神经计算的数学原理

“海棠”系列丛书

作者：刘俊宏

组织：哈尔滨工业大学（深圳）

目录

第一章 回归模型

内容提要

- 线性模型
- 量化评价
- 模型求解
- 多项式回归
- 正则化处理
- 算法实验

让我们以一个**线性回归**（Linear Regression）的例子开始我们的神经计算之旅。假设我们要预测去哈工大的通勤时间，并有如下数据：

- 距离 (Km): [2.7, 4.1, 1.0, 5.2, 2.8]
- 时段 (1: if weekday, 0: if weekend): [1, 1, 0, 1, 0]
- 通勤时间 (min): [25, 33, 15, 45, 22]

我们希望找到一个函数 $f : R^2 \to R$ 满足 $f(D, T_d) \approx T_c$，其中 D 表示距离，T_d 表示时段，T_c 表示通勤时间。

1.1 线性模型

我们首先尝试构建一个线性回归模型来预测通勤时间，其一般表达式如 (1.1) 所示，其中 d 为变量的个数。

$$f(x) = w_0 + w_1 x_1 + \ldots + w_d x_d \tag{1.1}$$

为了后续方便，我们将其转化为矩阵的形式：$f(x) = W^{\mathsf{T}} X$，其中

$$\left[w_0, w_1, w_2, \ldots, w_d\right] = W^{\mathsf{T}}, \begin{bmatrix} x_0 = 1 \\ x_1 \\ \ldots \\ x_d \end{bmatrix} = X$$

笔记 我们约定所有初始向量都是列向量。在上述通勤的例子中，线性回归模型中的参数 $d = 2$，x_1 和 x_2 分别对应于 D 和 T_d 中的元素，w_0 是一个偏置项（Bias Term）。

1.2 量化评价

一旦我们初始了一个线性回归模型，我们就需要一种方法来量化地评价我们模型效果的好坏，即在本例子中我们需要一种方法来评价我们构建的模型是否可以准确地预测通勤时间。在这里，我们使用**残差**（Residual）(1.2) 来衡量第 i 个数据点的期望输出（真实的通勤时间）和模型输出（预测的通勤时间）之间的差距。

$$e^{(i)} = y^{(i)} - W^{\mathsf{T}} x^{(i)} \tag{1.2}$$

同时，我们使用**均方误差** (Mean Square Error, MSE) (1.3) 来综合考虑所有数据点的残差情况。根据**一阶必要最优性条件**（First-Order Necessary Optimality Condition），当均方误差 $C(W)$ 的一阶导数 $C'(W) = 0$ 时 $C(W)$ 取得极值。又根据**二阶充分最优性条件**（Second-Order Sufficiency Conditions），$C(W)$ 的二阶导数 $C''(W) = \frac{1}{n} X^{\mathsf{T}} X$ 恒大于零，所以 $C'(W) = 0$ 时 $C(W)$ 取得最小值。

$$C(W) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - W^{\mathsf{T}} x^{(i)}\right)^2 \tag{1.3}$$

笔记 通常，标准的均方误差表达式为：$C(W) = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - W^{\mathsf{T}} x^{(i)}\right)^2$。但在机器学习，特别是在使用梯度下降

法进行优化时，将标准的均方误差缩放为 $C(W)=\frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)}-W^{\mathsf{T}}x^{(i)}\right)^2$ 有利于计算的便利，这在下一小节的计算演示中大家会感受到。

1.3 模型求解

这里使用了一个巧妙地方法求解 $C'(W^*)=0$ 的解析解 W^*，首先我们假设第 i 个数据点的特征向量为 $x^{(i)}$，其中第 i 行表示第 i 个数据点的特征向量，第 j 列表示特征向量中的第 j 个特征。接着我们可以假设一个维度为 $n\times d$ 的特征矩阵 X 和目标输出向量 Y。

$$x^{(i)}=\begin{bmatrix}x_0^{(i)}=1\\ \cdots\\ x_j^{(i)}\\ \cdots\\ x_d^{(i)}\end{bmatrix},X=\begin{bmatrix}x^{(1)\mathsf{T}}\\ \cdots\\ x^{(n)\mathsf{T}}\end{bmatrix},Y=\begin{bmatrix}y^{(1)}\\ \cdots\\ y^{(n)}\end{bmatrix}$$

接着，我们首先推导 $XW-Y$ 这个公式看看会得到什么：

$$XW-Y=\begin{bmatrix}x^{(1)\mathsf{T}}\\ \cdots\\ x^{(n)\mathsf{T}}\end{bmatrix}W-\begin{bmatrix}y^{(1)}\\ \cdots\\ y^{(n)}\end{bmatrix}=\begin{bmatrix}x^{(1)\mathsf{T}}W-y^{(1)}\\ \cdots\\ x^{(n)\mathsf{T}}W-y^{(n)}\end{bmatrix}$$

由此，我们可以得到下述等式：

$$\begin{aligned}(XW-Y)^{\mathsf{T}}(XW-Y)&=\begin{bmatrix}x^{(1)\mathsf{T}}W-y^{(1)},&\cdots&,x^{(n)\mathsf{T}}W-y^{(n)}\end{bmatrix}\begin{bmatrix}x^{(1)\mathsf{T}}W-y^{(1)}\\ \cdots\\ x^{(n)\mathsf{T}}W-y^{(n)}\end{bmatrix}\\&=\sum_{i=1}^{n}\left(X^{(i)\mathsf{T}}W-y^{(i)}\right)^2\\&=2nC(W)\end{aligned}$$

上述等式 $2nC(W)=(XW-Y)^{\mathsf{T}}(XW-Y)$ 描述了 $C(W)$ 与 X，W，Y 之间的关系，于是我们可以进一步推理得到一个更明确的等式 (1.4)：

$$\begin{aligned}C(W)&=\frac{1}{2n}(XW-Y)^{\mathsf{T}}(XW-Y)\\&=\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}-Y^{\mathsf{T}})(XW-Y)\\&=\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}XW-W^{\mathsf{T}}X^{\mathsf{T}}Y-Y^{\mathsf{T}}XW+Y^{\mathsf{T}}Y)\\&=\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}XW-2W^{\mathsf{T}}X^{\mathsf{T}}Y+Y^{\mathsf{T}}Y)\end{aligned}\tag{1.4}$$

笔记 在推导等式 (1.4) 的时候可能会对最后两步推导过程感到疑惑：

$$\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}XW-W^{\mathsf{T}}X^{\mathsf{T}}Y-Y^{\mathsf{T}}XW+Y^{\mathsf{T}}Y)=\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}XW-2W^{\mathsf{T}}X^{\mathsf{T}}Y+Y^{\mathsf{T}}Y)$$

这里解释一下，这是因为 $W^{\mathsf{T}}X^{\mathsf{T}}Y$ 实际上会得到一个实数，并且 $W^{\mathsf{T}}X^{\mathsf{T}}=(XW)^{\mathsf{T}}$。于是我们可以获得等价关系：

$$W^{\mathsf{T}}X^{\mathsf{T}}Y=(W^{\mathsf{T}}X^{\mathsf{T}}Y)^{\mathsf{T}}=Y^{\mathsf{T}}XW$$

笔记 同时，请注意看 X，W，Y 的组织形式，所以这里写成 $XW-Y$ 的形式才是正确并方便我们的推导，而不是 $Y-W^{\mathsf{T}}X$ 的形式。

通过等式 (1.4) 我们可以进一步求解 $C'(W)$ 与 X,W,Y 之间的关系。我们首先通过观察可以发现：1. $W^{\mathsf{T}}X^{\mathsf{T}}XW$

是二次项，2. $W^\mathsf{T}X^\mathsf{T}Y$ 是一次项，3. $Y^\mathsf{T}Y$ 是常数项。于是我们便可以快速得到等式 (1.5):

$$C'(W) = \frac{1}{2n}(2X^\mathsf{T}XW - 2X^\mathsf{T}Y) \tag{1.5}$$

根据上一小节的推论，当 $C'(W^*) = 0$ 时取得最优的 W^*，此时 $C(W)$ 取得最小值。于是根据等式 (1.5) 可以得到 $X^\mathsf{T}XW^* = X^\mathsf{T}Y$。当 $X^\mathsf{T}X$ 可逆时便可以获得 W^* 的解析解 (1.6) 和模型的输出 Y^* (1.6)：

$$\begin{cases} W^* = (X^\mathsf{T}X)^{-1}X^\mathsf{T}Y \\ Y^* = XW^* = X(X^\mathsf{T}X)^{-1}X^\mathsf{T}Y \end{cases} \tag{1.6}$$

证明 看完了 W^* 的解析解的整个推导过程，有些读者可能会有这样一个疑问：为什么 W 作为一个向量却可以将其视为一个“整体”进行求导操作？这涉及到多元微积分和矩阵求导的核心概念，下面开始阐述其数学原理。

1. 标量对向量的导数：梯度的定义

当函数 $f(X)$ 的输出是一个标量，而输入 X 是一个向量时，其导数被称为梯度（Gradient），记作 $\nabla_X f(X)$。梯度本身是一个向量，其每个分量是函数 f 对 X 的每个分量的偏导数。

例如，如果 $W = [w_0, w_1, ..., w_d]^\mathsf{T}$，那么：

$$\nabla_W C(W) = \frac{\partial C(W)}{\partial W} = \left[\frac{\partial C(W)}{\partial w_0}, \frac{\partial C(W)}{\partial w_1}, \cdots, \frac{\partial C(W)}{\partial w_d}\right]^\mathsf{T}$$

2. 为什么推导过程看起来像是在对“一维未知量”求导？

要回答这个问题，我们需要理解的一个关键问题是：为什么 XW（或 $W^\mathsf{T}X^\mathsf{T}$）对 W 的梯度是 X（或 X^T）。

我们假设

$$Z(W) = XW$$

其中：

$$X = \begin{bmatrix} x_0^{(1)}, x_1^{(1)}, ..., x_d^{(1)} \\ ... \\ x_0^{(n)}, x_1^{(n)}, ..., x_d^{(n)} \end{bmatrix}, W = \begin{bmatrix} w_0 \\ ... \\ w_d \end{bmatrix}$$

接着我们对 $Z(W)$ 求一阶导，即求 $Z(W)$ 的雅可比矩阵：

$$\nabla_W Z(W) = \frac{\partial Z_i}{\partial W_k} = \frac{\partial(\sum_{j=0}^{d} X_{ij}W_j)}{\partial W_k} = X_{ik}$$

即

$$J = \begin{bmatrix} \frac{\partial Z_1}{\partial w_0} & \frac{\partial Z_1}{\partial w_1} & \cdots & \frac{\partial Z_1}{\partial w_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Z_n}{\partial w_0} & \frac{\partial Z_n}{\partial w_1} & \cdots & \frac{\partial Z_n}{\partial w_d} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{(n)} & x_1^{(n)} & \cdots & x_d^{(n)} \end{bmatrix} = X$$

所以雅可比矩阵的第 i 行就是 X 的第 i 行，即证：

$$\frac{\partial XW}{\partial W} = X$$

1.4 线性模型求解二分类问题

假设我们有下述期望的输入输出对 S：

$$S = \left\{(x^{(1)}, y^{(1)}), \cdots, (x^{(n)}, y^{(n)})\right\},\ y^{(i)} \in \{-1, +1\}$$

模型在上述分类问题的单样本表现可以用 0-1 损失函数 L 来衡量：

$$L(\alpha, \beta) = \mathbb{I}[\alpha \neq \beta] = \begin{cases} 1, & \text{if } \alpha \neq \beta \\ 0, & otherwise. \end{cases}$$

从而模型的整体效果我们可以用下面的函数 $C(W)$ 衡量：

$$C(W) = \frac{1}{n}\sum_{i=1}^{n} L(\text{sgn}(\hat{y}^{(i)}), y^{(i)}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\text{sgn}(W^{\mathsf{T}}x^{(i)}) \neq y^{(i)}]$$

$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0. \end{cases}$$

但上述公式有一个不方便的地方就是目前的 $C(W)$ 是一个不连续的分段函数，如果我们可以找到一个可微分的减函数 G 来代替 L 函数用于评估单个样本上的模型效果，那么我们就可以得到一个易于求解的优化问题。

这里我们引入**间隔**（Margin）的概念，并定义间隔为模型的预测值与期望值的乘积，数学表达式为 $y\hat{y}$，即 $yW^{\mathsf{T}}x$。我们现在考虑基于间隔的函数 G 来衡量模型在单个样本上的效果，下面列举了三个可选的 G 函数形式：

$$G(y\hat{y}) = \max\{0, 1 - y\hat{y}\}$$

$$G(y\hat{y}) = \frac{1}{2}(\max\{0, 1 - y\hat{y}\})^2$$

$$G(y\hat{y}) = \log(1 + \exp(-y\hat{y}))$$

这里我们考虑 G 函数的第二种形式：

$$G(y\hat{y}) = \frac{1}{2}(\max\{0, 1 - y\hat{y}\})^2 = \frac{1}{2}\left(\max\left\{0, 1 - yW^{\mathsf{T}}x\right\}\right)^2 = \begin{cases} 0, & \text{if } yW^{\mathsf{T}}x \geq 1 \\ \frac{1}{2}(1 - yW^{\mathsf{T}}x)^2, & otherwise. \end{cases}$$

