人工智能创新实践案例解析

2025 年 1 月

前言

人工智能已经成为当今科技发展的核心驱动力。对于许多初学者来说，人工智能算法往往显得高深莫测。作为一名从事人工智能研究和大学教育十多年的教师，我深刻体会到理论与实践之间的鸿沟。许多学生可以看懂算法的公式，却对算法的实际计算过程一知半解；他们熟悉卷积核的概念，但面对一张真实的像素矩阵，竟不知如何计算卷积特征。

正是基于这样的观察和思考，我们决定编写《人工智能创新实践案例解析》这本书。我们的目标是写一本让高中生都能看懂的人工智能入门书。这本书不追求高深的理论推导，而是聚焦于算法的具体实现过程，通过完整的数值算例，手把手地带你一步步计算，真正理解人工智能算法是如何运作的。我们希望，无论是高中生、本科生，还是对人工智能感兴趣的普通读者，都能通过这本书轻松入门，并掌握算法的核心思想。本书的特色与原则如下。

1. 零门槛，仅需中学数学基础

本书的所有算例均以最基础的数学知识为起点，仅需中学阶段的代数、函数和概率知识即可理解。我们摒弃复杂的公式堆砌，转而用具体的数值计算展示算法的每一步，确保读者能够“看得见、算得清”。

2. 完整算例，手动计算

书中每个实践项目均配有完整的数值算例，从数据输入到最终结果，全部通过手动计算完成。例如，在讲解线性回归时，我们会用具体的数据点一步步计算权重更新；在讲解神经网络时，我们会用简单的数字演示前向传播和反向传播的全过程。这种“手算”方式能帮助读者真正内化算法的逻辑。

3. 面向两类读者：普通用户与开发者

普通用户只需关注算例部分，无需接触任何代码，即可理解算法的核心计算流程。对于开发者，我们提供配套的完整源代码，但书中不会涉及代码讲解，以免分散对算法本质的注意力。代码的详细解析将放在配套视频中，供需要动手实现的读者进一步学习。

4. 既适合自学，也适合教学

本书的设计兼顾了教材的系统性和自学的友好性。每一章均包含清晰的算例、习题和延伸思考题，既可作为本科低年级或中学人工智能课程的教材，也可作为爱好者自学的参考书。

如果你曾经被晦涩的算法推导或复杂的编程实现劝退，这本书将是你的理想

选择。我们相信，人工智能的学习不应是少数人的专利，而应是每个对科技充满好奇的人都能掌握的技能。通过具体的数值计算，你会发现，人工智能算法的本质并不神秘——它只是数学与逻辑的巧妙结合。

参与这本书编写的包括：汪金龙、马伟平、周航、魏孜晗、郑浩宇、李昊洋、王晶、邹俊申、刘嘉、金一凡等。

感谢中国计量大学人工智能专业的 2021 级、2022 级、2023 级、2024 级的同学们，你们是面对时代变革勇于进取的一代，有幸和你们一起成长。

这本书的内容还在迅速迭代中，如果你发现还有不足之处，我们还很快改进，也感谢你的耐心。

最后，感谢所有为本书提供建议和支持的学生和朋友们。书中难免存在不足之处，欢迎读者批评指正，我们将不断改进，让更多人以最轻松的方式走进人工智能的世界。

杨力

2025 年 4 月 1 日

1 神经网络

1.1 神经网络简介

神经网络一种模仿人脑工作方式的智能计算系统，其发展历史可以追溯到1940年代。1958年提出的感知机（Perceptron）是首个可训练的神经网络。它的输入层直接连接到输出层，可以认为是单层神经网络。

随后研究者提出了多层感知机（Multi-layer Perceptron，简称MLP），它是3层及以上的神经网络，它包含输入层、1个或多个隐藏层、输出层。MLP到现在依然被广泛使用。

1990年代研究者首次提出了卷积神经网络（convolutional neural Network，简称CNN），用于手写字符识别。

2010年代开始，深度学习（Deep Learning）的概念逐渐被提出，深度学习就是层数特别“深”的神经网络（也有学者提出神经网络并不是深度学习的唯一实现途径）。2012年，基于卷积神经网络的AlexNet横空出世，在大规模图像识别任务中大放异彩，深度学习流行起来。

2016年提出的Transformer架构，以及2022年开始备受关注的大模型，都是基于深度学习的进一步发展。

1.2 神经元

神经元是神经网络的基本组成部分。下图是一个典型的神经元模型：包含有 3 个输入值，1 个输出值。输入 1、输入 2、输入 3，分别与权重值 1、权重值 2、权重值 3 相乘后再求和（加权求和），通过一非线性函数后得到输出结果。

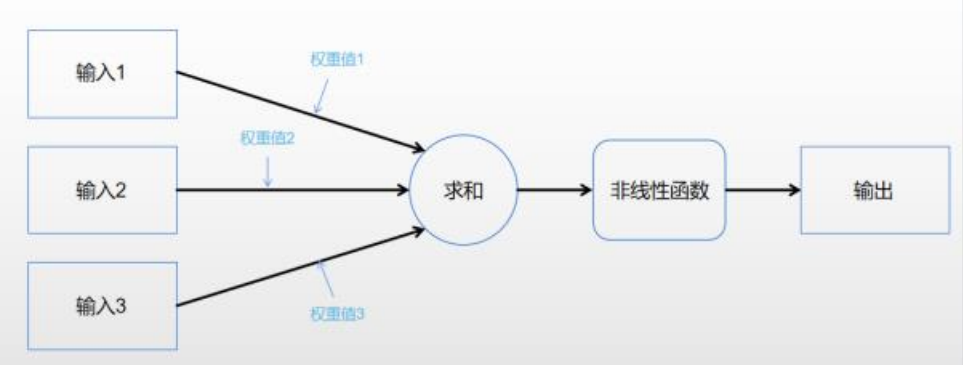


图 简易神经元模型

常见的激活函数有 Sigmoid 函数、tanh 函数和 ReLU 函数等。Sigmoid 函数对输入 x 进行运算，得到一个介于 0 到 1 之间的输出值。

数学计算式为：$F(x)=\frac{1}{1+e^{-x}}$，如图 1.4 所示。

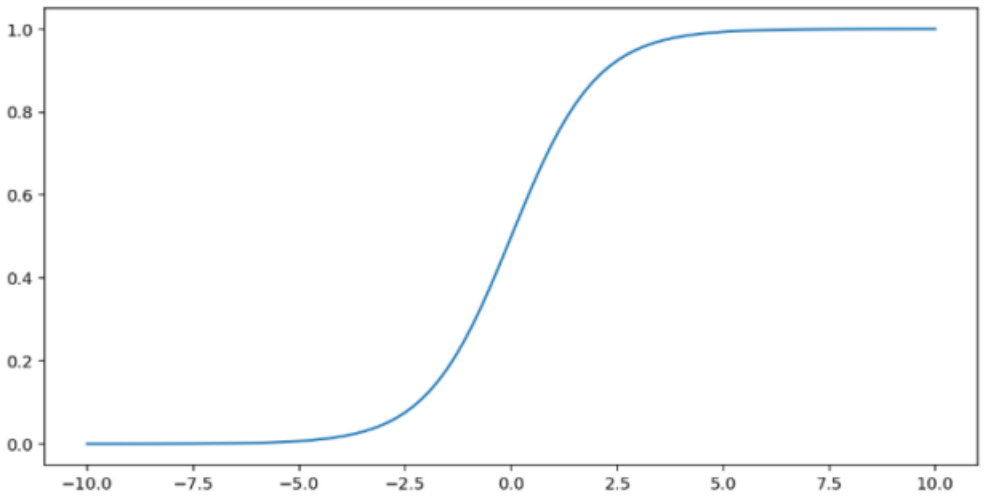


图 Sigmoid 函数的图像

tanh（简称“双曲正切”）函数对输入 x（取值为负无穷到正无穷之间的任意一个数）进行运算，得到一个介于 -1 到 1 之间的输出值。

数学计算式为：$F(x)=\tanh(x)$，如图 1.5 所示。

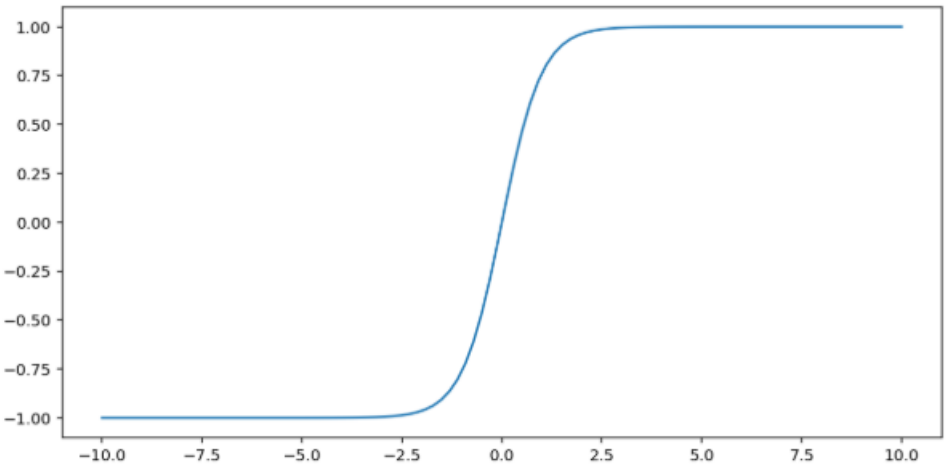


图　tanh 函数的图像

ReLU（简称“修正线性单元激活函数”）对输入 x（取值为负无穷到正无穷之间的任意一个数）进行一特定运算：如果输入值 x 小于 0，则返回 0；如果输入值 x 大于或等于 0，则返回输入值。

