机器学习讲义(L15-A)：深度学习基础

授课教师：王贝伦 / 助教：张嘉琦，黄旭，谈笑，徐浩卿

1 神经元（Neurons）

1.1 神经元的结构

神经网络是一种模拟人脑的神经网络以期能够实现类人工智能的机器学习技术。人脑中神经元结构如下图，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。

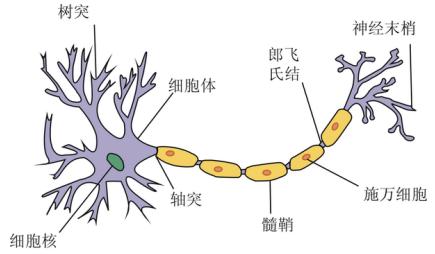

Figure 1: 人类神经元结构

科学家们参考了生物神经元的结构，创造了神经元模型，神经元模型是一个包含输入，输出与计算功能的模型。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。

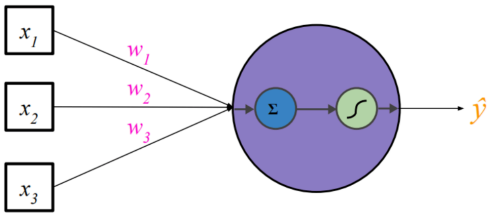

Figure 2: 神经元结构

如图所示，这个神经元模型由 3 个输入、2 个计算（这里为求和函数和激活函数中的 Sigmoid 函数）和 1 个输出组成，而在输入指向求和函数的连接上的 w 即是权值，每个连接都对应一个权值。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。

我们以 z 来表示求和的结果，sigmoid 作为激活函数，那么这个神经元模型的计算过程为：

$$z = \sum_{i=1}^{3} w_i x_i \tag{1}$$

$$\hat{y} = sigmoid(z) = \frac{e^z}{1+e^z} \tag{2}$$

图二中展示的是略去了截距的神经元结构，如果添加截距（bias），那么 (1) 式会变作 $z = \sum_{i=1}^{3} w_i x_i + b$，截距的作用同样是为了提高网络的预测效果。

而在图二中被圈起来的部分即可认为是一个神经元，接受外界的输入后进行线性计算和非线性转换，再向外界输出结果。

1.2 神经元与感知机、逻辑回归

早在 L08 的讲义中，我们就已经介绍过与感知机（Perceptron）有关的内容，感知机可被视作最简单的神经网络，仅由一个神经元模型组成，感知机的表达式 $f(x) = sign(w \cdot x + b)$，其中 $sign$ 符号函数就是最早的激活函数。

使用 Sigmoid 函数作为神经元里的激活函数，能将任意的实数值映射到 (0,1) 区间，Sigmoid 函数给神经元的输出赋予了概率的意义，这使得模型的理论基础更加扎实，也使得模型能被用于解决二元分类问题，比如当 sigmoid 神经元的输出大于 0.5 时，则预测类别为 1，否则预测类别为 0。值得注意的是，在这种情况下，sigmoid 神经元其实就是二元逻辑回归模型。

