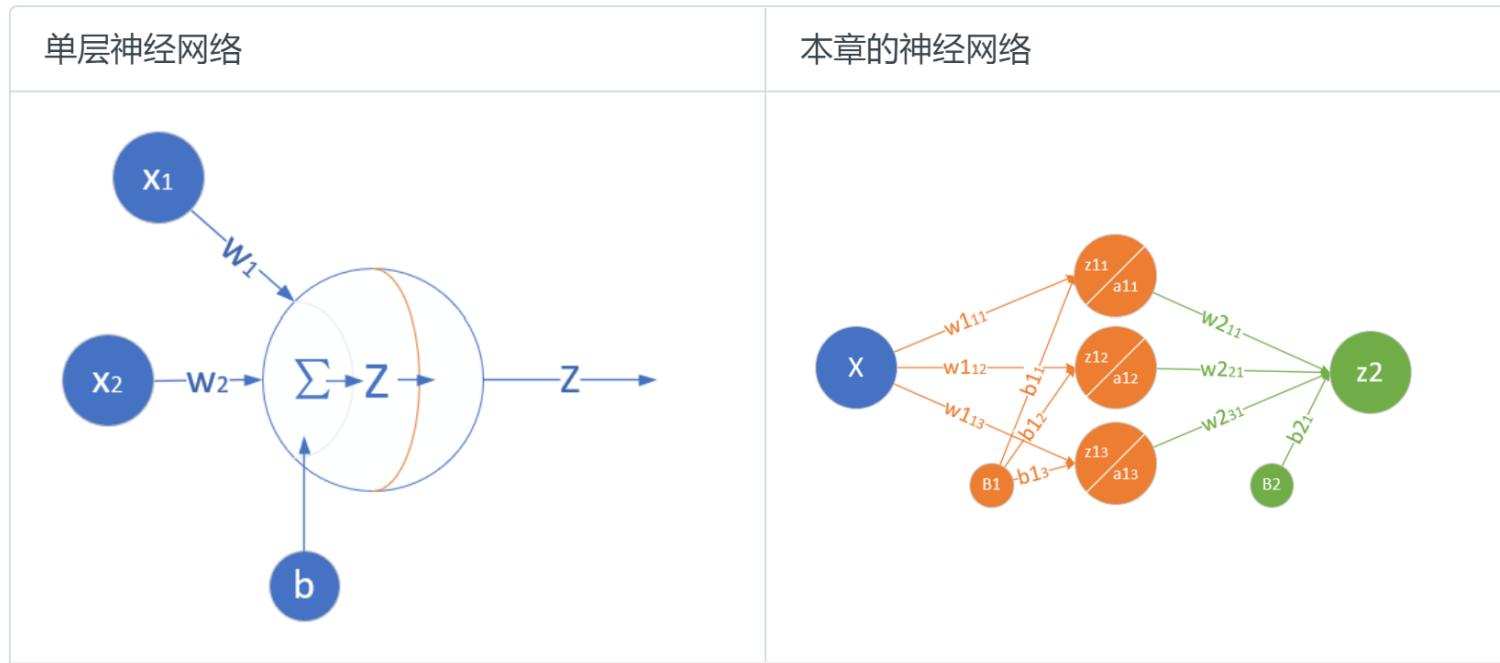
均方差损失函数：

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

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。

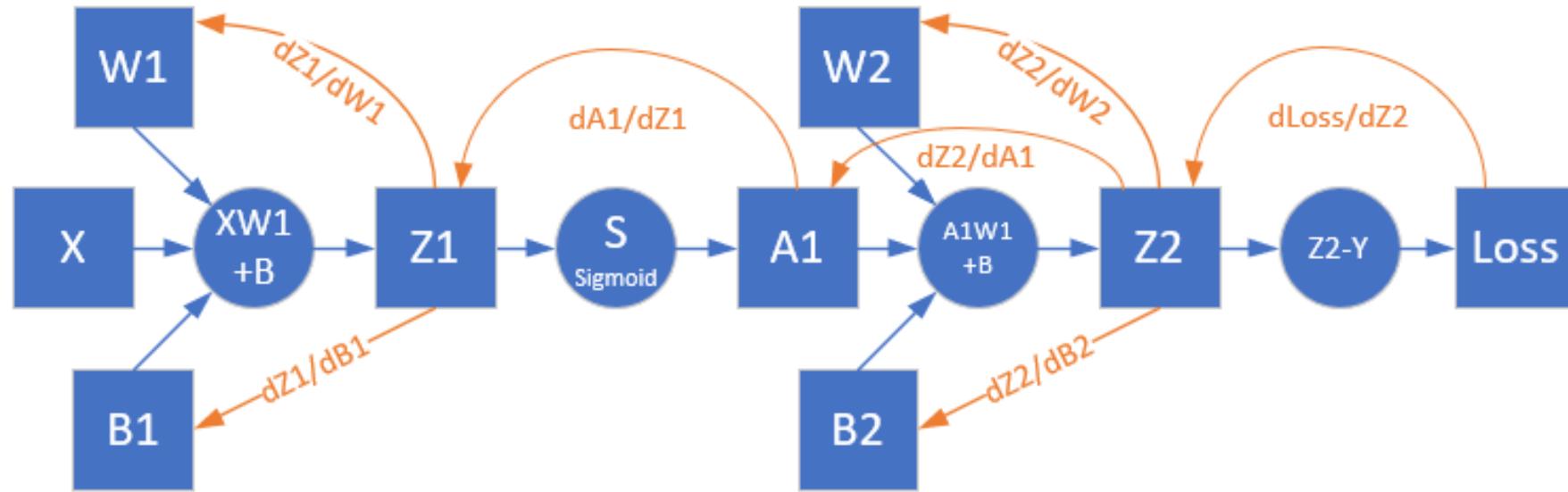
求B2的梯度

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

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：

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

