如何理解神经网络——信息量、压缩与智能

gtj

2025 年 3 月 3 日

目录

1 从函数拟合开始

1.1 最简单的规律——简单线性回归

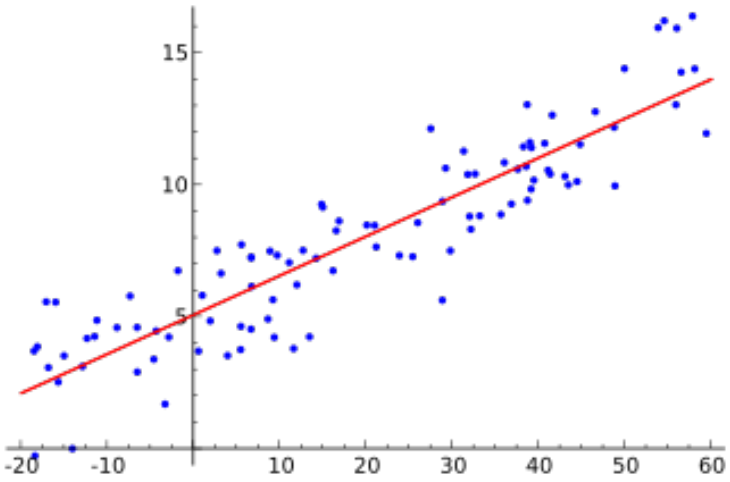


图 1: 线性回归示意图

图源: Wikipedia

虽然 Linear Regression 线性回归 的名字叫做“ Regression 回归 ”，但是事实上我更喜欢叫做 Linear Fitting 线性拟合。它的目的是找到一条直线尽可能“贴近”数据点。在这一基础上，我们可以发现数据之间的规律，从而做出一些预测。不过这里有几个问题：

- 为什么要用直线？为什么不用曲线？
- 为什么要用直线拟合数据点？这有什么用？
- “贴近”数据点的标准是什么？为什么要选择这个标准？

我认为用直线的原因无非两点：一是直线 $y = kx + b$ 简单且意义明确，又能处理不少的问题。几何上直线作为基本对象，尺子就能画出；代数上只需要加减乘除，一次函数我们也很早就学过了。而它的思想一路贯穿到了微积分的导数并延申到了线性代数。二是许多曲线的回归可以转为线性回归（见后文）。例如指数型的 $y = k\mathrm{e}^{\alpha x}$ 取对数变为 $z = \alpha x + \ln k$，又如分式型的 $y = (\alpha x + \beta)^{-1}$ 取倒数转化为 $z = \alpha x + \beta$，从而归结为线性拟合。因此带着线性拟合经验再去考虑曲线会更轻松。

至于其意义：一是找到数据的规律，二是做出预测。拟合的系数可以用于测算数据之间的关系，斜率 k 表明输出对输入的敏感程度。一个经典例子是广告投放的 Marginal Benefit 边际效益 [1]，

[1]边际效益：经济学概念，每增加单位投入，产出会增加多少单位

在一定范围内拟合收益与投入的关系，可以估算当前的边际效益，从而决定是否继续投放。而物理上，比值定义法定义的各种物理量，如电阻、电容等，最常用的测算方式都是线性拟合。例如测量电源输出的若干组电压和电流数据，并拟合出直线，斜率的绝对值是电源的内阻，同时截距顺带给出了电源的电动势，这样测得的数据就可以用于预测电源的输出情况。对我们所处的世界有定量的认识是科学的基础。可测量的数据和数学模型来描述、解释和预测自然现象是科学的基本方法，也是拟合的根本目的。

既然有了基本思路，那么如何选择“贴近”的标准呢？直接去度量一堆散点和直线的接近程度多少有点霰弹枪[2]打移动靶的感觉，但是我们总是可以计算子弹打到了几环。换言之，两个相差的部分才是关键的，**残差**（Residual）的概念由此产生。取出每个点实际值和拟合值的差，就得到了这样一个列表[3]（其中根据拟合函数 $\hat{y}_i = kx_i + b$ 计算出预测值）：

$$\mathbf{r} = [r_1, r_2, \ldots, r_n] = [y_1 - \hat{y}_1, y_2 - \hat{y}_2, \ldots, y_n - \hat{y}_n]$$

度量数据点与直线间偏差这一问题就转为了度量残差与 0 的偏差。还记得勾股定理吗？直角坐标系内一点到 0 的距离是坐标平方和的平方根，只不过这里残差列表是个 n 维的向量，度量它偏离原点的程度就是向量的**模**（Norm）。这个模越小，说明拟合的效果越好。这样我们就自然地引入了度量拟合效果的量化标准，不过实际应用中出于方便（特别是计算上的方便），通常省去开根号的一步，直接采用残差的平方和，此外还会除以样本点数得到“平均”的残差平方。习惯上称之为**均方误差**（Mean Squared Error）(MSE)：

$$\mathrm{MSE} = \frac{1}{n}|\mathbf{r}|^2 = \frac{1}{n}\sum_{i=1}^{n} r_i^2$$

在踏出下一步之前，我想这里有一点点思考的空间。例如：

- 为什么要用平方和而不是直接相加呢？

  这是因为直接相加会有正负相互抵消的可能，度量出的偏差为 0 甚至为负实在是不合理，因此至少要保证每一项都是正数。但是这又引出下一个问题。

- 为什么不用绝对值呢？绝对值也是正的啊。

  从正态分布的角度看，选用平方和自有它的道理。但是即使读者并不熟悉这些统计的背景，也可以从另一个角度理解：平方和的确是一个更好的度量方式，因为它对大的偏差更加敏感。例如一个残差为 2 的点和一个残差为 4 的点，直接绝对值相

[2]霰弹枪：一种枪，射出的子弹像雨点一样散开

[3]记号说明：对于变量，无论是一维变量还是多维变量，一律采用斜体。对于具体的数据点，视是否为向量决定使用黑体还是斜体。例如 $r = y - \hat{y}$ 表示的是方程，所以全部采用斜体。但是具体数据的残差计算，例如 $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$，是对数据点的向量运算，所以采用正体。

加的话是 6，在这里残差为 4 的点贡献了 $4/6 \approx 66.7\%$ 的偏差。而它们的平方和是 $2^2 + 4^2 = 20$，残差为 4 的点贡献提升到了 $4^2/20 = 80\%$，更加凸显出了 4 的偏差，反映了我们更“关注”这一大偏差的想法，更符合通常对“偏差”的直观认识。

- 为什么要除以样本点数 n 呢？

  一方面是为了跨数据集比较，数据集的大小通常有区别，就像买东西的重量。这正如不能光看价格不看质量就评价 5 元 2 斤的苹果贵还是 3 元 1 斤的苹果贵，因此需要一个“单位”来衡量。另一方面，看完下一个问题你就会明白其中的精妙之处。

- 这里直接把所有的残差平方加了起来，但如果有的点重要一些怎么办？

  先说明一下这样的需求并非空想，有时测量条件决定了不同点的可靠性并不相同。以一个精度 1% 的表为例，测量得到 1.00, 2.00, 3.00 时它们本身允许的误差分别是 0.01, 0.02, 0.03，而非相同。也就是说我们会觉得 1.00 的测量值从残差的大小上[4]更为可靠，这时似乎应该衡量一下点的“重要性”。如果你想说一个点很重要怎么办？直观上来讲你可能会想把它重复几遍，例如如果你很关心 r_1，你可能会想，这还不简单吗？在误差列表中把 r_1 重复 3 遍就好，就像这样：

$$\text{Refined } \mathbf{r} = [r_1, r_1, r_1, r_2, r_3, \ldots, r_n]$$

  这时再计算均方误差呢，变成了 $n+2$ 个点，一种我们设想的“改善的”(Refined)均方误差公式就变成了这样：

$$\text{Refined MSE} = \frac{1}{n+2}\left(2r_1^2 + \sum_{i=1}^{n+2} r_i^2\right)$$