数学计算式为：$F(x)=\max(0,x)$，如图 1.6 所示。

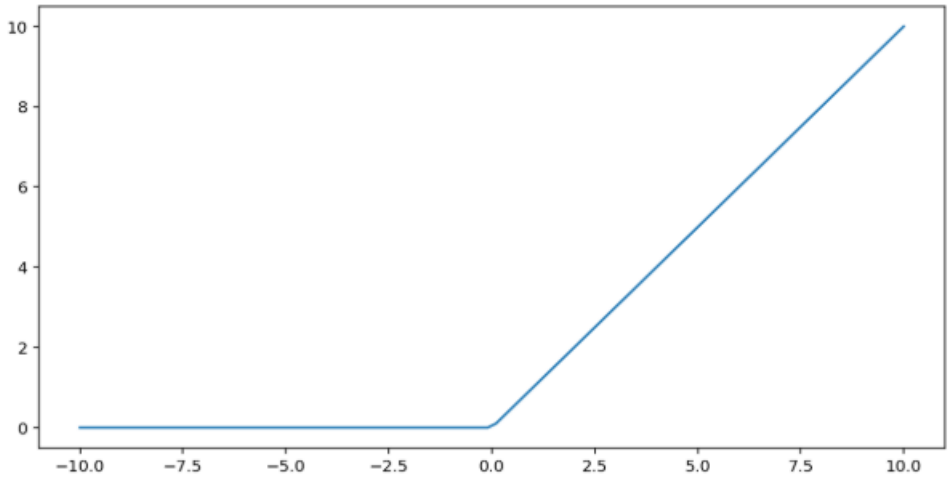


图　ReLU 函数的图像

1.3 两层神经网络（无偏置项）：以简易性别识别为例

这里我们引入一个简单算例，根据一个人的身高和体重，判断他（她）的性别。我们采用一个两层神经网络来解决这个问题。如图 1.7 所示。

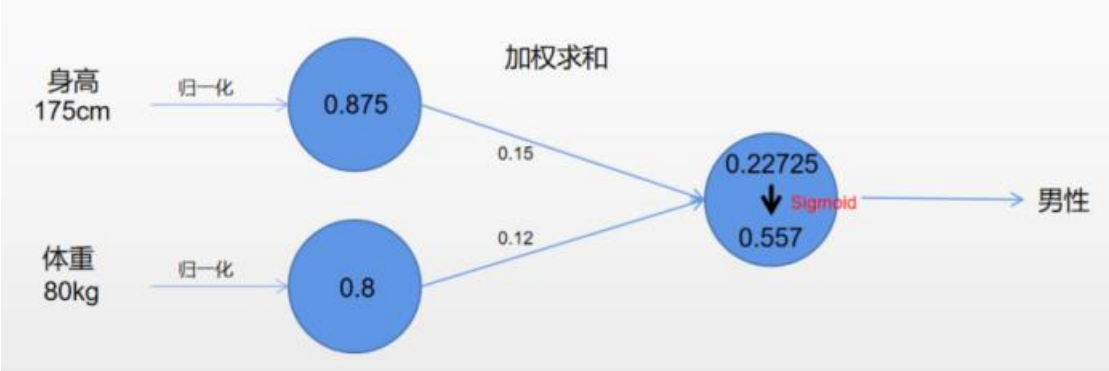


图 无偏置项的以简易性别识别为例的两层神经网络模型算例示意图

输入为身高 175cm、体重 80kg，输出为性别概率（假定概率大于 0.5 预测结果为男，其余情况预测结果为女）。

假设一个人的身高最大值是 200cm，体重最大值是 100kg，则归一化将输入值变换到[0, 1]区间，计算过程为：

$$175/200 = 0.875$$

$$80 / 100 = 0.8$$

假设计算身高的权重为 0.15，计算体重的权重为 0.12。

计算过程为

$$0.875\times0.15+0.8\times0.12=0.13125+0.096=0.22725$$

Sigmoid 函数可以将输出值转化为（0，1）区间的值，相当于概率值。概率值的计算式为：

$$\frac{1}{1+e^{-0.22725}} \approx 0.557$$

算得结果为 55.7%，大于 0.5。即预测出此人为男性。

1.4 两层神经网络（有偏置项）：以简易性别识别为例

下面展示一个带有偏置项的两层神经网络的性别识别计算案例。

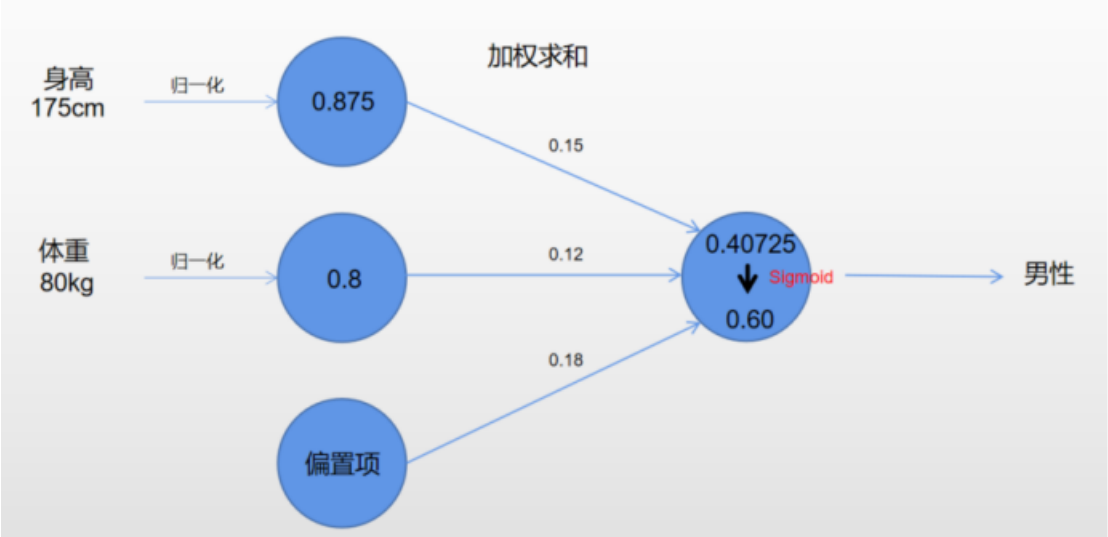


图 含一个偏置项的以简易性别识别为例的两层神经网络模型算例示意图

输入为身高 175cm、体重 80kg，输出为性别概率（假定概率大于 0.5 预测结果为男，其余情况预测结果为女），网络结构为 2 个节点的输入层、1 个值为 0.18 的偏置项、1 个单节点的隐藏层、Sigmoid 激活函数层、1 个节点的输出层。

计算步骤如下。

（1）对输入数据做归一化

假设一个人的身高最大值是 200cm，体重最大值是 100kg，归一化将输入值变换到[0, 1]区间，计算过程为：

$$175 / 200 = 0.875$$

$$80 / 100 = 0.8$$

（2）加权求和

假设计算身高的权重为 0.15，计算体重的权重为 0.12，偏置项为 0.18。

计算过程为：

$$0.875 \times 0.15 + 0.8 \times 0.12 + 0.18 = 0.13125 + 0.096 + 0.18 = 0.40725$$

（3）Sigmoid 计算

Sigmoid 函数的输出为（0，1）区间，相当于把输出值转化为概率值，计算公式为：

$$\frac{1}{1+e^{-0.40725}} \approx 0.60$$

算得结果为 60.0%，大于 0.5,即预测此人为男性。

1.5 三层神经网络：以运动偏好预测为例（二分类问题）

在前面小节中，我们通过两层神经网络解决了简易性别识别的问题。然而，对于更复杂的任务，仅有两层网络可能无法充分捕捉输入特征之间的复杂关系。

在本小节，我们引入一个新的案例，根据一个人的身高和体重，判断此人是否爱运动。我们构建一个简单的三层神经网络来预测一个人是否爱运动。图 1.9 展示了三层神经网络模型的示例图。除了输入层和输出层，这个网络还包含一个隐藏层，能够学习输入数据的高维特征，提升模型的预测能力。

网络结构为 2（输入）→3（隐藏层 1）→1（输出），各隐藏层内使用 ReLU 激活函数，并用 Sigmoid 激活函数输出概率（≥0.5 为“喜欢运动”）。

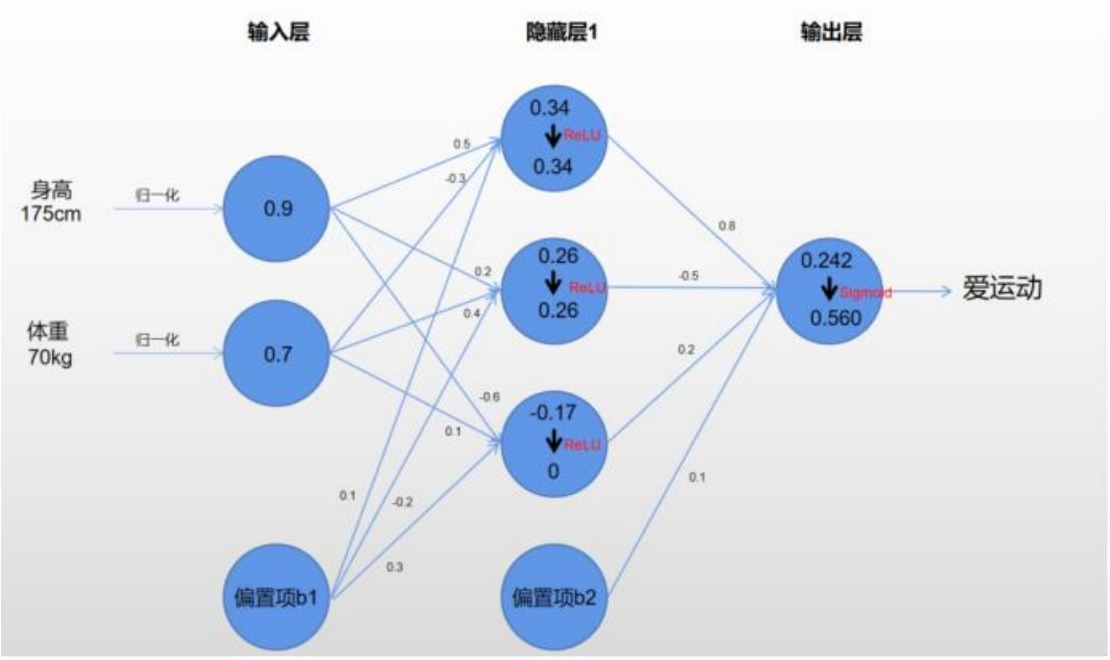


图 以运动偏好预测为例的三层神经网络模型算例示意图

假设有一个人身高 175cm、体重 70kg，和前一节算例类似，经过归一化后，身高（0.9）和体重（0.7）。

（1）输入层：x1=0.9（身高），x2=0.7（体重）。

（2）隐藏层 1 含 3 个神经元，假设经过训练后，得到优化的权重和偏置如下：

权重 $W_1=\begin{bmatrix}0.5 & -0.3\\ 0.2 & 0.4\\ -0.6 & 0.1\end{bmatrix}$，偏置 $b_1=\begin{bmatrix}0.1\\ -0.2\\ 0.3\end{bmatrix}$。

（3）输出层含 1 个神经元，参数如下：

权重 $W_2=\begin{bmatrix}0.8 & -0.5 & 0.2\end{bmatrix}$，偏置 $b_2=0.1$。

2）前向传播计算步骤

输入层 → 隐藏层

计算隐藏层的加权输入 z1 和激活输出 a1（使用 ReLU 激活函数）：

$$z_1=W_1*x+b_1=\begin{bmatrix}0.5*0.9+(-0.3)*0.7+0.1\\ 0.2*0.9+0.4*0.7+(-0.2)\\ (-0.6)*0.9+0.1*0.7+0.3\end{bmatrix}=\begin{bmatrix}0.34\\ 0.26\\ -0.17\end{bmatrix}$$

$$a_1=\mathrm{Re}\,LU(z_1)=\begin{bmatrix}\max(0,0.3975)\\ \max(0,0.5225)\\ \max(0,-0.255)\end{bmatrix}=\begin{bmatrix}0.34\\ 0.26\\ 0\end{bmatrix}$$

隐藏层 1 → 输出层

计算输出层的加权输入 z2 和最终概率输出 y（使用 Sigmoid 激活函数）：

$$z_2=W_2*a_1+b_2=0.8*0.34+(-0.5)*0.26+0.2*0+0.1=0.242$$

令 $F(x)=\dfrac{1}{1+e^{-x}}$，当 $x=z_2=0.242$时，

最终概率输出 $y\approx 0.560(>0.5)$，即此人喜欢运动。

1.6 多层感知机：以运动偏好预测为例（二分类问题）

多层感知机是层数为 3 层或以上的神经网络。这里我们构建一个简单的四层神经网络来预测一个人是否喜欢运动。输入为归一化后的身高（0.875）和体重（0.7），网络结构为 2（输入）→3（隐藏层 1）→2（隐藏层 2）→1（输出），各隐藏层内使用 ReLU 激活函数，并用 Sigmoid 激活函数输出概率（≥0.5 为“喜欢运动”）。图 1.9 展示了一个以是否热爱运动为预测目标的三层神经网络模型的示例图。

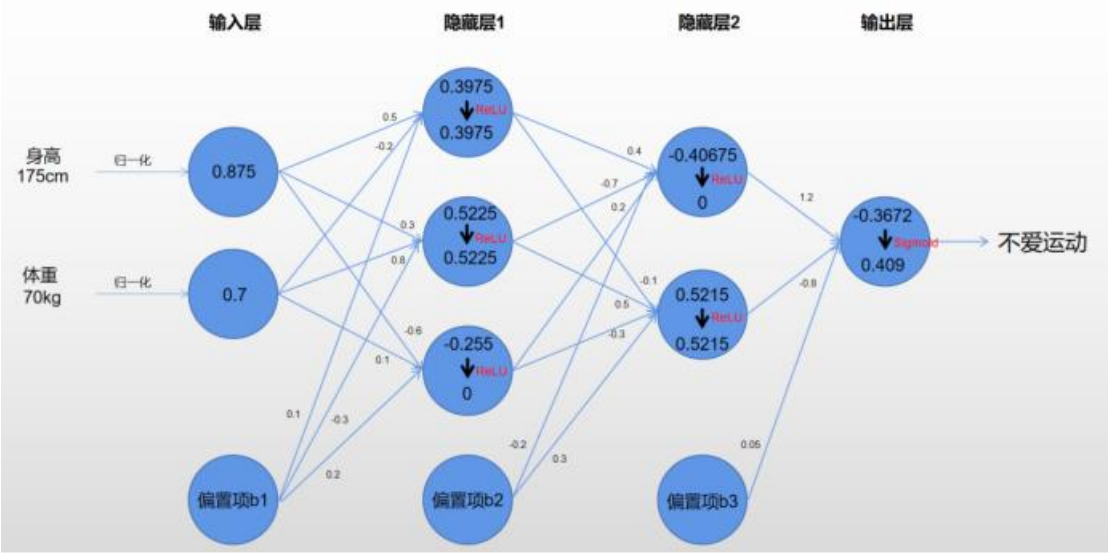


图 以运动偏好预测为例的四层神经网络模型算例示意图

网络结构与参数

（1）输入层：x1=0.875（身高），x2=0.7（体重）。

（2）隐藏层 1 含 3 个神经元，假设经过训练都得到优化后的权重和偏置项如下：

权重 $W_1 = \begin{bmatrix} 0.5 & -0.2 \\ 0.3 & 0.8 \\ -0.6 & 0.1 \end{bmatrix}$，偏置 $b_1 = \begin{bmatrix} 0.1 \\ -0.3 \\ 0.2 \end{bmatrix}$。

隐藏层 2 含 2 个神经元，参数如下：

权重 $W_2 = \begin{bmatrix} 0.4 & -0.7 & 0.2 \\ -0.1 & 0.5 & -0.3 \end{bmatrix}$，偏置 $b_2 = \begin{bmatrix} -0.2 \\ 0.3 \end{bmatrix}$。

（3）输出层含 1 个神经元，参数如下：

权重 $W_3=[1.2 \quad -0.8]$，偏置 $b_3=0.05$。

2）前向传播计算步骤

输入层 → 隐藏层 1

计算隐藏层 1 的加权输入 z1 和激活输出 a1（使用 ReLU 激活函数）：

$$z_1=W_1*x+b_1=\begin{bmatrix}0.5*0.875+(-0.2)*0.7+0.1\\0.3*0.875+0.8*0.7+(-0.3)\\(-0.6)*0.875+0.1*0.7+0.2\end{bmatrix}=\begin{bmatrix}0.4375-0.14+0.1\\0.2625+0.56-0.3\\-0.525+0.07+0.2\end{bmatrix}=\begin{bmatrix}0.3975\\0.5225\\-0.255\end{bmatrix}$$

$$a_1=\mathrm{Re}\,LU(z_1)=\begin{bmatrix}\max(0,0.3975)\\\max(0,0.5225)\\\max(0,-0.255)\end{bmatrix}=\begin{bmatrix}0.3975\\0.5225\\0\end{bmatrix}$$

隐藏层 1 → 隐藏层 2

计算隐藏层 2 的加权输入 z2 和激活输出 a2（继续使用 ReLU）：

$$z_2=W_2*a_1+b_2=\begin{bmatrix}0.4*0.3975+(-0.7)*0.5225+0.2*0+(-0.2)\\(-0.1)*0.3975+(0.5)*0.5225+(-0.3)*0+0.3\end{bmatrix}=\begin{bmatrix}-0.40675\\0.5215\end{bmatrix}$$

$$a_2=\mathrm{Re}\,LU(z_2)=\begin{bmatrix}\max(0,-0.40675)\\\max(0,0.5215)\end{bmatrix}=\begin{bmatrix}0\\0.5215\end{bmatrix}$$

