深度学习入门

咩咩宇

2025 年 9 月

目录

1 线性回归与梯度下降法

1.1 线性回归模型定义

设数据集 $D = \{\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_m, y_m\}\}$, 其中 $x_i = (x_{i1}; x_{i2}; \ldots; x_{id}), y_i \in \mathbb{R}$, 线性回归就是是学的一个线性模型尽可能的准确的预测实际输出值。

通俗的说就是求属性和结果之间的线性关系。线性回归模型的函数表达式可以用下面的式子来表达:

$$f(x) = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

当然也可以用向量的形式来表达:

$$f(x) = w^Tx + b$$

从上面的式子来，可以得出线性回归模型就是要求得到一组最优的 w_i 和 b 来确定线性模型，使这个模型无限逼近现有的数据 x_i 和结果 $f(x_i)$ 之间的关系。

1.2 误差函数

现在用一个公式计算输出和真实值间的误差:

$$loss = (f(x) - y)^2 = (wx - y)^2$$

当然数据是有很多的，要计算所有数据真实值和输出之间的误差和并计算出平均值，这个函数为均方误差函数，也是线性回归模型的损失函数。

$$J(x) = \frac{1}{2m}\sum_{i=1}^{m}(f(x_i) - y_i)^2$$

线性回归模型就是寻找一组参数使误差函数最小

1.3 穷举法

穷举 w 的值，去找到使误差最小的 w

1.4 最小二乘法

1.4.1 简介

最小二乘法（Least Squares Method）是一种数学优化技术，它通过最小化误差的平方和来寻找数据的最佳函数匹配。对于给定的一组观测数据 (x_i, y_i)（$i = 1, 2, \ldots, n$），假设拟合函数为 $y = f(x; \theta)$（其中 θ 是待确定的参数，如线性拟合中的斜率和截距），最小二乘法要找到合适的 θ，使得以下目标函数的值最小:

可以将其想象成「找最贴合数据的规律」的方法。例如，对于一组零散的数据点（如不同身高对应的体重数据），我们希望拟合一条线（或曲线），使这些点尽可能「靠近」这条线。具体而言：

1. 计算每个数据点到拟合线的「误差」（如实际值与拟合值的差值）；

2. 对所有误差进行平方（避免正负误差抵消，同时放大较大误差的影响）；

3. 调整拟合函数的参数，使误差的平方和最小，此时的拟合线即为最优匹配。

「二乘」对应数学中的「平方」操作（历史翻译中「乘」可表示平方含义），而「最小」则指该方法的核心是最小化「误差的平方和」。因此，这种通过最小化误差平方和来确定最优拟合模型的方法被称为「最小二乘法」。

$$\frac{\partial \text{Loss}}{\partial w} = \frac{\partial (wx - y)^2}{\partial w} = 2(wx - y)x$$

1.4.2 向量形式求解

线性回归模型的向量表达式如下:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$$

为了方便理解原理的同时也方便计算，我们将参数 b 纳入到矩阵 $\boldsymbol{w}$ 中，此时数据特征矩阵 $\boldsymbol{x}$ 则为：

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1d} & 1 \\ x_{21} & x_{22} & \dots & x_{2d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{md} & 1 \end{pmatrix}$$

矩阵 $\boldsymbol{w}$ 为：

$$\boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_m \\ b \end{pmatrix}$$

得到线性回归模型的向量表达式如下式所示：

$$f(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{w}$$

对其求解最优解：

很显然 $\boldsymbol{X}$ 和 $\boldsymbol{w}$ 都是一个矩阵，利用最小二乘法对这个矩阵求最优的 $\boldsymbol{w}$ 矩阵参数。计算的步骤如下所示：

$$\begin{aligned} J(w) &= \frac{1}{2}(J(w)-Y)^2 \\ &= \frac{1}{2}(Xw-Y)^2 \\ &= \frac{1}{2}(Xw-Y)^{\mathrm{T}}(Xw-Y) \\ &= \frac{1}{2}(w^{\mathrm{T}}X^{\mathrm{T}}-Y^{\mathrm{T}})(Xw-Y) \\ &= \frac{1}{2}(w^{\mathrm{T}}X^{\mathrm{T}}Xw-Y^{\mathrm{T}}Xw-w^{\mathrm{T}}X^{\mathrm{T}}Y+Y^{\mathrm{T}}Y) \end{aligned}$$

$J(w)$ 越小，说明损失函数的值越小，效果也就越好；需要我们去使 $J(w)$ 变小，也就是希望导数为 0

现在针对 $J(w)$ 求导数，首先要知道如下的知识点：

$$\frac{\partial AB}{\partial B}=A^{\mathrm{T}},\quad \frac{\partial A^{\mathrm{T}}B}{\partial A}=B,\quad \frac{\partial X^{\mathrm{T}}AX}{\partial X}=2AX$$

按这个规律对函数进行求导：

$$\begin{aligned} \frac{\partial J(w)}{\partial w} &= \frac{1}{2}\left(\frac{\partial w^{\mathrm{T}}X^{\mathrm{T}}Xw}{\partial w}-\frac{\partial Y^{\mathrm{T}}Xw}{\partial w}-\frac{\partial w^{\mathrm{T}}X^{\mathrm{T}}Y}{\partial w}\right) \\ &= \frac{1}{2}\left[2X^{\mathrm{T}}Xw-(Y^{\mathrm{T}}X)^{\mathrm{T}}-(X^{\mathrm{T}}Y)\right] \\ &= \frac{1}{2}\left[2X^{\mathrm{T}}Xw-2(X^{\mathrm{T}}Y)\right] \\ &= X^{\mathrm{T}}Xw-X^{\mathrm{T}}Y \end{aligned}$$

令导数为 $\frac{\partial J(w)}{\partial w}=0$，解得：

$$\begin{aligned} X^{\mathrm{T}}Xw-X^{\mathrm{T}}Y &= 0 \\ X^{\mathrm{T}}Xw &= X^{\mathrm{T}}Y \\ w &= (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}Y \end{aligned}$$

但是不是所有的矩阵都有可逆矩阵，$(X^{\mathrm{T}}X)^{-1}$ 可能无解，最小二乘法不能帮我们找到一组参数 w 使我们的损失函数最小。

1.5 梯度下降法

梯度下降法基于函数的梯度信息。函数在某一点的梯度方向是函数在该点增长最快的方向，那么其反方向就是函数值下降最快的方向。在优化问题中，我们希望找到目标函数的最小值，因此每次迭代时沿着目标函数梯度的反方向更新模型参数，不断朝着使目标函数值减小的方向前进，逐步逼近函数的最小值点

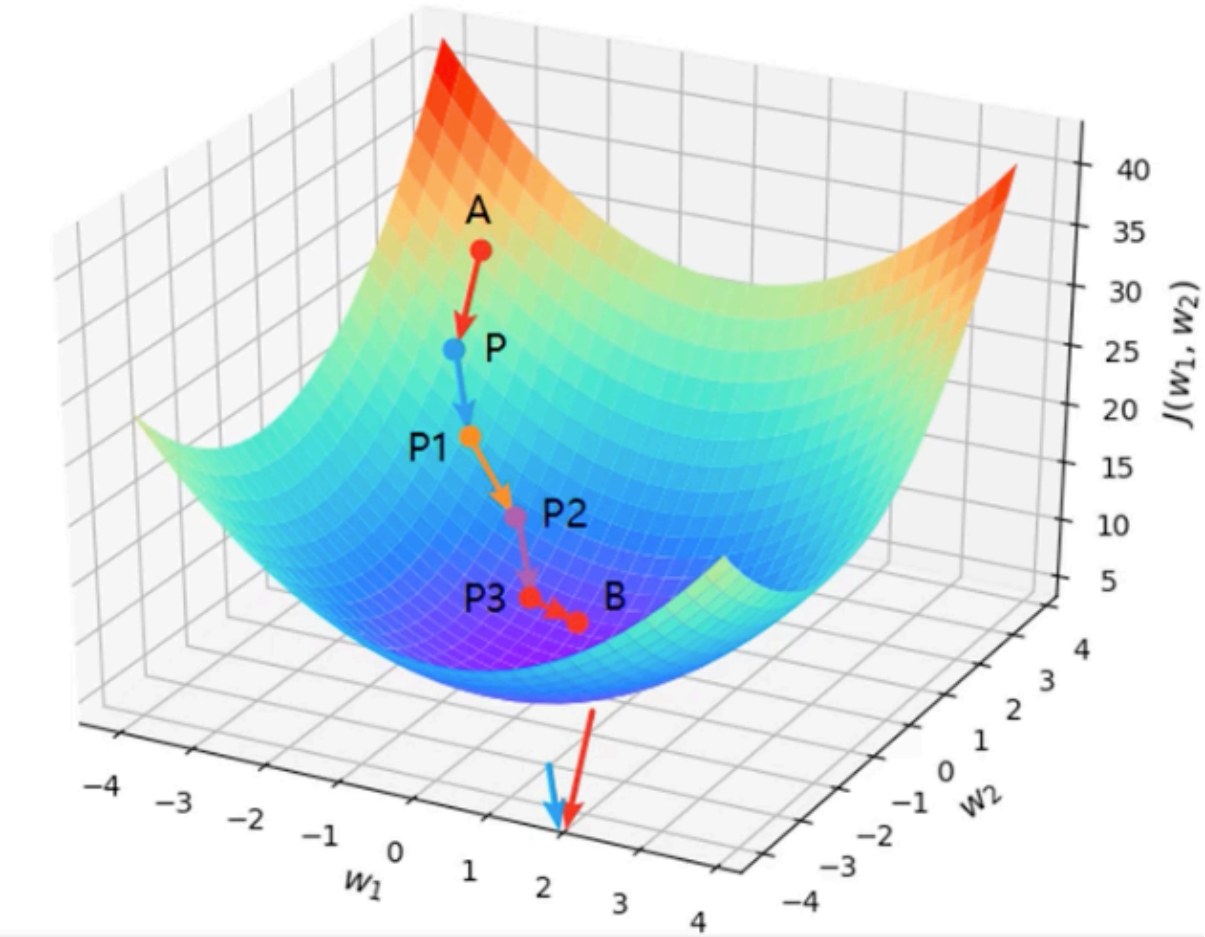


图 1: 梯度下降示意图

梯度下降法参数更新的计算公式如下所示:

$$w = w - a\frac{\partial J(w)}{\partial w}$$

设置 a 学习率——> 求解梯度——> 梯度参数更新

2 逻辑回归算法模型

2.1 逻辑回归算法原理

2.1.1 逻辑回归算法公式

sigmoid 函数的定义如下：

$$g(z) = \frac{1}{1 + e^{-z}}$$

求导公式如下：

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$$

在 sigmoid 函数中，公式中，e 为欧拉常数，自变量 z 可以取任意实数，函数的值域为 $[0, 1]$，这就相当于将输入的自变量值映射到 0-1 之间；sigmoid 的函数图像如下图所示：