  只不过这样的方式无疑有点“笨重”，再仔细想想呢？如果把 $1/(n+2)$ 乘到每一项上，就像这样：

$$\text{Refined MSE} = \frac{3}{n+2}r_1^2 + \frac{1}{n+2}r_2^2 + \cdots + \frac{1}{n+2}r_{n+2}^2$$

  再对照着上面的列表看一看，$3/(n+2)$ 不正好表明在大小为 $n+2$ 的列表中 r_1 出现了 3 此吗？频次就这样和权重(Weight)(系数)联系起来了。我们也没必要守着重复 3 次或者 5 次这种固定的规则——至少自然可没有限制重要性之间的比例刚好是整数。这样一来只需要一个权重列表就可以了，权重乘在残差平方前，这就引出了 **加权误差**(Weighted Error)，大权重表示更重要，略微改写一下公式得到：

$$\text{Weighted MSE} = \sum_{i=1}^{n} w_i r_i^2$$

[4]残差的大小：严谨的说称作 绝对误差(Absolute Error)

这里为了方便起见，假设了权重的和为 1，即 $\sum_{i=1}^{n} w_i = 1$，如果不为 1，可以先计算误差再除以权重的和。由此可以根据实际情况调整不同点的重要性，也可以看出，之前的均方误差不过是因为在 n 个数中每个残差变量都出现了 1 次，所以权重都设为了 $1/n$。在重要性可变时，加权均方误差无疑提供了一种更加“通用”的 Measurement Metrics 度量方式 。

使用的工具已经准备好了，目标也已经明确了，那么可以开始拟合了。当然，为了简单起见，这里还是只考虑无权重的情况。我们要做的是找到一组最优的 Optimal Parameter 参数值 $\hat{k}, \hat{b}$ 使得均方误差最小，从这一点可以窥见贯穿整个机器学习的核心思想——最小化 Minimize Loss 损失(误差)。形式上，公式会这么写：

$$\hat{k}, \hat{b} = \arg\min_{k,b} \mathrm{MSE} = \arg\min_{k,b} \frac{1}{n} \sum_{i=1}^{n} (y_i - kx_i - b)^2$$

但是它并没有那么神秘：arg 是 argument 的缩写[5]，min 则是 minimize 的缩写。上面的式子完全可以读作“ Find the parameter values k, b that minimize the MSE 找到参数值 $\hat{k}, \hat{b}$ 使得均方误差最小 ”。虽然项很多，但这本是上只是一个二次函数，所以无论是配方法、对 k, b 分别求导还是使用矩阵方法，都可以很容易地求解。不过我很喜欢另一个较少被人提及的视角——从线性代数和几何的角度来看待这个问题。我们回头看看残差的表达式：

$$\begin{aligned}
\mathbf{r} &= [r_1, r_2, \ldots, r_n] \\
&= [y_1 - (kx_1 + b), y_2 - (kx_2 + b), \ldots, y_n - (kx_n + b)] \\
&= [y_1, y_2, \ldots, y_n] - (k[x_1, x_2, \ldots, x_n] + b[1, 1, \ldots, 1])
\end{aligned}$$

我们暂时用一个这样的记号，记拟合所用的函数在这些数据点上的取值

$$\mathbf{x}^0 = [1, 1, \ldots, 1]$$
$$\mathbf{x}^1 = [x_1, x_2, \ldots, x_n]$$

并记输出 $\mathbf{y} = [y_1, y_2, \ldots, y_n]$，那么残差就可以写成 $\mathbf{r} = \mathbf{y} - (k\mathbf{x}^1 + b\mathbf{x}^0)$。这样一来，我们的目标是找到 k, b 使得 $\mathbf{r}$ 的模最小。写到这里，从代数上看可能依然不够直观，让我们换个角度看看。

[5]Argument: 自变量，数学优化中函数的输入变量。然而 arg min 中的 arg 仅仅表明在优化算法看来 k, b 是可变的、待优化的自变量。但是从拟合模型外看过去，它们是固定的参变量，通常意义上仍称作 Parameter。这里 Argument 与 Parameter 的区别一定程度上体现了视角的转换。

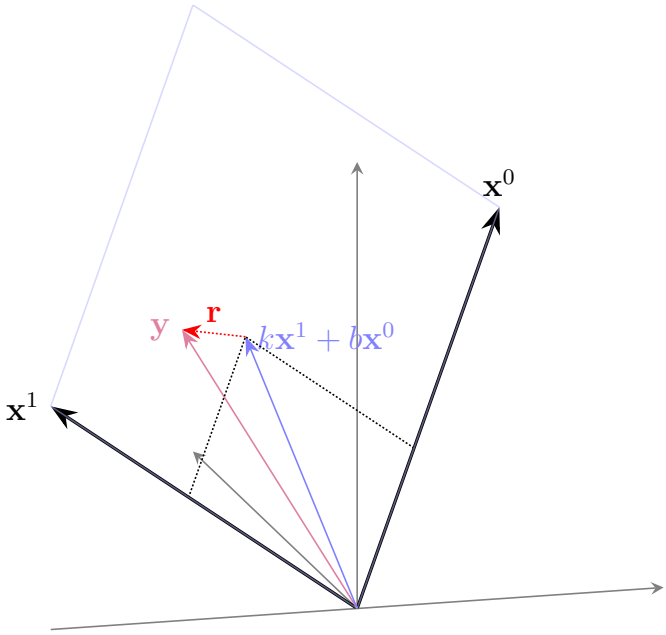


图 2: 从几何的角度看残差

从几何上，$k\mathbf{x}^1 + b\mathbf{x}^0$ 落在 $\mathbf{x}^0$ 与 $\mathbf{x}^1$ 确定的平面上，求 $\mathbf{r} = \mathbf{y} - (k\mathbf{x}^1 + b\mathbf{x}^0)$ 的最小值实际上就是从点向平面做垂线并求垂线长。平面上的点恰好表示了那些可以精准拟合的数据，而偏离平面的部分则暗示了无论怎么用直线拟合都会有误差。不得不说从几何上看确实清晰很多，事实上也有人从几何角度给出了推导，不过掠过这些细节，仅保留一个直观的印象也无大碍。本节的几篇推荐阅读中都用不同的方法解答了如何最小化误差，有详细的推导，因此这里不再赘述。但是我认为如果读者有一些基础的统计知识而且想记住线性回归推导出的结果，那么结论值得一提，不过跳过也无妨。计算出来的结论是这样的：

首先要计算的是样本中心点，对 b 的导数项为 0 推出最优的直线必然经过样本中心点 $(\bar{x}, \bar{y})$，其中

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

即均值(Mean)。

看斜率之前先看看方差(Variance)和协方差(Covariance)，方差[6]的表达式是

$$\mathrm{Var}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

[6]此注释写给学过数理统计的读者：此处并非样本方差(Sample Variance)，样本方差除以的是 $n-1$

是不是感觉很熟悉？这不就是自变量相对均值的 MSE 吗？而协方差的表达式是

$$\mathrm{Cov}(\mathbf{x},\mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})$$