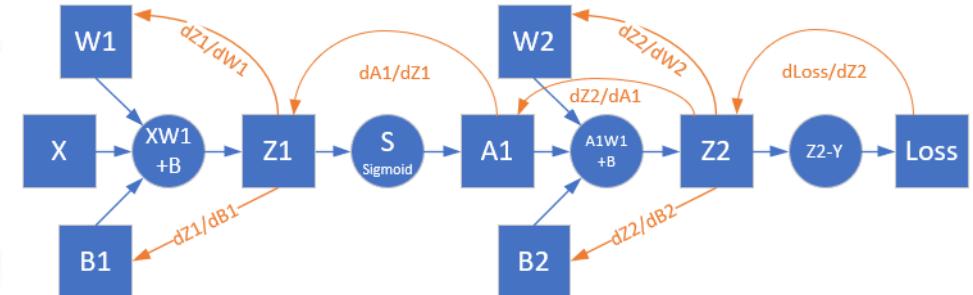
所以最后到达Z1的误差矩阵是：

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

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：

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

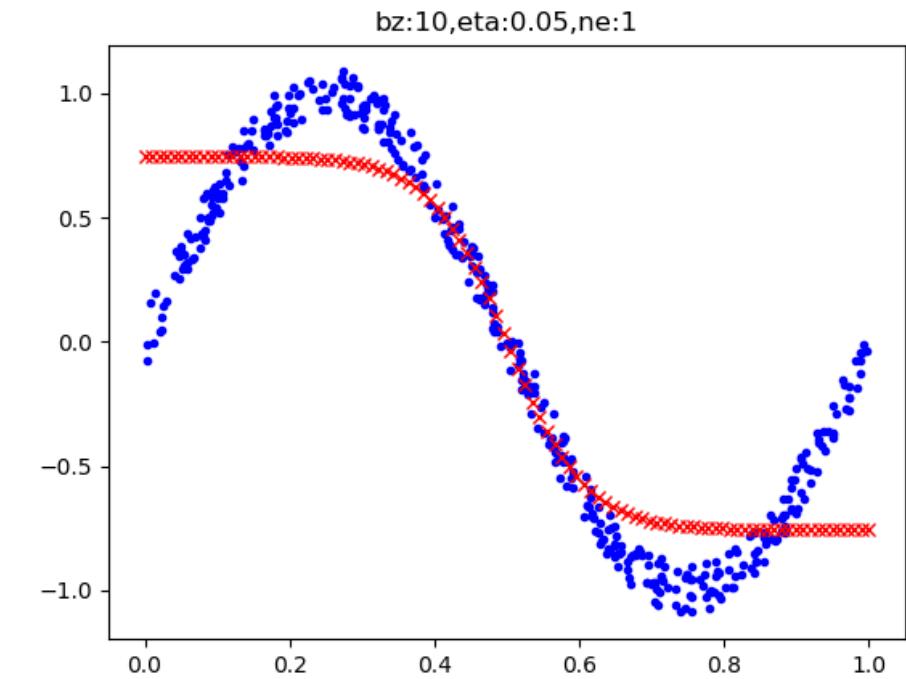
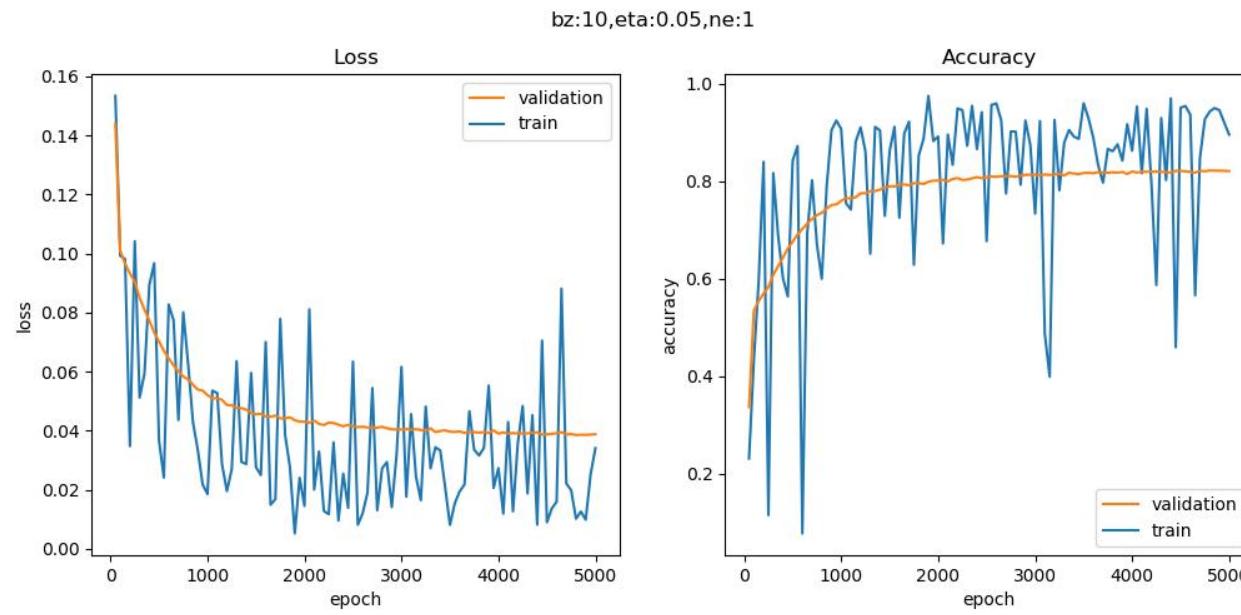
$$dB1 = dZ1\tag{12}$$



