6 - 深度学习基础

强化学习与深度学习的关系

强化学习解决的是序列决策问题。
深度学习解决的是“打标签”问题，即给定一张图片，我们需要判断这张图片是猫还是狗。

深度强化学习是基于大量的样本来对相应算法进行迭代更新并且达到最优的。

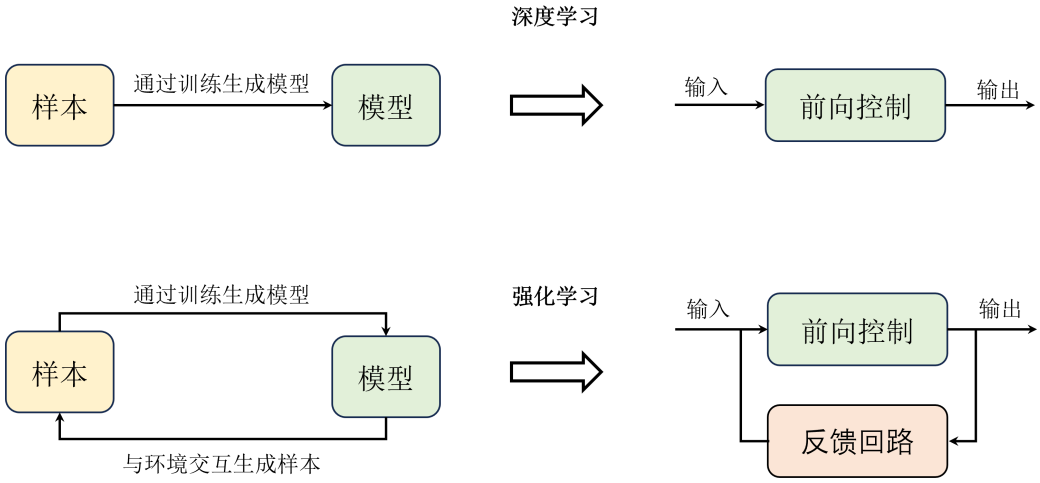


图 6-2 深度学习与强化学习示例

线性回归

严格来说，线性模型并不是深度学习模型，而是传统的机器学习模型，但它是深度学习模型的基础，在深度学习中相当于单层的神经网络。在线性模型中，应用较为广泛的两个基础模型就是线性回归和逻辑回归，通常分别用于解决回归和分类问题，尽管后者也可以用来解决回归问题。

设实例点集合 $\{x, y\}$ ，$x = \{x_1, x_2, \ldots, x_n\}$。那么 y 可以表示为

$$f(x; w, b) = w_1x_1 + w_2x_2 + \ldots + w_mx_m + b = w^Tx + b$$

其中 w 和 b 是模型的参数，$f(x; w, b)$ 是模型的输出

> **更多详情：线性回归**

梯度下降

最优化问题的解决方法也有很多种，例如最小二乘法、牛顿法等，但目前最流行的方法还是梯度下降。其基本思想如下。

- **初始化参数：选择一个初始点或参数的初始值。**
- **计算梯度：在当前点计算函数的梯度，即函数关于各参数的偏导数。梯度指向函数值增加最快的方向。**
- **更新参数：按照负梯度方向更新参数，这样可以减少函数值。这个过程在神经网络中一般是以反向传播算法来实现的。**
- **重复上述二三步骤，直到梯度趋近于 0 或者达到一定迭代次数。**