隐藏层 2 → 输出层

计算输出层的加权输入 z3 和最终概率输出 y（使用 Sigmoid 激活函数）：

$$z_3=W_3*a_2+b_3=1.2*0+(-0.8)*0.5215+0.05=-0.3672$$

令 $F(x)=\dfrac{1}{1+e^{-x}}$，当 $x=z_3=-0.3672$时，

最终概率输出 $y\approx 0.409(<0.5)$，即此人不喜欢运动。

1.7 多层感知机：以运动偏好预测为例（三分类问题）

前面我们介绍的都是二分类问题，接下来我们介绍一个多分类的案例，即根据一个人的身高、体重、每日运动时间，判断此人喜欢游泳、跑步、健身中的哪一种运动。

如图 1.10 所展示的是一个四层神经网络模型算例示意图，因权重值数量过多，未在图中标出。网络结构为 3（输入）→4（隐藏层 1）→3（隐藏层 2）→3（输出），各隐藏层使用 ReLU 激活函数，并用 Softmax 激活函数输出概率（哪个概率值最大，即表示喜欢哪一种运动）。输入节点个数和输出节点个数是固定的，隐藏层的节点数量可以修改。

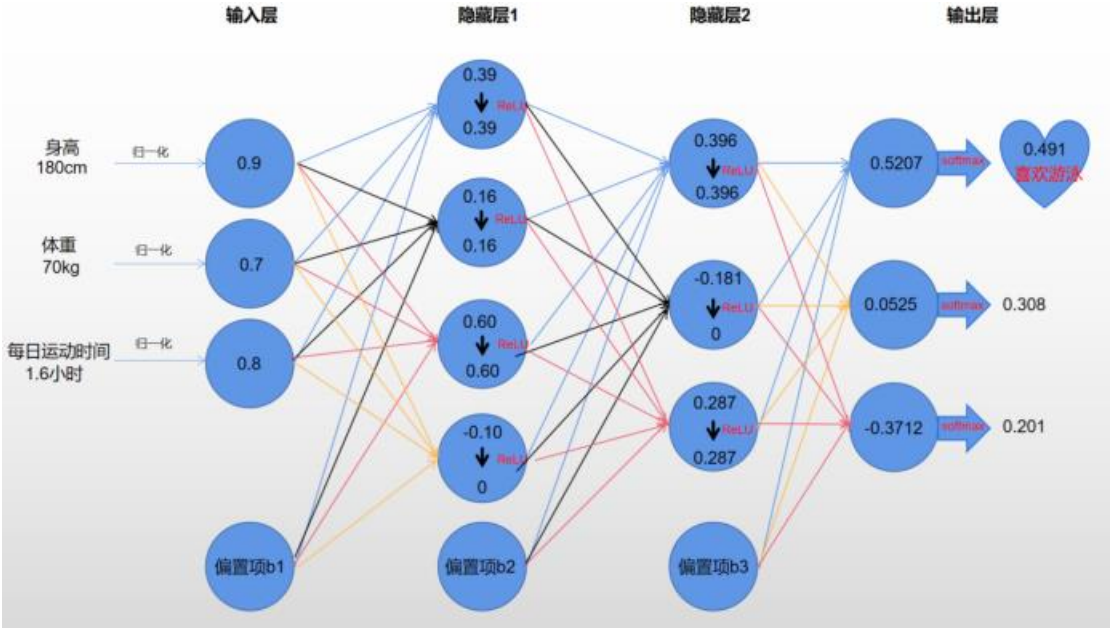


图 以运动喜好为例的三层神经网络模型算例示意图

输入为归一化后的身高（0.9）、体重（0.7）、每日运动时间（0.8）。

1）网络结构与参数

（1）输入层：x1=0.9（身高），x2=0.7（体重），每日运动时间（0.8）。

（2）隐藏层 1 含 4 个神经元，假设经过训练优化后的权重和偏置随机初始化如下：

权重 $W_1=\begin{bmatrix}0.2 & -0.3 & 0.4\\ 0.5 & 0.1 & -0.2\\ -0.4 & 0.6 & 0.3\\ 0.3 & -0.5 & 0.1\end{bmatrix}$，偏置 $b_1=\begin{bmatrix}0.1\\ -0.2\\ 0.3\\ -0.1\end{bmatrix}$。

隐藏层 2 含 3 个神经元，参数如下：

权重 $W_2=\begin{bmatrix}0.4 & -0.5 & 0.2 & 0.1\\ -0.3 & 0.6 & -0.1 & 0.2\\ 0.5 & 0.2 & -0.4 & 0.3\end{bmatrix}$，偏置 $b_2=\begin{bmatrix}0.2\\ -0.1\\ 0.3\end{bmatrix}$。

（3）输出层含 3 个神经元（使用 Softmax 激活函数，对应“游泳”、“跑步”、“健身”），参数如下：

权重 $W_3=\begin{bmatrix}0.7 & -0.2 & 0.5\\ -0.3 & 0.6 & -0.1\\ 0.4 & 0.3 & -0.8\end{bmatrix}$，偏置 $b_3=\begin{bmatrix}0.1\\ 0.2\\ -0.3\end{bmatrix}$。

2）前向传播计算步骤

（1）输入层 → 隐藏层 1

计算隐藏层 1 的加权输入 z1 和激活输出 a1（使用 ReLU 激活函数）：

$$z_1=W_1*x+b_1=\begin{bmatrix}0.2*0.9+(-0.3)*0.7+0.4*0.8+0.1\\ 0.5*0.9+0.1*0.7+(-0.2)*0.8+(-0.2)\\ (-0.4)*0.9+0.6*0.7+0.3*0.8+0.3\\ 0.3*0.9+(-0.5)*0.7+0.1*0.8+(-0.1)\end{bmatrix}=\begin{bmatrix}0.39\\ 0.16\\ 0.60\\ -0.10\end{bmatrix}$$

$$a_1=\mathrm{Re}\,LU(z_1)=\begin{bmatrix}\max(0,0.39)\\ \max(0,0.16)\\ \max(0,0.60)\\ \max(0,-0.10)\end{bmatrix}=\begin{bmatrix}0.39\\ 0.16\\ 0.60\\ 0\end{bmatrix}$$

（2）隐藏层 1 → 隐藏层 2

计算隐藏层 2 的加权输入 z2 和激活输出 a2（继续使用 ReLU）：

$$z_2=W_2*a_1+b_2=\begin{bmatrix}0.4*0.39+(-0.5)*0.16+0.2*0.60+0.1*0+0.2\\ (-0.3)*0.39+0.6*0.16+(-0.1)*0.60+0.2*0+(-0.1)\\ 0.5*0.39+0.2*0.16+(-0.4)*0.60+0.3*0+0.3\end{bmatrix}=\begin{bmatrix}0.396\\ -0.181\\ 0.287\end{bmatrix}$$

$$a_2=\mathrm{Re}\,LU(z_2)=\begin{bmatrix}\max(0,0.396)\\ \max(0,-0.181)\\ \max(0,0.287)\end{bmatrix}=\begin{bmatrix}0.396\\ 0\\ 0.287\end{bmatrix}$$

（3）隐藏层 2 → 输出层

计算输出层的加权输入 z3 和最终概率输出 y（使用 softmax 激活函数）：

$$z_3 = W_3 * a_2 + b_3 = \begin{bmatrix} 0.7*0.396 + (-0.2)*0 + 0.5*0.287 + 0.1 \\ (-0.3)*0.396 + 0.6*0 + (-0.1)*0.287 + 0.2 \\ 0.4*0.396 + 0.3*0 + (-0.8)*0.287 + (-0.3) \end{bmatrix} = \begin{bmatrix} 0.5207 \\ 0.0525 \\ -0.3712 \end{bmatrix}$$

softmax 的数学计算式为 $$y_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

在本例中 n=3，因为在本例中仅有三个预测概率要计算。

$$x_1 = 0.5207, x_2 = 0.0525, x_3 = -0.3712$$

$$e^{0.5207} \approx 1.683, e^{0.0525} \approx 1.054, e^{-0.3712} \approx 0.690$$

输出概率为 $$y = \begin{bmatrix} \frac{1.683}{1.683+1.054+0.690} \\ \frac{1.054}{1.683+1.054+0.690} \\ \frac{0.690}{1.683+1.054+0.690} \end{bmatrix} \approx \begin{bmatrix} 0.491 \\ 0.308 \\ 0.201 \end{bmatrix}$$

3）预测结果

由最终的计算结果可知，此人喜欢游泳的概率为 49.1%，喜欢跑步的概率为 30.8%，喜欢健身的概率为 20.1%。由此可以得出结论：此人最有可能对游泳表现出较高的偏好。

1.8 多层感知机：以年龄预测为例（回归问题）

使用神经网络除了可以解决分类问题外，还可以解决回归问题。回归用于建立输入特征与输出变量之间的关系，它的目标是预测一个或多个自变量（输入特征）对应的因变量（目标值）的数值。

本节的内容是通过一个人的身高、体重、每日步数来预测出人的年龄，网络结构为 3（输入）→4（隐藏层 1）→3（隐藏层 2）→1（输出），各隐藏层内使用 ReLU 激活函数。如图 1.11 所展示的是以年龄预测为例的三层神经网络模型算例示意图，因权重值数量过多，便不在图中展示，以免影响观感，无非是图 1.9 的再复杂化，但原理相同。

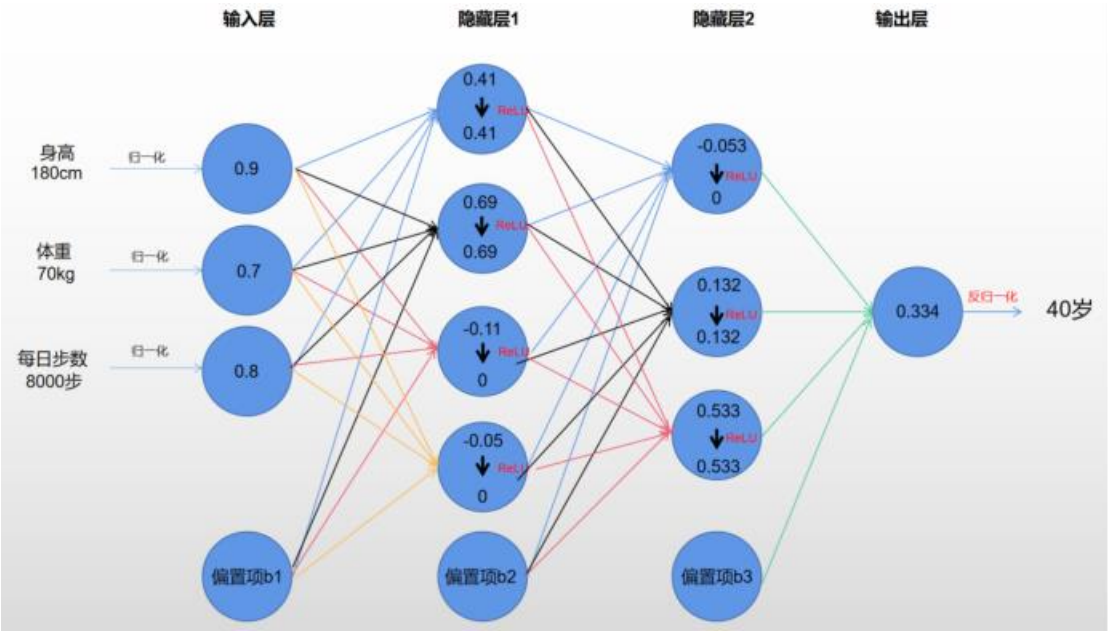


图 以年龄预测为例的三层神经网络模型算例示意图

输入为归一化后的身高（0.9）、体重（0.7）、每日步数（0.8）。

网络结构与参数

（1）输入层：x1=0.9（身高），x2=0.7（体重），x3=0.8（每日步数）。

（2）隐藏层 1 含 4 个神经元，权重和偏置随机初始化如下：

权重 $W_1=\begin{bmatrix}0.5 & 0.2 & -0.1\\ -0.3 & 0.4 & 0.6\\ 0.1 & -0.2 & 0.3\\ -0.2 & 0.5 & -0.4\end{bmatrix}$，偏置 $b_1=\begin{bmatrix}-0.1\\ 0.2\\ -0.3\\ 0.1\end{bmatrix}$。

隐藏层 2 含 3 个神经元，参数如下：

权重 $W_2=\begin{bmatrix}0.3 & -0.4 & 0.2 & 0.5\\ -0.2 & 0.6 & -0.1 & 0.3\\ 0.4 & 0.1 & -0.5 & 0.2\end{bmatrix}$，偏置 $b_2=\begin{bmatrix}0.1\\ -0.2\\ 0.3\end{bmatrix}$。

（3）输出层含 1 个神经元，参数如下：

权重 $W_3=\begin{bmatrix}0.8 & -0.6 & 0.4\end{bmatrix}$，偏置 $b_3=0.2$。

前向传播计算步骤

（1）输入层 → 隐藏层 1

计算隐藏层 1 的加权输入 z1 和激活输出 a1（使用 ReLU 激活函数）：

$$z_1=W_1*x+b_1=\begin{bmatrix}0.5*0.9+0.2*0.7+(-0.1)*0.8+(-0.1)\\ (-0.3)*0.9+0.4*0.7+0.6*0.8+0.2\\ 0.1*0.9+(-0.2)*0.7+0.3*0.8+(-0.3)\\ (-0.2)*0.9+0.5*0.7+(-0.4)*0.8+0.1\end{bmatrix}=\begin{bmatrix}0.45+0.14-0.08-0.1\\ -0.27+0.28+0.48+0.2\\ 0.09-0.14+0.24-0.3\\ -0.18+0.35-0.32+0.1\end{bmatrix}=\begin{bmatrix}0.41\\ 0.69\\ -0.11\\ -0.05\end{bmatrix}$$

$$a_1=\mathrm{Re}\,LU(z_1)=\begin{bmatrix}\max(0,0.41)\\ \max(0,0.69)\\ \max(0,-0.11)\\ \max(0,-0.05)\end{bmatrix}=\begin{bmatrix}0.41\\ 0.69\\ 0\\ 0\end{bmatrix}$$