于是我们有如下形式的 $C(W)$ 函数：

$$C(W) = \frac{1}{2n}\sum_{i=1}^{n}\left(\max\left\{0, 1 - y^{(i)}W^{\mathsf{T}}x^{(i)}\right\}\right)^2$$

其中 $C_i(W) = G(y^{(i)}\hat{y}^{(i)})$，接着我们可以求得 $C_i(W)$ 的一阶导表达式：

$$\nabla C_i(W) = \begin{cases} 0, & \text{if } y^{(i)}W^{\mathsf{T}}x^{(i)} \geq 1 \\ -(1 - y^{(i)}W^{\mathsf{T}}x^{(i)})y^{(i)}x^{(i)}, & otherwise. \end{cases}$$

又因为 $y^2(i) = 1$，所以 $\nabla C_i(W)$ 的表达式 (1.7) 可以简化为：

$$\nabla C_i(W) = \begin{cases} 0, & \text{if } y^{(i)}W^{\mathsf{T}}x^{(i)} \geq 1 \\ (W^{\mathsf{T}}x^{(i)} - y^{(i)})x^{(i)}, & otherwise. \end{cases} \tag{1.7}$$

通过上面的学习我们知道最优的模型权重 W^* 必须满足 $\frac{1}{n}\sum_{i=1}^{n}\nabla C_i(W^*) = 0$，但即使我们知道了等式左右两边的完整表达式，我们仍然缺乏一种有效的手段来求解上面的等式，即难以求出 W^* 的解析解。所以退而求其次，我们将在下面章节介绍利用随机梯度下降来求 W^* 的数值解（近似解）的方法。

1.5 多项式回归

在推导出线性回归模型的解后，我们其实也可以用上述公式求解多项式回归的 W^* 的解析解。多项式回归是一种基础的非线性回归模型，其基本形式如公式 (1.8) 所示：

$$f(x) = w_0 + w_1x_1 + w_2x^2 + \ldots + w_mx^m = W^{\mathsf{T}}\phi(x) \tag{1.8}$$

其中 $\phi(x) = \left[1, x, x^2, \cdots, x^m\right]^{\mathsf{T}}$，我们同样设：

$$X = \begin{bmatrix} \phi(x)^{(1)\mathsf{T}} \\ \ldots \\ \phi(x)^{(n)\mathsf{T}} \end{bmatrix}, Y = \begin{bmatrix} y^{(1)} \\ \ldots \\ y^{(n)} \end{bmatrix}$$

同样可以得到相同的解析解形式：

$$\begin{cases} W^* = (X^\mathsf{T}X)^{-1}X^\mathsf{T}Y \\ Y^* = XW^* = X(X^\mathsf{T}X)^{-1}X^\mathsf{T}Y \end{cases}$$

1.6 正则化处理

理想情况下，我们希望模型在训练样本上表现良好同时又不至于太复杂，从而实现良好的泛化（既不过拟合也不过欠拟合）。解决这个问题的一种方法是鼓励较小的权重（这样任何特征都不会对预测产生太大的影响）。这被称为**正则化**（Regularization）。

给定一个数据集 $S = \{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}$ 和一个正则化强度 $\lambda > 0$，我们同样希望找到下列**正则化最小二乘回归**函数（Regularized Least Square Regression） (1.9)，也被称为岭回归（Ridge Regression），的最小值：

$$C(W) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - W^\mathsf{T}X^{(i)}\right)^2 + \frac{\lambda}{2}||W||_2^2 \tag{1.9}$$

其中 $||W||_2$ 表示 W 的二范数（2-norm）。现在函数 (1.9) 有两个优化目标：函数右边第一项 $\frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - W^\mathsf{T}X^{(i)}\right)^2$ 是误差项（经验风险），用来衡量模型对训练数据的拟合程度；函数右边第二项 $\frac{\lambda}{2}||W||_2^2$ 是正则化项（结构风险），用来对模型的复杂度进行惩罚，当模型的某些权重变得过大，正则化项会变大并超过误差项的影响，从而导致总成本 $C(W)$ 变大。

定义 1.1 (范数)

对于一个向量 $v = (v_1, ..., v_d) \in R^d$，范数被定义为：

$$||V||_n = (|v_1|^n + |v_2|^n + ... + |v_d|^n)^{\frac{1}{n}}$$

二范数也被称为欧几里得范数（Euclidean Norm）。

♣

接下来我们开始求解上述函数的 W^* 值，前面我们已经证明过 $2nC(W) = (XW - Y)^\mathsf{T}(XW - Y)$，于是我们便可以得知：

$$C(W) = \frac{1}{2n}(XW - Y)^\mathsf{T}(XW - Y) + \frac{\lambda}{2}||W||_2^2$$

同时通过公式 (1.5)，我们可知 $C(W)$ 的一阶导为：

$$\begin{aligned} C'(W) &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + (||W||_2^2)' \\ &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + ((|w_0|^2 + |w_1|^2 + ... + |w_d|^2)^{\frac{1}{2}\times 2})' \\ &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + (|w_0|^2 + |w_1|^2 + ... + |w_d|^2)' \\ &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + 2(|w_0| + |w_1| + ... + |w_d|) \\ &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + 2W \end{aligned}$$

将 $C'(W^*) = 0$ 代入上式可得：

$$\begin{aligned} \frac{1}{n}(X^\mathsf{T}XW^* - X^\mathsf{T}Y) + \lambda W^* &= 0 \\ \frac{1}{n}X^\mathsf{T}XW^* + \lambda W^* &= \frac{1}{n}X^\mathsf{T}Y \\ (\frac{1}{n}X^\mathsf{T}X + \lambda\mathbb{I})W^* &= \frac{1}{n}X^\mathsf{T}Y \\ W^* &= (\frac{1}{n}X^\mathsf{T}X + \lambda\mathbb{I})^{-1}(\frac{1}{n}X^\mathsf{T}Y) \end{aligned}$$

于是我们得到了公式 (1.9) 的解析解 $W^* = (\frac{1}{n}X^\mathsf{T}X + \lambda\mathbb{I})^{-1}(\frac{1}{n}X^\mathsf{T}Y)$。这个正则化后的解析解有一个非常好的性质

那就是不要求 $X^\mathsf{T}X$ 是可逆的（invertible），因为 $\frac{1}{n}X^\mathsf{T}X+\lambda\mathbb{I}$ 几乎总是可逆的（一个矩阵可逆的充要条件是它的所有特征值都不为零，而 $\frac{1}{n}X^\mathsf{T}X+\lambda\mathbb{I}$ 的所有特征值都严格大于零。）

1.7 算法实验

现在让我们回到最开头的案例，我们想预估我们上学的通勤时间，现在我们有下列数据：

- 距离 (Km): [2.7, 4.1, 1.0, 5.2, 2.8]
- 时段 (1: if weekday, 0: if weekend): [1, 1, 0, 1, 0]
- 通勤时间 (min): [25, 33, 15, 45, 22]

代如上述公式 (1.6) 求解出的输出应该为：

$$W^* = \begin{bmatrix} 6.08613445378149 \\ 6.53361344537816 \\ 2.11274509803922 \end{bmatrix}, Y^* = \begin{bmatrix} 25.8396358543418 \\ 34.9866946778712 \\ 12.6197478991597 \\ 42.1736694677872 \\ 24.3802521008403 \end{bmatrix}, Y = \begin{bmatrix} 25 \\ 33 \\ 15 \\ 45 \\ 22 \end{bmatrix}$$

其中 W^* 为模型的最优权重，Y^* 为模型的最优预测输出，Y 为模型的期望输出（即真实数据）。上述例子在 GitHub 仓库提供了 Matlab 代码（Linear_Regression.m）实现，其可视化的结果如图 1.1 所示。

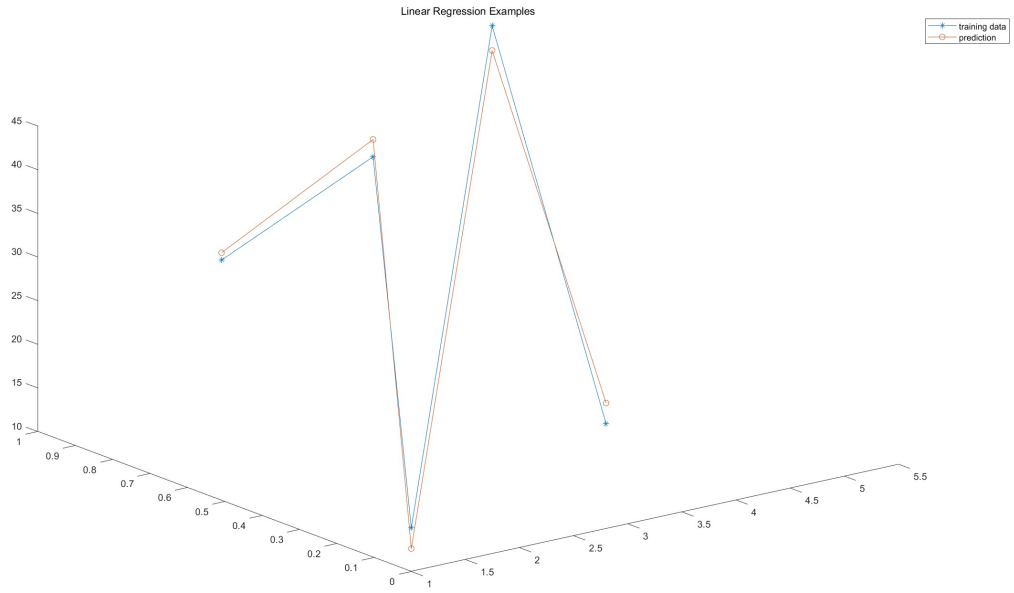


图 1.1: 通勤时间的回归建模结果

笔记 大家可能发现了一个现象：即使是 W^* 的解析解，我们构建的线性模型也没有完美拟合数据分布。这里要注意的是我们求解的是 $C(W)$ 的最小值，但是 $C(W)$ 的最小值并不一定为 0！

第二章 梯度下降

内容提要

- 算法原理
- 随机梯度下降

2.1 算法原理

梯度下降（Gradient Descent）是最小化目标函数 $C(W)$ 的通用算法之一（由柯西于 1847 年首次提出）。它是一种迭代算法，从 $W^{(t)}$ 开始，每次迭代都会产生一个新的 $W^{(t+1)}$，其公式 (2.1) 如下：

$$W^{(t+1)} = W^{(t)} - \eta_t \nabla C(W^{(t)}),\ (t = 0, 1, ..., n) \tag{2.1}$$

其中 η_t 称为**学习率**（Learning Rate）或**步长**（Step Size）。因此梯度下降 (2.1) 使用负梯度作为搜索方向，并沿着该方向移动 η_t 的距离。

接下来我们证明梯度下降 (2.1) 的合理性与可行性。根据一阶泰勒近似 $f(x) \approx f(a) + f'(a)(x-a)$ 的形式，那么存在下列近似关系：

$$C(W^{(t+1)}) \approx C(W^{(t)}) + \nabla C(W^{(t)})^{\mathsf{T}}(W^{(t+1)} - W^{(t)})$$

接着将公式 (2.1) 代入上述式子中可以得到：

$$\begin{aligned} C(W^{(t+1)}) &\approx C(W^{(t)}) + \nabla C(W^{(t)})^{\mathsf{T}}(W^{(t)} - \eta \nabla C(W^{(t)}) - W^{(t)}),\ (\eta \to 0) \\ &\approx C(W^{(t)}) + \nabla C(W^{(t)})^{\mathsf{T}}(-\eta \nabla C(W^{(t)})) \\ &\approx C(W^{(t)}) - \eta \nabla C(W^{(t)})^{\mathsf{T}} \nabla C(W^{(t)}) \end{aligned}$$

因为 $\nabla C(W)^{\mathsf{T}} \nabla C(W) = ||\nabla C(W)||_2^2 > 0$，所以即证 $C(W^{(t+1)}) < C(W^{(t)})$。

笔记 我们现在可以尝试利用梯度下降法来求解线性回归问题。

对于最小二乘回归 (1.4)，我们已经得到了 $C(W)$ 的一阶导形式 (1.5)，套用梯度下降公式 (2.1) 可以直接得到 W^* 数值解的求解方式：

$$\begin{aligned} W^{(t+1)} &= W^{(t)} - \eta_t \nabla C(W^{(t)}) \\ &= W^{(t)} - \frac{\eta}{n}(X^{\mathsf{T}} X W^{(t)} - X^{\mathsf{T}} Y) \end{aligned}$$