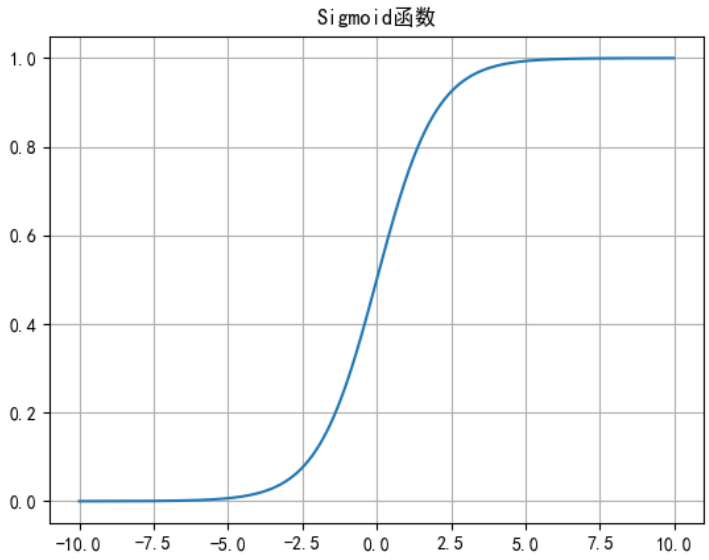


图 2: Sigmoid 函数

下面我们将函数的自变量转化为利用输入的数据特征来表示，以此来利用 sigmoid 函数对其进行映射。

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n + b$$

$$z = W^\top X + b$$

此时自变量就转变成输入的数据特征了，特征利用向量 X 表示。同时特征还有对应的权重参数向量 W 来表示这个数据特征的重要程度，同时还有偏置参数 b。因此，逻辑回归模型可以用如下的公式来表达：

$$g(X) = \frac{1}{1 + e^{-W^\top X + b}}$$

因此对于一个二分类的问题，此时正例和反例的函数表达式就如下式所示：

预测结果为正例的表达式：

$$p(y = 1|X) = \frac{1}{1 + e^{-W^\top X + b}}$$

预测结果为反例的表达式：

$$p(y = 0|X) = \frac{e^{-W^\top X + b}}{1 + e^{-W^\top X + b}} = 1 - p(y = 1|X)$$

在函数的计算推导过程中，如果分别考虑正反两例情况，计算起来就特别麻烦，因此可以将上述两个例子合并起来得到如下公式：

$$p(y|X) = p(y|X)^y \left[1 - p(y|X)\right]^{1-y} \quad \text{，其中} y \in \{0, 1\}$$

加上参数 b 可以写成这个公式:

$$P(y_i \mid x_i; w, b) = \hat{y}_i^{y_i} \cdot (1 - \hat{y}_i)^{1-y_i}$$

2.1.2 交叉熵损失函数

我们希望模型参数 w, b 能让所有样本的预测概率与真实标签尽可能匹配，即最大化“所有样本真实标签出现的总概率”（似然函数）：

$$L(w, b) = \prod_{i=1}^{m} P(y_i \mid x_i; w, b) = \prod_{i=1}^{m} \left[\hat{y}_i^{y_i} \cdot (1 - \hat{y}_i)^{1-y_i}\right]$$

其中 m 是样本总数，$L(w, b)$ 称为似然函数。

但直接优化乘积形式的似然函数不方便（数学计算复杂），因此通常转为最大化对数似然（利用对数将乘积转为求和，简化计算）：

$$\log L(w, b) = \sum_{i=1}^{m} \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\right]$$

而机器学习中习惯最小化损失函数，因此将对数似然取负号，得到交叉熵损失函数：

$$J(w, b) = -\frac{1}{m} \log L(w, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y_i \log \sigma(w^\top x_i + b) + (1 - y_i) \log\left(1 - \sigma(w^\top x_i + b)\right)\right]$$

这里除以 m 是为了对损失取平均值，让损失大小不受样本数量影响。

2.1.3 参数 w 的更新

参数 w 的偏导数的计算步骤如下所示：

$$\begin{aligned}
\frac{\partial J(w,b)}{\partial w} &= -\frac{1}{m}\sum_{i=1}^{m}\frac{\partial\left(y_i\log\sigma(\omega^\top x_i+b)+(1-y_i)\log\left(1-\sigma(\omega^\top x_i+b)\right)\right)}{\partial w} \\
&= -\frac{1}{m}\sum_{i=1}^{m}\left[y_i\frac{\sigma(\omega^\top x_i+b)\left(1-\sigma(\omega^\top x_i+b)\right)}{\sigma(\omega^\top x_i+b)}x_i\right. \\
&\quad \left.+(1-y_i)\frac{-\sigma(\omega^\top x_i+b)\left(1-\sigma(\omega^\top x_i+b)\right)}{1-\sigma(\omega^\top x_i+b)}x_i\right] \\
&= -\frac{1}{m}\sum_{i=1}^{m}\left[y_i x_i-\sigma(\omega^\top x_i+b)x_i\right] \\
&= \frac{1}{m}\sum_{i=1}^{m}\left(\sigma(\omega^\top x_i+b)-y_i\right)x_i
\end{aligned}$$

所以 $\frac{\partial J(w,b)}{\partial w}$ 的计算结果如下：

$$\frac{\partial J(w,b)}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma(w^\top x_i+b)-y_i\right)x_i$$

最终 w 值的更新公式如下

$$w = w - a\frac{\partial J(w,b)}{\partial w}$$

2.2 参数 b 的更新

参数 b 的偏导数的计算步骤如下所示：

$$\begin{aligned}
\frac{\partial J(w,b)}{\partial b} &= -\frac{1}{m}\sum_{i=1}^{m}\frac{\partial\left(y_i\log\sigma(\omega^\top x_i+b)+(1-y_i)\log\left(1-\sigma(\omega^\top x_i+b)\right)\right)}{\partial b} \\
&= -\frac{1}{m}\sum_{i=1}^{m}\left[y_i\frac{\sigma(\omega^\top x_i+b)\left(1-\sigma(\omega^\top x_i+b)\right)}{\sigma(\omega^\top x_i+b)}\right. \\
&\quad \left.+(1-y_i)\frac{-\sigma(\omega^\top x_i+b)\left(1-\sigma(\omega^\top x_i+b)\right)}{1-\sigma(\omega^\top x_i+b)}\right] \\
&= -\frac{1}{m}\sum_{i=1}^{m}\left[y_i\left(1-\sigma(\omega^\top x_i+b)\right)+(y_i-1)\sigma(\omega^\top x_i+b)\right] \\
&= \frac{1}{m}\sum_{i=1}^{m}\left(\sigma(\omega^\top x_i+b)-y_i\right)
\end{aligned}$$

所以 $\frac{\partial J(w,b)}{\partial b}$ 的计算结果如下：

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma(\omega^{\top}x_i + b) - y_i\right)$$

最后 b 值的更新公式如下

$$b = b - a\frac{\partial J(w,b)}{\partial b}$$

2.3 回归和分类的区别

深度学习分为有监督 和无监督 的学习

其中有监督学习就是给答案，然后去训练，最后根据答案去评定模型的优劣；无监督学习就是不给答案，让模型自己训练，不评分

有监督学习中大致可以分为两大任务，一种是回归任务，一种是分类任务。

1. 回归任务所预测的结果是无限的、不确定的连续数值。这样的机器学习任务就是回归任务。

2. 分类任务所输出的结果为离散值，预测的结果也是一个有限的数值来代表种类。

2.4 分类模型评价指标

2.4.1 准确率（Accuracy）

表示预测正确的样本数占总样本数的比例，用于评估模型整体的预测准确性。

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

TP（True Positive）表示真正例，即实际为正类且被模型正确预测为正类的样本数；

TN（True Negative）表示真负例，即实际为负类且被模型正确预测为负类的样本数；

FP（False Positive）表示假正例，即实际为负类但被模型错误预测为正类的样本数；

FN（False Negative）表示假负例，即实际为正类但被模型错误预测为负类的样本数。

2.4.2 精确率（Precision）

精确率又叫查准率，精确率告诉我们在预测出来的正类样本中，真正为正类的比例是多少。

$$\text{Precision} = \frac{TP}{TP + FP}$$

2.4.3 召回率（Recall）

也称为灵敏度（Sensitivity）或真正例率（True Positive Rate，TPR），也叫做查全率，召回率告诉我们所有的正类样本中，有多少被识别出来了。

$$\text{Recall} = \frac{TP}{TP + FN}$$

2.4.4 F1 值

是精确率和召回率的调和平均数，综合考虑了模型的精确率和召回率，能更全面地反映模型的性能。

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.5 回归模型评价指标

和分类模型不同的是，回归模型的输出是一个连续的值，或者可以说是不确定的值。那么用精确度和准确度等评价指标就不那么合理了。从回归模型的任务性质来说，我们希望得到的回归预测值尽可能的接近于真实值，也就是预测值和真实值之间的误差尽可能小。