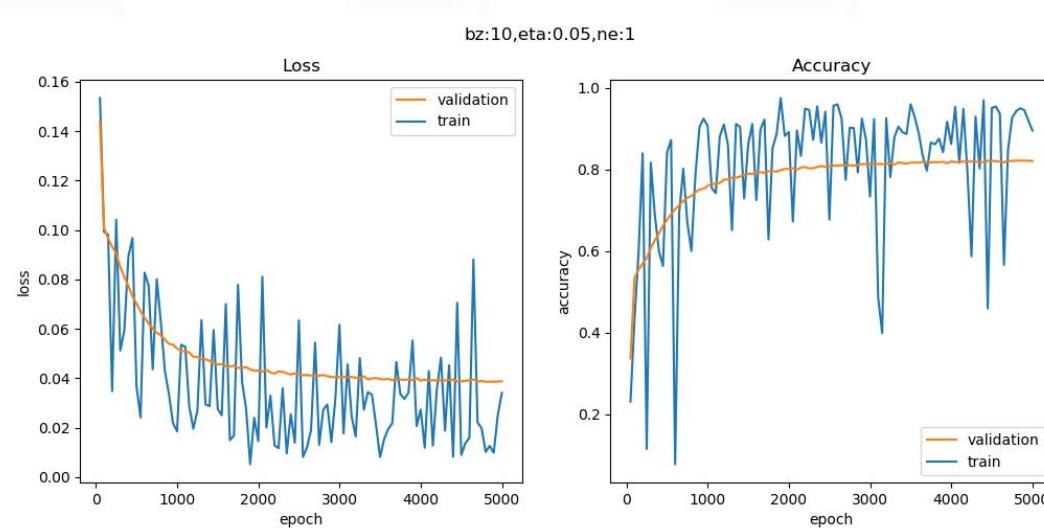
曲线拟合



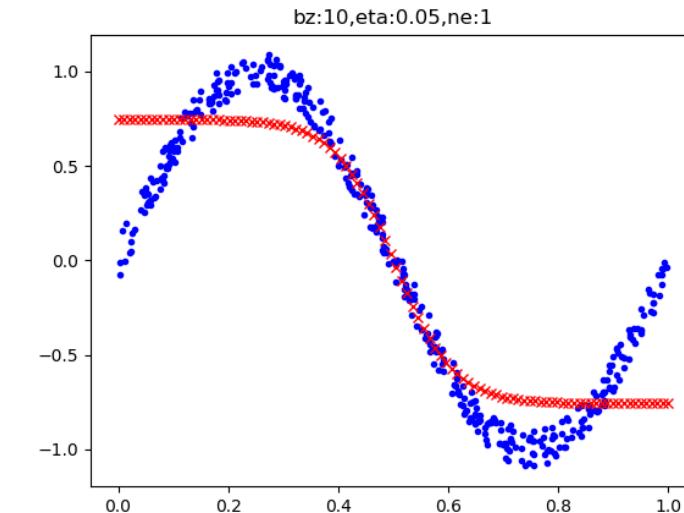
正弦曲线的拟合 - 隐层只有一个神经元的情况



正弦曲线的拟合 - 隐层只有一个神经元的情况

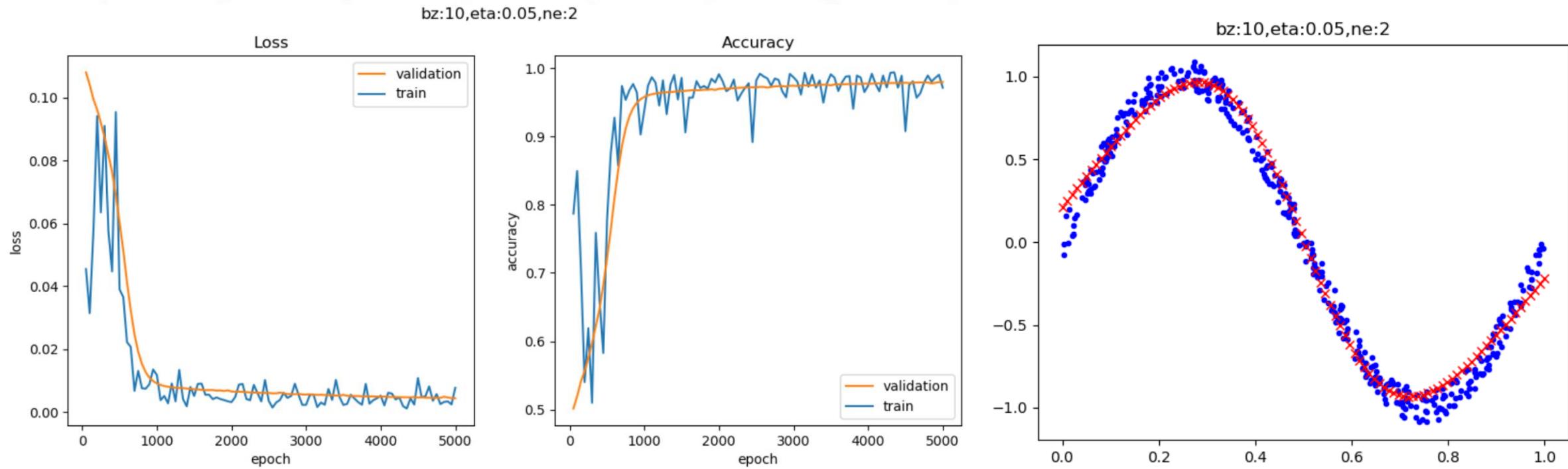


损失值到0.04附近就很难下降了



最后的测试集精度值为85.7%，不是很理想。所以隐层1个神经元是基本不能工作的，这比单层神经网络的线性拟合强一些，距离目标还差很远

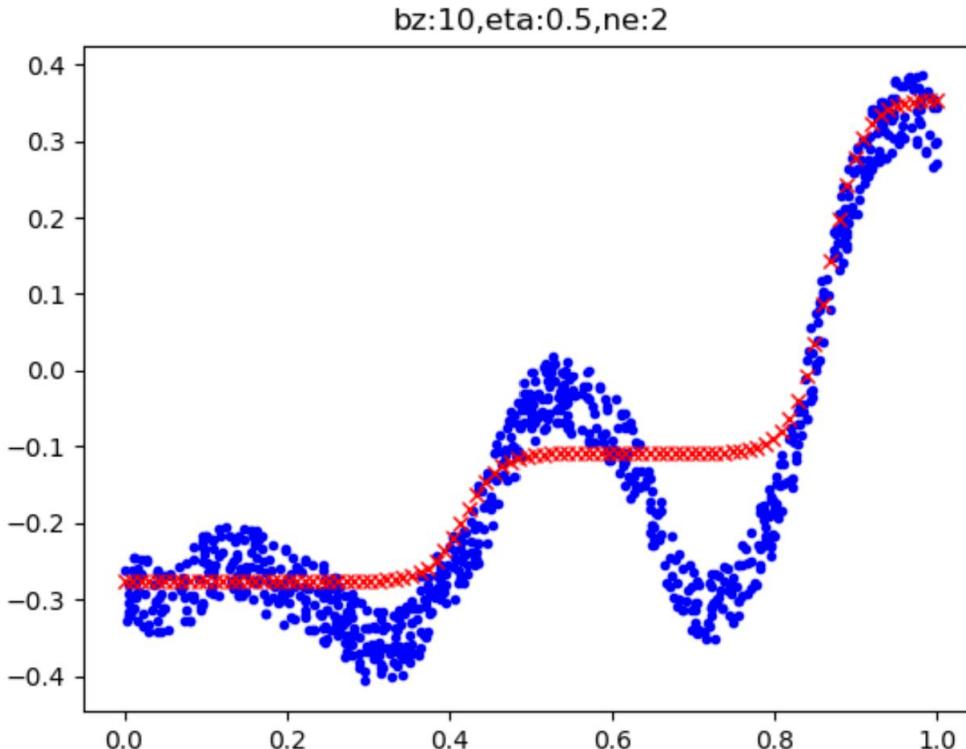
正弦曲线的拟合 - 隐层只有一个神经元的情况