（2）隐藏层 1 → 隐藏层 2

计算隐藏层 2 的加权输入 z2 和激活输出 a2（继续使用 ReLU）：

$$z_2=W_2*a_1+b_2=\begin{bmatrix}0.3*0.41+(-0.4)*0.69+0.2*0+0.5*0+0.1\\ (-0.2)*0.41+0.6*0.69+(-0.1)*0+0.3*0+(-0.2)\\ 0.4*0.41+0.1*0.69+(-0.5)*0+0.2*0+0.3\end{bmatrix}=\begin{bmatrix}-0.053\\ 0.132\\ 0.533\end{bmatrix}$$

$$a_2=\mathrm{Re}\,LU(z_2)=\begin{bmatrix}\max(0,-0.053)\\ \max(0,0.132)\\ \max(0,0.533)\end{bmatrix}=\begin{bmatrix}0\\ 0.132\\ 0.533\end{bmatrix}$$

（3）隐藏层 2 → 输出层

计算输出层的加权输入 z3 和反归一化结果：

$$z_3=W_3*a_2+b_3=0.8*0+(-0.6)*0.132+0.4*0.533+0.2=0.334$$

反归一化输出

假设年龄归一化范围为[20,80]岁，对应[0,1]，则预测年龄=20+0.334×60≈40岁。

2 卷积神经网络基础

卷积神经网络，顾名思义，它是一种神经网络，其特别之处是引入了卷积运算，这种卷积运算特别适合于处理图像数据。卷积神经网络是 1990 年代因图像识别任务而提出来的，当时已经具备了深度学习网络结构的形态，但并未引起广泛应用，直到 2012 年层数大大增加的 AlexNet 提出，才引爆了深度学习的应用。

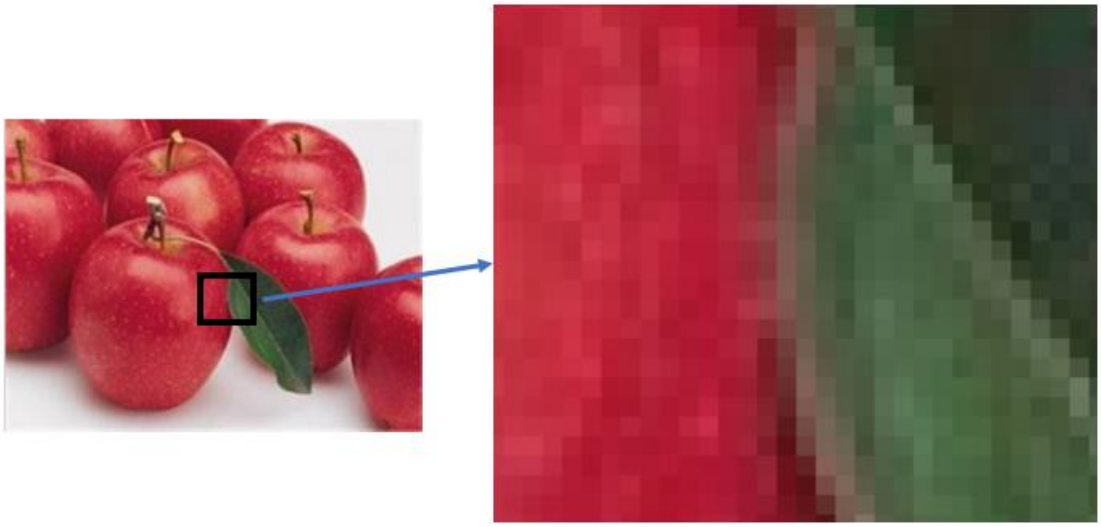
无独有偶，2017 年 Transformer 提出，直到五年之后 GPT 基于 Transformer 把网络规模扩大，才引爆了 Transformer 的应用。

2.1 什么是图像

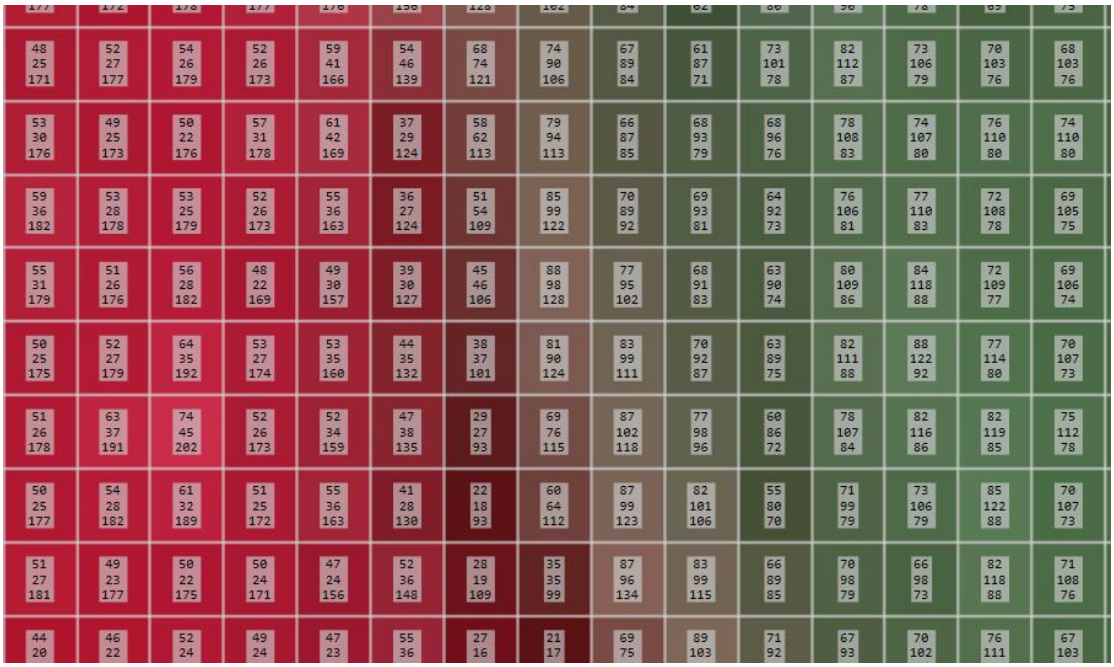
在介绍卷积神经网络之前，我们先来看看什么是图像数据。如下图，是一张普通图像，图像颜色有红、绿、白三种主要颜色。

图 一张普通图像

把这张图局部放大，会发现它是由很多不同颜色的方格组成的，如下图。每个方格称为像素。我们把一张图像水平和竖直方向各有多少个像素，称为图像的分辨率。常见的分辨率例如 1920*1080，即 200 万像素，意思是水平有 1920 个像素，竖直方向有 1080 个像素。现在我们用相机或手机拍摄的照片，往往动辄几千万像素。



图像中每个像素包含 3 个值，这是我们说的三原色 RGB（红、绿、蓝），取值范围是[0, 255]。如下图，对每个像素的 3 个值进行了可视化，这里像素值显示的顺序是 BGR，因此你会看到红色区域的 R 值比较大，绿色区域的 G 值比较大。



在计算机中，[0, 255]范围的值刚好用一个字节来表示，因此一个像素在计算机中占 3 个字节。

2.2 第一个卷积神经网络例子：手写字符识别

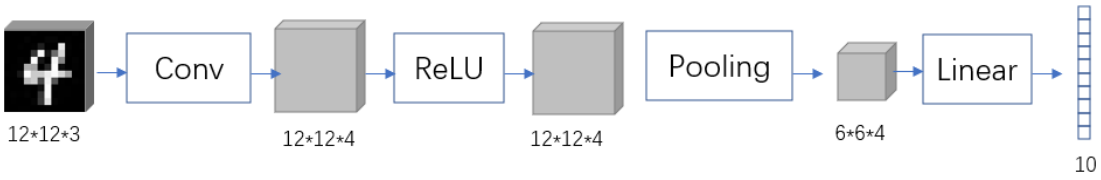
下面通过一个简单且实用的手写字符识别案例来讲解卷积神经网络的原理。

我们的任务是对 0-9 的手写数字图片进行分类。输入计算机中进行训练，经过大量训练，能通过输入的图像信息识别出对应的数字，并将该数字输出。

（此处补充一张手写字符识别的图）

原始的手写字符分辨率是 28*28，有 0~9 一共 10 个类别，我们把这个数据集保存为 jpg 格式，共享在本书配套的代码包中。

为了便于在书中展示，我们把图像进一步缩小为 12*12 分辨率，作为卷积神经网络的输入。网络结构如下图，输入为 12*12*3 的图像，经过 3*3 卷积，分辨率不变，通道数变为 4，经过 ReLU 激活函数，维度不变；经过 Pooling，宽和高降为一半，即 6*6*4；最后经过全连接，输出 10 个节点，代表 10 个类别。



图：一个简化的卷积神经网络结构，用于手写字符识别

如下图，手写字符图像“4”，在训练好的模型中经过一次卷积、激活、最大池化、展平与全连接过程，识别出数字 4 过程为例。

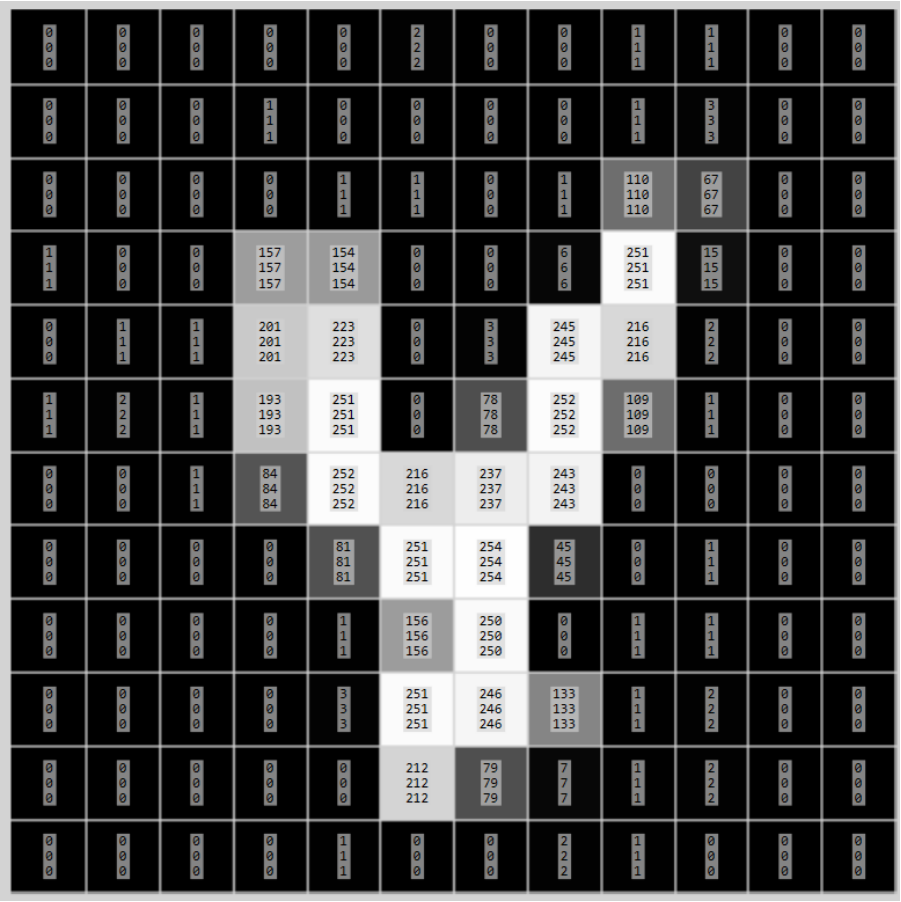


图 2.1(a) 12*12*3 的手写字符图像“4”

①卷积操作

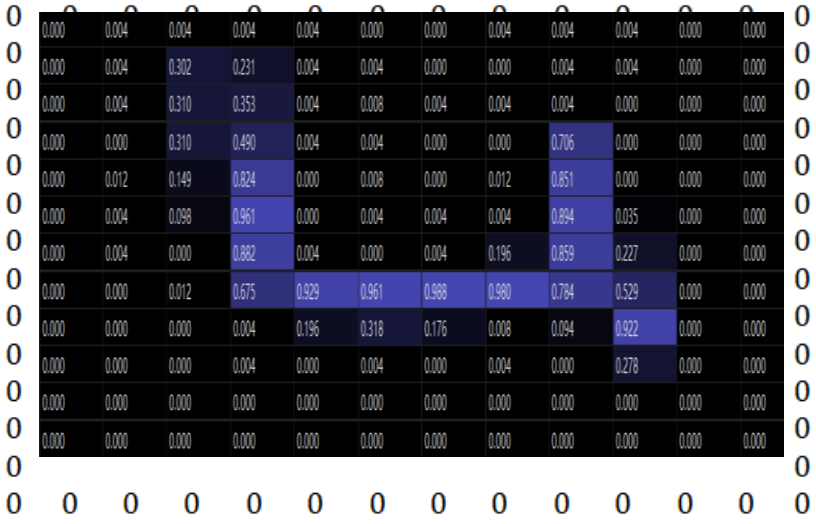


图 2.2(a) 向外填充一个单位

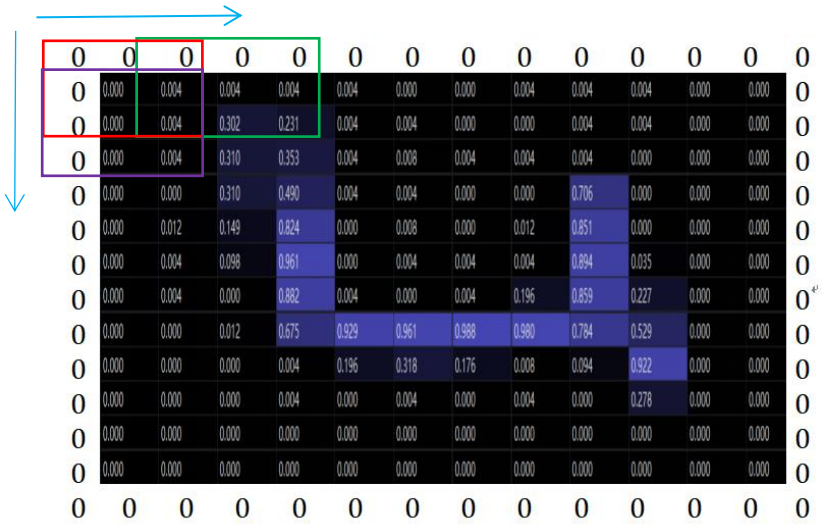


图 2.3(b) 卷积过程窗格移动过程

采用 3*3*3 的卷积核（K）进行检测，这个过程就相当于先把图像外围扩展一圈，扩展的值全部取零（向外填充 1 个单位，如图 2.2(a)），再把把卷积核作为一个窗口以固定的步幅（此处步幅为 1）从扩展后的图像左上角依次滑动（如图 2.2(b)，每次滑动时卷积核与窗口内对应元素相乘再相加，最后再加上偏置系数。以输入图像的左上角$I_{(0,0)}$处为例。