它把方差中的平方项换成了 x 和 x 的 Cross Term 交叉项，并由此体现出了 Correlation 相关关系。接下来计算的是斜率 k，它的表达式是

$$\hat{k} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$$

虽然分子分母都是求和式，看起来有些复杂，但是总结起来其实就是协方差除以自变量的方差，即 $k = \mathrm{Cov}(\mathbf{x},\mathbf{y})/\mathrm{Var}(\mathbf{x})$，如果把协方差看作一种乘法[7]，那么 $k = (\mathbf{x}\cdot\mathbf{y})/(\mathbf{x}\cdot\mathbf{x})$ 看起来确实挺像那么回事的。

这样一来，通过点-斜率式方程就可以得到最优的直线，那么直线拟合就告一段落了。

推荐阅读

- 如果你想了解“回归”与“Least Squares 最小二乘”的含义：
  用人话讲明白线性回归*Linear Regression* - 化简可得的文章 - 知乎
  https://zhuanlan.zhihu.com/p/72513104
- 如果你想阅读从求导法到线性代数方法的详尽公式推理：
  非常详细的线性回归原理讲解 - 小白*Horace*的文章 - 知乎
  https://zhuanlan.zhihu.com/p/488128941
- 如果你想详细了解了线性回归中的术语、求解过程与几何诠释：
  机器学习 — 算法笔记-线性回归（*Linear Regression*） - *iamwhatiwant*的文章 - 知乎
  https://zhuanlan.zhihu.com/p/139445419

[7]此注释写给熟悉线性代数的同学：在 向量空间内积 的意义上这几乎正确

1.2 多项式拟合

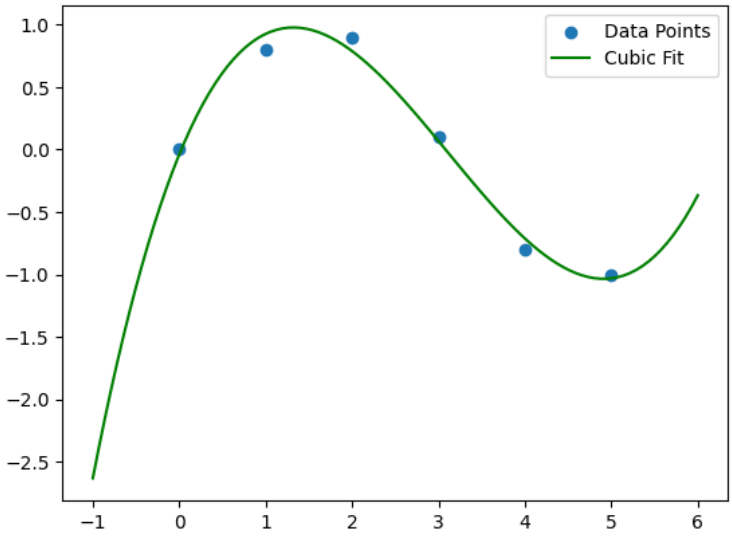


图 3: 多项式拟合示意图（图为 3 次拟合）
图源: GeeksforGeeks

线性拟合虽然很好，但是如果拿到了明显不线性的一堆数据，那么线性拟合就显得有些力不从心了。不过既然都是拟合，能做一次的那按理来讲也能做多次。 多项式拟合 (Polynomial Fitting) 就是这样一种思路，只是预测 $\hat{y}$ 从 $kx+b$ 变成了 $a_0+a_1x+\cdots+a_mx^m$ [8]，其中 m 是多项式的次数。而均方误差的表达式甚至几乎不用变，仍然是

$$\mathrm{MSE}=\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y})^2$$

只不过展开后是一系列的多项式项，待拟合的参数从两个变成了 $m+1$ 个。但是如果观察一下，这个式子仍然是一个（多变量的）二次函数，所以最小化的方法也是一样的。多项式自有多项式的好，能加的项多了，拟合的灵活性也就大了，误差显然会更小。然而与线性拟合相比，它虽然有 解析解 (Analytical Solution)，但不再像线性拟合一样可以逐项明确说出意义，而是只剩下一堆矩阵运算把这些参数算出来。因此相比于记下公式，形成一个整体上的印象显得尤为重要。

上一小节中，我们从图像看到了这种拟合的几何解释，而多项式拟合也是相似的，还是从 $\mathbf{r}$ 的表达式入手

$$\mathbf{r}=\mathbf{y}-(a_0\mathbf{x}^0+a_1\mathbf{x}^1+\cdots+a_m\mathbf{x}^m)$$

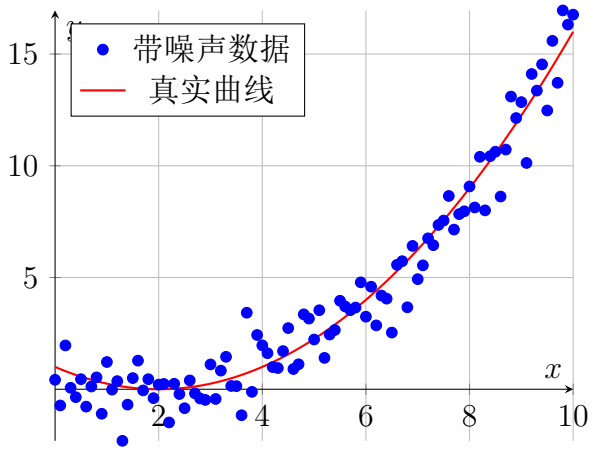
[8]记号说明：虽然习惯上幂次从大到小排列，但是为了下标和幂次的统一性，所以这里选择从常数项到最高次项排列

对比之前的表达式，当 $a_0, a_1, \ldots, a_m$ 变化时，预测得到的结果 $\hat{\mathbf{y}} = a_0\mathbf{x}^0 + a_1\mathbf{x}^1 + \cdots + a_m\mathbf{x}^m$ 也会在一个 $m+1$ 维的空间中变化，正如之前的平面，这个空间也是一个 $m+1$ 维的子空间。求最小模的 $\mathbf{r}$ 又回到了从点到子空间的垂线问题。虽然不得不承认：想象从一个高维的 n 维空间中向 $m+1$ 维的子空间做垂线确实有些困难，但是这多少离我们的几何直觉更近了一些。

系数的意义不那么明确了，但是误差下来了，这是好事吗？也不一定，灵活性的另一面是潜在的 **过拟合**（Overfitting）。前文中做线性拟合的时候有一个重要的假设是测量得到数据带有一定的误差。拟合的直线滤去了大部分的误差，留下了重要的趋势。但是如果灵活性太高，拟合的多项式会过于贴合数据，甚至把误差也拟合进去了。即使在给定的数据上做到了很小的误差，预测新数据的能力却可能会大打折扣。

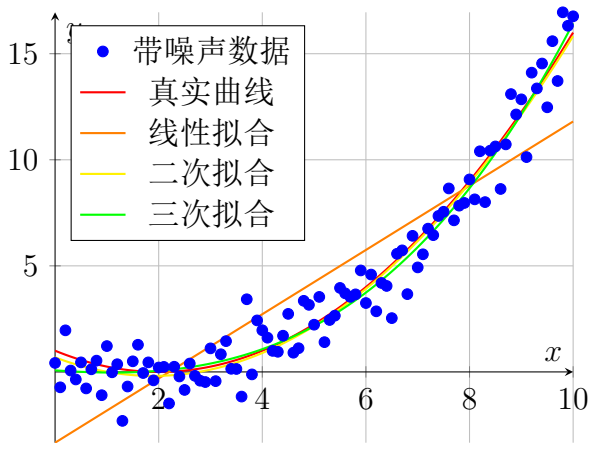
拿做题打个比方：使用直线拟合明显不线性的数据是方法错了，只能说是没完全学会。但是用接近数据量的参数来拟合数据，留给它的空间都够把结果“背下来”了，捕捉到了数据的细节，却忽略了数据背后的规律，化成了一种只知道背答案的自我感动。在几道例题上能做到滴水不漏，但是一遇到新题就束手无策。