继续套用通勤时间计算的例子，GitHub 仓库提供了 Matlab 代码（Gradient_Descent_for_Linear_Regression.m）实现。

2.2 随机梯度下降

通过上面的学习，我们可以知道 $\nabla C(W^{(t)})$ 的一般形式：

$$\nabla C(W^{(t)}) = \frac{1}{n} \sum_{i=1}^{n} \nabla C_i(W^{(t)})$$

其中 $C_i(W^{(t)})$ 表示的是模型对数据点 i 的预测值与期望值之间的差距。通过上述式子我们不难发现，模型需要遍历所有的数据才可以计算出梯度，如果数据特别多这种方式的计算消耗将会特别大。为了解决这个问题，人们发明了一种梯度下降法的变体叫做**随机梯度下降**（Stochastic Gradient Descent，SGD）(2.2)。

$$W^{(t+1)} = W^{(t)} - \eta_t \nabla C_i(W^{(t)}),\ (\eta_t = C/\sqrt{t}) \tag{2.2}$$

其中 $\nabla C_i(W^{(t)})$ 在每次迭代时将从数据集中进行随机且均匀地抽样。随机梯度是真实梯度的带有噪声的估计，如果学习率太大且固定不变，那么噪声的影响会被放大从而导致权重的更新步伐会非常不稳定，甚至可能无法收

敛到最小值点，而是被噪声推着越跑越远。所以我们在训练过程中一般会逐步降低学习率，一个典型的选择是：

$$\eta_t = \frac{C}{\sqrt{t}}$$

其中 C 是一个需要根据实际情况调整的超参数，t 是迭代次数。也就是说，随着迭代次数的增加，我们使用的学习率会越来越小。对于梯度下降与随机梯度下降之间的差异，我们可以总结以下几点：

- 如果我们想要一个高精度的解决方案，并且样本量较小，那么梯度下降是好的选择。但如果我们想要一个精度适中的解决方案，并且问题规模较大，那么使用随机梯度下降可能是更好的选择。
- 随机梯度下降的计算成本与样本大小无关，因此 SGD 对于大规模问题尤其有效。
- 梯度下降是平滑的，因为负梯度是函数值减小的方向。随机梯度下降是不稳定的，因为随机梯度是对真实梯度的带有噪声的估计。

笔记 为什么要采用 $\eta_t = \frac{C}{\sqrt{t}}$ 形式的学习率设计？

为了确保 SGD 最终能够收敛到最优解（或其附近），我们必须克服噪声的影响。这里我们遵从 Robbins-Monro 算法收敛条件，其为随机近似问题提供了几乎必然收敛（Almost Surely Convergence）的充分条件。应用于 SGD 的语境下，我们希望学习率序列满足下列条件：

1. $\sum_{t=1}^{\infty} \eta_t \to \infty$
2. $\sum_{t=1}^{\infty} \eta_t^2 < \infty$

第一个条件确保算法拥有足够的“能量”从任意初始点到达最优解，即使初始点离得很远。第二个条件确保梯度估计中的噪声（方差）能够被逐渐衰减至零，从而保证算法最终稳定在最优解附近。接下来我们证明 $\eta_t = \frac{C}{\sqrt{t}}$（其中 $C > 0$ 是一个常数）的形式满足上述两个条件。

条件一：$\sum_{t=1}^{\infty} \eta_t = \sum_{t=1}^{\infty} \frac{C}{\sqrt{t}} \to \infty$。这个我们可以通过将其与一个著名的发散级数（调和级数）进行比较来证明这个级数是发散的。对于任意 $t \geq 1$，我们有：

$$\frac{1}{\sqrt{t}} \geq \frac{1}{t}$$

我们知道调和级数 $\sum_{t=1}^{\infty} \frac{1}{t}$ 是发散的，一个简便的证明是：

$$\begin{aligned}\sum_{t=1}^{\infty} \frac{1}{t} &\geq \int_1^n \frac{1}{x} dx \\ &= \ln x \Big|_1^n \\ &= (\ln x - \ln 1) \to \infty\end{aligned}$$

由于 $\frac{1}{\sqrt{t}}$ 中每一项都大于或等于 $\frac{1}{t}$ 的对应项，所以 $\sum_{t=1}^{\infty} \frac{1}{\sqrt{t}}$ 也发散，即 $\sum_{t=1}^{\infty} \eta_t = \sum_{t=1}^{\infty} \frac{C}{\sqrt{t}} \to \infty$

条件二：$\sum_{t=1}^{\infty} \eta_t^2 = \sum_{t=1}^{\infty} \left(\frac{C}{\sqrt{t}}\right)^2 = \sum_{t=1}^{\infty} \frac{C^2}{t} < \infty$。现在我们来看这个级数是否收敛，我们首先进行如下变换：

$$\sum_{t=1}^{\infty} \eta_t^2 = C^2 \sum_{t=1}^{\infty} \frac{1}{t}$$

调和级数（Harmonic Series）$\sum_{t=1}^{\infty} \frac{1}{t}$ 在前面我们已经证明是发散的，所以实际上 $\eta_t = \frac{C}{\sqrt{t}}$ 并不严格满足 Robbins-Monro 的第二个条件。

接下来让我们重新审视 $\eta_t = \frac{C}{\sqrt{t}}$ 的设计。尽管 η_t 发散，但其发散速度是对数级的（类似于 $\ln(t)$）。在有限步（例如 $T = 10^6$）的迭代中，累积的 $\sum_{t=1}^{T} \eta_t^2$ 仍然是一个可控的有限值。但在实践中如果采用 $\eta_t = \frac{C}{t}$ 的形式，其确实满足 Robbins-Monro 的两个条件，但其衰减速度过快，在实践中性能往往不如 $\frac{C}{\sqrt{t}}$。所以，$\frac{C}{\sqrt{t}}$ 的衰减速度在“保证充分搜索”($\sum \eta_t = \infty$) 和“抑制噪声”之间提供了一个非常好的实践折衷。它比 $\frac{C}{t}$ 衰减得慢，允许算法在后期仍然保持一定的探索能力，从而在实践中通常获得更好的性能。

2.2.1 小批量随机梯度下降

在了解了基本的随机梯度下降算法之后，接下来给大家介绍随机梯度下降的衍生算法，**小批量随机梯度下降** (Minibatch Stochastic Gradient Descent)。小批量随机梯度下降不再每次只抽取一个样本计算梯度，而是抽取多个样本，其数学表达式为 (2.3)：

$$W^{(t+1)} = W^{(t)} - \frac{\eta_t}{b} \sum_{i \in B} \nabla C_i(W^{(t)}) \tag{2.3}$$

其中 B 为随机抽取的样本的集合，$b = \mathsf{length}(B)$。

这里我们可以采用小批量随机梯度下降的方法继续再次求解上述的通勤时间预测问题。我们设定 $C = 0.06$，$b = 2$，且当 $|W^{(t+1)} - W^{(t)}| < 0.02$ 时停止迭代。GitHub 仓库提供了 Matlab 代码（MiniSGD_for_Linear_Regression.m）实现。

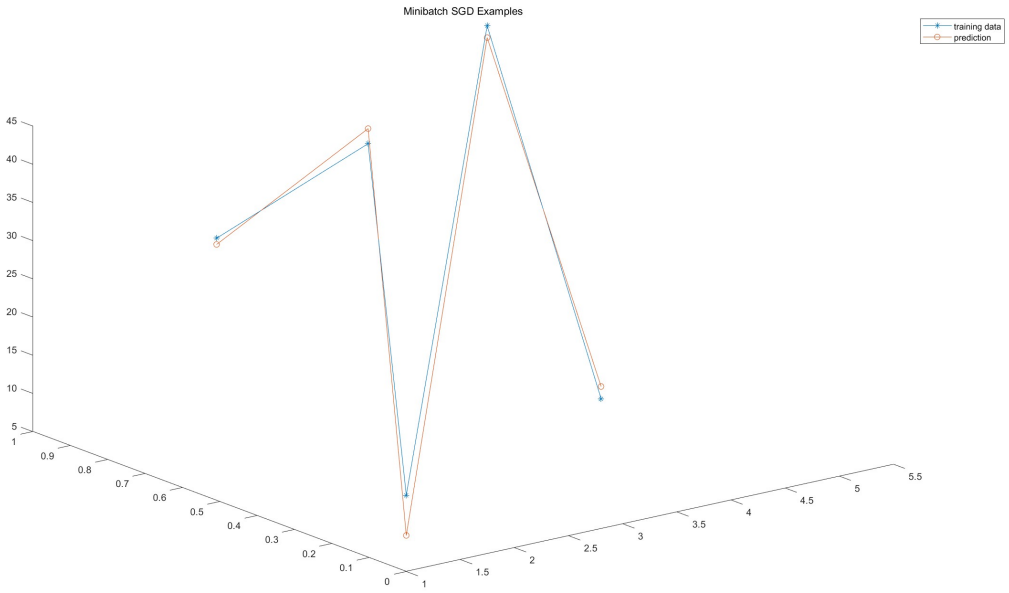


图 2.1: 小批量随机梯度下降的回归建模结果

笔记 这里有两种采样方式：重复采样和不重复采样。这两个概念的核心区别在于从总体中抽取样本后是否将样本个体放回总体中再进行下一轮采样。一般我们采用不重复采样。

2.2.2 求解线性二分类问题

在 1.4 小节中我们已经推导出了 $\nabla C_i(W)$ 的表达式 (1.7)，但我们难以直接求解 W^* 的解析解，这时通过随机梯度下降法我们就可以方便快速地迭代出 W^* 的数值解：

$$W^{(t+1)} = W^{(t)} - \eta_t \nabla C_i(W^{(t)}) = \begin{cases} W^{(t)}, & \text{if } y^{(i)} {W^{(t)}}^{\mathsf{T}} X^{(i)} \geq 1 \\ W^{(t)} - \eta_t (W^{\mathsf{T}} x^{(i)} - y^{(i)}) x^{(i)}, & otherwise. \end{cases}$$

我们在 GitHub 仓库提供了利用随机梯度下降（及小批量随机梯度下降）求解花朵品种预测的二分类问题的 Matlab 代码（MiniSGD_for_Linear_Classification.m）实现。

第三章 神经网络

内容提要

- 感知器
- 单层神经网络
- 前馈神经网络
- 链式法则

3.1 感知器

感知器（Perceptron）算法是一种二分类的线性分类模型，也是最早最简单的人工神经网络之一。它在 1958 年由 Frank Rosenblatt 提出，是深度学习领域的重要基石。

让我们继续考虑上一章节的二分类问题，假设存在输入输出对 S：

$$S = \left\{(x^{(1)}, y^{(1)}), \cdots, (x^{(n)}, y^{(n)})\right\},\ y^{(i)} \in \{-1, +1\}$$

这里 y 的取值只是一种形式化的表示，目的在于方便模型进行二分类预测。同时，我们需要一个**激活函数**（activation function）将感知机的线性计算（加权求和）转换为一个非线性的决策（分类结果），从而让感知机能够执行分类任务而不仅仅是做线性回归。此外，激活函数的另一个关键意义在于：它使得感知器学习算法得以实现。在这里感知器使用的是一种叫单位阶跃函数的激活函数，结合线性模型的表达式 $y = W^{\mathsf{T}}x + b$ 我们可以得到感知器模型的输出 $\hat{y}$ 的一个完整表达式：

$$\hat{y} = \begin{cases} 1, & \text{if } (W^{\mathsf{T}}x + b) > 0 \\ -1, & otherwise. \end{cases}$$

接着我们继续使用间隔的概念，当 $y\hat{y} = yW^{\mathsf{T}}x < 0$ 时，这意味着模型没有正确预测该样本的类别，那么我们就需要更新模型的权重来让模型能够进行正确的预测。所以下面我们开始介绍感知器的核心算法步骤：

1. 初始化：将权重 W 和偏置 b 初始化为 0 或很小的随机数。
2. 判断是否所欲样本都被正确分类。如果都被正确分类则程序终止，否则跳转到步骤 3。
3. 对所有样本进行遍历，如果 $yW^{\mathsf{T}}x < 0$，则更新权重 $W = W + \eta(y - \hat{y})x$ 和偏置 $b = b + \eta(y - \hat{y})$。所有样本遍历完后跳转到步骤 2。

证明 这里我们来证明为什么 $W = W + \eta(y - \hat{y})x$ 能够有效更新权重并优化模型的输出。首先我们直接对 $yW_{new}^{\mathsf{T}}x$ 进行展开，有如下推导过程：

$$\begin{aligned} yW_{new}^{\mathsf{T}}x &= y(W_{old} + \eta(y - \hat{y})x)^{\mathsf{T}}x \\ &= yW_{old}^{\mathsf{T}}x + \eta(y^2 - y\hat{y})x^{\mathsf{T}}x \\ &= yW_{old}^{\mathsf{T}}x + C \end{aligned}$$