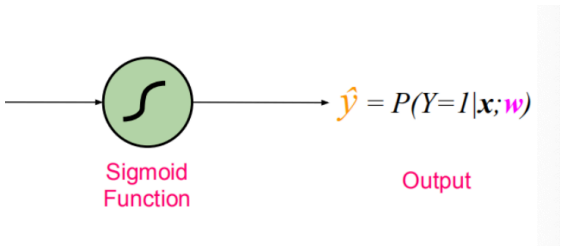

Figure 3: 概率意义

2 神经网络（Neuron Network）

神经网络则是多个神经元的搭建。在早期提出神经网络时，还没有提出隐藏层，那时的神经网络为单层神经网络，如图 4 所示。

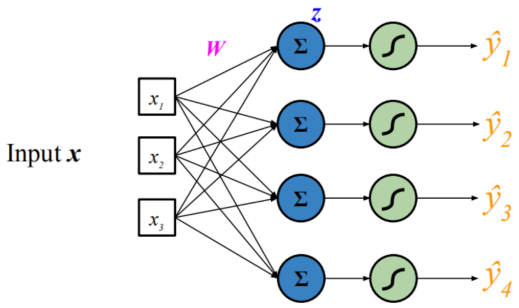

Figure 4: 单层神经网络

这时候的神经网络能解决的问题仅限于一些简单的线性问题，后来随着两层神经网络的提出，神经网络开始逐步能解决更加复杂的问题，能解决一些非线性分类问题。

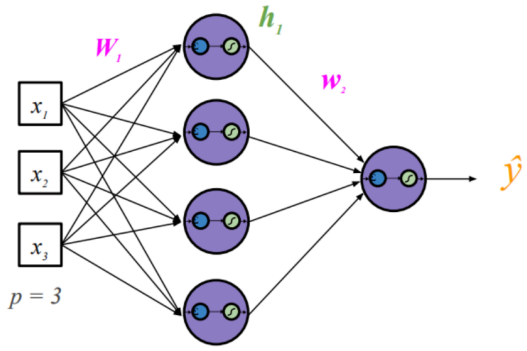

Figure 5: 两层神经网络

如图五所示，第一层神经元的的输出变成了第二层神经元的输入，我们不妨设第 i 层神经元的求和结果为 z_i，那么则有

$$\boldsymbol{z}_1 = \boldsymbol{W}_1^T \boldsymbol{x},\ \boldsymbol{h}_1 = sigmoid(\boldsymbol{z}_1) \tag{3}$$

$$z_2 = \boldsymbol{W}_2^T \boldsymbol{h}_1,\ \hat{y} = sigmoid(z_2) \tag{4}$$

在这种情况下，原来的第一层神经元被称为隐藏层，第二层则被称为输出层。

我们按照这样的方式，可以继续添加隐藏层，而隐藏层大于 1 的神经网络也就是我们耳熟能详多层神经网络，如图 6 所示。

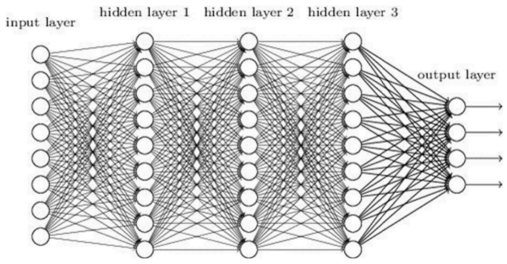

Figure 6: 多层神经网络

增加更多的层次有什么好处？更深入的表示特征，以及更强的函数模拟能力。更深入的表示特征可以这样理解，随着网络的层数增加，每一层对于前一层次的抽象表示更深入。在神经网络中，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的是由“边缘”组成的“形状”的特征，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。

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。

通过研究发现，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。

3 损失函数

我们在学习任何模型的时候，都需要一个指标去衡量模型的好坏，神经网络也不例外。

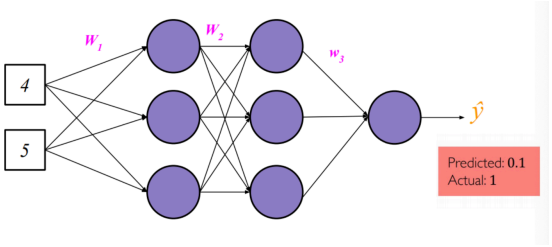

Figure 7: 预测值与实际值

在这里我们介绍三种损失函数作为衡量神经网络模型的指标。

3.1 二值交叉熵损失（Binary Cross Entropy Loss）

在二分类问题模型，真实样本的标签为 [0，1]，分别表示负类和正类。模型的最后通常会经过一个 Sigmoid 函数，输出一个概率值，这个概率值反映了预测为正类的可能性：概率越大，可能性越大。预测输出即 Sigmoid 函数的输出表征了当前样本标签为 1 的概率：为 $\hat{y} = P(y = 1|x)$，很明显，当前样本标签为 0 的概率就可以表达为：$1 - \hat{y} = P(y = 0|x)$，我们把两种情况整合在一起，即