2.5.1 平均绝对误差 (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

其中 y_i 为真实值，$\hat{y}_i$ 为回归预测值，n 为回归的数据个数。

2.5.2 均方误差 (MSE)

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

其中 y_i 为真实值，$\hat{y}_i$ 为回归预测值，n 为回归的数据个数。注意该公式也用于回归的损失函数，并且可导（MAE 绝对值不是处处可导的），即最小化均方误差。值越小，性能越好。

2.5.3 均方根误差 (RMSE)

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

其中 y_i 为真实值，$\hat{y}_i$ 为回归预测值，n 为回归的数据个数。其实是均方误差 MSE 开根号得到的，实质跟均方误差 MSE 是一样的。主要用于降低均方误差的数量级，防止均方误差 MSE 看起来很大。

2.5.4 评价绝对百分比误差（MAPE）

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\%$$

其中 y_i 为真实值，$\hat{y}_i$ 为回归预测值，n 为回归的数据个数。注意由于这里用了 y_i 作为分母，所以当测量真实值有数据为 0 时，即存在分母为 0 的情况，该指标公式就不可用了。值越小越好。

2.6 训练集、验证集与测试集

表 1: 训练集、验证集与测试集的关系对比

数据集类型	**用途**	**是否参与参数更新**	**核心作用**
训练集	用于模型学习参数（前向传播 + 反向传播）	是	让模型拟合数据规律
验证集	用于调整超参数（如学习率、网络深度）	否 (不通过计算梯度调整参数)	评估模型泛化能力，防止过拟合
测试集	用于最终评估模型真实性能	否	模拟模型在实际场景中的表现
关系：三者从原始数据中划分，互斥且独立，共同服务于模型优化与评估			

3 全连接神经网络

3.1 全连接神经网络架构

全连接神经网络 (MLP-Multilayer Perceptron) 的整体架构如下

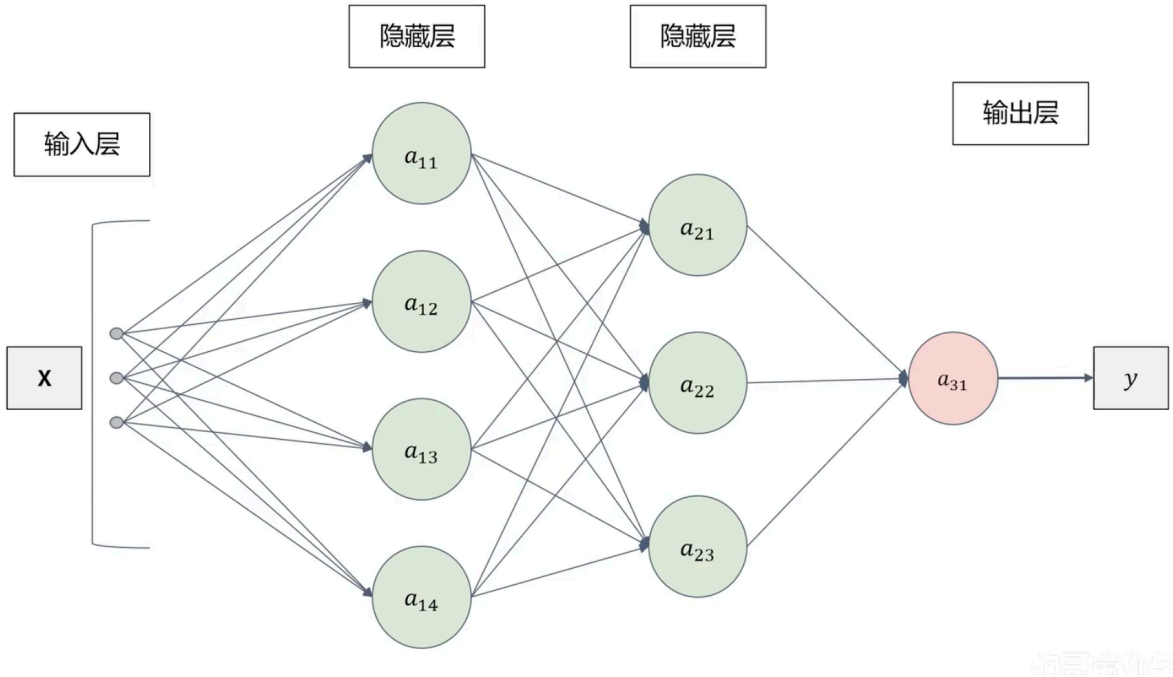


图 3: 全连接神经网络模型图

全连接神经网络的结构单元如下

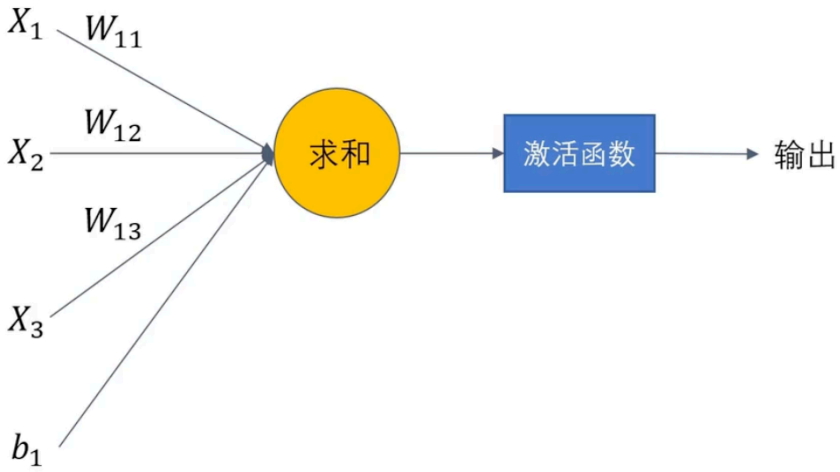


图 4: 全连接神经网络的结构单元

其中 X_1, X_2, X_3 表示一组数据 3 个特征输入

神经元的数学表达式如下所示:

$$a = h(w * x + b)$$

h 其实就是激活函数

3.2 激活函数

3.2.1 为什么要导入激活函数

在神经网路中加入激活函数，是为了让模型具备拟合非线性关系的能力。
若没有激活函数，每个神经元的输出就是输入的线性组合:

$$z = wx + b$$

此时，无论神经网络有多少层，整体输出仍是输入的线性组合，例如:
第一层输出:$z_1 = w_1 x + b_1$
第二层输出:
$z_2 = w_2 z_1 + b_2 = w_2(w_1 x + b_1) + b_2 = (w_2 w_1)x + (w_2 b_1 + b_2) = Wx + B$

3.3 常见激活函数

3.3.1 Sigmoid 函数

Sigmoid 函数的公式和导数如式所示:

$$y = \frac{1}{1 + e^{-z}} \implies y' = y(1 - y)$$

Sigmoid 函数优点

1. 简单、非常适用分类任务;