损失函数曲线和验证集精度曲线，都比较正常。而2个神经元的网络损失值可以达到0.004，少一个数量级。验证集精度到82%左右，而2个神经元的网络可以达到97%

正弦曲线的拟合 - 复合函数的拟合

隐层只有两个神经元的情况



两个神经元的拟合效果图，拟合情况很不理想，和正弦曲线只用一个神经元的情况类似。

正弦曲线的拟合 - 复合函数的拟合

隐层只有三个神经元的情况

损失函数曲线和验证集精度曲线，都比较正常

隐层3个神经元
学习率=0.5
批量=10

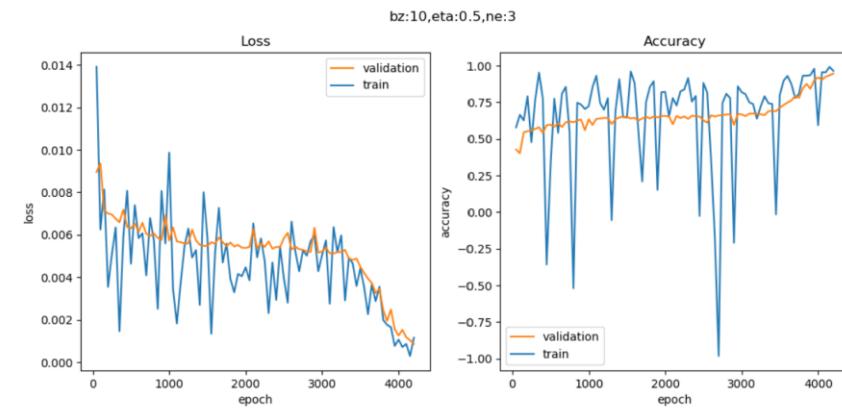
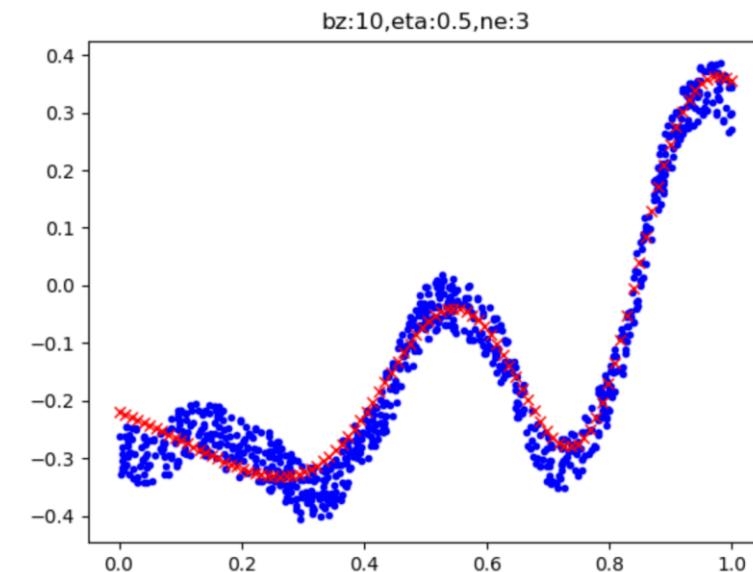