$$P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y} \tag{5}$$

我们当然是希望这个概率越大越好，我们引入 log 函数，它并不会改变函数本身的单调性，$\log P(y|x)$ 越大越好，即它的相反数越小越好，所以损失函数为

$$L = -\log P(y|x) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \tag{6}$$

就是二值交叉熵损失函数的表达式。

3.2 交叉熵损失（Cross Entropy Loss）

而对于交叉熵损失函数来说，其数学原理与二值交叉熵损失函数基本相似，但更适用于多分类问题，这里用的并不是 Sigmoid 函数，而是 Softmax 函数，每个输出表达该样本属于某一类的概率，共有三类，Softmax 函数的输出表征了当前样本标签为 1 的概率（即预测值与真实值一致的概率）：

$$\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}} = P(y_i = 1|x) \tag{7}$$

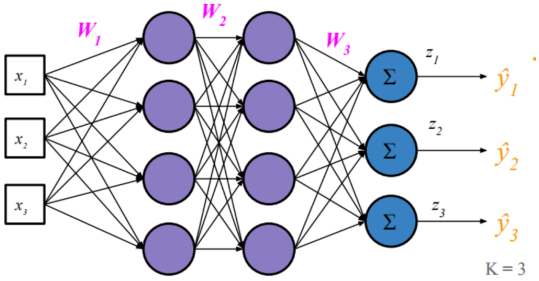

Figure 8: 多分类的神经网络

我们可以类比二值交叉熵损失函数，写出交叉熵损失函数的表达式：

$$L = -\sum_j y_j \log \hat{y}_j \tag{8}$$

其中 y_j 为指示变量，如果该类别与样本类别相同就是 1，否则就是 0。

3.3 均方差损失（Mean Squared Error Loss）

这是比较常见的损失函数，其定义为：

$$L = \frac{1}{n}\sum_{i}^{n}(\hat{y}_i - y_i)^2 \tag{9}$$

4 反向传播（Backpropagation）

得到预测结果的过程是一个前向传播的过程，在我们一开始生成神经网络的过程中，每个连接上的权值是随机生成的，我们需要最小化我们的损失函数。
前向传递输入数据直至输出产生误差，反向传播误差信息更新权重矩阵，我们需要将信息反向传播回整个神经网络，以得到最优的全局参数矩阵。

我们通过一个简单的例子来解释怎么利用反向传播来计算多层神经网络中的梯度。如图9所示，我们现在有一个简单的网络，输出为 $f(x,y,z)=(x+y)z$。当 $x=-2, y=5, z=-4$ 时，我们想要计算函数 $f(x,y,z)$ 分别关于 x,y,z 的导数。

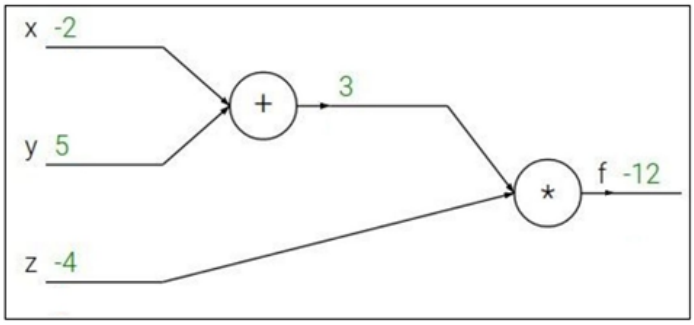

Figure 9: 简单神经网络示例。

对于这样的情况，通常，我们会用链式求导法则来计算梯度。

定义：链式求导法则

对于一个函数 $f(g(x))$，它关于 x 的导数为

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x} \tag{10}$$

在图9展示的例子中，我们假设 $q=x+y$，我们能得到 $f=qz$。对于 q 我们有

$$\begin{aligned} \frac{\partial q}{\partial x} &= 1 \\ \frac{\partial q}{\partial y} &= 1 \end{aligned} \tag{11}$$