举个例子，在下面这个数据集上试图拟合，我们在二次函数 $y = 0.25x^2 - x + 1$ 上添加了标准正态分布的噪声，即实际上 $y = 0.25x^2 - x + 1 + \mathcal{N}(0,1)$ [9]。

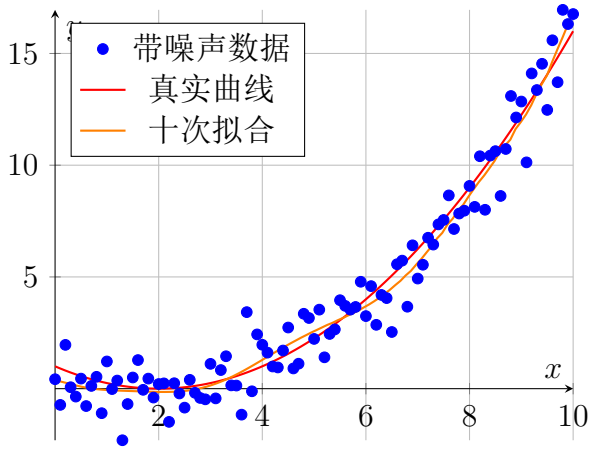


那么现在我们来试试用不同次数的多项式拟合这个数据集。不难看出线性拟合的线与数据点还是相差不少，因为它没能提供可以制造数据“弯曲”形状的项，2 次曲线的效果几乎和真实曲线一样，即使提升到 3 次也没有太明显的改变。

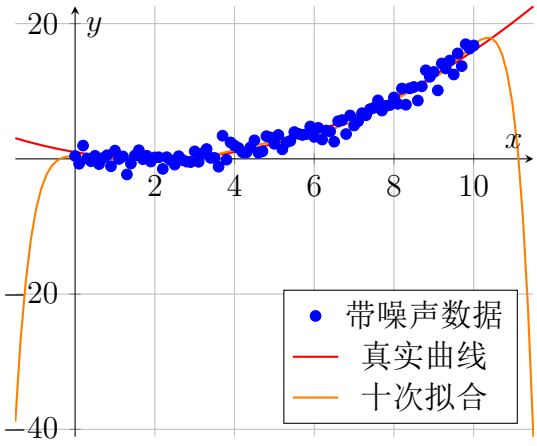
[9] $\mathcal{N}(0,1)$：表示一个服从 标准正态分布 的变量，均值为 0，方差为 1



但是如果继续增加次数呢？先来看看十次的拟合效果。



你可能会想，虽然是稍微歪了一点，不过这看起来还行吧。但是如果你把 x 的范围稍微扩大一点，你就会发现势头完全不对了。



一旦离开了拟合的区域，十次拟合的曲线就直勾勾地弯向无穷远，这是因为它把噪声

也拟合进去了，从而给出了 泛化性 (Generalization)[10]极差的结果。这就是过拟合的危害。因为参数量与样本点数量并没有非常显著的差别（10 个参数，100 个样本点），所以从去噪声的角度看，结果过拟合并不奇怪——过滤掉噪声需要更多的数据。

当然解决办法并不是没有，要解决问题先要找到问题的根源。既然得到的函数行为不符合预期，那么很自然地我们会想问，这个函数的系数怎么样呢？在上面这个具体的例子中，函数的表达式是

$$\begin{aligned}\hat{y} = &-0.000004129005667x^{10} + 0.000200033877258x^9 - 0.004061827595427x^8 \\ &+ 0.044810202155712x^7 - 0.291097682876070x^6 + 1.129425113256322x^5 \\ &- 2.542208192992861x^4 + 3.091776048493755x^3 - 1.584353162512058x^2 \\ &- 0.267618886184698x + 0.362912675589959\end{aligned}$$

简单估算一下就会发现，例如 3, 4 次项的系数都在个位数级别，再乘以 x 的 3 次方、4 次方数值就会变得很大。10 次方项的系数看起来只有 4.1×10^{-6}，但是乘上 $\mathbf{x}$ 中最大值的 10 次方，也就是 10^{10} 后，这个数值同样会飙升到上万的级别。一堆上万级别的数加在一起，倒不如说顶着舍入误差[11]还能够回归到原来的数据集上已经是奇迹了。也就只有 MSE 可以限制一下它在数据集内的行为，出了预定义的范围，这个高次函数大概就放飞自我了。

不过如果一定要用高次函数，补救的办法也不是没有。既然这些系数导致了很大的数值，那限制一下这些数值就好了，这就是 **正则化** (Regularization) 的思路。我们在待优化的函数上加上一个 惩罚项 (Penalty Term)，同样地使用平方求和的形式，只不过这次是对系数进行惩罚，为了让系数尽量小，即尽量贴近于 0，自然想到把它们的平方也加起来（再乘以一个权重），优化的目标[12]变成了

$$\text{Loss} = \text{MSE} + \underbrace{\lambda \sum_{i=1}^{10} a_i^2}_{\text{正则化项}}$$

可调参数 λ 表明我们希望在多大程度上抑制系数。然而仍然有一个致命的问题：不同系数对最终结果的影响不同。例如在 $x = 10$ 这一点上，x^{10} 项的系数对结果的影响远远大于 x 项的系数，即使是 4.1×10^{-6} 这样微小的 10 次项系数也会导致非常大的数值。平方

[10]泛化性：预测原有数据集以外点的能力

[11]舍入误差：就像手动计算时保留几位小数一样，计算机计算的并不是“实数”，而是具有一定精度的浮点数，同样也有误差。例如对于 32-bit 的浮点数，只能精确到 7 个十进制位，这意味着从万位向后数到第 7 位，从百分位就可能已经不准确，在此之后的数位就不太可靠了。

[12]Loss：损失，与前文单纯使用 MSE 时相同，我们希望让它尽可能小，它的每一项包含了我们对拟合结果的一个美好“祝愿”，MSE 项希望它误差减小，正则化项希望它系数正常。

后这一系数变得十分微小，原本用于约束系数大小的正则化项对它的影响更是微乎其微。那么怎么办呢？

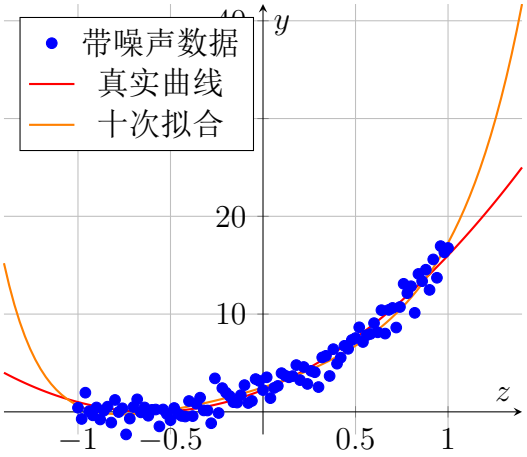
再仔细想想问题在哪里，我们发现一路下来导致问题的根源都是 x 的高次项即使在系数很小时也会导致数值爆炸。但是如果把 x 放到 $[-1,1]$ 的闭区间内呢？这样即使是 x^{10} 项，x 的 10 次方也不会超过 1，系数再怎么大，至少在小范围内也不会导致数值爆炸，这样一来正则化项才能发挥其约束作用。