输入 12*12 的矩阵：

$$I_{(0,0)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0.004 \\ 0 & 0 & 0.004 \end{bmatrix}$$

经过一次卷积后的 3*3*3 卷积和：

$$S_{(0,0)} = f(I_{(0,0)} \cdot K_1 + I_{(0,0)} \cdot K_2 + I_{(0,0)} \cdot K_3) = -0.010096$$

其中，

$$K_1 = \begin{bmatrix} 0.050 & 0.173 & 0.270 \\ 0.165 & 0.604 & -0.416 \\ 0.145 & 0.329 & -0.449 \end{bmatrix} \quad K_2 = \begin{bmatrix} -0.113 & 0.355 & 0.320 \\ 0.286 & 0.340 & -0.514 \\ 0.307 & 0.549 & -0.416 \end{bmatrix}$$

$$K_3 = \begin{bmatrix} 0.121 & 0.140 & 0.437 \\ 0.366 & 0.388 & -0.491 \\ 0.133 & 0.465 & -0.238 \end{bmatrix}$$

加上偏置分量$b_1 = [0.006 \quad 0.093 \quad -0.009 \quad 0.003]$（输出为 4 通道），对应输出量：

$$y_{(0,0)} = f(S_{(0,0)} + b_1) = [-0.004096 \quad 0.082904 \quad -0.019096 \quad -0.007096]$$

按照上诉方法推导出其余位置对应的输出量，整理得到输入图像对应的卷积层输出矩阵。

$$Y_{conv1} = \begin{bmatrix} -0.004096 & \cdots & 0.006 \\ \vdots & \ddots & \vdots \\ 0.006 & \cdots & \end{bmatrix}$$

$$Y_{conv2} = \begin{bmatrix} 0.082904 & \cdots & 0.093 \\ \vdots & \ddots & \vdots \\ 0.093 & \cdots & \end{bmatrix}$$

$$Y_{conv3} = \begin{bmatrix} -0.019096 & \cdots & -0.009 \\ \vdots & \ddots & \vdots \\ -0.009 & \cdots & -0.009 \end{bmatrix}$$

$$Y_{conv4} = \begin{bmatrix} -0.007096 & \cdots & 0.003 \\ \vdots & \ddots & \vdots \\ 0.003 & \cdots & 0.003 \end{bmatrix}$$

对比在计算机中运算结果（如图 2.3），在误差允许的情况下，二者结果相同。

	0	1	2	3	4	5	6	7	8	9	10	11
0	-0.025	0.023	0.590	0.599	0.132	-0.003	-0.023	-0.020	-0.007	0.001	-0.011	-0.019
1	-0.030	-0.491	0.592	1.523	0.515	0.014	-0.006	-0.020	-0.010	-0.000	-0.008	-0.019
2	-0.031	-0.925	0.160	2.388	1.049	0.005	-0.002	0.063	1.333	0.396	-0.013	-0.019
3	-0.023	-0.937	-0.508	3.136	1.584	0.008	-0.001	-1.103	2.472	1.448	-0.019	-0.019
4	-0.038	-0.670	-1.537	3.588	2.297	0.005	-0.012	-2.324	2.901	2.666	0.002	-0.019
5	-0.041	-0.378	-2.524	3.348	2.889	-0.016	0.010	-2.238	2.996	3.306	0.160	-0.019
6	-0.031	-0.147	-2.726	2.795	4.738	2.456	2.201	0.048	3.182	4.089	0.645	-0.019
7	-0.024	-0.038	-2.376	-0.549	1.974	1.477	1.076	0.178	1.811	4.702	1.539	-0.019
8	-0.019	-0.035	-0.958	-1.461	-0.547	0.417	0.756	0.521	-0.697	2.914	2.112	-0.019
9	-0.019	-0.019	-0.031	-0.285	-0.411	0.073	0.429	0.090	-1.736	0.653	1.582	-0.019
10	-0.019	-0.019	-0.024	-0.018	-0.019	-0.018	-0.019	-0.018	-0.400	0.045	0.349	-0.019
11	-0.019	-0.019	-0.019	-0.019	-0.019	-0.019	-0.019	-0.019	-0.019	-0.019	-0.019	-0.019

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.084	-0.092	-0.456	0.099	0.345	0.100	0.091	0.081	0.083	0.096	0.106	0.093
1	0.076	-0.620	-1.221	0.315	0.935	0.095	0.093	0.075	0.080	0.109	0.108	0.093
2	0.078	-1.304	-1.971	0.538	1.675	0.093	0.109	-0.315	-0.894	0.887	0.100	0.093
3	0.078	-1.244	-2.441	-0.197	2.511	0.086	0.108	-1.624	-1.761	2.460	0.093	0.093
4	0.071	-0.938	-3.169	-1.057	3.557	0.085	0.092	-3.493	-1.822	3.852	0.132	0.093
5	0.058	-0.420	-3.963	-1.466	4.264	0.087	-0.039	-4.126	-1.738	3.825	0.415	0.093
6	0.078	-0.138	-3.970	-1.862	2.274	-0.783	-1.120	-4.312	-0.932	3.139	1.191	0.093
7	0.085	0.074	-3.080	-2.082	-0.178	-1.009	-0.974	-1.995	-0.274	1.373	2.533	0.093
8	0.093	0.067	-1.441	-2.196	-1.555	-0.705	0.050	0.258	-1.220	0.341	3.088	0.093
9	0.093	0.093	0.078	-0.354	-0.577	0.066	0.613	0.155	-2.436	0.169	2.100	0.093
10	0.093	0.093	0.085	0.094	0.091	0.094	0.091	0.094	-0.531	0.154	0.530	0.093
11	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093

	0	1	2	3	4	5	6	7	8	9	10	11
0	-0.003	0.128	0.434	0.573	0.238	-0.007	-0.000	0.002	0.000	-0.002	-0.011	-0.009
1	0.002	0.480	1.094	0.525	0.015	-0.004	0.004	0.002	-0.003	-0.023	-0.023	-0.009
2	-0.000	0.832	0.957	-0.286	-0.497	-0.009	-0.016	0.288	0.782	0.707	-0.018	-0.009
3	0.000	0.784	1.219	-0.185	-0.614	-0.021	-0.022	1.175	1.764	-0.180	-0.009	-0.009
4	0.006	0.586	1.567	0.258	-1.265	-0.010	-0.005	2.206	0.549	-1.800	0.028	-0.009
5	0.011	0.265	2.230	-0.038	-2.248	-0.010	0.076	2.590	0.572	-1.952	0.176	-0.009
6	-0.000	0.106	2.228	0.008	-1.206	2.447	2.737	4.712	1.302	-1.577	0.131	-0.009
7	-0.005	-0.003	1.821	0.146	-0.328	1.516	1.614	1.402	-1.345	-1.713	-0.317	-0.009
8	-0.009	0.004	0.758	-0.085	-1.610	-2.497	-3.171	-3.326	-1.764	-1.509	-2.216	-0.009
9	-0.009	-0.009	-0.000	0.216	-0.048	-0.861	-1.034	-0.285	1.178	-1.762	-2.379	-0.009
10	-0.009	-0.009	-0.005	-0.017	-0.013	-0.017	-0.013	-0.017	0.307	-0.576	-0.600	-0.009
11	-0.009	-0.009	-0.009	-0.009	-0.009	-0.009	-0.009	-0.009	-0.009	-0.009	-0.009	-0.009

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.003	-0.610	-1.234	-0.879	-0.232	-0.007	0.008	0.009	0.002	-0.001	0.003	0.003
1	0.006	-0.016	-0.615	-0.602	-0.139	-0.018	-0.016	-0.004	0.011	0.015	0.009	0.003
2	0.014	0.202	-0.185	-0.400	-0.032	0.018	0.022	-1.419	-1.800	-0.677	0.005	0.003
3	-0.018	0.500	-0.102	-0.774	-0.177	0.005	-0.014	-0.326	-1.204	-0.187	0.003	0.003
4	0.019	0.311	0.304	-0.624	0.066	0.002	0.014	0.359	-0.454	0.157	-0.031	0.003
5	0.011	0.305	0.973	0.069	0.420	0.013	-0.379	0.121	-0.666	-0.144	-0.187	0.003
6	0.014	0.052	1.107	-1.374	-3.733	-5.382	-5.074	-2.490	-1.281	-0.554	-0.291	0.003
7	0.006	0.031	1.949	3.332	3.147	2.831	3.597	4.234	2.049	-0.356	-0.306	0.003
8	0.003	0.011	0.450	1.686	2.809	3.045	2.651	2.343	3.058	1.589	0.827	0.003
9	0.003	0.003	0.014	0.139	0.421	0.542	0.356	0.164	1.264	1.377	0.709	0.003
10	0.003	0.003	0.006	0.007	0.008	0.007	0.008	0.007	0.185	0.286	0.141	0.003
11	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003

图 2.4 卷积后的结果

②ReLU 激活

对于上述卷积层输出矩阵Y_{conv}，仅仅是对输入量进行了线性变换，为了实现更复杂的非线性关系，引入了 ReLU 激活。于是可以将输出的卷积层矩阵Y_{conv}用 ReLU 函数进行激活，其中大于等于零的部分保持不变，小于零的部分变成零。激活后的矩阵如下。

$$Y_{ReLU1}=\begin{bmatrix}0 & \cdots & 0.006\\ \vdots & \ddots & \vdots\\ 0.006 & \cdots & 0.006\end{bmatrix}$$

$$Y_{ReLU2}=\begin{bmatrix}0.82904 & \cdots & 0.093\\ \vdots & \ddots & \vdots\\ 0.093 & \cdots & 0.093\end{bmatrix}$$

$$Y_{ReLU3}=\begin{bmatrix}0 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & 0\end{bmatrix}$$

$$Y_{ReLU4}=\begin{bmatrix}0 & \cdots & 0.003\\ \vdots & \ddots & \vdots\\ 0.003 & \cdots & 0.003\end{bmatrix}$$

对比在计算机中运算结果（如图 2.4），在误差允许的情况下，二者结果相同。可见上述计算方法是合理的。

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.000	0.023	0.590	0.599	0.132	0.000	0.000	0.000	0.000	0.001	0.000	0.000
1	0.000	0.000	0.592	1.523	0.515	0.014	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	0.000	0.160	2.388	1.049	0.005	0.000	0.063	1.333	0.396	0.000	0.000
3	0.000	0.000	0.000	3.136	1.584	0.008	0.000	0.000	2.472	1.448	0.000	0.000
4	0.000	0.000	0.000	3.588	2.297	0.005	0.000	0.000	2.901	2.666	0.002	0.000
5	0.000	0.000	0.000	3.348	2.889	0.000	0.010	0.000	2.996	3.306	0.160	0.000
6	0.000	0.000	0.000	2.795	4.738	2.456	2.201	0.048	3.182	4.089	0.645	0.000
7	0.000	0.000	0.000	0.000	1.974	1.477	1.076	0.178	1.811	4.702	1.539	0.000
8	0.000	0.000	0.000	0.000	0.000	0.417	0.756	0.521	0.000	2.914	2.112	0.000
9	0.000	0.000	0.000	0.000	0.000	0.073	0.429	0.090	0.000	0.653	1.582	0.000
10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.045	0.349	0.000
11	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

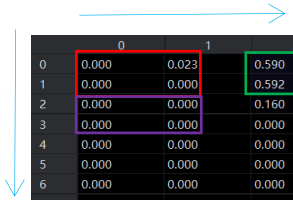
	0	1	2	3	4	5	6	7	8	9	10	11
0	0.084	0.000	0.000	0.099	0.345	0.100	0.091	0.081	0.083	0.096	0.106	0.093
1	0.076	0.000	0.000	0.315	0.935	0.095	0.093	0.075	0.080	0.109	0.108	0.093
2	0.078	0.000	0.000	0.538	1.675	0.093	0.109	0.000	0.000	0.887	0.100	0.093
3	0.078	0.000	0.000	0.000	2.511	0.086	0.108	0.000	0.000	2.460	0.093	0.093
4	0.071	0.000	0.000	0.000	3.557	0.085	0.092	0.000	0.000	3.852	0.132	0.093
5	0.058	0.000	0.000	0.000	4.264	0.087	0.000	0.000	0.000	3.825	0.415	0.093
6	0.078	0.000	0.000	0.000	2.274	0.000	0.000	0.000	0.000	3.139	1.191	0.093
7	0.085	0.074	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.373	2.533	0.093
8	0.093	0.067	0.000	0.000	0.000	0.000	0.050	0.258	0.000	0.341	3.088	0.093
9	0.093	0.093	0.078	0.000	0.000	0.066	0.613	0.155	0.000	0.169	2.100	0.093
10	0.093	0.093	0.085	0.094	0.091	0.094	0.091	0.094	0.000	0.154	0.530	0.093
11	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093	0.093