接着我们开始分析 $C = \eta(y^2 - y\hat{y})x^{\mathsf{T}}x$ 这一项，其中 $\eta > 0$，$y^2 > 0$，$x^{\mathsf{T}}x > 0$，$y\hat{y} < 0$，所以我们可以知道 C 这个常数项恒大于 0，即 $yW_{new}^{\mathsf{T}}X > yW_{old}^{\mathsf{T}}X$。所以感知器的更新迭代策略是有效的。

3.2 单层神经网络

感知机的一个局限性在于它只能构建线性分类器。具体而言，它根据样本位于超平面某一侧的位置对其进行分类。同时感知机还具有以下缺陷：

1. 感知机仅在数据线性可分时才能收敛，对于线性不可分的数据（即不存在能够完全区分正负样本的超平面）则无法停止迭代。
2. 权值 W 仅针对误分类样本进行调整（正确分类的样本完全不被考虑）。

3. 多个感知机可以形成复杂的决策边界（分段线性：例如通过两个感知机将空间划分为四个区域，从而实现正负样本的分离），但训练难度较大。

所以我们得想办法改进感知机，一个思路就是用可微的非线性函数替换掉感知机原始的激活函数，这里我们使用 Sigmoid 函数 (3.1)：

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.1}$$

Sigmoid 函数可以将区间 $(-\infty, +\infty)$ 映射到 $(0,1)$，且 $\sigma'(x) = \sigma(x)(1-\sigma(x))$。单层神经网络与感知机的唯一区别在于：前者使用高度非线性的 sigmoid 函数替代了感知机中的单位阶跃函数。此外 Sigmoid 函数可以输出任意实数值，这意味着单层神经网络能够应用于回归问题。

证明 这里我们来证明 $\sigma'(x) = \sigma(x)(1-\sigma(x))$：

$$\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} \\
\sigma'(x) &= \frac{-(-e^{-x})}{(1+e^{-x})^2} \\
\sigma'(x) &= \frac{1}{1+e^{-x}} \bullet \frac{e^{-x}+1-1}{1+e^{-x}} \\
\sigma'(x) &= \frac{1}{1+e^{-x}}(1 - \frac{1}{1+e^{-x}}) \\
\sigma'(x) &= \sigma(x)(1-\sigma(x))
\end{aligned}$$

现在让我们重新考虑非线性回归问题，套用公式 (1.3) 我们可以快速得知单层神经网络的均方误差表达式 (3.2)。需要注意的是公式 (3.2) 多了一个新的需要迭代训练的参数 b，它表示偏置项，其维度等于单层网络中神经元的个数。

$$C(W,b) = \frac{1}{2n}\sum_{i=1}^{n}(\sigma(W^{\mathsf{T}}x^{(i)} + b) - y^{(i)})^2 \tag{3.2}$$

不同于线性回归问题，我们难以直接求解单层神经网络的 $\nabla C(W^*, b^*) = 0$ 的解析解，但我们任然可以通过梯度下降法求解 W^* 和 b^* 的近似数值解：

$$W^* = W - \eta\nabla C(W,b),\ b^* = b - \eta\nabla C(W,b)$$

即求解 (3.3)

$$w_i^* = w_i - \eta\frac{\partial C(W,b)}{\partial w_i},\ b_i^* = b_i - \eta\frac{\partial C(W,b)}{\partial b_i} \tag{3.3}$$

这里 w_i 代表的就是神经网络里面的一个神经元的权重，所以 $W = [w_0, \cdots, w_n]$ 表示的是一个具有 n 个神经元的单层神经网络的权重。我们将在下一小节展示如何利用一元链式法则求解公式 (3.3)。

3.3 链式法则

为了便于大家理解单层神经网络的梯度传播过程，让我们简单回顾一下计算复合函数导数的**链式法则**（Chain Rule）。它将复合函数的导数分解为简单函数的导数。

3.3.1 一元链式法则

在这里首先需要说明的一个概念是，一元链式法则中的“一元”表示的是在复合函数中只嵌套了一层函数。假设一个有关函数 f 和 g 的复合函数表达形式：$(f \circ g)(x) = f(g(x))$。**一元链式法则**（Univariate Chain Rule）定义了一元复合函数的求导法则为：

$$\frac{d}{dx}f(g(x)) = f'(g(x)) * g'(x)$$

简单来说，$f(g(x))$ 的导数是：外部函数 f 对内部函数 $g(x)$ 的导数乘以内部函数 g 对 x 的导数。

现在让我们考虑公式 (3.2) 的一个简化形式：

$$C(w_i, b_i) = \frac{1}{2}(\sigma(w_i x + b_i) - y)^2$$

其中参数 $w_i, b_i \in \mathbb{R}$。接着我们令 $C = \frac{1}{2}p^2$，$p = \sigma(q) - y$，$q = w_i x + b_i$，根据一元链式法则我们便有如下求导推理：

$$\begin{aligned}\frac{\partial C}{\partial w_i} &= \frac{\partial C}{\partial p}\frac{\partial p}{\partial q}\frac{\partial q}{\partial w_i} \\ &= p\sigma'(q)x \\ &= (\sigma(q) - y)\sigma'(q)x \\ &= (\sigma(w_i x + b_i) - y)\sigma'(w_i x + b_i)x\end{aligned} \tag{3.4}$$

以及

$$\begin{aligned}\frac{\partial C}{\partial b_i} &= \frac{\partial C}{\partial p}\frac{\partial p}{\partial q}\frac{\partial q}{\partial b} \\ &= p\sigma'(q) \\ &= (\sigma(q) - y)\sigma'(q) \\ &= (\sigma(w_i x + b_i) - y)\sigma'(w_i x + b_i)\end{aligned} \tag{3.5}$$

其中 $w_i \in W = [w_0, \cdots, w_n]$，所以对于单层神经网络我们可以使用一元链式法则分别对不同的 w_i 和 b_i 求导并使用梯度下降法进行迭代。而在多层神经网络中，下一层神经元的输入依赖上一层神经元的输出，所以会存在多个函数嵌套，这在后面章节会介绍。

3.3.2 多元链式法则

多元链式法则（Multivariate Chain Rule）的求导规则定义为：

$$\frac{\partial f(u_1(x), \cdots, u_m(x))}{\partial x} = \sum_{j=1}^{m} \frac{\partial f}{\partial u_j(x)} \bullet \frac{\partial u_j(x)}{\partial x}$$

简单来说，如果一个复合函数中有包含了多个单独的函数，那么多元链式法则告诉我们可以先分别对每个函数求偏导再相加。例如，假设我们有如下复合函数 $f(x, y) = y + e^{xy}$，其中 $x(t) = \cos t$，$y(t) = t^2$。我们通过链式法则对其求导的计算过程为：

$$\frac{df}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} = (ye^{xy}) * (-\sin t) + (1 + xe^{xy}) * 2t$$

3.4 前馈神经网络

前馈神经网络（Feed-Forward Neural Networks）是一种处理非线性数据的强大模型，其基本思想是使用多个神经元且相互之间进行有向的数据传递。

- 前馈神经网络中的神经元以有向无环图（Directed Acyclic Graph）的形式相互连接，这意味着如果数据沿着神经元连接的方向流动，数据将永远不会回到之前访问过的神经元。
- 神经元会被被分组为不同的层（Layer）。一般完整的前馈神经网络包含一个输入层（Input Layer）、多个隐藏层（Hidden Layers）和一个输出层（Output Layer）。通过构建多层神经网络，我们可以得到了具有增强表达能力的高度非线性模型。
- 通常输入层的神经元数量会与输入数据的特征维度对齐，同理输出层的神经元数量会与期望输出数据的特征维度对齐（具体取决于学习任务）。
- 在最简单的情况下，每一层的每个神经元都与下一层中的每个神经元相连，这就组成了一个全连接神经网络（Fully Connected Neural Network）。我们称其为前馈神经网络或多层感知器（Multi-Layer Perceptron，MLP）。

3.4.1 权重和偏置

接下来我们将以图 3.1 的形式展示一个简单的前馈神经网络结构方便大家理解。我们借用图论中的概念，使用边连接不同的神经元（即节点）。每条边都与一个称为权重（weight）的数字相关联，每个神经元都与一个称为偏置（bias）的数字相关联。权重和偏置都为可训练的参数，即我们进行模型训练就是为了确定这些参数。在图 3.1 中，L 表示层数；m 表示网络的宽度，即一层网络所包含的神经元的数据；w_{jk}^{ℓ} 表示第 $\ell-1$ 层第 k 神经元向第 ℓ 层第 j 神经元传递数据的权重；b_j^{ℓ} 表示第 ℓ 层第 j 神经元的偏置；$\sigma(\cdot)$ 表示激活函数，一般只应用在隐藏层和输出层的神经元上。现在让我们以第 ℓ 层第 j 神经元举例，其 z_j^{ℓ} 的计算数学表达式为：

$$z_j^{\ell} = \sum_{k=1}^{m} w_{jk}^{\ell} a_k^{\ell-1} + b_j^{\ell}$$

z_j^{ℓ} 是一个线性函数，如之前的章节所介绍的，为了让模型能够拟合非线性数据，需要将激活函数应用在 z_j^{ℓ} 上得到 a_j^{ℓ}：

$$a_j^{\ell} = \sigma(z_j^{\ell})$$

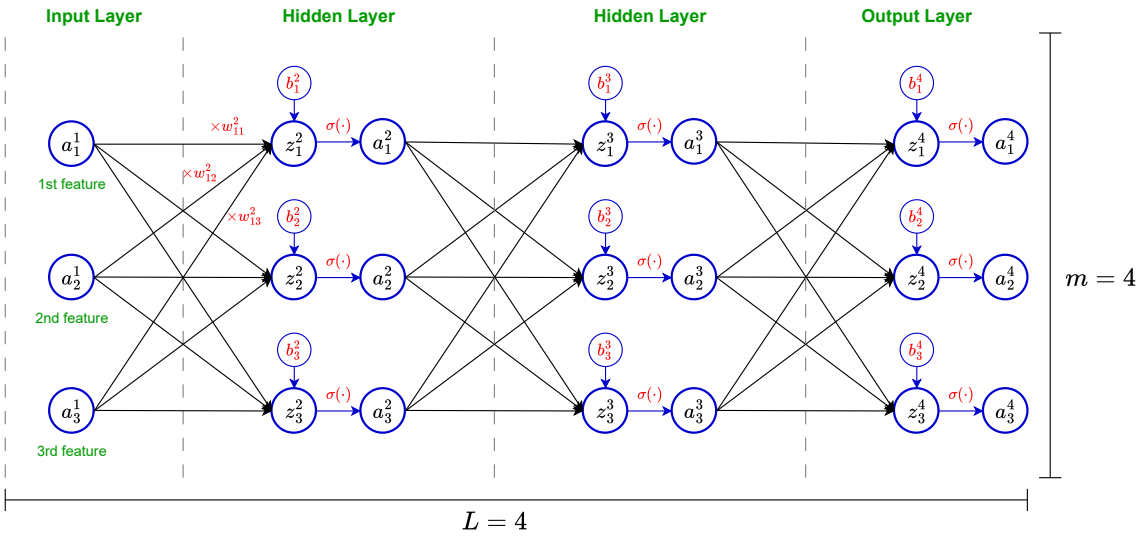


图 3.1: 前馈神经网络的结构示意图

图 3.1 清晰展示了前馈神经网络的数据流动与计算过程，但为了计算与编程方便，我们将其转化为**矢量化**（Vectorization）表示，即用矩阵和向量的形式进行表示与计算。我们令

$$W^{\ell} = \begin{bmatrix} w_{11}^{\ell} & w_{12}^{\ell} \cdots & w_{1m}^{\ell} \\ w_{21}^{\ell} & w_{22}^{\ell} \cdots & w_{2m}^{\ell} \\ \vdots & \ddots & \vdots \\ w_{m1}^{\ell} & w_{m2}^{\ell} \cdots & w_{mm}^{\ell} \end{bmatrix}, \ z^{\ell} = \begin{bmatrix} z_1^{\ell} \\ z_2^{\ell} \\ \vdots \\ z_m^{\ell} \end{bmatrix}, \ a^{\ell} = \begin{bmatrix} a_1^{\ell} \\ a_2^{\ell} \\ \vdots \\ a_m^{\ell} \end{bmatrix}, \ b^{\ell} = \begin{bmatrix} b_1^{\ell} \\ b_2^{\ell} \\ \vdots \\ b_m^{\ell} \end{bmatrix}, \ (\ell = 2, ..., L)$$

于是我们便有如下等式：

$$a^{\ell} = \sigma(z^{\ell}) = \sigma(W^{\ell} a^{\ell-1} + b^{\ell}), \ (\ell = 2, ..., L)$$