Sigmoid 函数缺点

1. 反向传播训练时有梯度消失的问题;
2. 输出值区间为 $(0, 1)$，关于 0 不对称;
3. 梯度更新在不同方向走得太远，使得优化难度增大，训练耗时。

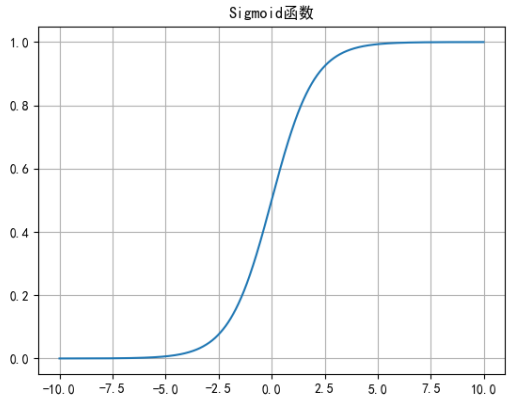


图 5: Sigmoid 函数

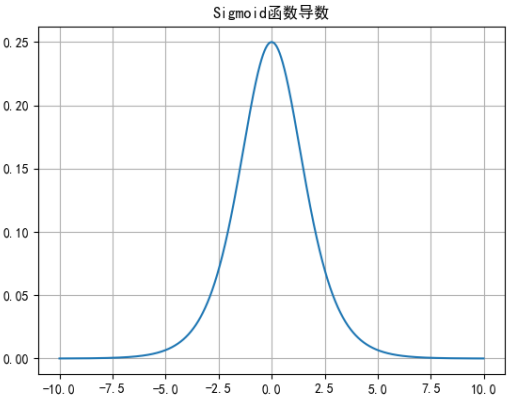


图 6: Sigmoid 函数导数

3.3.2 Tanh 函数

Tanh 函数的公式和导数如式所示：

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}} \implies y' = 1 - y^2$$

Tanh 函数优点

1. 解决了 Sigmoid 函数输出值非 0 对称的问题；
2. 训练比 Sigmoid 函数快，更容易收敛

Tanh 函数缺点

1. 1. 反向传播训练时有梯度消失的问题；
2. 2. Tanh 函数和 Sigmoid 函数非常相似。

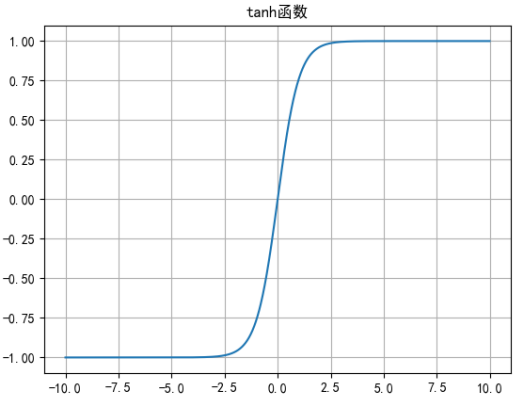


图 7: Tanh 函数

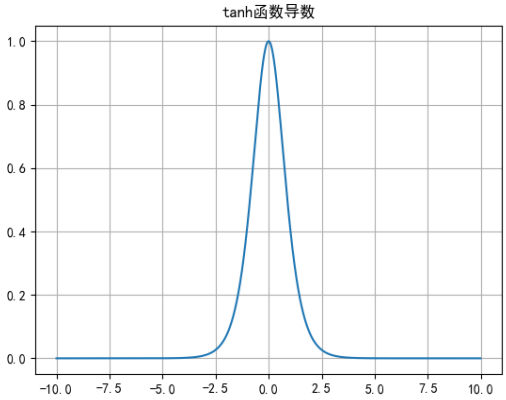


图 8: Tanh 函数导数

3.3.3 ReLU 函数

ReLU 函数的公式和导数如式所示：

$$y = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} \Longrightarrow y' = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

ReLU 函数优点：

1. 解决了梯度消失的问题；
2. 计算更为简单，没有 Sigmoid 函数和 Tanh 函数的指数运算；

ReLU 函数缺点：

1. 训练时可能出现神经元死亡；

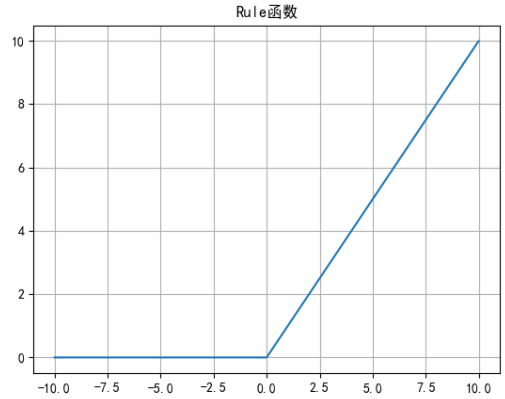


图 9: ReLU 函数

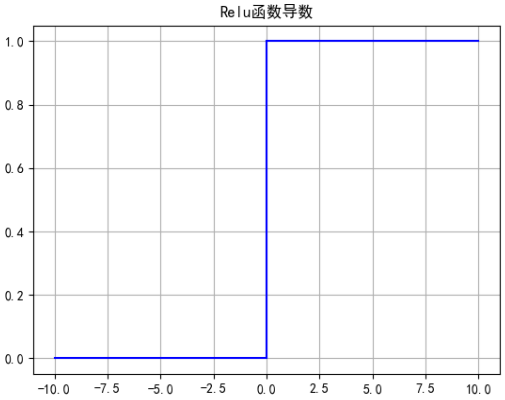


图 10: ReLU 函数导数

3.3.4 Leaky ReLU 函数

Leaky ReLU 函数的公式和导数如式所示：

$$y = \begin{cases} z & \text{if } z > 0 \\ az & \text{if } z \leq 0 \end{cases} \Longrightarrow y' = \begin{cases} 1 & \text{if } z > 0 \\ a & \text{if } z \leq 0 \end{cases}$$

Leaky ReLU 函数优点：

1. 解决了 ReLU 的神经元死亡问题；

Leaky ReLU 函数缺点：

1. 无法为正负输入值提供一致的关系预测（不同区间函数不同）；

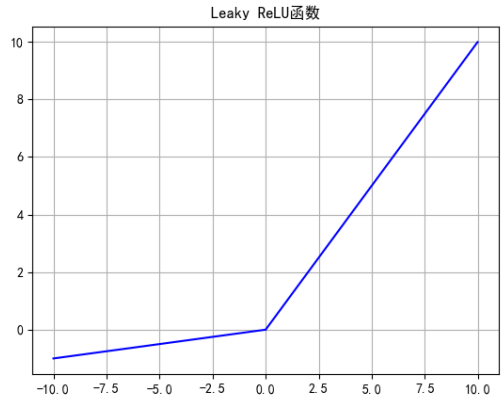


图 11: Leaky ReLU 函数

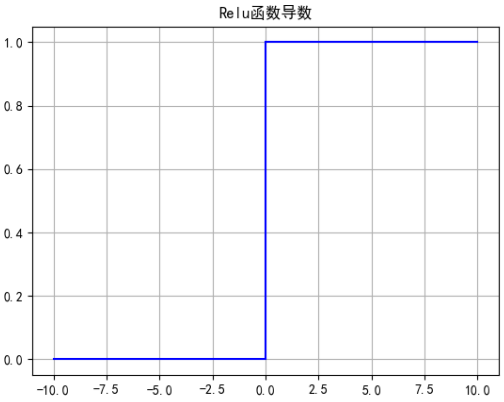


图 12: Leaky ReLU 函数导数

3.3.5 SoftMax 激活函数

SoftMax 函数的公式和导数如下所示：

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \implies y_i' = y_i(1 - y_i)$$

SoftMax 函数的值域是在 [0，1] 之间的，并且存在多个输出。

3.4 前向传播

前向传播是神经网络模型推理的过程，给 $x_1, x_2, \ldots, x_i$ 求 y 的过程。

3.5 反向传播

反向传播的过程就是通过链式求导法则求出 w 和 b 的梯度的过程。

1. 前向传播

2. 计算误差

3. 计算梯度

4. 单向传播（参数更新）

4 卷积神经网络

4.1 简介

卷积神经网络（Convolutional Neural Network，简称 CNN）是一种受生物视觉系统启发设计的深度学习模型，专门用于处理网格结构数据（如图像、视频、音频等），核心优势是能高效提取数据的局部特征并具有平移不变性。

4.1.1 基本结构组成

典型 CNN 由以下层依次堆叠而成：

1. **输入层**：接收原始数据（如图像的像素矩阵，形状为 [， ，]，如 RGB 图像为 $[224, 224, 3]$）。
2. **卷积层（Convolutional Layer）**：
   - 核心组件是**卷积核**（如 3×3 或 5×5 的矩阵），通过滑动窗口与输入数据做卷积运算，生成**特征图**（Feature Map）。
   - 作用：提取局部特征（如边缘、纹理、颜色块等低级特征，或眼睛、车轮等高级特征）。
3. **激活层**：
   - 对卷积层输出应用非线性激活函数（如 ReLU），引入非线性能力，使网络能拟合复杂模式。
4. **池化层（Pooling Layer）**：
   - 常见有**最大池化（Max Pooling）**和**平均池化（Average Pooling）**，通过缩减特征图尺寸（如 2×2 池化将尺寸缩小一半），降低计算量并增强抗干扰性。
5. **全连接层（Fully Connected Layer）**：
   - 位于网络末尾，将池化层输出的特征图扁平化为一维向量，通过神经元全连接实现分类或回归。
6. **输出层**：
   - 根据任务输出结果（如分类任务用 Softmax 输出类别概率，回归任务用线性层输出连续值）。

4.1.2 应用场景

1. **计算机视觉**：
    - 图像分类（如识别猫狗、手写数字）；
    - 目标检测（如定位图像中的行人、车辆）；
    - 图像分割（如像素级划分道路、天空）；
    - 人脸识别、医学影像分析（如肿瘤检测）。
2. **其他领域**：
    - 视频分析（动作识别）；
    - 自然语言处理（文本分类，将文字视为 1D 序列）；
    - 音频处理（语音识别，将声波视为 1D 信号）。

4.1.3 经典模型

- **LeNet-5**：早期 CNN 模型，用于手写数字识别，奠定了 CNN 基本框架。
- **AlexNet**：2012 年 ImageNet 竞赛冠军，首次证明深度学习在图像识别中的优越性，使用 ReLU 和 GPU 加速。
- **VGGNet**：采用多个 3×3 卷积核堆叠，结构简洁，特征提取能力强。
- **ResNet**：引入残差连接（Residual Connection），解决深层网络梯度消失问题，可训练上千层网络。
- **Inception（GoogLeNet）**：通过多尺度卷积核并行提取特征，在精度和效率间取得平衡。

4.1.4 核心优势

相比全连接网络，CNN 通过局部感受野和权值共享**大幅减少参数数量**，同时利用池化增强**平移/缩放不变性**，使其在处理图像等网格数据时效率更高、泛化能力更强，成为计算机视觉领域的主流模型。

