4.1 多层感知机重点摘录与练习解答

（1）隐藏层：从线性到非线性

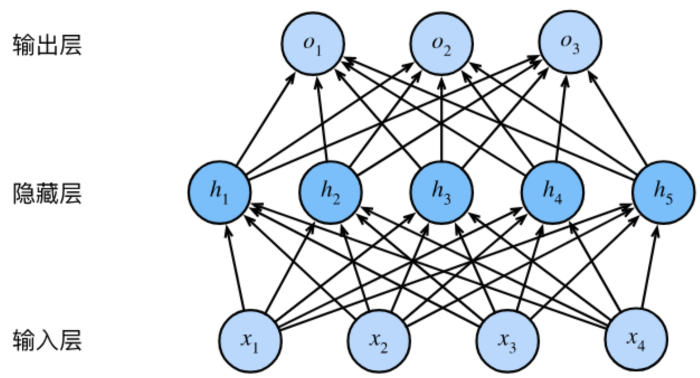

图 1: 隐藏层的加入

如同之前的章节一样，我们通过矩阵 $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ 来表示 n 个样本的小批量，其中每个样本具有 d 个输入特征。对于具有 h 个隐藏单元的单隐藏层多层感知机，用 $\boldsymbol{H} \in \mathbb{R}^{n\times h}$ 表示隐藏层的输出，称为隐藏表示（hidden representations）。在数学或代码中，$\boldsymbol{H}$ 也被称为隐藏层变量（hidden-layer variable）或隐藏变量（hidden variable）。因为隐藏层和输出层都是全连接的，所以我们有隐藏层权重 $\boldsymbol{W}^{(1)} \in \mathbb{R}^{d\times h}$ 和隐藏层偏置 $\boldsymbol{b}^{(1)} \in \mathbb{R}^{1\times h}$ 以及输出层权重 $\boldsymbol{W}^{(2)} \in \mathbb{R}^{h\times q}$ 和输出层偏置 $\boldsymbol{b}^{(2)} \in \mathbb{R}^{1\times q}$。形式上，我们按如下方式计算单隐藏层多层感知机的输出 $\boldsymbol{O} \in \mathbb{R}^{n\times q}$:

$$\boldsymbol{H} = \boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)},$$

$$\boldsymbol{O} = \boldsymbol{H}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}.$$

这其实还是仿射变换，并没有得到任何改进，我们可以证明这一等价性，即对于任意权重值，我们只需合并隐藏层，便可产生具有参数 $\boldsymbol{W} = \boldsymbol{W}^{(1)}\boldsymbol{W}^{(2)}$ 和 $\boldsymbol{b} = \boldsymbol{b}^{(1)}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}$ 的等价单层模型:

$$\boldsymbol{O} = (\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)})\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)} = \boldsymbol{X}\boldsymbol{W}^{(1)}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(1)}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}.$$

为了发挥多层架构的潜力，我们还需要在仿射变换之后对每个隐藏单元应用非线性的激活函数（activation function）σ。激活函数的输出（例如，$\sigma(\cdot)$）被称为活性值（activations）：

$$\boldsymbol{H} = \sigma(\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}),$$

$$\boldsymbol{O} = \boldsymbol{H}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}.$$

由于 $\boldsymbol{X}$ 中的每一行对应于小批量中的一个样本，出于记号习惯的考量，我们定义非线性函数 σ 也以按行的方式作用于其输入，即一次计算一个样本。我们在 :numref:‘subsec_softmax_vectorization‘ 中以相同的方式使用了 softmax 符号来表示按行操作。但是本节应用于隐藏层的激活函数通常不仅按行操作，也按元素操作。这意味着在计算每一层的线性部分之后，我们可以计算每个活性值，而不需要查看其他隐藏单元所取的值。对于大多数激活函数都是这样。

为了构建更通用的多层感知机，我们可以继续堆叠这样的隐藏层，例如 $\boldsymbol{H}^{(1)} = \sigma_1(\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)})$ 和 $\boldsymbol{H}^{(2)} = \sigma_2(\boldsymbol{H}^{(1)}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)})$，一层叠一层，从而产生更有表达能力的模型。