这里给大家展开上述等式，验证一下上述的矢量化表示 $z_j^{\ell} = W^{\ell} a^{\ell-1} + b^{\ell}$ 是否正确：

$$\begin{bmatrix} Z_1^{\ell} \\ Z_2^{\ell} \\ \vdots \\ Z_m^{\ell} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{m} w_{1k}^{\ell} a_k^{\ell-1} + b_1^{\ell} \\ \sum_{k=1}^{m} w_{2k}^{\ell} a_k^{\ell-1} + b_2^{\ell} \\ \vdots \\ \sum_{k=1}^{m} w_{mk}^{\ell} a_k^{\ell-1} + b_m^{\ell} \end{bmatrix} = \begin{bmatrix} w_{11}^{\ell} & w_{12}^{\ell} \cdots & w_{1m}^{\ell} \\ w_{21}^{\ell} & w_{22}^{\ell} \cdots & w_{2m}^{\ell} \\ \vdots & \ddots & \vdots \\ w_{m1}^{\ell} & w_{m2}^{\ell} \cdots & w_{mm}^{\ell} \end{bmatrix} \begin{bmatrix} a_1^{\ell-1} \\ a_2^{\ell-1} \\ \vdots \\ a_m^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^{\ell} \\ b_2^{\ell} \\ \vdots \\ b_m^{\ell} \end{bmatrix}$$

接着，在图 3.2 中我们以前馈神经网络的输出层神经元的计算流程举例，让大家能够更直观地理解前向传播过程。

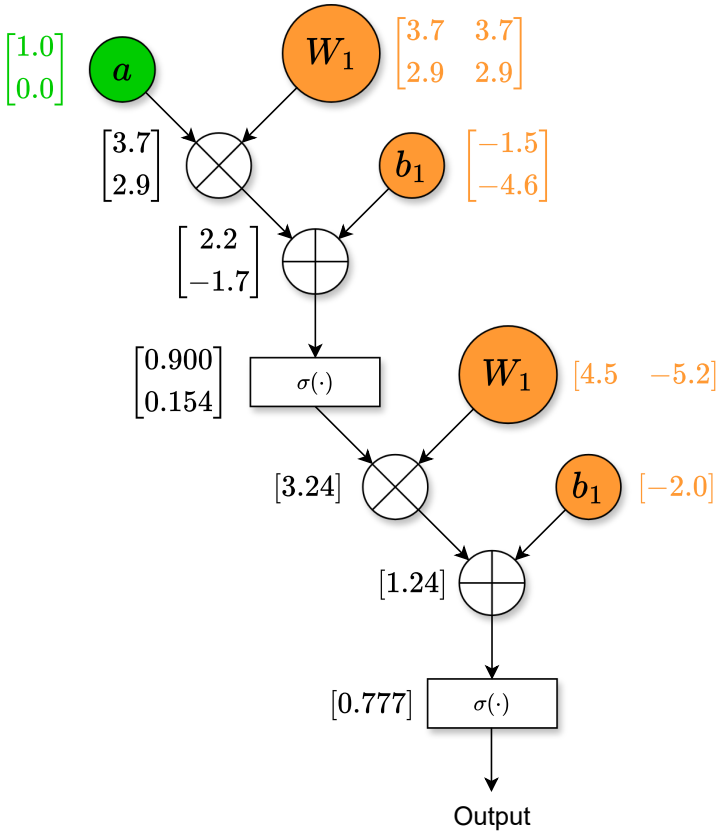


图 3.2: 前向传播与结果输出

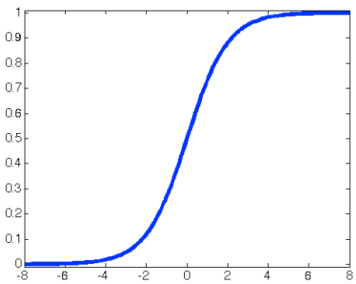
除了图 3.2 中所使用的 Sigmoid 激活函数，这里介绍另外两种激活函数：ReLU 激活函数和 Tanh 激活函数。这些激活函数对于神经网络构建非线性模型至关重要。它们的函数曲线分别如图 3.3 所示。

- ReLU 激活函数计算简单，因为它只涉及最大值算子。它是一个连续函数，但从定义可以看出 ReLU 在零点处不可微。ReLU 因其简单性而被广泛应用于现代神经网络。
- Sigmoid 和 Tanh 是可微的。然而这两个函数使用指数函数，所以需要更多的计算量（指数函数比最大值算子更难计算）。Sigmoid 和 Tanh 的函数曲线非常相似，他们的区别在于 Sigmoid 函数的输出值在 $[0, 1]$ 之间，而 Tanh 的输出值在 $[-1, 1]$ 之间。

笔记 这里可能会有人有疑问：ReLU 是一个线性函数，那么使用 ReLU 作为激活函数的神经网络本质上不还是线性回归模型吗？

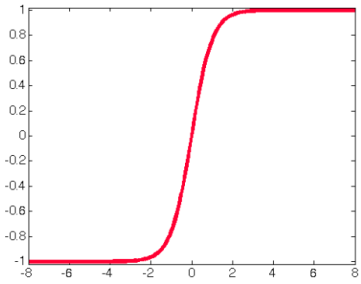
如果读者期望一个直接的回答，那么答案是：是的，线性函数组合的结果仍然是线性函数。ReLU 函数在正区间 $(x > 0)$ 确实是线性的，但关键点在于整个神经网络通过 ReLU 激活函数引入了“分段线性”的性质，从而可以让模型通过组合多个小段的直线去近似逼近非线性的曲线，实现接近非线性回归的能力。让我们进一步阐述并解释这个现象：

1. 单个 ReLU 是分段线性的，但神经网络整体是近似非线性的。虽然 ReLU 有一个比较特殊的“拐点”$(x = 0)$，当 $(x > 0)$ 时 ReLU 是线性的，当 $(x = 0)$ 时 ReLU 是不可导的，当 $(x = 0)$ 时 ReLU 的值是零。正是 $(x = 0)$ 这个“拐点”打破了整个模型的纯粹线性。
2. 在使用 ReLU 的深度网络的前向传播的过程，网络会根据不同输入数据动态地选择激活哪些神经元。对于不同的输入样本，激活函数 ReLU 能够控制神经元的激活（输出 > 0）或关闭（输出 $= 0$），这相当于为每



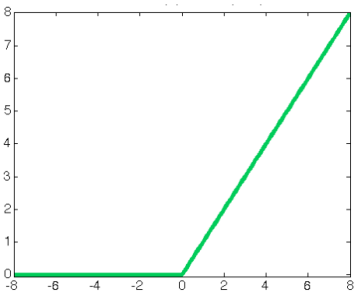
$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

$$(-\infty, \infty) \mapsto (0, 1)$$



$$\text{Tanh}(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

$$(-\infty, \infty) \mapsto (-1, 1)$$



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

$$(-\infty, \infty) \mapsto (0, \infty)$$

图 3.3: 三种激活函数

个输入样本都创建了一个独特的子网络结构。

为了进一步解释 ReLU 激活函数是如何让神经网络拥有用线性回归去近似非线性回归的能力，让我们考虑一个非常简单的例子：

假设我们使用一个 3 层神经网络去拟合 $y = x^2$ 这条非线性函数曲线，其中第一层输入层包含一个神经元，第二层隐藏层包含两个神经元，第三层输出层包含一个神经元，只有隐藏层和输出层使用 ReLU 激活函数。通常一个线性模型难以拟合 $y = x^2$ 这条曲线，但是上述的 3 层神经网络却可以构建 $y = |x|$ 的函数形式去拟合 $y = x^2$。其中的一个解法是令 $w_{11}^2 = 1,\ w_{21}^2 = -1,\ b_1^2 = 0,\ b_2^2 = 0,\ w_{11}^3 = 1,\ w_{12}^3 = 1,\ b_1^3 = 0$。

3.4.2 训练前馈神经网络

现在我们讨论如何从数据集中训练前馈神经网络。正如我们之前讨论的，训练模型就是对权重 W 和偏置 b 进行迭代更新，此时 W^i 表示的是第 i 层神经网络的权重，是一个一维向量；b^i 表示的是第 i 层神经网络的偏置，也是一个一维向量。我们继续使用均方误差来衡量前馈神经网络在样本 (x, y) 上的性能，有如下表达式：

$$C_{(x,y)}(W^2, ..., W^L, b^2, ..., b^L) = (y - \sigma(W^L(\sigma...\sigma(W^2x + b^2)) + b^L))^2$$

于是我们期望最小化的函数表达式为：

$$C(W, b) = \frac{1}{n} \sum_{(x,y)\in S} C_{(x,y)}(W^2, ..., W^L, b^2, ..., b^L)$$

其中 S 表示训练数据的集合。我们使用梯度下降法求解上述最小化问题，即有迭代过程：

$$W^2 \Leftarrow W^2 - \frac{\eta}{n} \sum_{(x,y)\in S} \frac{\partial C_{(x,y)}(W^2, ..., W^L, b^2, ..., b^L)}{\partial W^2}$$

$$W^3 \Leftarrow W^3 - \frac{\eta}{n} \sum_{(x,y)\in S} \frac{\partial C_{(x,y)}(W^2, ..., W^L, b^2, ..., b^L)}{\partial W^3}$$

$$\vdots$$

$$W^L \Leftarrow W^L - \frac{\eta}{n} \sum_{(x,y)\in S} \frac{\partial C_{(x,y)}(W^2, ..., W^L, b^2, ..., b^L)}{\partial W^L}$$

第四章 反向传播算法

内容提要

- 链式法则求解优化
- 反向传播原理

4.1 链式法则求解优化

通过之前的章节我们已经得到公式 (3.4) 和公式 (3.5)，即有：

$$\frac{\partial C}{\partial w_i} = (\sigma(w_i x + b_i) - y)\sigma'(w_i x + b_i)x$$

以及

$$\frac{\partial C}{\partial b_i} = (\sigma(w_i x + b_i) - y)\sigma'(w_i x + b_i)$$

通过上述表达式我们可以应用梯度下降法更新模型的权重和偏置参数。但这些表达式在计算机中实现起来并不高效，其根本原因是存在一些重复计算：$\sigma(w_i x + b_i) - y$ 和 $\sigma'(w_i x + b_i)$ 同时出现在了 $\frac{\partial C}{\partial w_i}$ 和 $\frac{\partial C}{\partial b_i}$ 的计算过程中。因此，我们可以用更结构化的方式进行计算，其基本思想是引入一些中间变量（Intermediate Variables），这样我们就可以存储一些中间结果从而避免重复计算。

我们知道对于单个神经元而言存在以下计算误差的过程：

$$C = \frac{1}{2}(\sigma(wx + b) - y)^2$$

其中 $w, b \in \mathbb{R}$。于是如图 3.1 所展示的计算过程，我们设置一些中间变量：

$$z = wx + b$$

$$a = \sigma(z)$$

$$C = \frac{1}{2}(a - y)^2$$

同时令

$$V_2 = \frac{\partial C}{\partial z} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial z} = (a - y) * \sigma'(z)$$

于是我们便有如下等式：

$$\frac{\partial C}{\partial w} = (\sigma(wx + b) - y)\sigma'(wx + b)x = (a - y) * \sigma'(wx + b)x = V_2 * x$$

$$\frac{\partial C}{\partial b} = (\sigma(wx + b) - y)\sigma'(wx + b) = (a - y) * \sigma'(wx + b)x = V_2$$

通过上述推导过程我们可以发现 $V_2 = \frac{\partial C}{\partial z}$ 被使用了两次，所以我们可以在计算完成后将其存储起来从而避免重复计算。

4.2 反向传播原理

4.2.1 链式法则计算图

这里先用计算图 4.1 直观展示一下链式法则的计算原理。其中 $y_i \in \{y_1, y_2, \cdots, y_n\}$ 是包含 x 函数，z 为 $\{y_1, y_2, \cdots, y_n\}$ 的多元复合函数。根据链式法则我们有：

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \cdot \frac{\partial y_i}{\partial x}$$

同时结合计算图 4.1，这里介绍两种类型的梯度：

- **反向传播梯度**（Backpropagated Gradient）。来自最终输出的梯度，即 $\frac{\partial z}{\partial y_i}$。
- **局部梯度**（Local Gradient）。除输出节点外其他节点的梯度，即 $\frac{\partial y_i}{\partial x}$。