因此，根据链式求导法则，我们可以得到

$$\begin{aligned} \frac{\partial f}{\partial x} &= \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z \\ \frac{\partial f}{\partial y} &= \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = z \\ \frac{\partial f}{\partial z} &= q = x+y \end{aligned} \tag{12}$$

将 x,y,z 的值代入就能得到我们需要的导数值。可以看到，在计算导数值时，我们从网络的输出 f 利用链式法则倒推回网络的输出，逐层计算导数值。这就是为什么这种方法被称为**反向传播**。

我们现在用一个深度学习中常见的结构来解释反向传播在更复杂的情况下的使用。如图10所示，我们现在用一个具有三层结构的网络来完成二分类问题。网络的输入层接受有三个变量 $x=[x_1,x_2,x_3]^T$ 的样本，中间层有四个变量 $h=[h_1,h_2,h_3,h_4]^T$，最终的输出是样本 x 属于类别 1（$y=1$）的概率。特别地，输入层和中间层之间的系数用 W_1 表示而中间层与输出层之间的系数用 W_2 来表示。

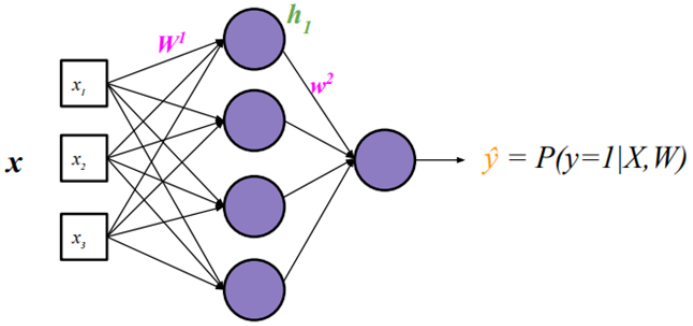

Figure 10: 三层神经网络示例。

如图11所示，在神经网络中，输入的数据 x 先与系数 W_1 相乘得到中间层的输入 $W_1^T x$，然后经过 sigmoid 函数映射到 $[0,1]$ 之间。这时得到的 sigmoid$(W_1^T x)$ 是中间层的输出，会输入到最后的输出层来计算最终网络输出。网络输出的运算方式与前一步类似。

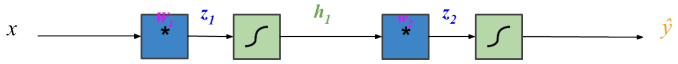

Figure 11: 三层神经网络运算流程。

经过运算之后，对于一个样本 x，网络会预测出它属于类别 1 的概率 $\hat{y}$。利用 $\hat{y}$ 与 y 计算出预测损失 $\mathcal{L}$ 之后，我们就可以利用梯度下降来更新网络中的系数 W_1, W_2。

$$\begin{aligned} W_1^{(t+1)} &= W_1^{(t)} - \eta\frac{\partial \mathcal{L}}{W_1^{(t)}} \\ W_2^{(t+2)} &= W_2^{(t)} - \eta\frac{\partial \mathcal{L}}{W_2^{(t)}} \end{aligned} \tag{13}$$

其中 t 表示迭代的代数。现在我们的重点就变成了怎么求 $\frac{\partial \mathcal{L}}{W_1^{(t)}}$ 与 $\frac{\partial \mathcal{L}}{W_2^{(t)}}$。这里我们以求解关于 W_2 的导数为例。假设我们选择交叉熵作为损失函数，网络中的所有变量可以表示为：

$$\begin{aligned} \mathcal{L} &= f_5 = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y}) \\ \hat{y} &= f_4 = \frac{e^{z_2}}{1+e^{z_2}} \\ z_2 &= f_3 = W_2^T h_1 \\ h_1 &= f_2 = \frac{e^{z_1}}{1+e^{z_1}} \\ z_1 &= f_1 = W_1^T x \end{aligned} \tag{14}$$