根据万能逼近定理，我们甚至可以知道，这样的方式可以近似任何函数。

（2）激活函数

① ReLU函数

最受欢迎的激活函数是修正线性单元（Rectified linear unit, ReLU），因为它实现简单，同时在各种预测任务中表现良好。ReLU 提供了一种非常简单的非线性变换，给定元素 x，ReLU函数被定义为该元素与0的最大值：

$$\mathrm{ReLU}(x) = \max(x, 0).$$

通俗地说，ReLU函数通过将相应的活性值设为0，仅保留正元素并丢弃所有负元素，并且是分段线性的。

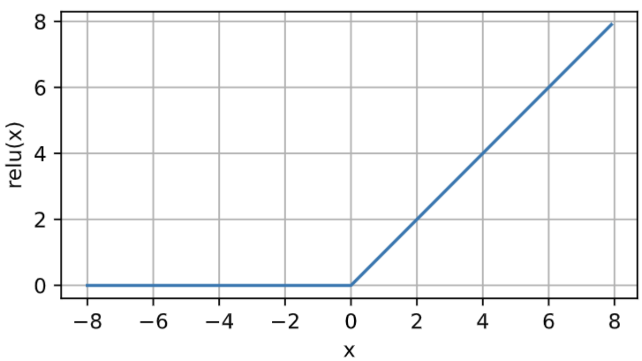

图 2: ReLU函数

当输入为负时，ReLU 函数的导数为0，而当输入为正时，ReLU 函数的导数为1。注意，当输入值精确等于 0 时，ReLU 函数不可导。在此时，我们默认使用左侧的导数，即当输入为 0 时导数为 0。我们可以忽略这种情况，因为输入可能永远都不会是 0.

使用ReLU的原因是，它求导表现得特别好：要么让参数消失，要么让参数通过。这使得优化表现得更好，并且ReLU减轻了困扰以往神经网络的梯度消失问题。

② sigmoid函数

[对于一个定义域在 $\mathbb{R}$ 中的输入，sigmoid函数将输入变换为区间(0, 1)上的输出]。因此，sigmoid通常称为挤压函数（squashing function）：它将范围（-∞, ∞) 中的任意输入压缩到区间 (0, 1) 中的某个值:

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

该函数是一个平滑的、可微的阈值单元近似。当我们想要将输出视作二元分类问题的概率时，sigmoid仍然被广泛用作输出单元上的激活函数（sigmoid可以视为softmax的特例）。然而，sigmoid在

隐藏层中已经较少使用，它在大部分时候被更简单、更容易训练的ReLU所取代。在后面关于循环神经网络的章节中，我们将描述利用sigmoid单元来控制时序信息流的架构。

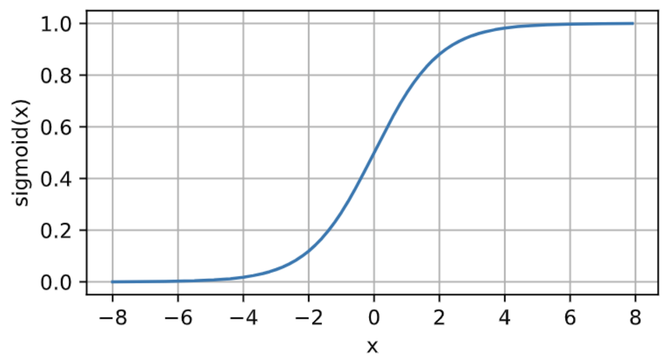

图 3: sigmoid函数

sigmoid 函数的导数为下面的公式：

$$\frac{d}{dx}\text{sigmoid}(x) = \frac{\exp(-x)}{(1+\exp(-x))^2} = \text{sigmoid}(x)(1-\text{sigmoid}(x)).$$

③ **tanh函数**

与sigmoid函数类似，[tanh(双曲正切)函数也能将其输入压缩转换到区间(-1, 1)上]。tanh函数的公式如下:

$$\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}.$$

下面我们绘制tanh函数。注意，当输入在0附近时，tanh函数接近线性变换。函数的形状类似于sigmoid函数，不同的是tanh函数关于坐标系原点中心对称。

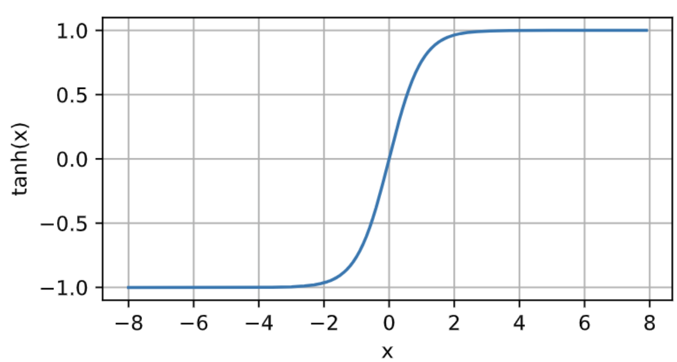

图 4: tanh函数

（**3**）问题解答

2、增加迭代周期的数量。为什么测试精度会在一段时间后降低？我们怎么解决这个问题？

解：