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.000	0.128	0.434	0.573	0.238	0.000	0.000	0.002	0.000	0.000	0.000	0.000
1	0.002	0.480	1.094	0.525	0.015	0.000	0.004	0.002	0.000	0.000	0.000	0.000
2	0.000	0.832	0.957	0.000	0.000	0.000	0.000	0.288	0.782	0.707	0.000	0.000
3	0.000	0.784	1.219	0.000	0.000	0.000	0.000	1.175	1.764	0.000	0.000	0.000
4	0.006	0.586	1.567	0.258	0.000	0.000	0.000	2.206	0.549	0.000	0.028	0.000
5	0.011	0.265	2.230	0.000	0.000	0.000	0.076	2.590	0.572	0.000	0.176	0.000
6	0.000	0.106	2.228	0.008	0.000	2.447	2.737	4.712	1.302	0.000	0.131	0.000
7	0.000	0.000	1.821	0.146	0.000	1.516	1.614	1.402	0.000	0.000	0.000	0.000
8	0.000	0.004	0.758	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.216	0.000	0.000	0.000	0.000	1.178	0.000	0.000	0.000
10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.307	0.000	0.000	0.000
11	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.003	0.000	0.000	0.000	0.000	0.000	0.008	0.009	0.002	0.000	0.003	0.003
1	0.006	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.011	0.015	0.009	0.003
2	0.014	0.202	0.000	0.000	0.000	0.018	0.022	0.000	0.000	0.000	0.005	0.003
3	0.000	0.500	0.000	0.000	0.000	0.005	0.000	0.000	0.000	0.000	0.003	0.003
4	0.019	0.311	0.304	0.000	0.066	0.002	0.014	0.359	0.000	0.157	0.000	0.003
5	0.011	0.305	0.973	0.069	0.420	0.013	0.000	0.121	0.000	0.000	0.000	0.003
6	0.014	0.052	1.107	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.003
7	0.006	0.031	1.949	3.332	3.147	2.831	3.597	4.234	2.049	0.000	0.000	0.003
8	0.003	0.011	0.450	1.686	2.809	3.045	2.651	2.343	3.058	1.589	0.827	0.003
9	0.003	0.003	0.014	0.139	0.421	0.542	0.356	0.164	1.264	1.377	0.709	0.003
10	0.003	0.003	0.006	0.007	0.008	0.007	0.008	0.007	0.185	0.286	0.141	0.003
11	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003

图 2.5 ReLU 后的结果

③最大池化



	0	1	2	3	4	5	6	7	8	9	10	11
0	0.000	0.023	0.590	0.599	0.132	0.000	0.000	0.000	0.000	0.001	0.000	0.000
1	0.000	0.000	0.592	1.523	0.515	0.014	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	0.000	0.160	2.388	1.049	0.005	0.000	0.063	1.333	0.396	0.000	0.000
3	0.000	0.000	0.000	3.136	1.584	0.008	0.000	0.000	2.472	1.448	0.000	0.000
4	0.000	0.000	0.000	3.588	2.297	0.005	0.000	0.000	2.901	2.666	0.002	0.000
5	0.000	0.000	0.000	3.348	2.889	0.000	0.010	0.000	2.996	3.306	0.160	0.000
6	0.000	0.000	0.000	2.795	4.738	2.456	2.201	0.048	3.182	4.089	0.645	0.000
7	0.000	0.000	0.000	0.000	1.974	1.477	1.076	0.178	1.811	4.702	1.539	0.000
8	0.000	0.000	0.000	0.000	0.000	0.417	0.756	0.521	0.000	2.914	2.112	0.000
9	0.000	0.000	0.000	0.000	0.000	0.073	0.429	0.090	0.000	0.653	1.582	0.000
10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.045	0.349	0.000
11	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

图 2.3 最大池化窗格移动过程

对于上述激活后的矩阵Y_{ReLU}对应维度较大，为了降低计算复杂度同时保留图像边缘特征，提出了最大池化的概念。例如将经过激活的矩阵Y_{ReLU}做 2*2 的最大池化，就是将 2*2 的窗口自左上起步在矩阵Y_{ReLU}上依次滑动（窗口不重叠，如图 2.3），每次取窗口内的最大值进行输出。最大池化后的矩阵如下。

$$Y_{pool1} = \begin{bmatrix} & \\ & \end{bmatrix}$$

$$Y_{pool2} = \begin{bmatrix} & \\ & \end{bmatrix}$$

$$Y_{pool3} = \begin{bmatrix} & \\ & \end{bmatrix}$$

$$Y_{pool4} = \begin{bmatrix} & \\ & \end{bmatrix}$$

通过对计算机中运算结果（如图 2.5）进行对比，在误差允许的情况下，二者结果相同。

	0	1	2	3	4	5
0	0.023	1.523	0.515	0.000	0.001	0.000
1	0.000	3.136	1.584	0.063	2.472	0.000
2	0.000	3.588	2.889	0.010	3.306	0.160
3	0.000	2.795	4.738	2.201	4.702	1.539
4	0.000	0.000	0.417	0.756	2.914	2.112
5	0.000	0.000	0.000	0.000	0.045	0.349

	0	1	2	3	4	5
0	0.084	0.315	0.935	0.093	0.109	0.108
1	0.078	0.538	2.511	0.109	2.460	0.100
2	0.071	0.000	4.264	0.092	3.852	0.415
3	0.085	0.000	2.274	0.000	3.139	2.533
4	0.093	0.078	0.066	0.613	0.341	3.088
5	0.093	0.094	0.094	0.094	0.154	0.530

	0	1	2	3	4	5
0	0.480	1.094	0.238	0.004	0.000	0.000
1	0.832	1.219	0.000	1.175	1.764	0.000
2	0.586	2.230	0.000	2.590	0.572	0.176
3	0.106	2.228	2.447	4.712	1.302	0.131
4	0.004	0.758	0.000	0.000	1.178	0.000
5	0.000	0.000	0.000	0.000	0.307	0.000

	0	1	2	3	4	5
0	0.006	0.000	0.000	0.009	0.015	0.009
1	0.500	0.000	0.018	0.022	0.000	0.005
2	0.311	0.973	0.420	0.359	0.157	0.003
3	0.052	3.332	3.147	4.234	2.049	0.003
4	0.011	1.686	3.045	2.651	3.058	0.827
5	0.003	0.007	0.008	0.008	0.286	0.141

图 2.6 池化后的结果

④展平与全连接

最大池化后的矩阵Y_{pool}对应的网络结构较为复杂，如果直接将其进行全连接，会大大增加计算难度，因此可以先进行展平操作，将多维向量变成一维向量，展平后的矩阵如下。

$$\mathrm{x} = [\quad \cdots \quad]$$

将展平后的矩阵$\mathbf{x}$进行全连接，将图像对应到相应的数字类型，具体过程为将展平后的矩阵$\mathbf{x}$与权重矩阵$\mathbf{W}$对应位数相乘再相加，最后加上偏置分量，通过激活函数。全连接后的结果如下。

$$\mathrm{Z} = f(\mathrm{W} \cdot \mathrm{x} + \mathrm{b}_2) = [\quad]$$

其中，

$$\mathrm{W} = \begin{bmatrix} -0.081 & \cdots & -0.009 \\ \vdots & \ddots & \vdots \\ -0.017 & \cdots & 0.194 \end{bmatrix}$$

$$b_2 = [-0.223 \quad 0.192 \quad \ldots \quad -0.157 \quad 0.048]$$

对比计算机算出来的最终结果（如图 2.6），在允许有点误差的情况下，二者结果仍相同，因此不难看出上述方法是正确的。

```
tensor([[ -2.4567, -10.2763,   1.6618,  -5.3785,   7.1892,  -4.9864,   1.9331,
          -0.7353,  -4.0777,  -2.7269]], grad_fn=<AddmmBackward0>)
tensor([4])
```

图 5.7 最终结果

经过结果的对比，不难看出理论公式推到计算与实际计算机执行结果吻合，这不仅验证了卷积神经网络（CNN）各层运算逻辑的数学严谨性，更表明手写字符识别模型训练参数与架构设计在数值计算层面具备高度一致性。

2.3 戴口罩识别案例

人脸戴口罩识别的实现的功能是，打开摄像头，拍摄到人脸，判断人脸是否戴口罩。算法实现的流程图如下图所示。首先输入摄像头拍摄到的原图，通过人脸检测，将原图中的人脸框出来，然后把原图上人脸区域内容抠图，并 resize（缩放）到标准大小，比如 28×28，最后通过一个二分类器判断它是戴口罩/不戴口罩。

这里人脸检测模块有很多开源的实现方式，例如 opencv 自带的 FaceDetectorYN 类可以实现。二分类模块在手写字符识别基础上修改即可训练和测试。

数值计算过程

背景说明：
我们使用一张 5×5 的灰度图模拟戴口罩的人脸图像，像素值如下：
- 背景为白色（255）
- 口罩区域为灰色（100）
- 其他面部区域为肤色（50）

输入图像（5×5）：

$$I = \begin{bmatrix} 255 & 255 & 100 & 100 & 255 \\ 255 & 100 & 50 & 50 & 255 \\ 100 & 50 & 50 & 50 & 100 \\ 100 & 50 & 50 & 50 & 100 \\ 255 & 255 & 100 & 100 & 255 \end{bmatrix}$$

1. 卷积操作
使用与手写字符识别相同的 3×3 卷积核，进行特征提取，卷积核为：

$$K = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

偏置 b = 0，采用 valid 卷积，步长为 1，输出尺寸为 3×3。
计算位置 (0,0)
取输入图像的左上角 3×3 区域：

$$I_{(0,0)} = \begin{bmatrix} 255 & 255 & 100 \\ 255 & 100 & 50 \\ 100 & 50 & 50 \end{bmatrix}$$

计算卷积和：

$$\begin{aligned} S_{(0,0)} &= (1 \times 255) + (0 \times 255) + (-1 \times 100) \\ &\quad + (1 \times 255) + (0 \times 100) + (-1 \times 50) \\ &\quad + (1 \times 100) + (0 \times 50) + (-1 \times 50) \\ &= 255 - 100 + 255 - 50 + 100 - 50 \\ &= 410 \end{aligned}$$

加上偏置 b = 0 后：

$$y_{(0,0)} = 410$$

其他位置

类似计算其他 3×3 区域，得到卷积层输出矩阵：

$$Y_{\text{conv}} = \begin{bmatrix} 410 & 205 & -410 \\ 305 & 50 & -305 \\ 255 & 155 & -255 \end{bmatrix}$$

2. ReLU 激活

使用 ReLU 激活函数，负值变为 0，得到：

$$Y_{\text{ReLU}} = \begin{bmatrix} 410 & 205 & 0 \\ 305 & 50 & 0 \\ 255 & 155 & 0 \end{bmatrix}$$

3. 最大池化

采用 2×2 最大池化（stride = 1）。例如，取左上角区域的最大值 410，池化后的结果为：

$$Y_{\text{pool}} = \begin{bmatrix} 410 & 205 \\ 305 & 155 \end{bmatrix}$$

1. 展平与全连接层

将池化层输出展平成向量：

$$x=[410,205,305,155]$$

全连接层将 4 维向量映射到 2 类（戴口罩与未戴口罩）。以戴口罩为例，设权重向量为：

$$w = [0.1,\ 0.1,\ 0.1,\ 0.1]$$

偏置 $b_{\text{fc}} = 0$，计算得到：

$$z=0.1\times(410+205+305+155)=0.1\times1075=107.5$$

经过 Sigmoid 激活函数后：

$$y_{\text{hat}} = \frac{1}{1+e^{-107.5}} \approx 1.0$$

由于结果接近 1，模型预测该图像为戴口罩。

配套代码（戴口罩识别案例）：

```
import numpy as np

def conv2d(image, kernel, bias):
    k_h, k_w = kernel.shape
```

```
    i_h, i_w = image.shape
    output = np.zeros((i_h - k_h + 1, i_w - k_w + 1))
    for y in range(output.shape[0]):
        for x in range(output.shape[1]):
            roi = image[y:y+k_h, x:x+k_w]
            output[y, x] = np.sum(roi * kernel) + bias
    return output

def relu(x):
    return np.maximum(0, x)

def max_pooling(matrix, pool_size=2, stride=1):
    m_h, m_w = matrix.shape
    o_h = (m_h - pool_size) // stride + 1
    o_w = (m_w - pool_size) // stride + 1
    output = np.zeros((o_h, o_w))
    for y in range(o_h):
        for x in range(o_w):
            window = matrix[y*stride:y*stride+pool_size, x*stride:x*stride+pool_size]
            output[y, x] = np.max(window)
    return output

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# 戴口罩识别完整代码
I = np.array([
    [255, 255, 100, 100, 255],
    [255, 100, 50, 50, 255],
```

```
        [100, 50, 50, 50, 100],
        [100, 50, 50, 50, 100],
        [255, 255, 100, 100, 255]
], dtype=np.float32)

K = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]], dtype=np.float32)
b_conv = 0

# 卷积层
conv_result = conv2d(I, K, b_conv)
print("卷积结果:\n", conv_result)

# ReLU 激活
relu_result = relu(conv_result)
print("\nReLU 激活结果:\n", relu_result)

# 最大池化
pool_result = max_pooling(relu_result)
print("\n 池化结果:\n", pool_result)

# 全连接层与分类
flatten = pool_result.flatten()
w_fc = np.array([0.1, 0.1, 0.1, 0.1])
b_fc = 0
z = np.dot(flatten, w_fc) + b_fc

# Sigmoid 激活
y_hat = sigmoid(z)
```

print("\n 预测概率:", y_hat)

print("最终预测结果:", "戴口罩" if y_hat > 0.5 else "未戴口罩")

2.4 残差网络案例

背景说明：
残差网络通过引入残差连接 y=F(x)+x 来避免深层网络训练中的梯度消失问题。以下示例使用一个简单的二维向量展示残差连接的计算过程。

输入向量：

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

权重矩阵：

$$W = \begin{bmatrix} 0.5 & 0.2 \\ -0.3 & 0.8 \end{bmatrix}$$

1. 计算 F(x)