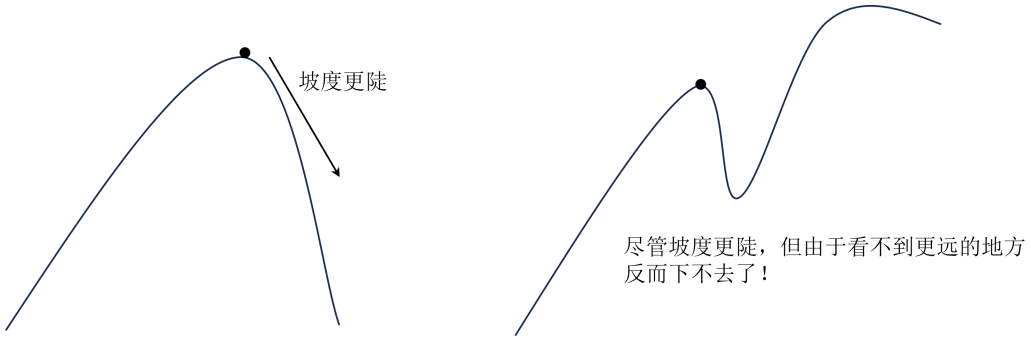


图 6-3 梯度下降示例

比较常用的梯度下降优化器有随机梯度下降（Stochastic Gradient Descent, SGD），小批量随机梯度下降（Mini Batch Stochastic Gradient Descent, Mini Batch SGD），ADAM优化器等等。

```
def stochastic_gradient_descent(X, y, learning_rate=0.01, n_iterations=1000):
        # 添加偏置项 x0=1 到输入数据 X 中
        X_b = np.c_[np.ones((X.shape[0], 1)), X]

        # 初始化模型参数
        theta = np.random.randn(X_b.shape[1], 1)

        # 随机梯度下降算法
        for iteration in range(n_iterations):
                for i in range(X.shape[0]):
                random_index = np.random.randint(X.shape[0])
                xi = X_b[random_index:random_index+1]
                yi = y[random_index:random_index+1]
                gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
                theta = theta - learning_rate * gradients

        return theta

def adam_optimizer(X, y, learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8,
n_iterations=1000):
        # 添加偏置项 x0=1 到输入数据 X 中
        X_b = np.c_[np.ones((X.shape[0], 1)), X]

        # 初始化模型参数
        theta = np.random.randn(X_b.shape[1], 1)

        # 初始化一阶和二阶矩估计
        m = np.zeros_like(theta) v = np.zeros_like(theta)

        # Adam 优化算法
        for iteration in range(1, n_iterations + 1):
                gradients = 2 * X_b.T.dot(X_b.dot(theta) - y)
                m = beta1 * m + (1 - beta1) * gradients
                v = beta2 * v + (1 - beta2) * gradients**2 m_hat = m / (1 -
beta1**iteration)
                v_hat = v / (1 - beta2**iteration)
                theta = theta - learning_rate * m_hat / (np.sqrt(v_hat) + epsilon) return
theta
```