操作上只需要把 $[0,10]$ 范围内的 x 线性地映射到 $[-1,1]$ 范围内。这很简单，令 $z = 0.2x - 1$ 再对 y 用 z 的多项式拟合，这种操作称作 **归一化**（Normalization）[13]。

事实上只需要一个很小的 λ 就可以在一定程度上抑制高次项的系数，这里我们取 $\lambda = 0.01$，优化的目标变为了

$$\text{Loss} = \text{MSE} + 0.01 \sum_{i=1}^{10} a_i^2$$

这时拟合出来的图像是这样的：



虽然图中拟合曲线与真值在数据集外确实仍然有显著的差距，但经过自变量归一化和参数正则化项的加入，拟合的曲线至少把数据的大体趋势成功地延伸到了数据集的一个邻域内，不至于像原本的那样惨不忍睹。

不过在实际应用中，几乎不会用到 5 次以上的多项式拟合。仍然是因为容易过拟合：就以 $[-1,1]$ 上的函数为例，x^5 与 x^7 的图像几乎是一样的，它们最大的差值仅为 0.12，这意味着如果允许的高次项太多，一点点微小的噪声就能让轻易地把五次项的系数“分给”七次项，或者反之。这导致拟合的数值稳定性很差，因此显然不太可靠。从这种影响的角度看，高次多项式拟合本身就有 **奇异性**（Singularity），解并不稳定（这种对微小噪声敏感的问题常称

[13] 归一化：调整数据到某个给定的范围内，使数据在不同场景下更加可比、更加数值稳定。前文计算误差时取平均实际上也是一种归一化。

为 病态问题 (Ill-posed Problem)。因此比正则化或者归一化更重要的是，我们应该意识到高次函数并不是万能的。当你觉得需要用到很高次的函数才能成功拟合时，不如先想想，多项式的假设真的合适吗？

我们注意到一个重要的事实：虽然拟合的参数在变化，但是拟合前仍然需要人为地设定多项式的次数，正则项（如果有的话）权重也需要人为设定。这些先验(Prior)的[14]参数通常称为 **超参数** (Hyperparameter)。如何用模型拟合固然是重要的问题，但是模型的结构，包括如何选取适当的超参数也有学问。因为它们通常不是直接从数据中学习的，而需要人为设定。有道是“学而不思则 欠拟合(Underfitting)，思而不学则过拟合”，我们只有在一定程度上了解问题的本质才能做出适当的选择。

推荐阅读

- 如果你想看更多关于多项式拟合的实战，可以阅读：
  多项式拟合的介绍与例子 - 姓甚名谁的文章 - 知乎
  https://zhuanlan.zhihu.com/p/366870301
- 如果你曾经想过拿问卷调查来做拟合，可以看看：
  理科生觉得哪些知识不知道是文科生的遗憾？ - 一只小猫咪的回答 - 知乎
  https://www.zhihu.com/question/270455074/answer/2374983755
- 这个比喻很好，同一问题下的其它回答也很有趣：
  人的大脑会不会出现“过拟合”病？ - 莲梅莉*usamimeri*的回答 - 知乎
  https://www.zhihu.com/question/625846838/answer/3250463511

[14]先验：在观测到数据之前，我们已经了解了一些数据特征。

1.3 高维的线性拟合

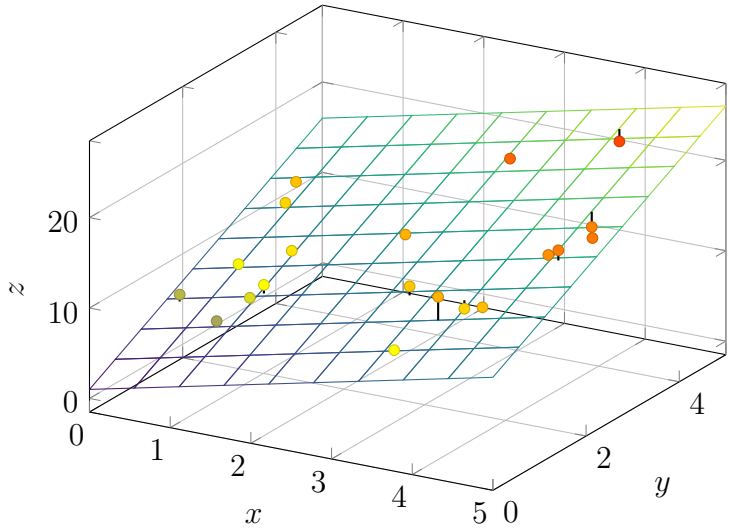


图 4: 高维线性拟合示例

第一节我们介绍了“简单线性回归”，即只有一个自变量的线性回归。但是在实际问题中，自变量往往不止一个，这时一元的线性回归就需要改成 多元线性回归 (Multiple Linear Regression)。不过按照我的习惯，文中仍然称为“拟合”。

现实世界中的数据往往是多维的，就以估计体重为例，不难发现年龄和身高就是两个可能相关的变量。如果我们想用一个模型来描述这种相关性的话，最简单的就是线性模型了，与之前的 $\hat{y} = kx + b$ 类似，自然想到用这样的函数[15]去拟合数据：