计算 F(x)=W×x

$$F(x) = \begin{bmatrix} 0.5 \times 1 + 0.2 \times 2 \\ -0.3 \times 1 + 0.8 \times 2 \end{bmatrix} = \begin{bmatrix} 0.5 + 0.4 \\ -0.3 + 1.6 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 1.3 \end{bmatrix}$$

2. 计算残差块的输出

将 x 加到 F(x)上：

$$y = F(x) + x = \begin{bmatrix} 0.9 \\ 1.3 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1.9 \\ 3.3 \end{bmatrix}$$

配套代码（残差网络案例）

```
import numpy as np


# 示例 3：残差网络案例


# 输入向量

x = np.array([1.0, 2.0])


# 权重矩阵 W

W = np.array([[0.5, 0.2],

              [-0.3, 0.8]])


# 计算 F(x)

F_x = np.dot(W, x)

print("F(x) =", F_x)


# 计算残差输出 y = F(x) + x

y = F_x + x

print("残差块输出 y = F(x) + x =", y)
```

2.4 章节总结

1. 手写字符识别案例：

1. 利用 5×5 图像、3×3 卷积核、加偏置、ReLU 激活、2×2 最大池化、全连接层和 Softmax 处理，实现数字分类（示例中预测数字 3）。

2. 戴口罩识别案例：

    1. 模拟 5×5 人脸图像，通过卷积、ReLU 激活、最大池化和全连接层（映射到二分类）实现戴口罩识别，利用 Sigmoid 得到类别概率。

3. 残差网络案例：

    1. 通过简单的二维向量和矩阵乘法，展示残差连接 y=F(x)+xy = F(x) + xy=F(x)+x 的计算过程，说明其在缓解梯度消失问题中的作用。

3 目标检测案例

3.1 目标检测是什么

目标检测是在一幅图像中找到指定类别的物体的类别和位置。如图 8，在图像中找到了行人和公交车，并且用不同颜色的矩形框把物体位置标识了出来。其中不同颜色代表了不同类别的物体。有意思的是，图中物体甚至发生了遮挡，只能看到一部分，算法也能正确检测出物体。

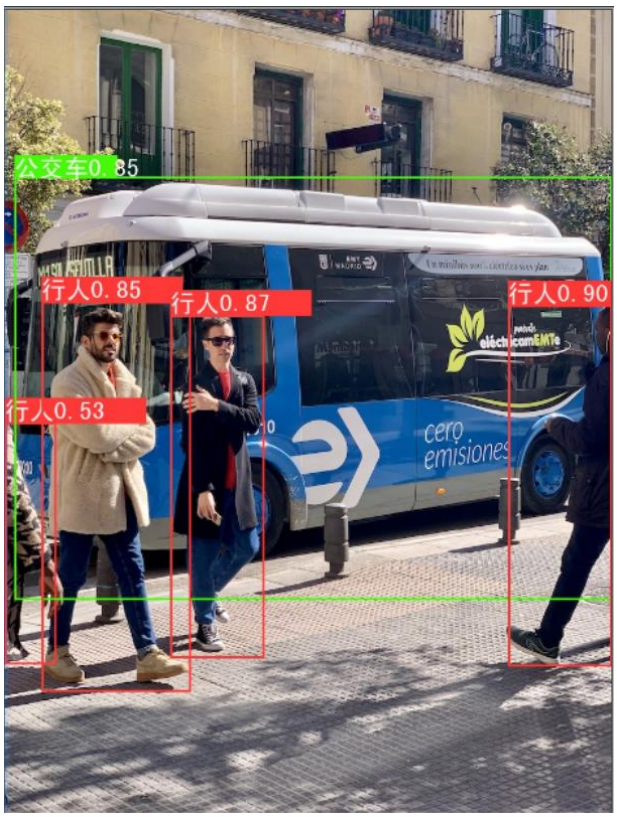


图 8 目标检测效果图

3.2 YOLO 算法原理

图 9 是 YOLO 目标检测的一个简化版原理图。首先是输入一张图像，经过 YOLO 模型处理后，输出一个 3x3x8 的数据块（也称为特征图，数学上也称为张量）。

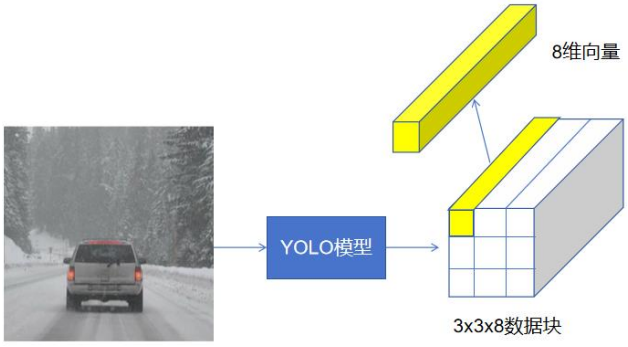


图 9 YOLO 简易原理图

那么这个 3x3x8 的数据块的含义是什么呢？图像经过 YOLO 模型处理后将原图拆解为 9（3*3）个 8 维向量，我们可以理解为此时的图像被划分成了一个 3*3 的格子，每一个格子都是一个八维向量，这个格子是虚拟的，不是真实存在

的，如图 3 所示

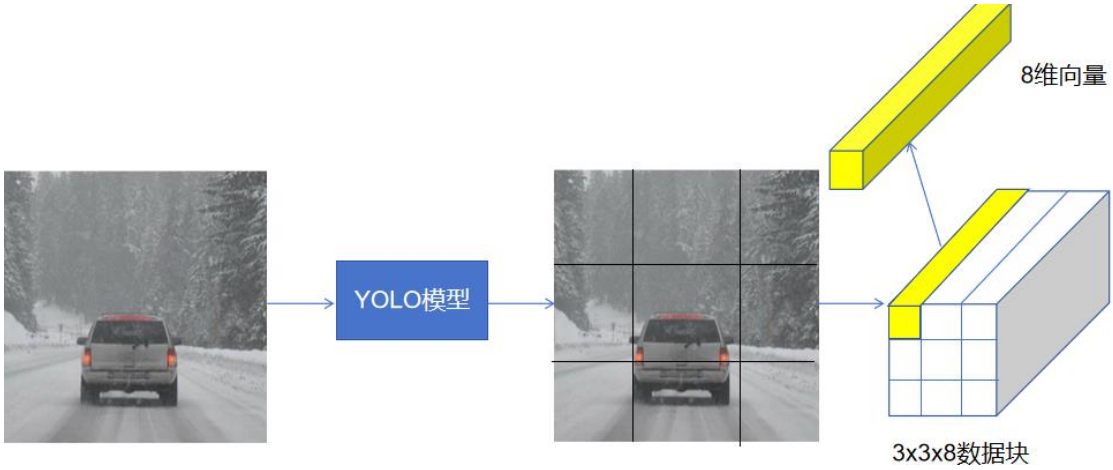


图 10 图像被划分为 3*3 的网格

这个 8 维向量有什么具体含义呢？我们以左上角黄色的 8 维向量为例，它表示原图的左上角的物体信息，即是否含有物体、矩形框位置、物体类别这 3 个信息，8 维向量的具体详细含义如图 11 所示。第 1 维向量表示了该图片内是否包含了物体。第 2 和第 3 维度表示的是矩形框的中心坐标 x,y，这个中心坐标是相当于整张图片来说的，例如这个车子在这个图片内的中心坐标 x,y 大致为(0.5,0.7)。第四和第五维度表示的是这个矩形框的宽和高。第 6、第 7 和第 8 维度的向量表示了检测物体的类别。

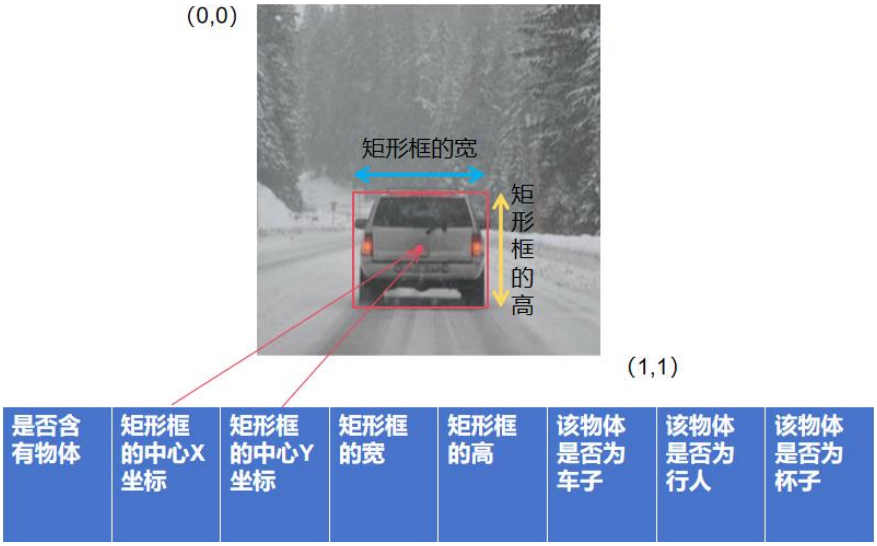


图 11 八维向量的具体含义

有人就会问了，如果一个格子里有多个物体怎么办，当前的八维向量不是只能输出一个物体吗？我们可以增加每个格子对应的向量的长度，也就是把输出的

向量维度由 8 变为 16，如图图 12 所示，16 维度的向量就能检测两种物体了。

还有个问题是物体分布在这么多格子里，那到底由哪个格子来检测这个物体呢？在 YOLO 中，物体的中心点落在哪个格子内，就由哪个格子来检测。

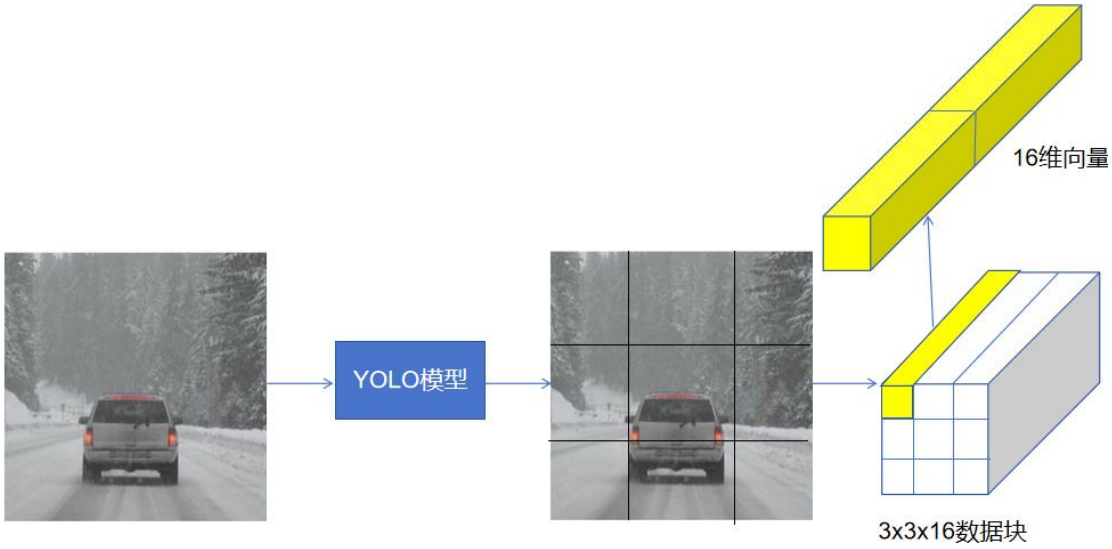


图 12 增加向量长度

在了解完 YOLO 输出向量的含义后，我们就要去了解 YOLO 是如何检测图像中的目标。在此之前，我们先了解一下在 YOLO 问世之前，人们是怎么去检测图像中物体的呢？

世界上第一个实时的人脸检测算法 VJ（全称）算法。如果你有一个人脸二分类器，可以判断一张图像是否为人脸。如何用这个二分类器检测一张大图上的所有人脸呢？我们都知道，要判断一张图片是否为人脸，那么人脸必须占据图像的主要部分。要在一张大图上检测出所有人脸，我们就得把视野缩小，由此将人脸在视野中的比重放大。所以我们可以使用一个小窗口代表缩小的视野，使用它去扫描整张图片，随后判断视野内是否为人脸，这种方法就叫做滑动窗口。由图 13 所示

图 13 滑动窗口示意图（这图还要修改 表现力不强）

比对一下人脸二分类器和人脸检测的区别。人脸二分类器是判断一张图像是否为人脸，它的前提是人脸必须占据图像的主要部分。人脸检测是在一张大图中找到所有的人脸的位置，由图 14 所示可以看出两者的区别。

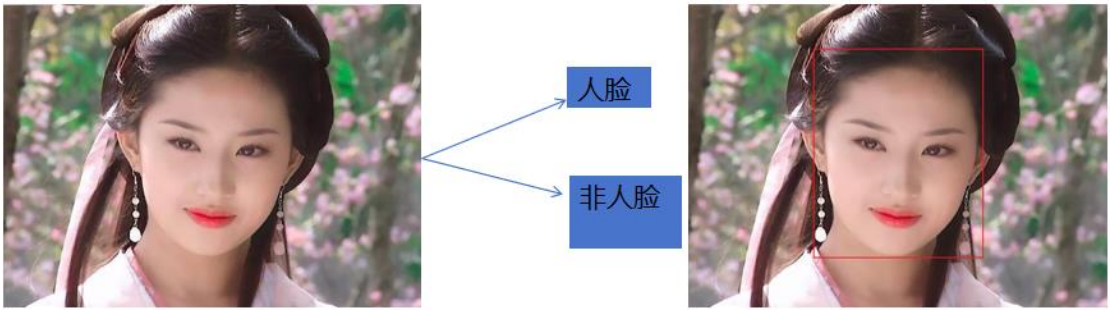


图 14 人脸识别和人脸检测的对比

在人脸检测中，当滑动窗口划到某个位置时，要判断是否为人脸，以及包围框的准确坐标。这时你求得的包围框坐标是相对于当前滑动窗口的坐标。那么，在 YOLO 中，输出的包围框是相对于什么的坐标呢？我们要提出一个新概念——anchor box，滑动窗口就是一种 anchor box 的特例。上面讲解滑动窗口时，我们可以发现滑动窗口只有一种宽高比（典型情况为正方形），这在检测一些宽高比差异很大的物体时，例如行人和车子时会有较大的问题。