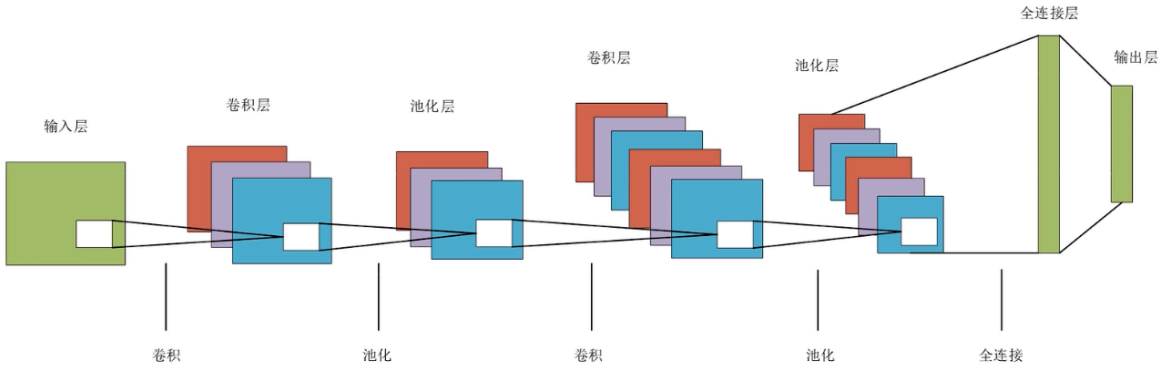


图 13: 卷积神经网络整体结构图

4.2 卷积层

卷积神经网络（CNN）的卷积层是特征提取的核心。

卷积层通过卷积核滑动、多通道融合、步长控制、填充调整等机制，实现了对局部特征的高效提取，同时通过参数共享减少计算量。这些特性使 CNN 在处理图像等网格数据时，既能捕捉局部细节，又能保持平移不变性

4.2.1 卷积核

卷积核是一个小型的二维矩阵，是卷积运算的核心参数，通过训练学习得到。每个卷积核负责提取一种特定的局部特征。

4.2.2 卷积运算

卷积层的核心操作是滑动窗口卷积，具体步骤如下：

1. 窗口滑动：卷积核在输入特征图上按固定步长（Stride）从左到右、从上到下滑动。
2. 元素相乘求和：在每个滑动位置，卷积核与输入特征图的对应局部区域进行元素相乘，再将结果求和，得到输出特征图上的一个像素值。
3. 偏置项（Bias）：求和结果通常会加上一个偏置项，再经过激活函数处理。

对于输入特征图 X 和卷积核 K，输出特征图 Y 上的元素 $Y(i,j)$ 计算为：

$$Y(i,j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} X(i+m, j+n) \times K(m,n) + b$$

其中，$k_h \times k_w$ 是卷积核尺寸，b 是偏置项。

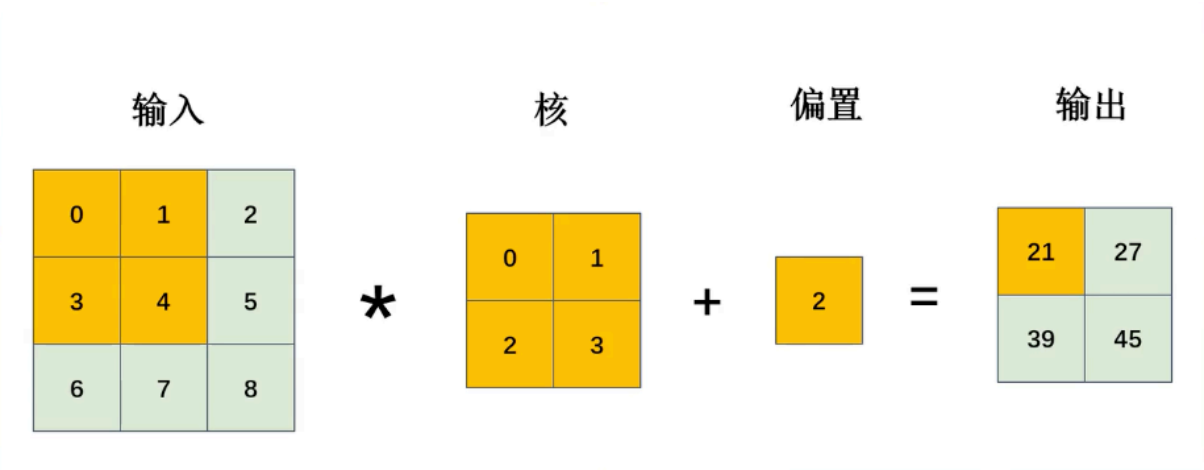


图 14: 卷积运算

4.2.3 步长 (Stride)

定义：卷积核每次滑动的像素数（横向和纵向步长可不同，通常设为相同值）。
影响：

1. 步长为 1：卷积核逐个像素滑动，输出特征图尺寸较大。
2. 步长为 2：卷积核每次滑动 2 个像素，输出特征图尺寸缩小（约为输入的 1/2），计算量减少。

4.2.4 填充 (Padding)

定义：在输入特征图的边缘添加额外像素（通常为 0），以控制输出特征图的尺寸。
作用：

1. 避免边缘特征被较少提取（边缘像素仅参与一次卷积，中心像素参与多次）。
2. 保持输出尺寸与输入一致（如“same padding”）。

计算公式:

$$OH = \frac{H + 2P - FH}{S} + 1$$
$$OW = \frac{W + 2P - FW}{S} + 1$$

4.2.5 参数共享

定义：同一卷积核在输入特征图的所有位置共享相同的权重参数。
优势：大幅减少参数数量（例如，3*3 卷积核对 224*224 输入仅需 9 个权重，而非 224*224*9 个），降低过拟合风险。

4.2.6 多通道卷积运算

输入多通道：若输入是多通道（如 RGB 三通道），每个卷积核需包含与输入通道数相同的子核（如 3 个 3*3 子核对应 RGB 输入）。每个子核与对应输入通道卷积后，所有结果求和得到单通道输出。

输出多通道：通过设置多个卷积核（如 64 个），可得到多个输出通道（64 通道），每个通道对应一个卷积核提取的特征。

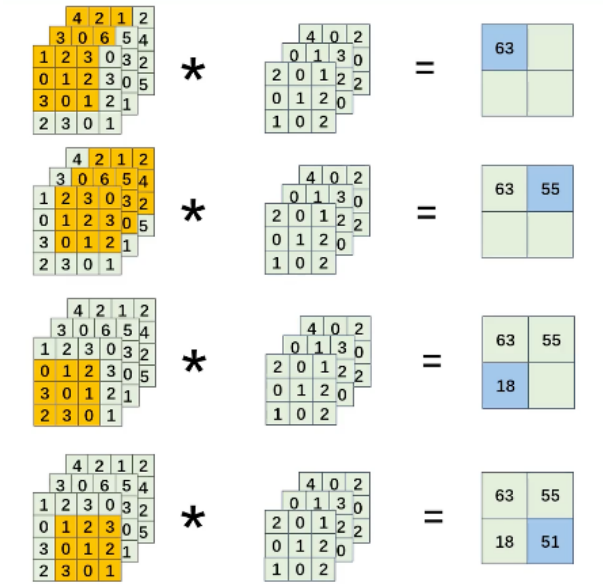


图 15: 多通道数据卷积运算

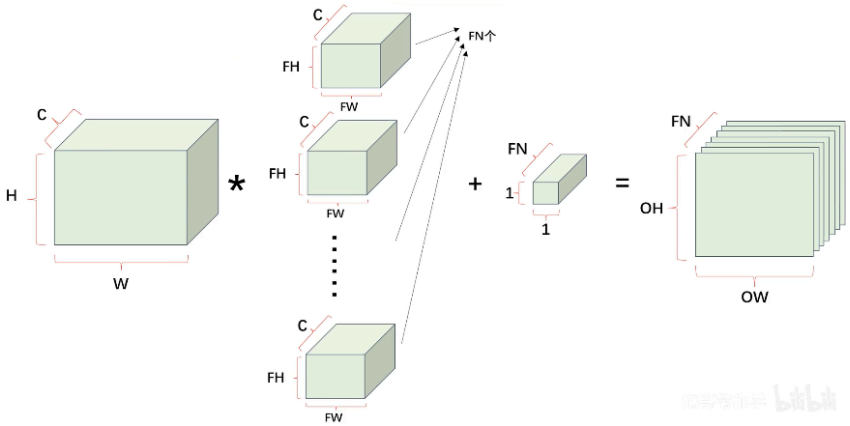


图 16: 卷积运算展示图

4.3 池化层

池化层是卷积神经网络（CNN）中的关键组件，主要作用是对卷积层输出的特征图进行下采样，在保留关键特征的同时减少数据维度，降低计算量。

4.3.1 核心作用

- **降维（下采样）**：减少特征图的尺寸（高度和宽度），降低后续层的计算复杂度。
- **增强平移不变性**：通过聚合局部区域的特征（如取最大值或平均值），使网络对输入的微小位移不敏感。
- **防止过拟合**：减少特征数量，降低模型对局部细节的过度依赖。

4.3.2 常见池化类型

1. 最大池化
    - **操作**：在池化窗口（如 2×2）内取最大值作为输出。
    - **优势**：能有效保留局部区域的显著特征（如边缘、纹理的强度），对噪声更鲁棒。
2. 平均池化
    - **操作**：在池化窗口内取平均值作为输出。
    - **优势**：能保留区域的整体信息，特征更平滑，但对噪声较敏感。

4.3.3 输出尺寸计算

$$OH = \frac{H + 2P - FH}{S} + 1$$
$$OW = \frac{W + 2P - FW}{S} + 1$$

4.3.4 特点总结

特性	最大池化	平均池化
特征保留	保留局部显著特征（如边缘）	保留区域整体信息
抗噪声能力	强（忽略小幅度噪声）	弱（噪声会被平均放大）
计算效率	高（只需比较最大值）	中（需计算平均值）
适用场景	特征检测（如分类、目标检测）	特征平滑（如生成式模型）