图9-15 三个神经元的训练过程中损失函数值和准确率的变化



正弦曲线的拟合 - 广义的回归/拟合

“曲线” 在这里是一个广义的概念，它不仅可以代表二维平面上的数学曲线，也可以代表工程实践中的任何拟合问题，比如房价预测问题，影响房价的自变量可以达到20个左右，显然已经超出了线性回归的范畴，此时我们可以用多层神经网络来做预测。

非线性回归的工作原理



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。

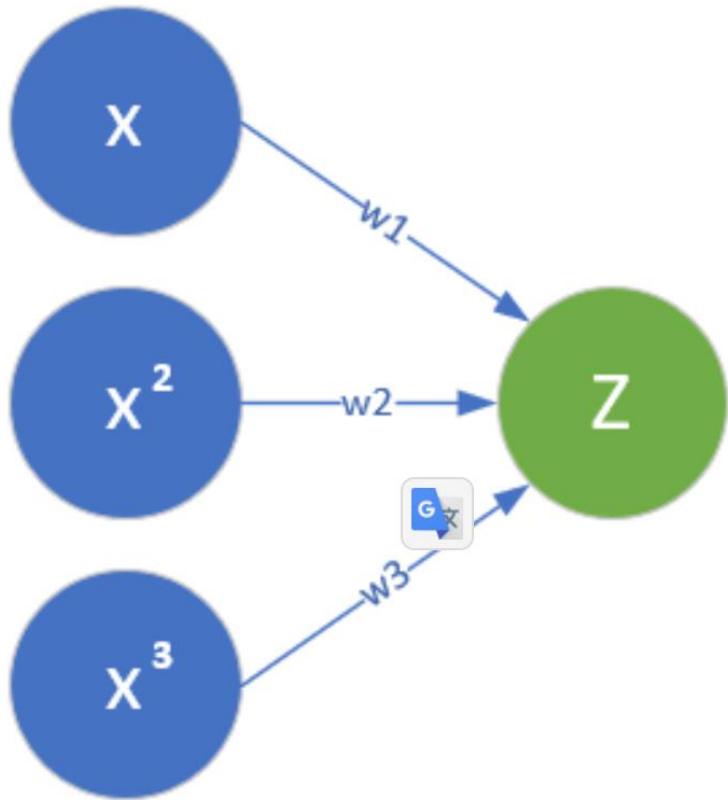
当我们使用双层神经网络时，在隐层只放置了三个神经元，就轻松解决了复合函数拟合的问题，效率高出十几倍，复杂度却降低了几倍。