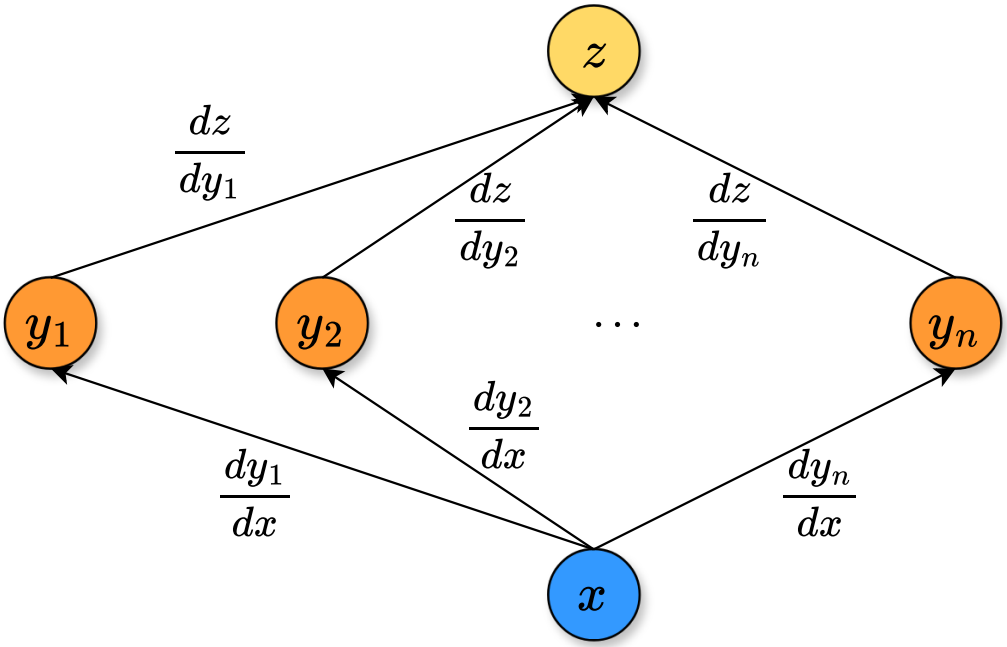


图 4.1: 链式法则计算图

要计算 z 相对于 x 的梯度，我们首先需要计算 z 相对于 x 的反向传播梯度 $\frac{dz}{dy_i}$，然后将其乘以局部梯度 $\frac{dy_i}{dx}$。这样就完成了一条路径的计算，最后我们只需要对所有路径求和。

4.2.2 线性回归计算图

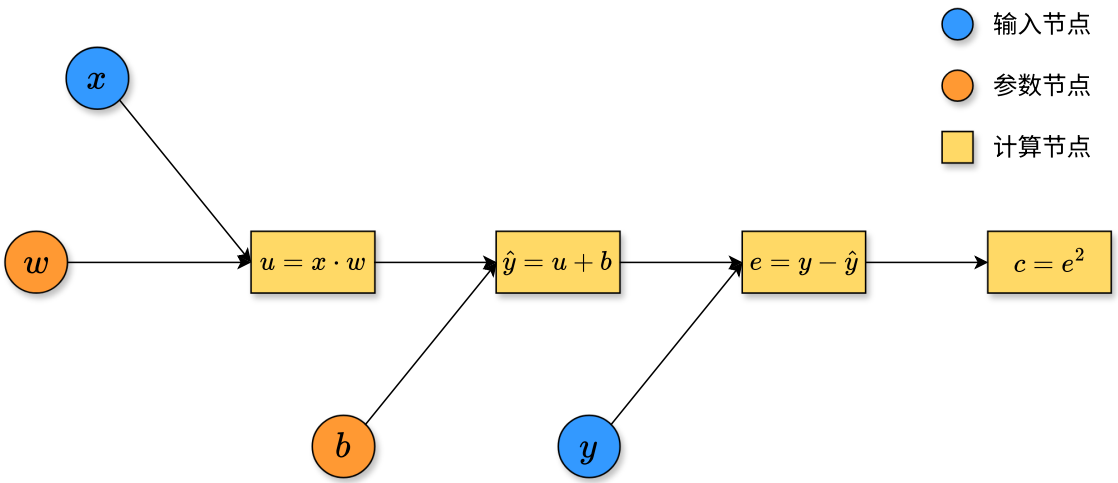


图 4.2: 线性回归计算图

我们通过之前章节的学习可以了解到神经网络是建立在线性回归模型的基础上的，所以现在让我们先探究线性回归模型中的反向传播算法。图 4.2 所示的线性回归简化计算流程包括了两个输入节点 x 和 y；两个参数节点 w 和 b；四个计算节点：中间变量 u，模型预测值 $\hat{y}$，残差项 e，损失 c。

下面我们给出一个实例分别展示计算图 4.2 计算损失 c 的前向传播与反向传播过程。对于前向传播过程 4.3，这里我们令 $w=5$，$x=1$，接着便可以然后计算后续节点的结果 $u=5$，$\hat{y}=6$，$c=16$。

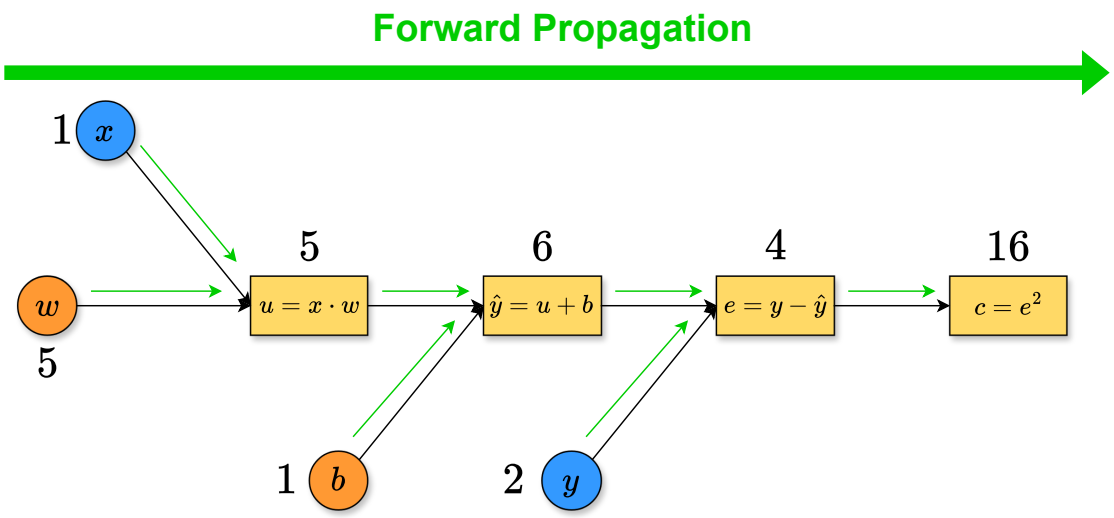


图 4.3: 线性回归的前向传播过程

在通过前向传播得到模型的预测输出 $\hat{y}$ 和损失（即误差估计）c 后，我们现在应用反向传播来计算上述示例的梯度。我们需要依次计算反向传播梯度 $\frac{\partial c}{\partial c}$、$\frac{\partial c}{\partial e}$、$\frac{\partial c}{\partial \hat{y}}$ 和 $\frac{\partial c}{\partial u}$。在此过程中，我们可以使用已经计算的结果。

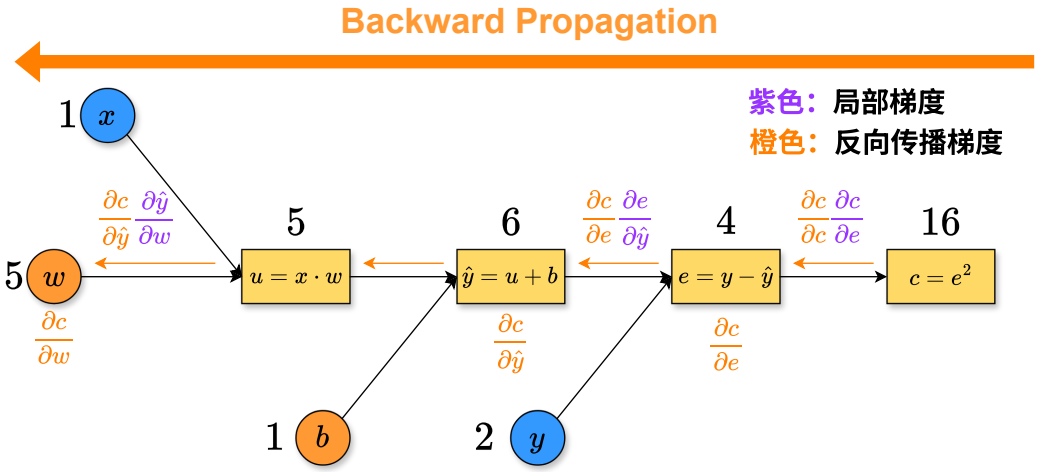


图 4.4: 线性回归的反向传播过程

通过计算图 4.4 所展示的反向传播过程，我们可以清晰地了解如何计算高效各个节点的梯度：

$$\frac{\partial c}{\partial e}=\frac{\partial c}{\partial c}\frac{\partial c}{\partial e}=\frac{\partial c}{\partial c}\frac{\partial e^2}{\partial e}=1\cdot 2e=8$$
$$\frac{\partial c}{\partial \hat{y}}=\frac{\partial c}{\partial e}\frac{\partial e}{\partial \hat{y}}=\frac{\partial c}{\partial e}\frac{\partial(\hat{y}-y)}{\partial \hat{y}}=2e\cdot 1=8$$
$$\frac{\partial c}{\partial w}=\frac{\partial c}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}=\frac{\partial c}{\partial \hat{y}}\frac{\partial xw}{\partial w}=2e\cdot x=8$$

4.3 反向传播梯度表达式

通过计算图 3.1 我们可以清晰地理解单个神经元的计算表达式为：

$$z_j^\ell = \sum_{k=1}^{m} w_{jk}^\ell a_k^{\ell-1} + b_j^\ell$$

那么该神经元有关权重和偏置的梯度可以表示为：

$$\frac{\partial C}{\partial w_{jk}^\ell} = \frac{\partial C}{\partial z_j^\ell} \cdot \frac{\partial z_j^\ell}{\partial w_{jk}^\ell} = \frac{\partial C}{\partial z_j^\ell} \cdot a_k^{\ell-1}$$

$$\frac{\partial C}{\partial b_j^\ell} = \frac{\partial C}{\partial z_j^\ell} \cdot \frac{\partial z_j^\ell}{\partial b_j^\ell} = \frac{\partial C}{\partial z_j^\ell}$$

其中 $a_k^{\ell-1}$ 为前一层神经网络中的神经元的输出，那么通过上式可以直观地看出我们只要计算出反向传播梯度 $\frac{\partial C}{\partial z_j^\ell}$ 就能够得到该神经元有关权重和偏置的梯度，进而利用梯度下降法更新模型的权重和偏置。

4.3.1 矢量化

为了方便计算和直观表示，现在我们再次结合计算图 3.1 将神经网络的梯度求解进行矢量化表示（即转化成雅可比矩阵的形式）。现在我们令：

$$\delta_j^\ell = \frac{\partial C}{\partial z_j^\ell}$$

于是我们便有：

$$\frac{\partial C}{\partial w_{jk}^\ell} = \delta_j^\ell \cdot a_k^{\ell-1}$$

$$\frac{\partial C}{\partial b_j^\ell} = \delta_j^\ell$$

那么对于某一层的神经网络的权重梯度求解有如下表达形式：

$$\begin{aligned}
\frac{\partial C}{\partial W^\ell} &= \begin{bmatrix} \frac{\partial C}{\partial w_{11}^\ell} & \cdots & \frac{\partial C}{\partial w_{1m}^\ell} \\ \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial w_{m1}^\ell} & \cdots & \frac{\partial C}{\partial w_{mm}^\ell} \end{bmatrix} \\
&= \begin{bmatrix} \delta_1^\ell a_1^{\ell-1} & \cdots & \delta_1^\ell a_m^{\ell-1} \\ \vdots & \ddots & \vdots \\ \delta_m^\ell a_1^{\ell-1} & \cdots & \delta_m^\ell a_m^{\ell-1} \end{bmatrix} \\
&= \begin{bmatrix} \delta_1^\ell \\ \vdots \\ \delta_m^\ell \end{bmatrix} \begin{bmatrix} a_1^{\ell-1}, \cdots, a_m^{\ell-1} \end{bmatrix} \\
&= \delta^\ell (a^{\ell-1})^\mathsf{T}
\end{aligned}$$

同理可得偏置的梯度表达式：

$$\frac{\partial C}{\partial b^\ell} = \delta^\ell$$

4.4 反向传播梯度的递归关系

通过上一节的推导我们知道了要计算单层神经网络的梯度，反向传播梯度 δ^ℓ 的求解是关键。那么在本节我们将给大家展示如何递归地求解完整神经网络中各神经层的梯度。简单来说，我们首先计算输出层的反向传播

梯度 δ^L，接着将 δ^{L-1} 看作 δ^L 的一个函数 $\delta^{L-1} = f(\delta^L)$ 进行求解，然后重复此递归过程直到到达输入层。

4.4.1 输出层的反向传播梯度

输出层是神经网络的最后一层，结合计算图 3.1 我们假设其中某一神经元的输出为 $a_j^L = \sigma(z_j^L)$，其中 $\sigma(\cdot)$ 代指任意激活函数。根据链式法则我们可以得到如下输出层神经元的反向传播梯度：

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \cdot \frac{\sigma(z_j^L)}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)\end{aligned}$$