5 循环神经网络

5.1 RNN 简介

RNN (Recurrent Neural Network)，即循环神经网络，是神经网络大家族中的一员。与普通神经网络最大的不同在于，RNN 专门设计用来处理和预测序列数据。相对于 CNN，RNN 相当于添加了时间维度这个信息。

正是因为其“记忆”能力，RNN 在处理序列相关的任务上表现得非常出色，尤其是在自然语言处理（NLP）领域

5.2 RNN 结构

RNN 的基本结构如下，当前时间段的输出由当前时间段的输入加上上一个时间段的输出以及偏置量产生。

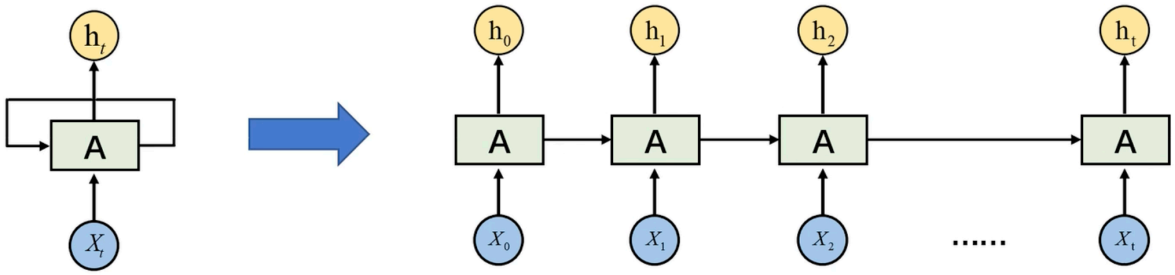


图 17: RNN 基本结构

可以分为两种视角来看待。

5.2.1 折叠起来的循环视角

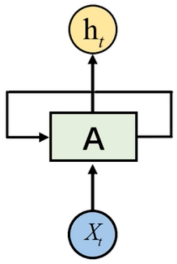


图 18: RNN 折叠起来的循环视角

5.2.2 按照时间展开的序列视角

将上面的循环结构按照时间步一个一个地“拉开”，就得到了展开后的结构。这就像一个多层的前馈神经网络，但有一个非常重要的特点：所有时间步的隐藏层都共享同一套权重参数。

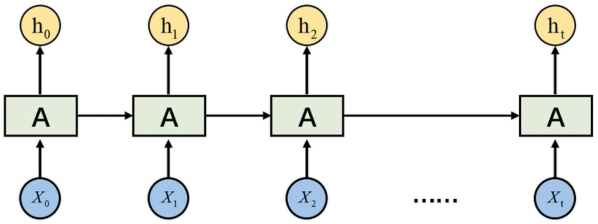


图 19: RNN 按照时间展开序列视角

5.2.3 RNN 整体结构

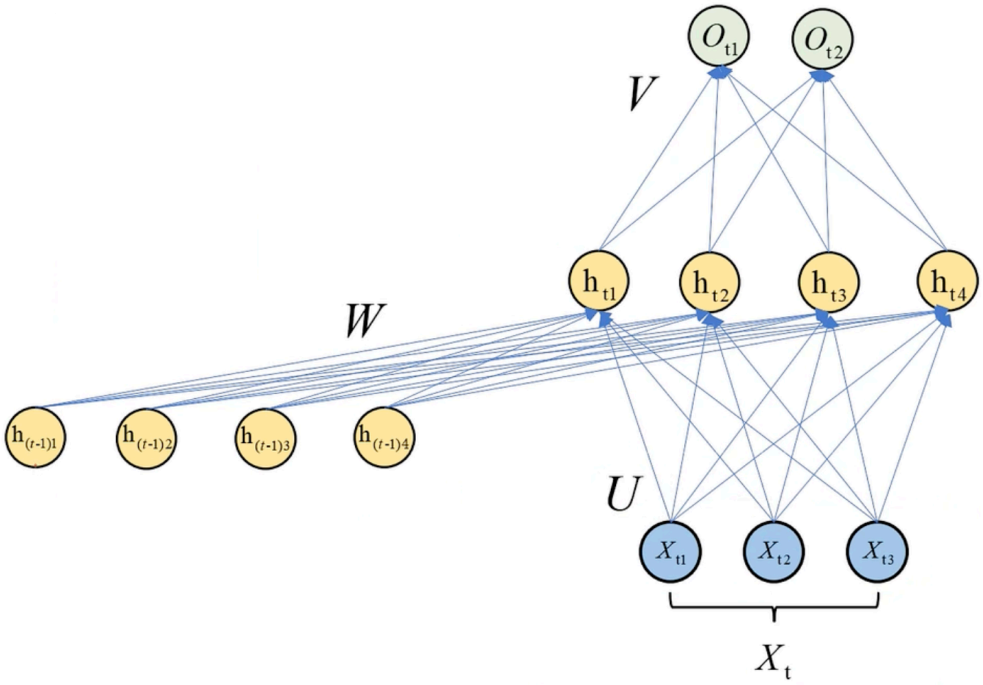


图 20: RNN 结构示意图

5.3 RNN 数学模型

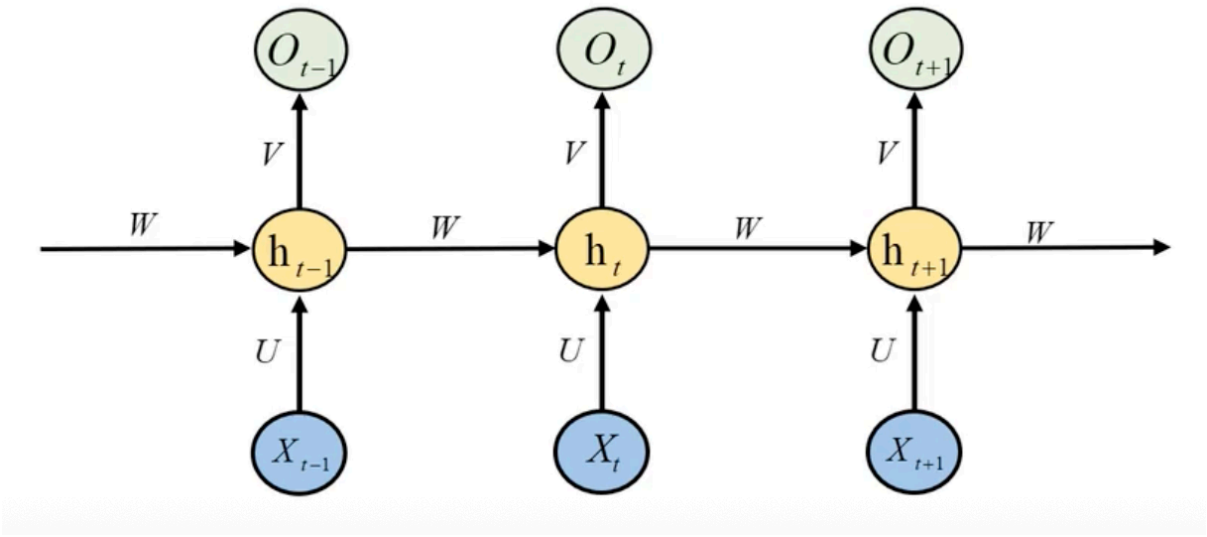
5.3.1 前向传播

隐藏状态更新公式

$$h_t = f(Ux_t + Wh_{t-1} + b)$$

输出公式

$$o_t = Vh_t + c$$



其中, 其中, U、W、V、b 和 c 在所有时间步上是相同的, 这就是 RNN 中的权重共享。

5.3.2 随时间反向传播（BPTT）

我们的目标是最小化整个序列的损失函数 L，它通常是所有时间步损失 L_t 的总和:

$$L = \sum_{t=1}^{T} L_t$$

为了更新权重（例如 W），我们需要计算损失函数 L 对 W 的梯度 $\frac{\partial L}{\partial W}$。根据链式法则，这个总梯度是每个时间步的损失对 W 的梯度之和:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

现在，我们来关注单个时间步的梯度 $\frac{\partial L_t}{\partial W}$。由于 W 在计算每个隐藏状态 h_k (其中 $k \leq t$) 时都被使用，所以根据链式法则，我们需要将 h_t 对 W 的影响一直追溯到过去的每一个时间步:

$$\frac{\partial L_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

在这个公式中，最关键、也是导致问题的部分是 $\frac{\partial h_t}{\partial h_k}$。这一项表示**时间步 k 的隐藏状态对未来时间步 t 的隐藏状态的影响**。

5.4 RNN 梯度爆炸和梯度消失

我们来展开这个关键项 $\frac{\partial h_t}{\partial h_k}$。根据链式法则，它是一系列雅可比矩阵（Jacobian Matrix）的连乘积：

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

现在，我们计算这个式子中的单个雅可比矩阵 $\frac{\partial h_i}{\partial h_{i-1}}$。回顾我们的隐藏状态更新公式 $h_i = f(Ux_i + Wh_{i-1} + b)$，对其求导：

$$\frac{\partial h_i}{\partial h_{i-1}} = \text{diag}(f'(h_i)) \cdot W^T$$

其中，$f'(h_i)$ 是激活函数 f 在 h_i 处的导数，diag() 表示将其构成一个对角矩阵。

将这个结果代入连乘公式中，我们得到：

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \text{diag}(f'(h_i)) \cdot W^T$$

这个公式包含了一个**长达 $(t-k)$ 次的矩阵连乘**。假设 W^T 的最大特征值的绝对值（谱范数）为 $\lambda_{\max}$，并且激活函数导数的上界为 γ。那么，这个连乘项的范数大致可以被估算为：