逻辑回归

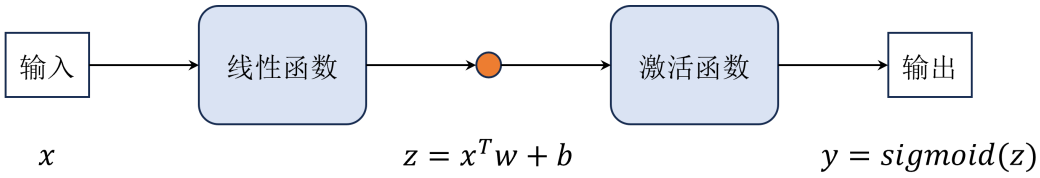


图 6-4 逻辑回归结构

从上述图，我们不难发现到逻辑回归相比于线性回归只多了一个激活函数。而这个激活函数，我们使用了 *Sigmoid* 激活函数，这就成了逻辑回归。

Sigmoid 激活函数的公式：

$$sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

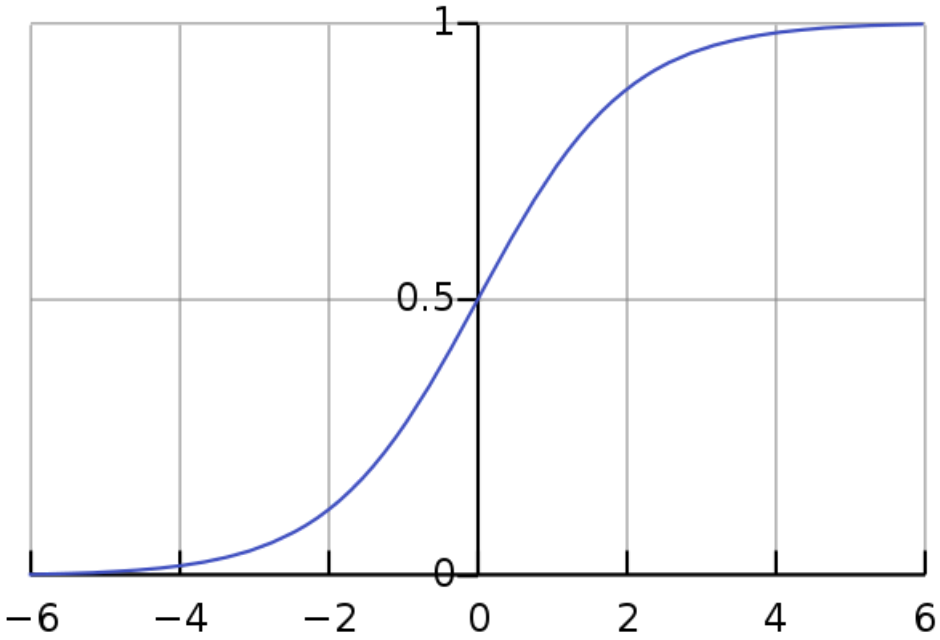


图 6-5 sigmoid 函数图像

为什么要加激活函数?
为了解决线性回归中无法解决的非线性问题。加入激活函数可以让神经网络所拟合出来的函数变为非线性函数。

> 更多细节：点击这里

全连接网络

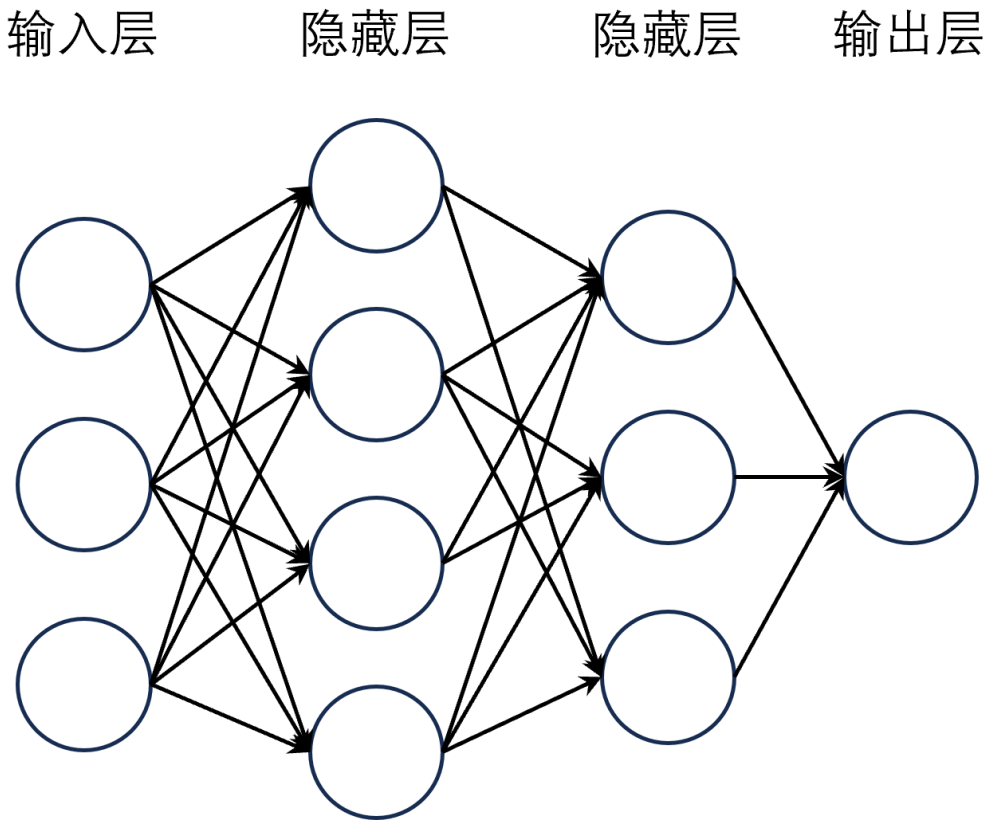


图 6-7 全连接网络

上图是全连接神经网络，也称为多层感知机，是最基础的神经网络模型。它拥有1层输入层，n层隐藏层和1层输出层。

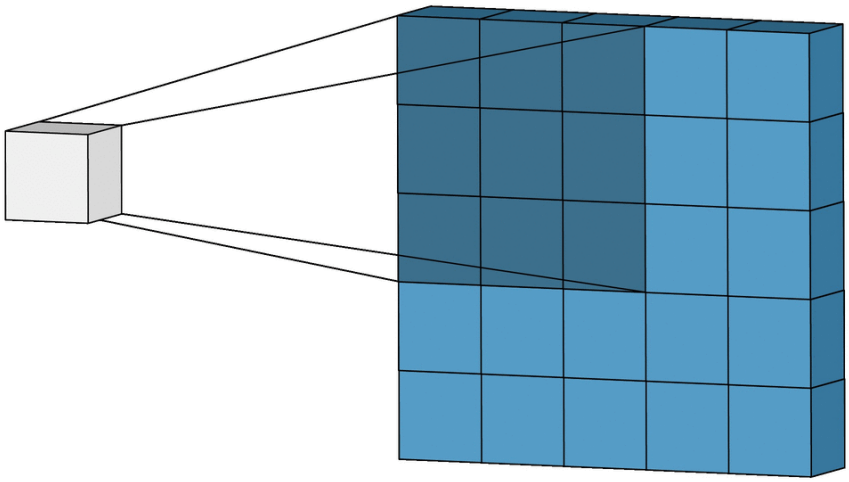
这激活函数为 σ。那每一层的公式为：

$$x^{(n)} = \sigma_n(W^{(n)}x^{(n-1)} + b^{(n)})$$

其中 W 为权重而 b 为偏执，是为了避免过拟合的项。

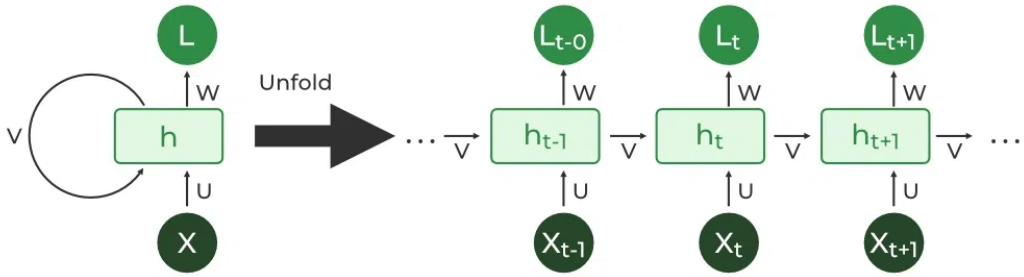