根据上述的式子，预测的损失可以表示为

$$\mathcal{L} = f_5(f_4(f_3(f_2(f_1(x))))) \tag{15}$$

下面我们根据链式法则来对 W_2 求导。

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W_2} &= \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_2} \\ &= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_2} \\ &= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot \left[\frac{e^{z_2}}{1+e^{z_2}}\left(1-\frac{e^{z_2}}{1+e^{z_2}}\right)\right] \cdot \frac{\partial z_2}{\partial W_2} \\ &= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot \left[\frac{e^{z_2}}{1+e^{z_2}}\left(1-\frac{e^{z_2}}{1+e^{z_2}}\right)\right] \cdot h_1 \\ &= (\hat{y}-y)h_1 \end{aligned} \tag{16}$$

最后一步是代入 $\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$ 化简得到的。计算 $\frac{\partial \mathcal{L}}{\partial W_1}$ 的过程与上述过程是类似的。

4.1 神经网络训练过程

这里，我们总结一下神经网络的训练过程：

1. 初始化所有系数（比如 W_1, W_2）
2. 将样本划分成多个批次（batch），对于每一批的样本：
   - 从输入到输出，“前向”运算，计算预测输出 $\hat{y}$ 和损失 $\mathcal{L}$
   - “反向”计算损失关于所有系数的梯度
   - 用随机梯度下降（或其他算法）更新系数值
3. 重复第 2 步直到收敛

5 构建深度神经网络

5.1 神经网络中的块结构（block-view）

由于反向传播（链式求导法则）的存在，我们可以把一个神经网络看成不同块的组合。如图12所示，在反向传播计算导数的过程中，某一块的导数是可以单独计算后在合并入整个计算链中的。这让我们能够任意更换神经网络中的一块（即图中的 f）来达到不同需求。从这个角度来看，构建一个深度神经网络就像是在搭积木。我们选择需要的块来搭建出所需的结构。

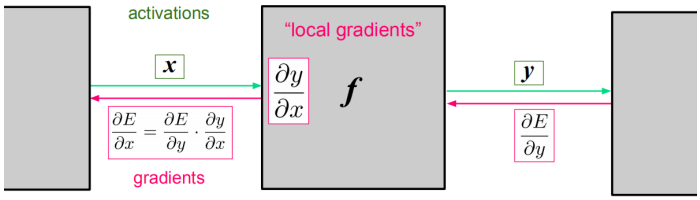

Figure 12: 神经网络的块结构。

图13是一种用来完成物体分类任务的名为 GoogleNet 的模型。可以看到，使用不同类型的块，我们可以构建出功能不同的神经网络模型。

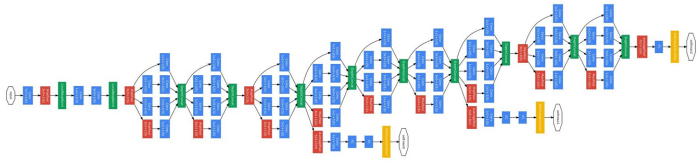

Figure 13: 用于物体分类的 GoogleNet 模型。

5.2 深度学习框架

在 Python 语言社区中，已经存在有很多深度学习的框架（图14）。这些框架提供基本的模块，能够自动完成反向传播的梯度计算，从而帮助你快速地构建所需的深度学习模型。