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq (\lambda_{\max} \cdot \gamma)^{t-k}$$

5.4.1 梯度爆炸 (Exploding Gradients)

如果 $(\lambda_{\max} \cdot \gamma) > 1$，那么当时间间隔 $(t-k)$ 增大时，该项会呈指数级增长。

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \to \infty \quad \text{as} \quad (t-k) \to \infty$$

后果:

- 梯度变得非常大，导致权重更新过大，跳过最优解。
- 数值计算上导致 `NaN` 或无穷大，使训练崩溃。

5.4.2 梯度消失 (Vanishing Gradients)

如果 $(\lambda_{\max} \cdot \gamma) < 1$，那么当时间间隔 $(t-k)$ 增大时，该项会呈指数级衰减至 0。这种情况在使用 tanh（导数 ≤ 1）或 sigmoid（导数 ≤ 0.25）时尤为常见。

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \to 0 \quad \text{as} \quad (t-k) \to \infty$$

后果:

- 模型无法学习到**长距离依赖关系**，来自遥远过去的梯度信号几乎为 0。
- 靠近输出层的权重正常更新，而靠近输入层的权重几乎不更新，训练停滞。

5.4.3 解决方案

梯度裁剪 (Gradient Clipping) 主要针对**梯度爆炸**。设定一个梯度阈值，如果梯度的范数超过这个阈值，就按比例缩小它。

改进 RNN 架构（根本性解决方案）主要针对**梯度消失**:

- **LSTM (Long Short-Term Memory)**: 引入门控机制和独立的细胞状态，梯度可以通过“传送带”顺畅流动。
- **GRU (Gated Recurrent Unit)**: LSTM 的简化版本，效率更高。

问题	梯度爆炸 (Exploding Gradient)	梯度消失 (Vanishing Gradient)
数学原因	梯度计算链式法则中，连乘项的范数**大于 1**，导致梯度随时间步呈指数级**增长**。	梯度计算链式法则中，连乘项的范数**小于 1**，导致梯度随时间步呈指数级**衰减**。
主要公式	$\left\|\prod \text{diag}(f'(h_i))W^T\right\| > 1$	$\left\|\prod \text{diag}(f'(h_i))W^T\right\| < 1$
主要影响	训练不稳定，损失函数出现 `NaN`，模型崩溃。	无法学习长距离依赖关系，模型“健忘”，训练过早停滞。
主要对策	**梯度裁剪 (Gradient Clipping)**	**LSTM / GRU 架构**，使用 ReLU 激活函数

表 2: 梯度消失与梯度爆炸对比

5.5 LSTM 基本架构与原理

5.5.1 记忆细胞

首先，**LSTM 依然是一个循环神经网络，因此 LSTM 的网络架构与 RNN 高度相似，同时 LSTM 也是需要遍历所有时间步，不断完成循环和嵌套的**。但不同的是，RNN 由输入层（输入 X_t）、隐藏层和输出层（输出 Y_t）构成，而 LSTM 由输入层（输入 X_t）、**记忆细胞（Memory Cell）**和输出层（输出 Y_t）构成，其中输入、输出层与 RNN 的输入、输出层完全一致，而记忆细胞是 LSTM 独有的结构。

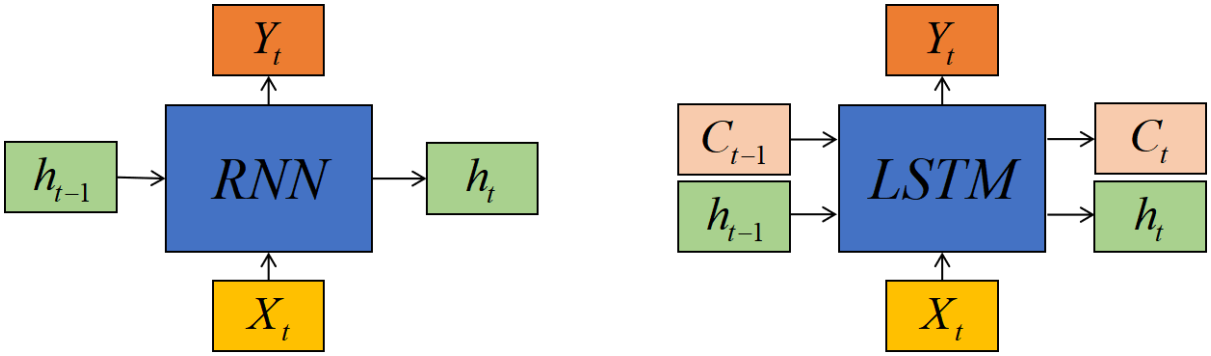


图 21: LSTM 单元图

这是一个复杂的流程，但在横向上，它可以被分割为 C 的传递和 h 的传递两条路径；在纵向上，它可以被分为如图所示的三个不同的路径：

1. **帮助循环网络选择吸纳多少新信息的输入门**
2. **帮助循环网络选择遗忘多少历史信息的遗忘门**，以及
3. **帮助循环网络选择出对当前时间步的预测来说最重要的信息、并将该信息输出给当前时间步的输出门**

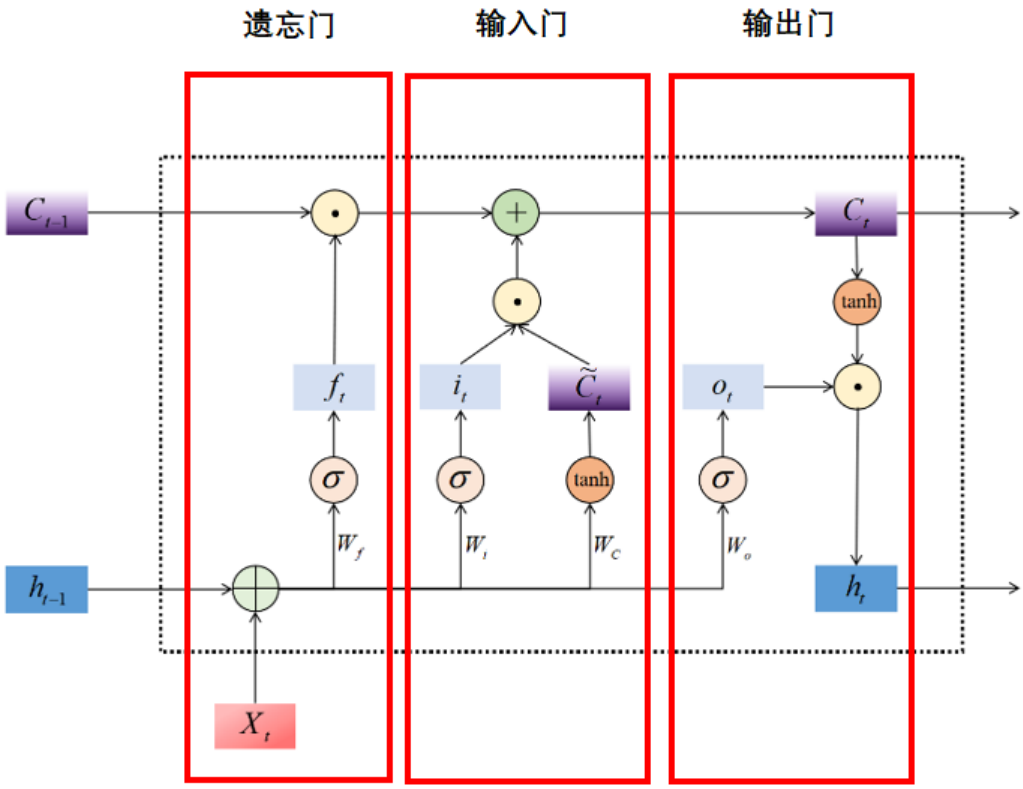
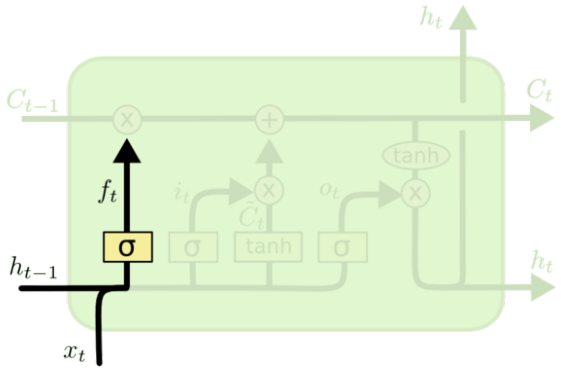


图 22: LSTM 流程图 1

5.5.2 遗忘门

遗忘门是决定要留下多少长期信息 C 的关键计算单元，其数学本质是令上一个时间步传入的 C_{t-1} 乘以一个 [0,1] 之间的比例，以此筛选掉部分旧信息。

在这个计算过程中，假设遗忘门令 C_{t-1} 乘以 0.7，那就是说遗忘门决定了要保留 70% 的历史信息，遗忘 30% 的历史信息，这 30% 的信息空间就可以留给全新的信息来使用。



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

图 23: LSTM 遗忘门

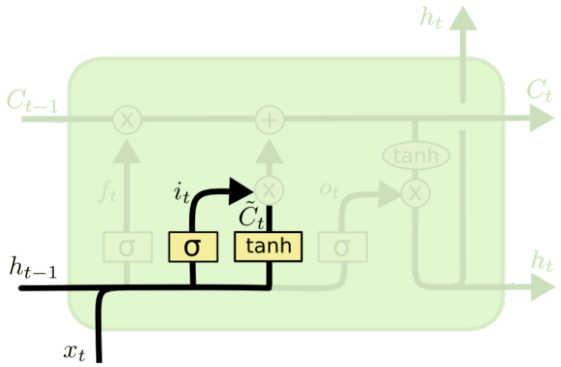
那这个比例是如何被计算出来的呢？如图所示，遗忘门会参考当前时间步的信息 X_t 与上一个时间步的短时信息 h_{t-1} 来计算该比例，其中 σ 是 sigmoid 函数，w_f 是动态影响最终权重大小的参数，f_t 就是 [0,1] 之间的、用于影响 C_{t-1} 的比例。