其中 $\sigma'(z_j^L)$ 和 $\frac{\partial C}{\partial a_j^L}$ 我们都可以较容易地得到解析解，其解析解表达式取决于具体使用的激活函数 $\sigma(\cdot)$ 和损失函数 $C(\cdot)$。例如对于一个 m 特征维度的回归问题，我们继续使用如下均方误差作为损失函数：

$$C(a_1^L, ..., a_m^L) = \frac{1}{2m}\sum_{k=1}^{m}(y_k - a_k^L)^2$$

那么对于第 j 个神经元有如下 $\frac{\partial C}{\partial a_j^L}$ 解析解表达式：

$$\frac{\partial C}{\partial a_j^L} = -\frac{1}{m}(y_j - a_j^L)$$

于是计算输出层反向传播梯度的一个关键值 δ_j^L 就可以被求解出来了，其表达式为：

$$\delta_j^L = -\frac{1}{m}\sigma'(z_j^L)(y_j - a_j^L)$$

笔记 损失函数（Loss Function）是一个衡量模型预测值与真实值之间差异程度的函数。它的核心作用是为模型提供一个明确的优化目标：通过调整模型参数，最小化这个损失值。

4.4.2 隐藏层的反向传播梯度

现在我们考虑隐藏层的反向梯度求解，根据之前的学习我们可以总结以下关系表达式：

$$z_k^{\ell+1} = \sum_j (w_{kj}^{\ell+1} a_j^\ell + b_k)$$

$$a_j^\ell = \sigma(z_j^\ell)$$

这里我们发现 $z_k^{\ell+1}$ 其实是 a_j^ℓ 的一个函数，所以根据链式法则我们有如下 $\frac{\partial C}{\partial a_j^\ell}$ 的表达式：

$$\frac{\partial C}{\partial a_j^\ell} = \sum_k \left(\frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial a_j^\ell}\right)$$

进而得到 δ_j^ℓ 的表达式 (4.1) 为：

$$\begin{aligned}\delta_j^\ell &= \frac{\partial C}{\partial z_j^\ell} = \frac{\partial C}{\partial a_j^\ell} \cdot \frac{\partial a_j^\ell}{\partial z_j^\ell} \\ &= \left(\sum_k \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial a_j^\ell}\right)\sigma'(z_j^\ell) \\ &= \sigma'(z_j^\ell)\sum_k (\delta_k^{\ell+1} w_{kj}^{\ell+1})\end{aligned} \tag{4.1}$$

笔记 这里需要注意以下几点：

1. $\ell \in \{2, 3, \cdots, L-1\}$，且由于我们从输出层开始反向计算梯度，所以隐藏层的反向梯度求解过程中 $\delta_k^{\ell+1}$ 的值是已知的。
2. 公式 (4.1) 表明 ℓ 层中的反向传播梯度可以根据 $\ell+1$ 层中的反向传播梯度来计算。
3. 通过计算得到的 δ^L 可以用于计算 δ^{L-1}，而 δ^{L-1} 可以用于计算 δ^{L-2}，以此进行递归求解。

我们总结一下目前推理论证得到的信息：损失函数对于 $w_{jk}^{\ell+1}$ 和 b_j^{ℓ} 的梯度可以通过反向传播梯度来表示，即：

$$\frac{\partial C}{\partial w_{jk}^{\ell}} = \delta_j^{\ell} \cdot a_k^{\ell-1}$$

$$\frac{\partial C}{\partial b_j^{\ell}} = \delta_j^{\ell}$$

而反向传播梯度可以用递归的方式从输出层开始反向递归求解，即：

$$\delta_j^{\ell} = \begin{cases} \sigma'(z_j^L) \cdot \frac{\partial C}{\partial a_j^L}, & \text{if } \ell = L \\ \sigma'(z_j^{\ell}) \sum_k (\delta_k^{\ell+1} w_{kj}^{\ell+1}), & otherwise. \end{cases}$$

4.4.3 矢量化

同样为了计算效率，我们将上述过程进行矢量化表示。我们使用 $\odot$ 表示矩阵的哈达玛积（Hadamard product），即矩阵对应元素相乘。例如 $[1,2] \odot [3,4] = [3,8]$。

当 $\ell < L$ 时，我们有如下输出层的方向传播梯度 δ^L 的矢量化表达形式：

$$\begin{bmatrix} \delta_1^L \\ \vdots \\ \delta_m^L \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial a_1^L} \cdot \sigma'(z_1^L) \\ \vdots \\ \frac{\partial C}{\partial a_m^L} \cdot \sigma'(z_m^L) \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial a_1^L} \\ \vdots \\ \frac{\partial C}{\partial a_m^L} \end{bmatrix} \odot \begin{bmatrix} \sigma'(z_1^L) \\ \vdots \\ \sigma'(z_m^L) \end{bmatrix} = \nabla_{a^L} C \odot \sigma'(Z^L)$$

当 $\ell < L$ 时，我们有如下隐藏层的方向传播梯度 δ^L 的矢量化表达形式：

$$\begin{bmatrix} \delta_1^{\ell} \\ \vdots \\ \delta_m^{\ell} \end{bmatrix} = \begin{bmatrix} \sigma'(z_1^{\ell}) \sum_k (\delta_k^{\ell+1} w_{k1}^{\ell+1}) \\ \vdots \\ \sigma'(z_m^{\ell}) \sum_k (\delta_k^{\ell+1} w_{km}^{\ell+1}) \end{bmatrix} = \begin{bmatrix} \sigma'(z_1^{\ell}) \\ \vdots \\ \sigma'(z_m^{\ell}) \end{bmatrix} \odot \begin{bmatrix} \sum_k (w^{\ell+1})_{1k}^{\mathsf{T}} \cdot \delta_k^{\ell+1} \\ \vdots \\ \sum_k (w^{\ell+1})_{mk}^{\mathsf{T}} \cdot \delta_k^{\ell+1} \end{bmatrix} = ((W^{\ell+1})^{\mathsf{T}} \cdot \delta^{\ell+1}) \odot \sigma'(Z^{\ell})$$

总结来说我们有如下方向传播梯度的矢量化表达形式：

$$\delta^{\ell} = \begin{cases} \nabla_{a^L} C \odot \sigma'(Z^L), & \text{if } \ell = L \\ ((W^{\ell+1})^{\mathsf{T}} \cdot \delta^{\ell+1}) \odot \sigma'(Z^{\ell}), & otherwise. \end{cases}$$

4.5 多层神经网络的反向传播过程

如图 4.5 所示，现在我们可以明确地阐述反向传播算法了。首先，我们利用前向传播计算模型的输出 a^L，然后我们利用反向传播过程计算反向传播梯度 $\delta^L, \delta^{L-1}, ..., \delta^2$，进而求解损失函数对于权重和偏置的梯度并使用梯度下降法更新模型参数。

4.6 前馈神经网络的梯度下降过程

反向传播算法提供了一种计算梯度的有效方法。我们可以将这些梯度放入梯度下降的框架中。注意，上述算法计算的是单个训练样本 (x, y) 的损失函数梯度。数据集 S 通常包含 n 个样本。

假设有一批训练样本 $S = \{(x^{(1)}, y^{(1)}), \cdots, (x^{(n)}, y^{(n)})\}$，我们定义的损失函数为 $C = \frac{1}{n} \sum_{i=1}^{n} C_i$，其中 C_i 代

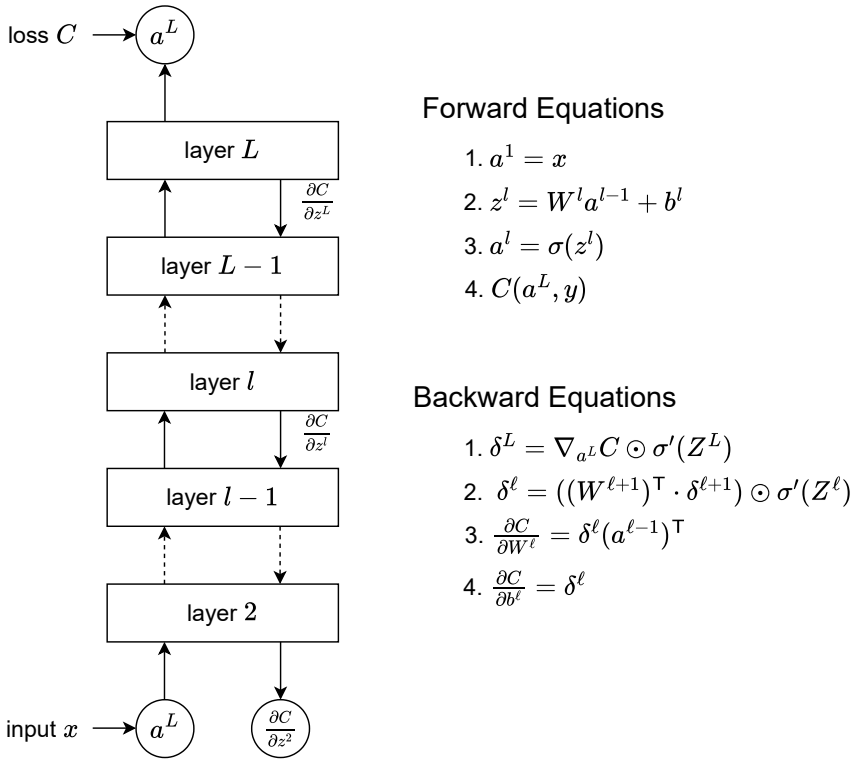


图 4.5: 多层神经网络的反向传播过程示意图

表第 i 个样本的损失函数。反向传播算法可以求解出损失函数的总体梯度：

$$\frac{\partial C}{\partial W^\ell} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial C_i}{\partial W^\ell}$$

$$\frac{\partial C}{\partial b^\ell} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial C_i}{\partial b^\ell}$$

接着我们便可以使用梯度下降法（或者其它基于梯度的优化方法）来优化权重和偏置的参数。在实际中训练的数据量通常很大，所以普遍使用小批量随机梯度下降法。

第五章 习题集

5.1 习题

练习一

1. (Gradient) The gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ is denoted by ∇f. Which of the following statements is correct?
   (A) The gradient ∇f is a vector with positive elements.
   (B) The gradient ∇f is a function which maps vectors to vectors.
   (C) The gradient ∇f is a function which maps vectors to scalars.
   (D) The gradient ∇f is a vector with negative elements.
2. (Minibatch SGD) Which of the following statement is not true (n is the sample size)?
   (A) If the batch size is 1, then minibatch SGD becomes SGD.
   (B) If the batch size is n, then minibatch SGD (sampling without replacement) becomes gradient descent.
   (C) If the batch size is n, then minibatch SGD (sampling with replacement) becomes gradient descent.
   (D) As compared to SGD, minibatch SGD builds a better gradient estimator by using more examples.
3. (Propagation) Which of the following statement is not true?
   (A) Forward propagation goes from the input layer to the output layer.
   (B) Backward propagation aims to to compute a gradient of a function.
   (C) Backward propagation is based on a chain rule.
   (D) Forward and backward propagation are two independent processes.
4. (Multi-Layer Perceptron) Consider a fully-connected MLP with 4 layers: 1 input layer, 1 output layer and 2 hidden layers. Assume the input layer has 6 nodes, the two hidden layers have 5 and 10 nodes respectively, and the output layer has 3 nodes. How many trainable parameters are there in this MLP?
   (A): 100
   (B): 128
   (C): 160
   (D): 180
5. (Minimization) Let a, b, c, d be four numbers. Consider two points $x_1 = (a, b)^\mathsf{T}$ and $x_2 = (c, d)^\mathsf{T}$. Consider the following minimization problem: $\min_{X \in \mathbb{R}^2} ||X - X_1||_2^2 + 2||X - X_2||_2^2$. Which of the following is the minimizer?
   (A): $\begin{bmatrix} \frac{a}{3} + \frac{c}{3} \\ \frac{b}{3} + \frac{d}{3} \end{bmatrix}$
   (B): $\begin{bmatrix} \frac{a}{2} + \frac{c}{2} \\ \frac{b}{2} + \frac{d}{2} \end{bmatrix}$
   (C): $\begin{bmatrix} \frac{a}{3} + \frac{2c}{3} \\ \frac{b}{3} + \frac{2d}{3} \end{bmatrix}$
   (D): $\begin{bmatrix} \frac{2a}{3} + \frac{c}{3} \\ \frac{2b}{3} + \frac{d}{3} \end{bmatrix}$
6. (Gradient Descent) Consider a binary classification problem with the following training examples:

```
X = [-0.5, -1, 0.5, -0.2, -0.8, -0.15, -1, 0;
    0.25, -0.1, 0, -0.3, 0, -0.5, 0, -0.25;
    -0.8, -0.1, 0.25, 0.2, -0.8, 0.05, -1, 0.25;
    -1, -1, 0.1, 0, -1, -0.25, -1, 0.1]
```

```
Y = [1, 1, -1, -1, 1, -1, 1, -1]
```

Suppose we consider a linear model for classification $X \rightarrow W^{\mathsf{T}}X$, where $W = (w_1, w_2, w_3, w_4)^{\mathsf{T}} \in \mathbb{R}^4$. We minimize the objective function (for simplicity we do not consider the bias in the linear model):

$$C(W) = \frac{1}{n}\sum_{i=1}^{n} C_i(W)$$

where

$$C_i(W) = \begin{cases} 0, & \text{if } y^{(i)}W^{\mathsf{T}}X^{(i)} \geq 1 \\ \frac{1}{2}(1 - y^{(i)}W^{\mathsf{T}}X^{(i)})^2, & otherwise. \end{cases}$$

Suppose we run gradient descent with $W^{(0)} = (0, 0, 0, 0)^{\mathsf{T}}$ and step size $\eta_t = \eta = 0.5$. What is $W^{(25)}$ (We only preserve three digits after the decimal point)?