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

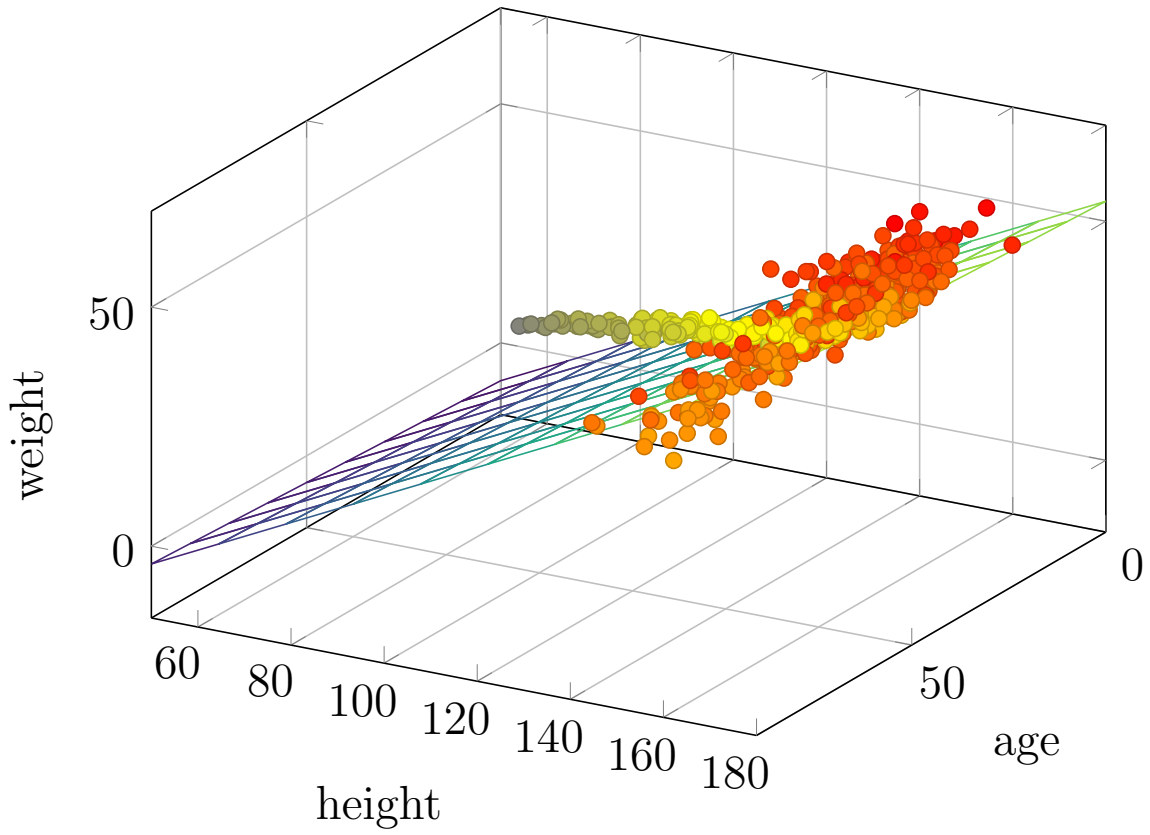
同样地，优化的目标仍然是最小化均方误差，即对 n 个数据点令

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

通过最小化误差得到 $w = [w_1, w_2, \cdots, w_d]$ 和 b 的值。这个过程与一元线性回归的过程是类似的，只不过自变量是一维时，可以在平面上直接画出拟合的直线，二维时可以在空间中画出平面，但是当维数增加到三维及以上时，拟合所用的线性函数就变为 超平面 (Hyperplane) 了，我们无法直观地看到这个超平面，但是可以猜测，它的原理差不多。

话不多说，先看看效果。这里以美国人类学家 Richard McElreath 搜集到的一个年龄、身高与体重 数据集 为例，它的分布与拟合出来的平面是这样的：

[15]记号说明：这里使用字母 w 表示 权重 (weight)，b 表示 偏置 (bias)，即常数项，d 表示的是空间的 维度 (dimension)。



通过拟合，我们可以得到一个超平面，它大致描述了数据的分布。这个超平面的方程是 $0.04676645038926784 \cdot \text{age} + 0.47766688346191755 \cdot \text{height} - 31.805656676953056 = \hat{\text{weight}}$，它比单纯使用身高或者年龄的拟合效果都要好一些。由此还可以量化地看到，年龄与身高都会影响体重，但是年龄是弱相关，而身高是强相关，这也符合我们的日常经验。

不过正如我们之前一直在做的一样，让我们看看更为直观的几何视角。仍然用 $\mathbf{x}^0$ 表示全 1 的向量，使用 $\mathbf{x}_{:1}$ 表示所有样本的第一个 Feature 特征 (分量)，$\mathbf{x}_{:2}$ 表示所有样本的第二个特征，以此类推[16]。那么多元线性拟合时的残差向量变为了[17]

$$\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - (b\mathbf{x}^0 + w_1\mathbf{x}_{:1} + w_2\mathbf{x}_{:2} + \cdots + w_d\mathbf{x}_{:d})$$

如果回顾一下我们在多项式拟合一节的内容，就会发现这和多项式时的残差向量

$$\mathbf{r} = \mathbf{y} - (a_0\mathbf{x}^0 + a_1\mathbf{x}^1 + a_2\mathbf{x}^2 + \cdots + a_m\mathbf{x}^m)$$

有着惊人的相似之处。细心的读者可能已经发现，如果令这些分量 $\mathbf{x}_{:1}, \mathbf{x}_{:2}, \ldots, \mathbf{x}_{:n}$ 分别为 $\mathbf{x}$ 的幂次组成的向量 $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^d$，那么我们得到的完完全全就是多项式拟合。这也就意味着，多项式拟合实际上可以视为多元线性拟合的一种特殊情况。

[16]记号说明：冒号表示取所有行，这是为了与 Python 中 Numpy, Torch 等库的列切片语法 $a[:, j]$ 对齐。

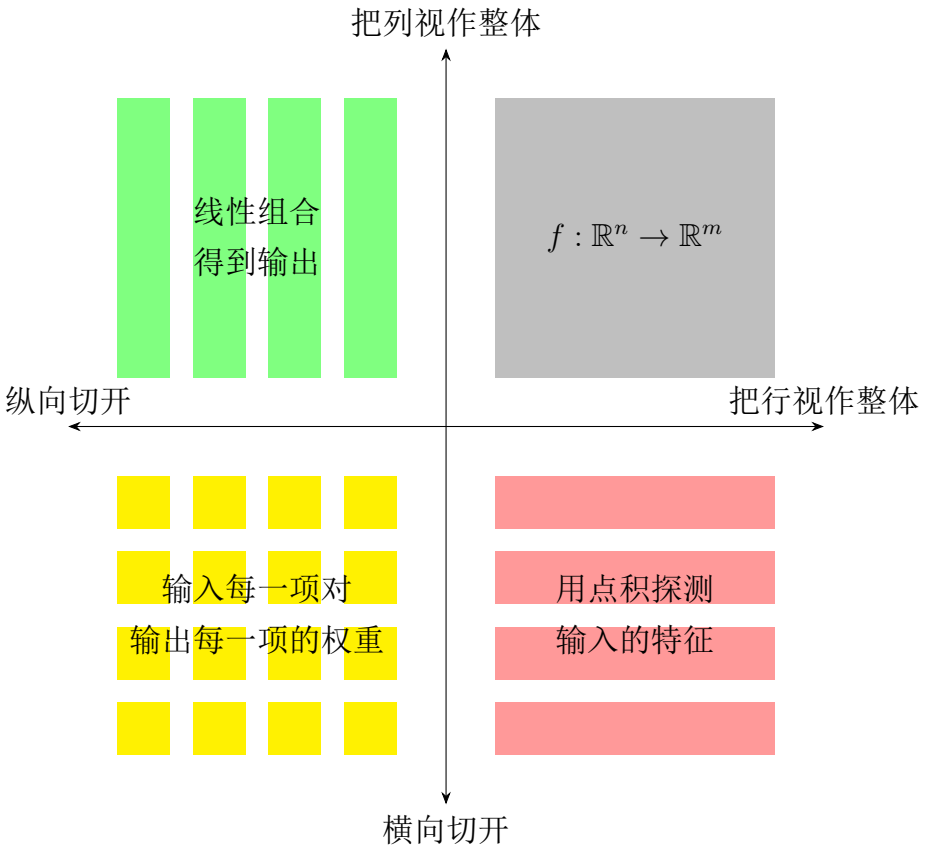
[17]记号说明：在公式中我特意将常数项放到了最前面，这是为了让它和多项式拟合的形式保持一致。

事已至此，我们似乎已经许多次遇到了这样一种情况：从一面看过去，是代数上，一组样本点上的线性拟合。但是从另一面看过去，确是在几何上找到高维空间的超平面中最接近给定点的向量。这里其实有不少精妙的数学原理[18]，但是考虑到这里的主题是机器学习，我将只带读者简要地复习(或者学习)一下线性代数，更为系统性地从几种略有差距的视角[19]体会矩阵的本质。

这里首先要感谢 The Art of Linear Algebra 这篇笔记，我初见就感到文中的插图画的非常精妙。它的思路是顺着 Gilbert Strang 教授书籍 Linear Algebra for Everyone 来的，我认为可以看成是一本矩阵图鉴，对理解矩阵运算有着极大的帮助。

3Blue1Brown 的线性代数系列 也是优质线性代数学习资源。这个制作精良的合集仅用不到两个小时的视频就清晰地从几何的角度讲明白了线性代数的基础知识，也是我入门线性代数的第一课。