Figure 14: 深度学习框架。

这些框架中，Tensorflow 和 Pytorch 是比较常用的几种框架。Tensorflow 是工业界最常用的深度学习框架，它同时提供了可视化工具来方便用户随时观测训练过程。但是 Tensorflow 的问题在与过于复杂，接口设计比较抽象难懂。反观 Pytorch，这是一种学界很常用的框架。它简洁易用，方便研究人员去观察网络低层次的运算。但是由于 Pytorch 在工业界很少使用，Pytorch 训练出的模型难以真正在实际应用中进行部署。在选择框架时，我们可以根据具体需要来选择不同的框架。

6 激活函数

在之前的内容中，我们频繁地使用了 sigmoid 函数

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{17}$$

我们在每一个神经元的运算完成之后会额外使用一个 sigmoid 函数来得到神经元输出，比如 $\sigma(W_1^T x)$。这类使用在网络神经元上用以计算神经元输出的函数被统称为激活函数（activation function）。注意到激活函数是非线性的函数，引入激活函数的目的就是为网络引入非线性，从而使得网络能解决更广泛的问题。

激活函数的类型有很多种。图15展示了几种激活函数。从实际经验来看，ReLU 函数能达到的效果最好。因此在实际应用中很多深度学习模型会将 ReLU 函数而非 sigmoid 函数作为激活函数。

Name	Plot	Equation	Derivative (w.r.t x)
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Rectifier (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

Figure 15: 激活函数

7 神经网络的优缺点总结

神经网络与我们之前介绍的传统的机器学习算法相比，最大的优势在于神经网络的性能优于传统方法。并且数据量越多，神经网络的性能会更好。在大数据时代中，因为我们拥有海量数据以及高运算性能的机器，神经网络的使用就变得广泛且必要。

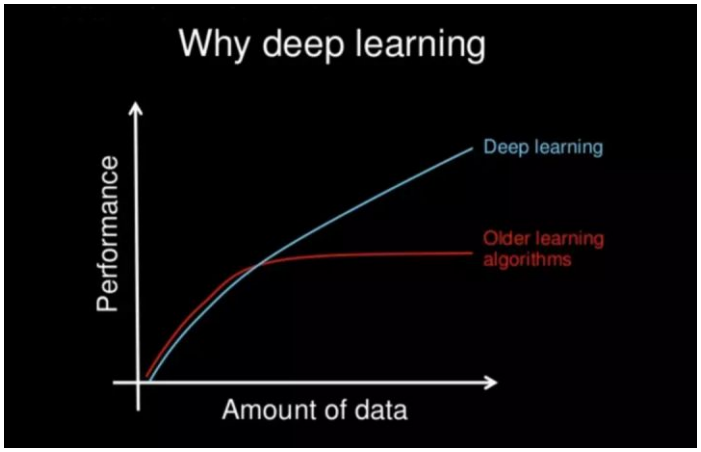

Figure 16: 神经网络与传统机器学习算法的对比。

神经网络或者说深度学习最大的缺点就是**不可解释性**。神经网络模型就像一个黑箱，我们输入数据然后得到输出，但是目前我们无法解释网络作出这样一个预测的原因。在实际应用中，模型的可解释性是非常重要的。假如银行使用神经网络来判断是否应该向用户放贷。对于你的信息，网络的输出认为你不应该获得贷款。这时候，你当然想要搞清楚银行作出这个决定的原因，而不希望获得“这是计算机模型的决定” 这样的答案。

神经网络的另一个问题就是所需样本数量很庞大。训练一个足够好的神经网络所需的样本数量往往是和模型复杂度成指数相关的。近几年提出的一些深度学习模型的复杂度往往很高，这就需要大量的数据来训练。这种规模的数据量在一些应用中是我们难以获得的。

引用

[1] Activation Function: https://en.wikipedia.org/wiki/Activation_function

[2] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[3] Tensorflow: https://www.tensorflow.org/

[4] Pytorch: https://pytorch.org/

[5] Backpropogation: https://en.wikipedia.org/wiki/Backpropagation

[6]https://www.cnblogs.com/subconscious/p/5058741.html

[7]https://blog.csdn.net/weixin_39844018/article/details/82886354