多项式为何能拟合曲线？

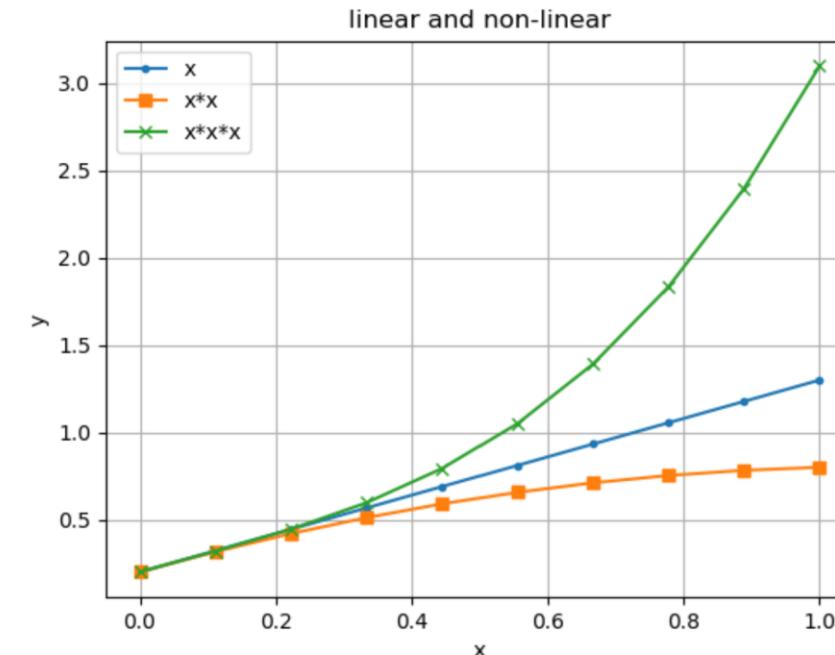
以正弦曲线拟合为例

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。

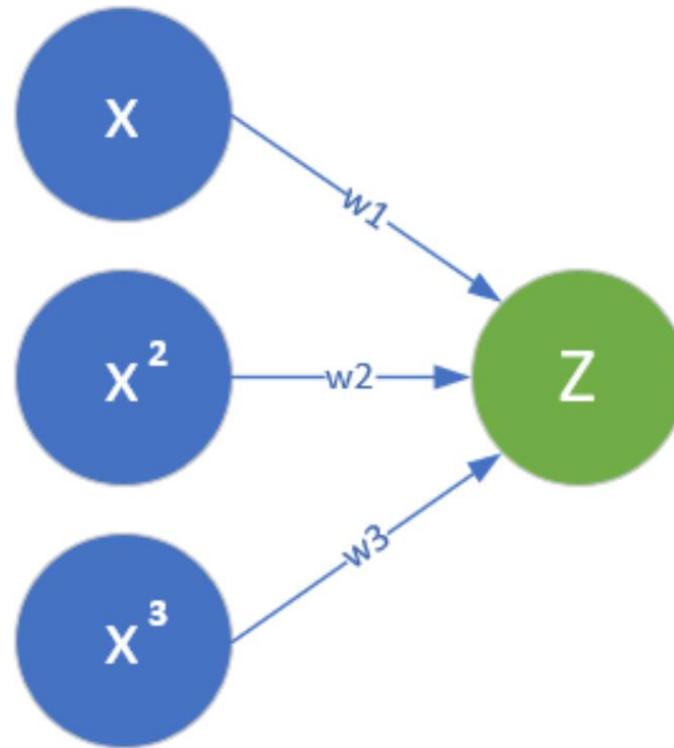


我们只使用了同一个x序列的原始值，却可以得到三种不同数值序列，这就是多项式拟合的原理。当多项式次数很高、数据样本充裕、训练足够多的时候，甚至可以拟合出非单调的曲线。