矩阵有很多种 Interpretation 解读，不过我觉得大致可以按照是否把行看作一个整体以及是否把列看作一个整体来分为四类。



[18]写给数学基础好的读者：这本质上体现了代数与几何的 Duality 对偶性。

[19]几种视角： Overall 整体 解读、 Row-wise 按行 解读、 Column-wise 按列 解读、 Element-wise 按元素 解读。

矩阵究竟是什么，我们的大学教了很多年，也没有完全搞清楚。从数学的角度看，可能 Linear Algebra Done Right 的思路比较好，搞了个向量空间起手，全程以映射(Map)的逻辑贯穿。但是在国内，大部分教材一上来前两章就是讲行列式的计算，教学内容逐渐搞僵化了。

既然是服务于机器学习，许多内容[20]我们一概砍掉，只留下最为基础的内容。第一种视角就是作为 $\mathbb{R}^n \to \mathbb{R}^m$ 的线性映射。使用矩阵的第一个重要目的就是把一个线性映射“打包”成一个符号，毕竟只有这样才能方便地书写、推导和计算。从工科的视角看来，一个“向量”无非是一个数组，而一个“线性映射”实际上就是吃进去一个数组，吐出来另一个数组的机器[21]。矩阵作为一个二维数组，忠实地记录了这个机器的所有参数。它的运算规则是这样的：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}$$

从输出的表达式中，我们自然引出了行的视角。如果把矩阵看作若干行：

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}$$

我们会发现输出 y 的每一个分量 y_i 都是输入 x 与行 a_i 的点积[22]。例如

$$y_1 = a_1 \cdot x = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n$$

诚然，每一行都是一个齐次的(Homogeneous)[23]线性函数，它的形式也只能是这种 x 的加权和(Weighted Sum)，但是相信一定会有读者好奇：点积衡量了两个向量的相似程度，那么这里做点积的几何意义是什么呢？答曰：探测输入的特征。

[20] 许多内容：线性方程组的求解、行列式、基变换(Change of Basis)、特征分解(Eigen Decomposition)、二次型(Quadratic Form)。

[21] 此注释写给编程基础较好的同学：这里的机器指的就是编程中的函数(Function)。

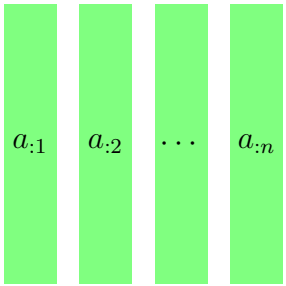
[22] 严格地说 a 是行向量，x 是列向量，这中间在数学上有一些差别，是矩阵乘法而不是点积。但是在计算机存储中，因为都是一维数组，从实用的角度并不需要纠结于此，这种记号混用(Abuse of Notation)就见怪不怪了。

[23] 齐次：指不带偏置(常数)项。

我们可以把矩阵的每一行看作一个特征检测器(Feature Detector)，它的方向表明了待检测的特征方向，与 x 的点积则说明了这个输入在这个特征上的响应强度(通常称为特征响应(Feature Response))。当 x 与特征的方向相近时响应的值为正，而当 x 与特征的方向相反时响应的值会为负，当 x 与特征几乎无关时响应的值会接近于零。这是点积的几何特性，至少从理论上为特征提取画出了一条路径。

这时如果考虑怎么样的输入可以接近预期的输出呢？根据几何解释，其实就是试图找到一个输入向量，让它尽可能通过这些特征检测器，使得每一个特征检测器的响应值与预期的响应(输出的对应分量)尽可能接近，并考虑使用均方误差来“惩罚”不接近的程度，我们一开始的代数解释便是如此。

但是如果改改输出的写法，把矩阵切成若干列，我们又有了一个不同的视角，不过这里我们用 $a_{:j}$ 表示它的列：



这样看来，矩阵的乘法也可以写成

$$Ax = \begin{bmatrix} a_{:1} & a_{:2} & \cdots & a_{:n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

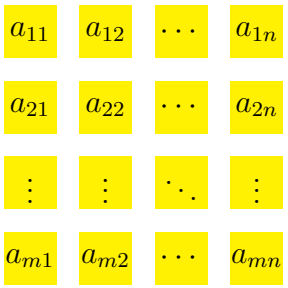
$$= x_1 a_{:1} + x_2 a_{:2} + \cdots + x_n a_{:n}$$

也就是说，输出写成了输入的线性组合。这时我们在输出的空间 $\mathbb{R}^m$ 操作，而输入的空间 $\mathbb{R}^n$ 仅仅是作为权重的载体，给出了这些列向量应该以怎么样的比例组合。

这时我们要怎么考虑用输出反推合适的输入这个问题呢？随着输入的变化，输出会变为列向量的不同组合方式，正如之前所说的，我们需要在这些列向量线性组合形成的超平面上找点，让它和预期的输出尽可能接近，这就是我们在前文提到的最小二乘的几何视角。

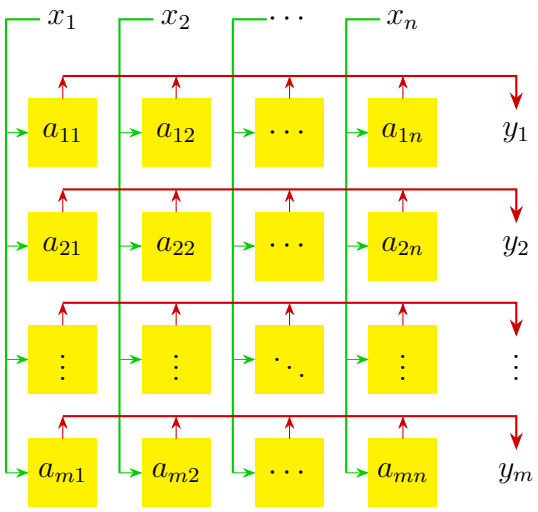
最后一种角度则带有更为浓重的解构(Distruction)[24]色彩，如果把矩阵看成一个数表，作为一个填了数字的 $m \times n$ 矩形：

[24]解构：哲学术语，通常指的是对一个结构或概念进行拆解、分析。



一般来说，我们的教材都是这么引入的，但是就像我刚才提到的一样，这种理解有着一股解构的色彩。如果没有解构后重新的 建构 (Construction) [25]，这种理解很容易让人迷失在行列式、特征值、特征向量等等复杂计算的汪洋大海中，从而忘记了矩阵的本质。那么这种解释有什么意义呢？我认为它的作用就在于 a_{ij} 体现了第 j 个输入对第 i 个输出的权重。

让我们看看下面这个示意图：



矩阵乘法可以看成这样，输入的每个分量沿着列地址线[26]输入到每一列的所有块，由每个元素乘上对应的权重后，将结果“上传”到对应的行地址线上，最后在行地址线上的所有块累积起来得到输出的每个分量。

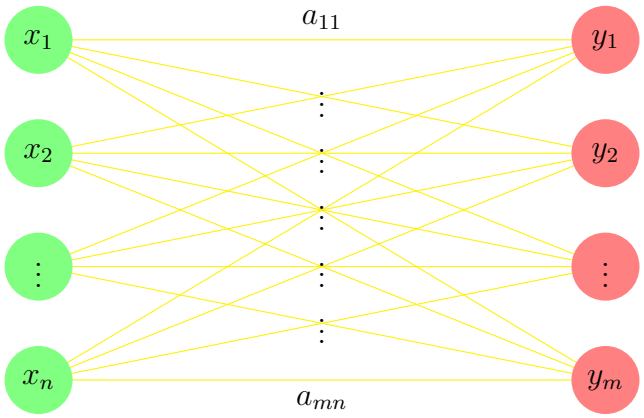
这个图还有另外一个很常见的呈现形式，把它看成一个无偏置的 线性层 (Linear Layer) [27]，我们通常是这样绘制的:

---

[25]建构：哲学术语，通常指的是对一个结构或概念进行重建、整合。

[26]地址线：计算机内存中的概念，虽然逻辑上内存是连续的，但是实际上内存寻址时有很多层，最底层时被选择的内存芯片是通过行和列寻址的，物理上由两条地址线输入 行/列地址选通信号 (Row/Column Address Strobe)。

[27]线性层：在后面章节的机器学习中会成为一个基本模块，本质上就是从输入 x 得到输出 $y = Ax + b$ 的过程，这里的无偏置即 $b = 0$。



每条从 x 到 y 的连线都代表有一个从 x 到 y 的权重，我们给 x_j 到 y_i 的连线赋予权重 a_{ij}，它就表明了输入 x_j 是如何影响输出 y_i 的。

至此我们已经从四个有差别但是又有联系的视角理解了矩阵的行为，不过我觉得我对于不同的解释还有一点观察。当降维时，特征提取(行的视角)体现的更明显，而当升维时，特征组合(列的视角)更为重要。降维伴随着对信息的压缩和精简，通过去掉不重要的部分，更接近事物的本质。升维不仅仅是增加维度，更是通过新的空间来赋予数据提供更多的可变性。在这个过程中，每一列代表了一个基向量，整个矩阵的列向量按比例组合出高维的结果。

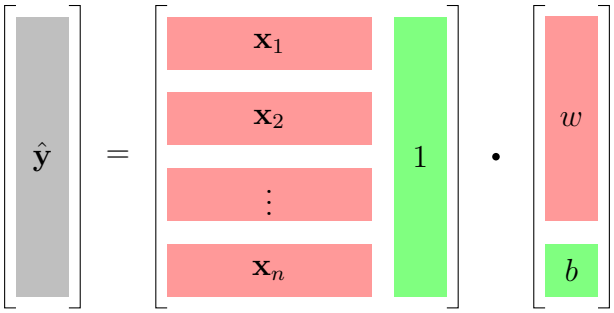
花了一些篇幅来复习线性代数，是时候回到多元线性拟合的问题上了。不过这次可以使用矩阵的语言来描述这个问题了。假设我们有 n 组 d 维的 x 的取值，它们组成了一个 $n \times d$ 的矩阵 $\mathbf{x}$[28]。我们的目标是找到一个 d 维的 w，使得数据集上 $x \cdot w$ 尽可能接近 y，现在 $\hat{y}$ 的表达式用点积的语言可以简洁地写为 $\hat{y} = x \cdot w + b$ 这种形式。

但是通过一点点技巧可以让问题更简洁，在拟合函数中，我们可以把 b 合并到 w 中，也就是

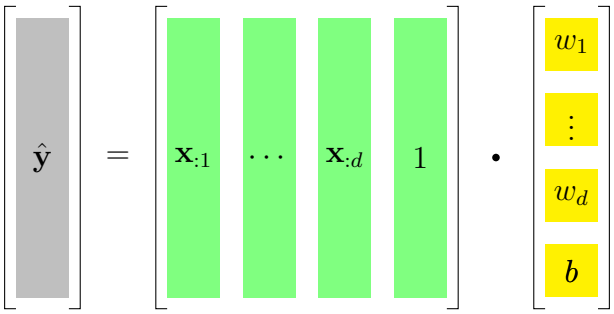
$$\begin{aligned}\hat{y} &= x \cdot w + b \\ &= x_1 w_1 + x_2 w_2 + \cdots + x_d w_d + b \\ &= \begin{bmatrix} x_1 & x_2 & \cdots & x_d & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{bmatrix}\end{aligned}$$

[28]记号说明：使用大写字母表示矩阵的比较多，但是这里为了美观和符号的一致性，我们仍然采用黑体小写字母表示所有数据点的集合。$\mathbf{x}_i$ 表示第 i 个数据点，$\mathbf{x}_{:j}$ 表示所有数据点的第 j 个分量，x_{ij} 表示第 i 个数据点的第 j 个分量，因为是标量，所以采取小写。而在方程 $x \cdot w$ 中，x, w 并非数据点的集合，而是变量，故虽然为向量，但是采用斜体。

对于样本，把所有的行并起来就变成了不需要额外添加偏置的 $\hat{\mathbf{y}} = \tilde{\mathbf{x}}\tilde{w}$。如果要说这个 $\tilde{\mathbf{x}}$ 是什么，它就是把 $\mathbf{x}$ 的每一行拼上一个 1：



但是如果改成用列的视角来看待矩阵 $\tilde{\mathbf{x}}$，我们会发现问题变成了用 $\mathbf{x}_{:1}, \mathbf{x}_{:2}, \cdots, \mathbf{x}_{:d}$ 与 $\mathbf{x}^0$ 的线性组合来贴近 $\mathbf{y}$。



诶？这不是函数拟合问题吗？如果把 $\mathbf{y}$ 和 $\mathbf{x}_{:j}$ 看作是关于 i 的函数（这里函数定义在 $\{1, 2, \cdots, n\}$ 上，换言之，记 $\mathbf{y}(1) = y_1, \mathbf{y}(2) = y_2, \cdots, \mathbf{y}(n) = y_n$，同理，设 $\mathbf{x}_{:j}(i) = x_{ij}$。那么我们的问题就是用 d 个函数 $\mathbf{x}_{:1}(i) \sim \mathbf{x}_{:d}(i)$ 与一个常值函数 $\mathbf{x}^0(i) \equiv 1$ 来拟合一个给定的函数 $\mathbf{y}(i)$。

去掉常数项（或者把常数项也当成一个 $\mathbf{x}_{:d+1}$ ）并把自变量 i 当成一元函数拟合问题中的 x 就可以发现：这与使用给定函数 $f_1(x) \sim f_d(x)$ 的线性组合，在若干数据点上拟合给定函数 $f(x)$ 这一问题没有任何差别。由此可见，用给定的函数集来拟合一个函数本质上与多元线性拟合有着完全相同的数学结构。

至此，我们至少已经初步理解了线性拟合。在这一章的最后，让我们做个总结。

- 最开始我们引入了一元的线性拟合，学会了最小二乘法与最小化损失以优化参数的基本思想，也学会了如何给数据加权。
- 接下来来到了多项式拟合，虽然是曲线，但是如果把 x 的方幂看作线性独立的分量，这仍然可以看作是一种线性拟合，在这里我们领略到了过拟合的危害，也理解了为什么要使用正则化与归一化方法。

- 在过拟合中，我们得到的更重要的启示是要意识到高次并不意味着万能，合适的才是最好。如果参数足以存下所有的数据，那么大概率就会过拟合。
- 最后我们引入了多元线性拟合，通过矩阵的不同解读，我们理解了特征提取与特征重组的逻辑。
- 从矩阵的行、列解读中，我们意识到多元线性拟合与函数的线性拟合本质上是一样的，这里有一种精妙的对应关系。

推荐阅读

- The Art of Linear Algebra 这篇文章非常好，但是如果你上不去 GitHub，进这个知乎回答看也行：
  如何快速理解线性代数？ - 丿小奇迹丨的回答 - 知乎
  https://www.zhihu.com/question/30726396/answer/3124578647
- 这篇文章推荐给数理统计与线性代数都学的较好的读者：
  回归分析—笔记整理（6)——多元线性回归（上） - 学弱猹的文章 - 知乎
  https://zhuanlan.zhihu.com/p/48541799

10 杂谈

10.1 矩阵式研究——场景与模型的排列组合

10.2 AI圈的常见行话

10.3 AI——生产力还是毁灭？

10.4 新人类与自由意志？