(A): $\begin{bmatrix} -0.721 \\ 1.262 \\ -1.204 \\ -0.484 \end{bmatrix}$

(B): $\begin{bmatrix} -0.527 \\ 1.220 \\ -1.047 \\ -0.445 \end{bmatrix}$

(C): $\begin{bmatrix} -0.536 \\ 1.260 \\ -1.257 \\ -0.565 \end{bmatrix}$

(D): $\begin{bmatrix} -0.637 \\ 1.120 \\ -1.056 \\ -0.395 \end{bmatrix}$

7. (Stochastic Gradient Descent) Let us consider the Problem 6, the same objective function. Suppose we run SGD to minimize loss function. Let $W^{(0)} = (0, 0, 0, 0)^{\mathsf{T}}$ and $\eta_t = \eta = 0.1$. Let $i_t = (t \mod 8) + 1$, i.e., $i_0 = 1, i_7 = 8, i_8 = 1$. What is $W^{(80)}$ (We only preserve three digits after the decimal point)?

(A): $\begin{bmatrix} -0.469 \\ 0.890 \\ -0.799 \\ -0.449 \end{bmatrix}$

(B): $\begin{bmatrix} -0.480 \\ 0.706 \\ -0.879 \\ -0.549 \end{bmatrix}$

(C): $\begin{bmatrix} -0.569 \\ 0.990 \\ -0.579 \\ -0.479 \end{bmatrix}$

(D): $\begin{bmatrix} -0.669 \\ 0.706 \\ -0.764 \\ -0.462 \end{bmatrix}$

8. (Momentum) Let us consider the Problem 6, the same objective function. Suppose we run Momentum to minimize loss function. Let $W^{(0)} = (0, 0, 0, 0)^\mathsf{T}$ and $\eta_t = \eta = 0.5$. Let the parameter α in the Momentum be 0.5. What is $W^{(25)}$ (We only preserve three digits after the decimal point)?

(A): $\begin{bmatrix} -0.798 \\ 1.939 \\ -1.697 \\ -0.408 \end{bmatrix}$

(B): $\begin{bmatrix} -0.798 \\ 1.849 \\ -1.667 \\ -0.422 \end{bmatrix}$

(C): $\begin{bmatrix} -0.698 \\ 1.839 \\ -1.657 \\ -0.402 \end{bmatrix}$

(D): $\begin{bmatrix} -0.598 \\ 1.829 \\ -1.557 \\ -0.502 \end{bmatrix}$

9. (Adaptive Gradient Descent) Let us consider the Problem 6, the same objective function. Suppose we run AdaGrad to minimize loss function. Let $W^{(0)} = (0, 0, 0, 0)^\mathsf{T}$ and $\eta_t = \eta = 0.1$. Let the parameter δ in the AdaGrad be 10^{-6}. Let $i_t = (t \mod 8) + 1$, i.e., $i_0 = 1, i_7 = 8, i_8 = 1$. What is $W^{(80)}$ (We only preserve three digits after the decimal point)?

(A): $\begin{bmatrix} -0.478 \\ 0.849 \\ -0.722 \\ -0.380 \end{bmatrix}$

(B): $\begin{bmatrix} -0.498 \\ 0.858 \\ -0.612 \\ -0.360 \end{bmatrix}$

(C): $\begin{bmatrix} -0.398 \\ 0.868 \\ -0.623 \\ -0.354 \end{bmatrix}$

(D): $\begin{bmatrix} -0.698 \\ 0.862 \\ -0.632 \\ -0.352 \end{bmatrix}$

10. (Perceptron) Let us consider the dataset in Problem 6. Suppose we apply the Perceptron algorithm to find a linear model. Suppose we initialize $W^{(0)} = (0, 0, 0, 0)^\mathsf{T}$. Suppose we go through the dataset once in this order from

$(x^{(1)}, y^{(1)})$ to $(x^{(8)}, y^{(8)})$. What is $W^{(8)}$?

(A): $\begin{bmatrix} -0.45 \\ 0.70 \\ -0.82 \\ -0.70 \end{bmatrix}$

(B): $\begin{bmatrix} -0.40 \\ 0.75 \\ -0.84 \\ -0.70 \end{bmatrix}$

(C): $\begin{bmatrix} -0.35 \\ 0.75 \\ -0.85 \\ -0.75 \end{bmatrix}$

(D): $\begin{bmatrix} -0.48 \\ 0.70 \\ -0.75 \\ -0.85 \end{bmatrix}$

练习二

1. We build a simple classification network with the goal of classifying 32 by 32 greyscale images into 10 classes. The network consists of 3 layers. The first two layers are convolutional layers, the output of the second convolutional layer is then stretched into a 1-D vector, and a fully connected layer is applied to obtain the output.For the convolution layers:
   - Ki corresponds to the number of filters for the ith layer.
   - Fi corresponds to filter size (assuming square filters, and the correct number of channels) for the ith layer.
   - Si corresponds to the stride for the ith layer.
   - Pi corresponds to the zero padding for the ith layer.

   We describe the fully connected layer using the format $n_{in} \rightarrow n_{out}$, where n_{in} is the number of neurons input into the layer, and n_{out} is the number of neurons output from the layer. In the network architectures described, check all feasible answers shown below.

   1. Layer 1: K1 = 1, F1 = 7, S1 = 1, P1 = 0 Layer 2: K2 = 2, F2 = 11, S2 = 1, P2 = 0 Layer 3: 512 -> 10
   2. Layer 1: K1 = 1, F1 = 3, S1 = 1, P1 = 0 Layer 2: K2 = 5, F2 = 3, S2 = 1, P2 = 1 Layer 3: 900 -> 10
   3. Layer 1: K1 = 1, F1 = 3, S1 = 1, P1 = 0 Layer 2: K2 = 2, F2 = 11, S2 = 1, P2 = 0 Layer 3: 1152 -> 11
   4. Layer 1: K1 = 1, F1 = 3, S1 = 1, P1 = 0 Layer 2: K2 = 1, F2 = 3, S2 = 1, P2 = 1 Layer 3: 900 -> 10

2. Given the following statements relating to CNNs. Check all true statements:

   1. CNNs are more efficient than MLPs for inputs with a large number of features.
   2. We can use Stochastic Gradient Descent to optimize both convolutional neural networks and fully connected networks.
   3. CNNs use a different version of backpropagation compared to fully connected networks called ”backpropagation through space”.
   4. The gradient for each weight in the convolutional kernel only ever depends on one pixel in the image/feature map it is applied to.
   5. CNNs can be made more memory efficient by downsampling the feature maps.

3. We apply N convolutional kernels of dimension H × W × C (H, W and C are the height, width and number of

channels respectively) with stride = 1 and padding = 0 to a 11 × 11 colour image. This results in a 7 by 7 feature map with 10 channels. What are the correct values for N, H, W, and C?

1. N = 3, H = 3, W = 3 and C = 10
2. N = 10, H= 5, W = 5 and C = 3
3. N = 10, H = 5, W=5 and C = 1
4. N = 10, H = 3, W = 3 and C=1
5. N = 3, H = 5, W = 5 and C = 10

4. Within a CNN we apply a 3 × 3 kernel with stride = 1 and padding = 1 to a 50 × 50 greyscale image to obtain an output image of size H × W. In order to produce a H × W output from the same input image, how many more parameters (including biases) would a fully connected layer require than the aforementioned convolutional layer?
   1. 6,250,490
   2. 2490
   3. 6,252,490
   4. 6,250,000
   5. 2500
5. Given the following statements about RNNs. Check all True statements:
   1. The forward pass of an RNN can be applied to individual elements of the same sequence at exactly the same time.
   2. RNNS are primarily used to model data with a grid like structure.
   3. RNNs are only suitable for sequence-to-sequence modelling.
   4. As well as a hidden state LSTMs also have an additional state called the cell state.
   5. RNNs can be used to learn latent representations of sequential data.
6. Given the following statements about Autoencoders. Check all True statements:
   1. Autoencoders implicitly perform clustering.
   2. Autoencoders can generate unseen data effectively.
   3. The encoder of an autoencoder can be used to train a classification network with limited training data.
   4. If an autoencoder was applied to the MNIST dataset, each class would have single latent representation.
   5. There may be some sparse locations in the latent space produced by an autoencoder.
7. Given the following statements about the bottleneck layer found in Autoencoders. Check all True statements:
   1. The bottleneck forces the autoencoder to learn fine-grained details specific to individual images.
   2. The bottleneck layer should be the same size as the output of the network.
   3. The number of neurons in the bottleneck layer and the length of the latent representation are the same in an autoencoder.
   4. The bottleneck layer should be smaller than both the input and output layers.
   5. We need a bottleneck layer to make sure the network doesn't learn the identity function.
8. Given the following statements about latent variables/features. Check all True statements:
   1. If we re-train an autoencoder twice with different random seeds it will learn the same features in latent space.
   2. Increasing the size of the latent vector reduces the reconstruction loss.
   3. Latent variables can be directly observed in the raw data.
   4. Training an autoencoder to represent latent variables is an example of unsupervised learning.
9. Given the following statements about the reparameterization trick. Check all True statements:
   1. The reparameterization trick for variational autoencoders utilises that we can sample a normally distributed variable with mean μ and variance σ^2 using the following: $z = \mu + \sigma^2\epsilon$, where $\epsilon \sim N(0, 1)$
   2. The reparameterization trick is required to allow gradients to flow through the sampling procedure.
   3. The reparameterization trick is only used at inference time, and not training time.
   4. The reparameterization trick for VAEs is required make sampling more efficient.

10. Given the following statements relating to variational autoencoders. Check all True statements:
    1. KL divergence is measured between the reconstructed image and the ground truth image.
    2. Adding KL divergence to the loss function improves the reconstruction accuracy of VAEs.
    3. The regularisation loss a function of the encoder parameters only.
    4. KL divergence ensures that the predicted distribution in the latent space is a Uniform distribution.
    5. Given two datapoints x_1 and x_2 and their latent means μ_1 and μ_2 respectively, to interpolate between x_1 and x_2 we decode the latent variable $\mu^* = (1-\alpha)\mu_1 + \alpha\mu_2$ for $\alpha \in [0,1]$

5.2 解

练习一

1. B
2. C
3. D
4. B. Solution: We have $5+10+3=18$ bias, $6*5+5*10+10*3=110$ weight, so we have 128 trainable parameters.
5. C. Solution: We set $f(X) = ||X-X_1||_2^2 + 2||X-X_2||_2^2$, that is $f(x_1,x_2) = (x_1-a)^2 + (x_2-b)^2 + 2(x_1-c)^2 + 2(x_2-d)^2$,then we have:
$$\begin{cases} \frac{\partial f(x_1,x_2)}{\partial x_1} = 2(x_1-a)+4(x_1-c)=0 \\ \frac{\partial f(x_1,x_2)}{\partial x_2} = 2(x_2-b)+4(x_2-d)=0 \end{cases} \rightarrow \begin{cases} x_1 = \frac{a}{3}+\frac{2c}{3} \\ x_2 = \frac{b}{3}+\frac{2d}{3} \end{cases}$$
6. B. See GitHub (Exercise_1_6.m)
7. A. See GitHub (Exercise_1_7.m)
8. D. See GitHub (Exercise_1_8.m)
9. A. See GitHub (Exercise_1_9.m)
10. C. See GitHub (Exercise_1_10.m)

练习二

1. 1,4
2. 1,2,5
3. 2
4. 3 Solution: According to the question, the size of output image is 50×50.For convolutional layer, the number of parameters (including biases) is
   Per filter:
   $$F \times F \times D$$
   Total number of parameters: weights + basis
   $$F \times F \times D \times K + K$$
   where F is the size of kernel, D is the number of channels of input, K is the number of filters. Thus, in this question, the number of parameters (including biases) is $3*3+1=10$.For fully connected layer,both size of input and output are 50×50, sothe number of parameters (including biases) is $(50 \times 50) \times 50 \times 50 + 50 \times 50 = 6252500$.To sum up, the answer is 6252490.
5. 4,5
6. 1,2,4

7. 3,4,5
8. 2,4
9. 2
10. 3,5