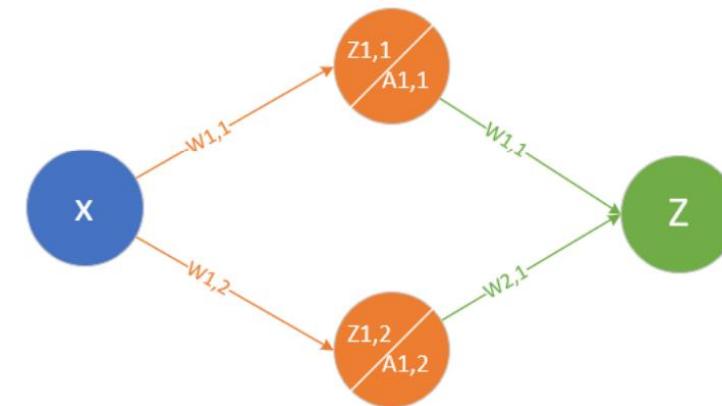
神经网络的非线性拟合工作原理

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

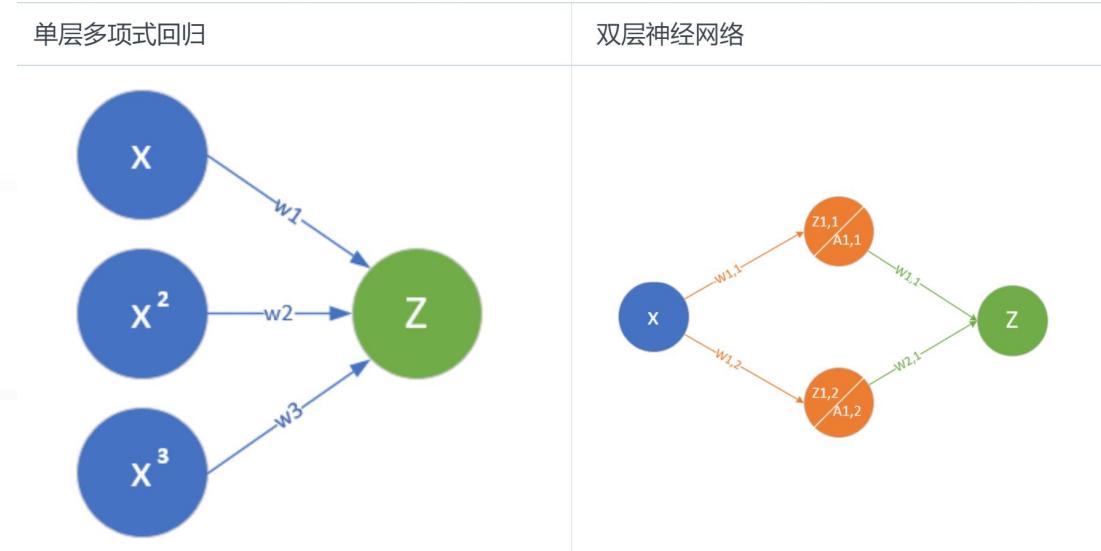
单层多项式回归



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

超参数优化的初步认识



超参数存在困难

超参数优化 (Hyperparameter Optimization) 主要存在两方面的困难：

1. 超参数优化是一个组合优化问题，无法像一般参数那样通过梯度下降方法来优化，也没有一种通用有效的优化方法。
2. 评估一组超参数配置 (Configuration) 的时间代价非常高，从而导致一些优化方法（比如演化算法）在超参数优化中难以应用。
对于超参数的设置，比较简单的方法有人工搜索、网格搜索和随机搜索。

避免权重矩阵初始化的影响

权重矩阵中的参数，是神经网络要学习的参数，所以不能称作超参数。

权重矩阵初始化是神经网络训练非常重要的环节之一，不同的初始化方法，甚至是相同的方法但不同的随机值，都会给结果带来或多或少的影响。

手动调整参数

手动调整超参数，我们必须了解超参数、训练误差、泛化误差和计算资源（内存和运行时间）之间的关系。手动调整超参数的主要目标是调整模型的有效容量以匹配任务的复杂性。有效容量受限于3个因素：

模型的表示容量；

学习算法与代价函数的匹配程度；

代价函数和训练过程正则化模型的程度。

超参数的作用

超参数	目标	作用	副作用
学习率	调至最优	低的学习率会导致收敛慢，高的学习率会导致错失最佳解	容易忽略其它参数的调整
隐层神经元数量	增加	增加数量会增加模型的表示能力	参数增多、训练时间增长
批大小	有限范围内尽量大	大批量的数据可以保持训练平稳，缩短训练时间	可能会收敛速度慢

通常的做法是，按经验设置好隐层神经元数量和批大小，并使之相对固定，然后调整学习率

网格搜索