更高级神经网络

卷积神经网络（Convolution Neural Network, CNN）



以上的图展示了CNN的工作原理，通过一个卷积核从原生图像得到一个具有该图像特征的特征图。CNN可以有效的提取一个更有效的数据方便训练。在同等深度的网络上，使用CNN可以大幅缩小模型的参数量，这可以帮助节省很多的训练要求。

循环神经网络（Recurrent Neural Network，RNN）



不同于CNN，RNN将工作机制专注于处理时间序列问题，例如语言，天气等对于具有依赖性的元素。RNN的工作原理是将以前学习到的知识传个当前层参与训练，这有助于神经网络模型学习到以前的内容和当前内容的关系。

练习题

逻辑回归与神经网络之间有什么联系？

逻辑回归中加入了激活函数，这和最基础的神经模型非常相似。他们都有输入层，隐藏层和输出层，区别在于神经网络不一定只有单一结构，但是逻辑回归就是一个使用了 $Sigmoid$ 激活函数的基础神经网络模型。

全连接网络、卷积神经网络、循环神经网络分别适用于什么场景？

全连接网络适用于一些比较基础的问题，不具有太多复杂数据的问题上。
卷积神经网络一般应用于图像识别，图像切割等图像问题。
循环神经网络常常适用于处理时间数列问题，例如文本生成，股票预测，天气预测等等。

循环神经网络在反向传播时会比全连接网络慢吗？为什么？

因为玄幻神经网络相较于全连接网络多了一个传递前一个时刻知识的机制，这导致了它比全连接网络多了一些参数，因此更新更慢。