在 LSTM 的设计逻辑之中，考虑 X_t 和 h_{t-1} 实际是在考虑离当前时间步最近的**上下文信息**，而权重 w_f 会受到**损失函数和算法整体表现的影响，不断调节遗忘门中计算出的比例 f 的大小**，因此遗忘门能够结合上下文信息、损失函数传来的梯度信息、以及历史信息共同计算出全新的、被留下的长期记忆 C_t。这个流程在实践中被证明是十分有效的。

5.5.3 输入门

输入门是决定要吸纳多少新信息来融入长期记忆 C 的计算单元，其数学本质是在当前时间步传入的所有信息上乘以一个 [0,1] 之间的比例，以筛选掉部分新信息，将剩余的新信息融入长期记忆 C。

在这个计算过程中，我们首先要计算出当前时间步总共吸收了多少全新的信息 $\tilde{C}_t$，这个计算全新信息的方式就与 RNN 中计算 h_t 的方式高度相似，因此也会包含用于影响新信息传递的参数 W_C 和 RNN 中常见的 tanh 函数。然后，我们要依据上下文信息（依然是 X_t 和 h_{t-1}）以及参数 W_i 来生成筛选新信息的比例 i_t。最后我们将二者相乘，并加入到长期记忆 C 当中。



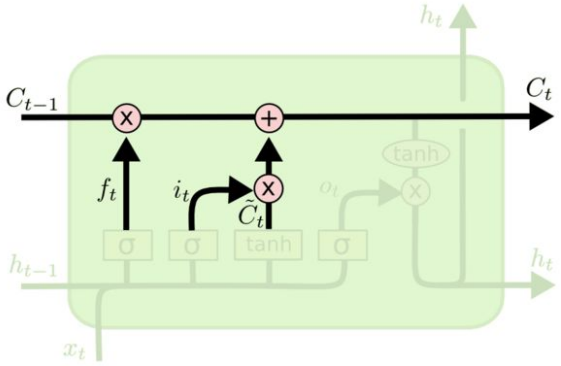
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

图 24: LSTM 输入门

可以看到，相比起 RNN 的数据输入过程，LSTM 的输入过程灵活了非常多——在输入门当中，我们不仅对输入数据加上了一个比例 i_t，还分别使用了两个受损失函数影响的权重 W_i 和 W_C 来控制新信息聚合和比例计算的流程。在这一比例和两大参数的作用下，输入数据可以被高度灵活地调节，以便满足最佳的损失函数需求。

5.5.4 更新细胞状态

当遗忘门决定了哪些信息要被遗忘，输入门决定了哪些信息要被加入到长期记忆后，就可以更新用于控制长期记忆的细胞状态了。如下图所示，上一个时间步的长期记忆将乘以遗忘门给出的比例 f_t，再加上新信息 $\tilde{C}_t$ 乘以新信息筛选的比例 i_t，同时考虑放弃过去的信息、容纳新信息，以此来构成传递给下一个时间步的长期信息 C_t。

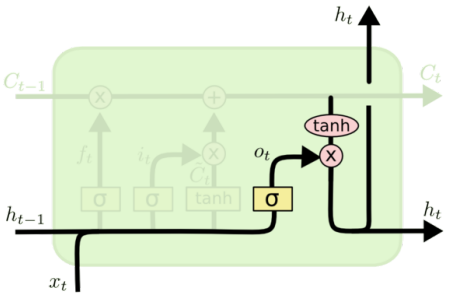


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

图 25: LSTM 更新细胞状态

5.5.5 输出门

最后我们来到了输出门。**输出门是从全新的长期信息 C_t 中筛选出最适合当前时间步的短期信息 h_t 的计算单元，其数学本质是令已经计算好的长期信息 C_t 乘以一个 [0,1] 之间的比例，以此筛选出对当前时间步最有效的信息用于当前时间步的预测**。具体流程如下所示：



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

这个流程分为三步:

1. 首先要借助上下文信息和权重 W_o 来求解出比例 o_t
2. 对长期信息 C_t 进行 tanh 标准化处理
3. 将 o_t 乘在标准化后的长期信息 C_t 之上，用于筛选出 h_t。

为什么要对长期信息 C_t 做标准化处理呢？在 LSTM 的论文中如此写到：Tanh 标准化可以限制有效长期信息 C_t 的数字范围，避免历史信息在传递过程中变得越来越大，同时还能为输出门赋予一定的非线性性质，这个过程被实践证明有助于保持训练稳定、还能够强化算法学习能力，因此在 LSTM 最终的设计中被保留下来。

5.6 LSTM 反向传播以及缓解梯度消失和梯度爆炸

梳理一下 LSTM 的数学流程:

遗忘门:

$$f_t = \sigma(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1})$$

输入门:

$$i_t = \sigma(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1})$$

潜在细胞状态:

$$\tilde{C}_t = \tanh(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1})$$

细胞状态更新:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

输出门:

$$o_t = \sigma(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1})$$

当下时刻输出:

$$h_t = o_t \cdot \tanh(C_t)$$

LSTM 在反向传播中的梯度求解链路可以被可视化为:

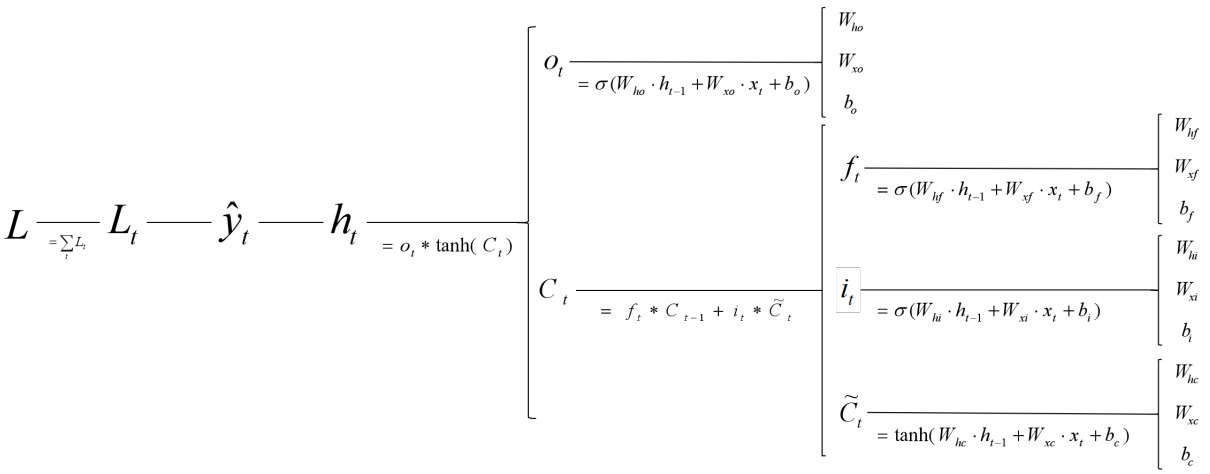


图 26: LSTM 梯度求解链路可视化

假设我们现在计算 $\frac{\partial L_t}{\partial W_{hf}}$，则有：

$$\frac{\partial L_t}{\partial W_{hf}} = \frac{\partial L_t}{\partial \hat{y}_t} * \frac{\partial \hat{y}_t}{\partial h_t} * \frac{\partial h_t}{\partial C_t} * \frac{\partial C_t}{\partial f_t} * \frac{\partial f_t}{\partial h_{t-1}} * \frac{\partial h_{t-1}}{\partial C_{t-1}} \cdots * \frac{\partial f_1}{\partial W_{hf}}$$

$$= \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} * \frac{\partial h_t}{\partial C_t} * \left[\frac{\partial C_t}{\partial C_{t-1}} * \frac{\partial C_{t-1}}{\partial C_{t-2}} \cdots * \frac{\partial C_2}{\partial C_1} \right] * \frac{\partial C_1}{\partial f_1} * \frac{\partial f_1}{\partial w_{hf}}$$

$$= \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} * \frac{\partial h_t}{\partial C_t} * [f_t * f_{t-1} * f_{t-2} \cdots * f_1] * \frac{\partial C_1}{\partial f_1} * \frac{\partial f_1}{\partial w_{hf}}$$

通过避开共享权重的相乘，LSTM 将循环网络梯度爆炸和梯度消失的危险性降低到了一般神经网络的水平。由于 f_t 在 0～1 之间，因此就意味着梯度爆炸的风险将会很小，至于会不会梯度消失，取决于 f_t 是否接近于 1。如果当下时刻的长期记忆比较依赖于历史信息，那么 f_t 就会接近于 1，这时候历史的梯度信息也正好不容易消失；如果 f_t 很接近于 0，那么就说明当下的长期记忆不依赖于历史信息，这时候就算梯度消失也无妨了。

f_t 在 0～1 之间这个特性决定了它梯度爆炸的风险很小，同时 f_t 表明了模型对历史信息的依赖性，也正好是历史梯度的保留程度，两者相互自洽，所以 LSTM 也能较好地缓解梯度消失问题。而计算其他 W 梯度值与计算 W_{hf} 的过程一样，因此，LSTM 同时较好地缓解了梯度消失/爆炸问题。