当有3个或更少的超参数时，常见的超参数搜索方法是网格搜索（grid search）。对于每个超参数，选择一个较小的有限值集去试验。然后，这些超参数的笛卡儿乘积（所有的排列组合）得到若干组超参数，网格搜索使用每组超参数训练模型。挑选验证集误差最小的超参数作为最好的超参数组合。

用学习率和隐层神经元数量来举例，横向为学习率，取值[0.1,0.3,0.5,0.7]；纵向为隐层神经元数量，取值[2,4,8,12]，在每个组合上测试验证集的精度。我们假设其中最佳的组合精度达到0.97，学习率=0.5，神经元数=8，那么这个组合就是我们需要的模型超参，可以拿到测试集上去做最终测试了。

随机搜索

随机搜索 (Bergstra and Bengio, 2012) , 是一个替代网格搜索的方法，并且编程简单，使用更方便，能更快地收敛到超参数的良好取值。

随机搜索过程如下：

首先，我们为每个超参数定义一个边缘分布，例如，Bernoulli分布或范畴分布（分别对应着二元超参数或离散超参数），或者对数尺度上的均匀分布（对应着正实值超参数）。例如，其中， $u(a,b)$ 表示区间 (a,b) 上均匀采样的样本。类似地， $\log_number_of_hidden_units$ 可以从 $u(\log(50),\log(2000))$ 上采样。与网格搜索不同，我们不需要离散化超参数的值。这允许我们在一个更大的集合上进行搜索，而不产生额外的计算代价。实际上，当有几个超参数对性能度量没有显著影响时，随机搜索相比于网格搜索指数级地高效。

Bergstra and Bengio (2012) 进行了详细的研究并发现相比于网格搜索，随机搜索能够更快地减小验证集误差（就每个模型运行的试验数而言）。

与网格搜索一样，我们通常会重复运行不同版本的随机搜索，以基于前一次运行的结果改进下一次搜索。

随机搜索能比网格搜索更快地找到良好超参数的原因是，没有浪费的实验，不像网格搜索有时会对一个超参数的两个不同值（给定其他超参数值不变）给出相同结果。在网格搜索中，其他超参数将在这两次实验中拥有相同的值，而在随机搜索中，它们通常会具有不同的值。因此，如果这两个值的变化所对应的验证集误差没有明显区别的话，网格搜索没有必要重复两个等价的实验，而随机搜索仍然会和其他超参数进行两次独立的探索。



Reactor

Thank You!