3.3 目标检测的评价指标

3.3.1 精确度（Precision）与召回率（Recall）

在目标检测技术中，精确度（Precision）与召回率（Recall）是衡量算法性能的两个核心指标。精确度是指检测出的目标中正确目标的比例，而召回率则是指所有真实目标中被正确检测出的比例。在实际应用中，这两个指标往往需要权衡，因为提高精确度可能会降低召回率，反之亦然。例如，在自动驾驶系统中，高召回率至关重要，因为漏检一个行人可能导致严重后果；而高精确度则有助于减少误报，避免不必要的紧急制动。研究者们经常引用“帕累托最优”原则来描述这种权衡，即在不损害一个指标的前提下尽可能提高另一个指标。例如，一个目标检测模型可能在特定数据集上达到 85%的精确度和 70%的召回率，这表明模型在识别目标时较为准确，但仍有改进空间以减少漏检。

3.3.2 平均精度均值（mAP）

在目标检测技术领域，平均精度均值（mAP）是衡量检测算法性能的关键指标之一。mAP 综合考虑了检测的精确度（Precision）和召回率（Recall），提供了一个全面评估模型在各种阈值下的平均表现的方法。例如，在一个包含多个类别的目标检测任务中，mAP 能够反映出模型在识别不同类别的目标时的准确性和完整性。一个高 mAP 值通常意味着模型在检测目标时既准确又全面，而一个低 mAP 值则可能表明模型在某些类别上存在漏检或误检的问题。

以深度学习为基础的目标检测算法，如 YOLO 系列和 Faster R-CNN，其性能的评估往往依赖于 mAP。例如，在 PASCAL VOC 或 COCO 等标准数据集上，mAP 被用作比较不同算法优劣的基准。在这些数据集上，mAP 的计算通常涉及对每个类别的检测结果进行平均，以确保算法在所有类别上都有良好的表现。研究者们通过不断优化算法，努力提高 mAP 值，以期达到更高的检测精度和更广的适用范围。

在实际应用中，mAP 的高低直接影响到目标检测技术的实用性和可靠性。例如，在自动驾驶汽车的视觉系统中，高 mAP 值意味着系统能够更准确地识别道路上的行人、车辆和其他障碍物，从而提高行车安全。正如安德鲁·恩格所言：

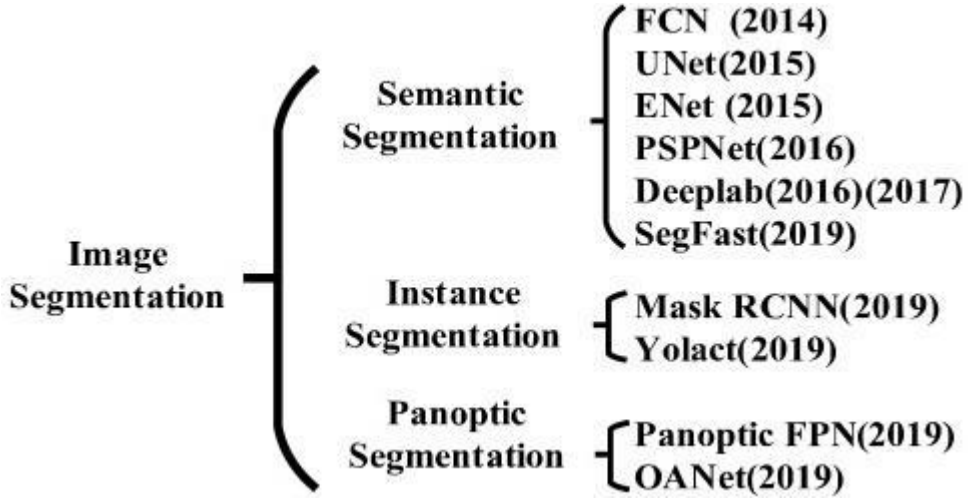
“在机器学习中，模型的性能评估是理解其能力的关键。”因此，mAP 作为评估指标，不仅为研究者提供了改进算法的依据，也为最终用户提供了衡量技术成熟度的尺度。

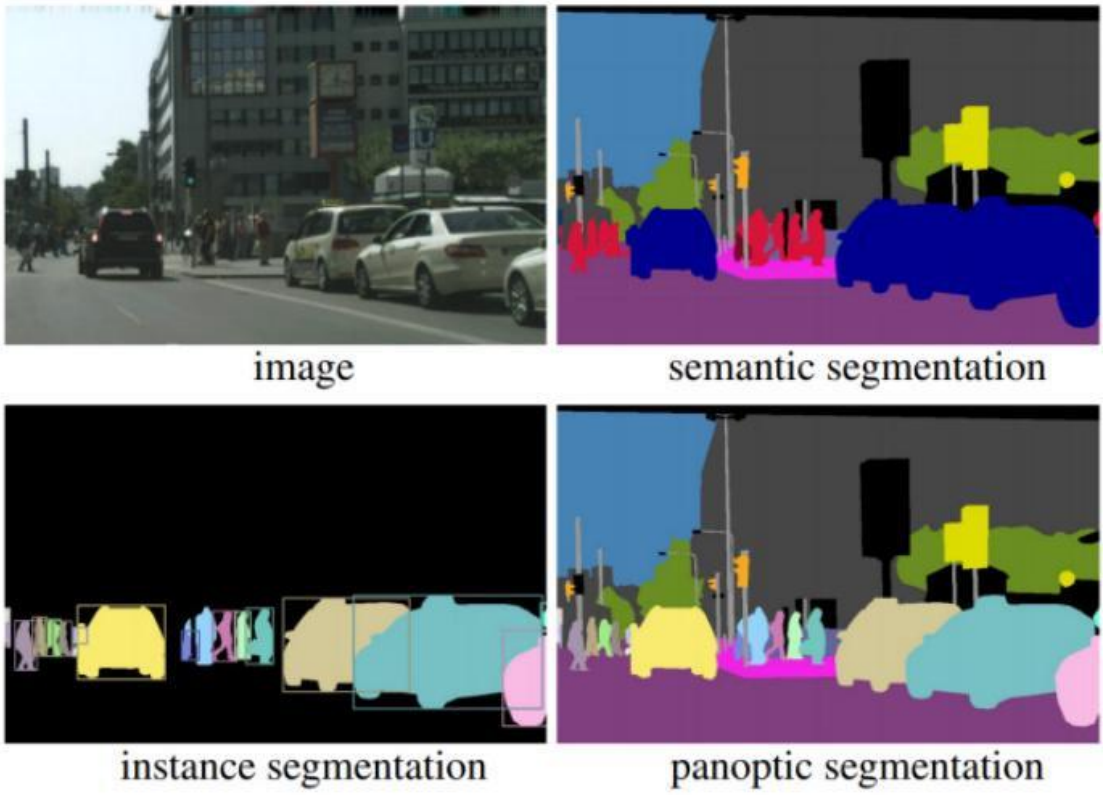
4 图像分割案例

4.1 图像分割的基本概念

图像分割就像用不同颜色的笔给照片涂色，把图片中的每个区域划分成不同类别。例如，把一张街道照片分成“道路”“汽车”“行人”等区域。所有分割技术的核心目标都是让计算机像人一样理解图片中的物体

图像分割主要包括以下三种任务：





（1）语义分割（Semantic Segmentation）

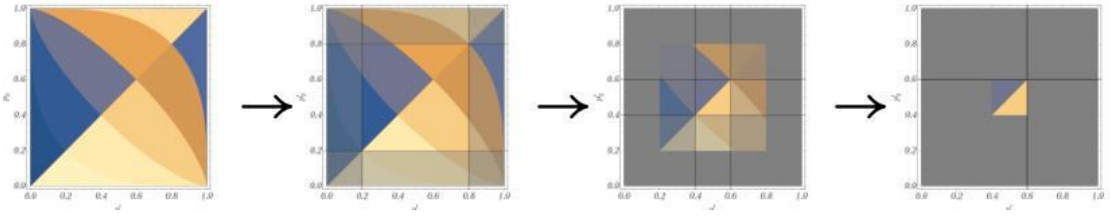
原理：为每个像素分配一个类别标签，但不区分同类物体的不同个体。

例子：一张图片中有 3 辆汽车，语义分割会将所有汽车的像素都标为“汽车”，但不会区分哪辆是 A 车，哪辆是 B 车。

算法：常用模型如 FCN、UNet。例如，UNet 通过编码器（缩小图片提取特征）和解码器（放大特征图恢复细节）完成分割。

计算案例：

i. 输入图片（如 224x224 像素）。
ii. 通过卷积层提取特征，得到高维特征图（如 28x28x512）。
iii. 上采样（反卷积）将特征图恢复到原图尺寸。
iv. 对每个像素用 Softmax 分类（如输出“汽车”或“行人”的概率）。



（2）实例分割（Instance Segmentation）

- **原理**：在语义分割基础上，区分同一类别的不同个体。例如，把 3 辆汽车分别标为“汽车 1”“汽车 2”“汽车 3”。
- **例子**：Mask R-CNN 模型先检测物体的边界框，再为每个框内的像素生成掩码（mask）。
- **算法步骤**（以 Mask R-CNN 为例）：
  i. **区域提议**：用 RPN（区域提议网络）生成候选框（例如 1000 个框）。
  ii. **分类与回归**：对每个框预测类别（如“汽车”）并调整框的位置。
  iii. **生成掩码**：在框内用小型神经网络预测每个像素是否属于该物体，生成二值掩码（0 或 1）。
- **计算案例**：

  i. 输入候选框为 1000 个，经筛选保留 50 个置信度高的框。
  ii. 对每个框内的 7x7 区域进行 ROI Align（精确对齐特征），输出 14x14 的特征。
  iii. 通过卷积层生成 28x28 的掩码，判断每个像素是否属于该物体。

(3) 全景分割（Panoptic Segmentation）

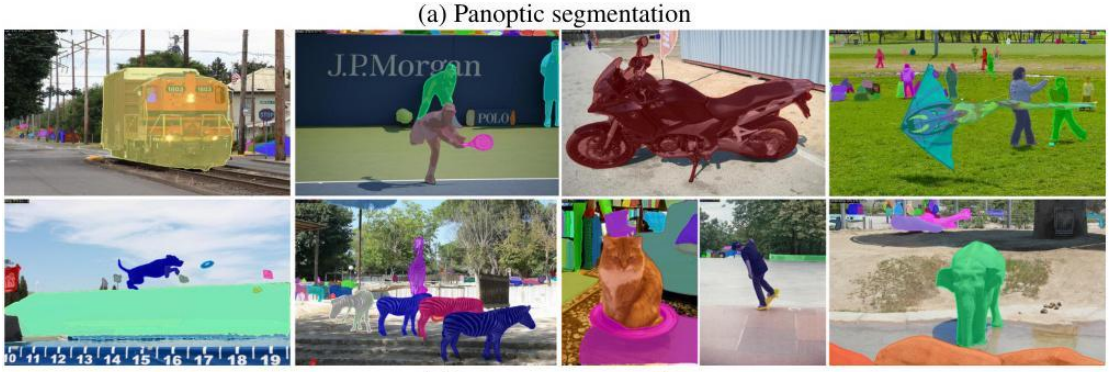
- **原理**：结合语义分割和实例分割，为每个像素分配**类别标签+实例 ID**。例如，“汽车-1”“汽车-2”“道路-无 ID”。
- **例子**：Panoptic FPN 模型通过特征金字塔提取多尺度特征，同时输出语义标签和实例 ID。
- **关键步骤**：
  i. **语义分支**：预测每个像素的类别（如“道路”“汽车”）。
  ii. **实例分支**：预测每个物体的掩码和 ID。

iii. **融合结果**：将两类结果合并，确保无重叠（例如优先保留实例分割的结果）。

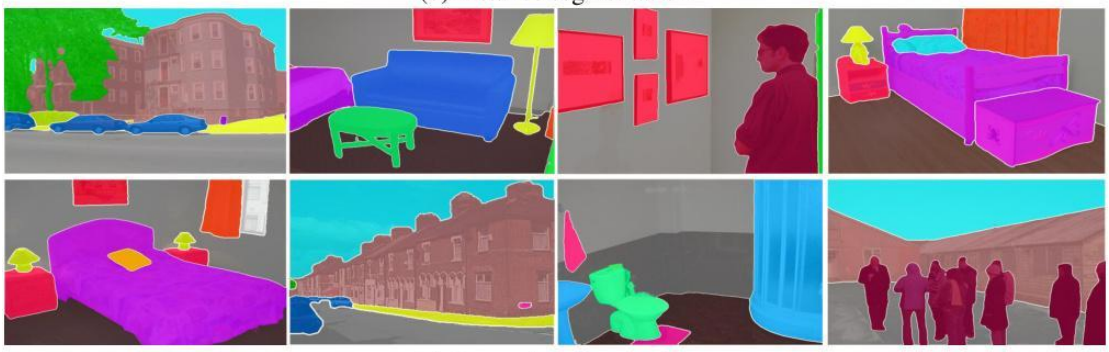
- **计算案例：**

i. 输入图片经 ResNet 提取特征，生成多尺度特征金字塔。
ii. 语义分支输出每个像素的类别概率（如“汽车”概率 0.9）。
iii. 实例分支输出每个物体的掩码和置信度。
iv. 最终结果：对每个像素选择置信度最高的类别和实例 ID

(a) Panoptic segmentation



(b) Instance segmentation



(c) Semantic segmentation

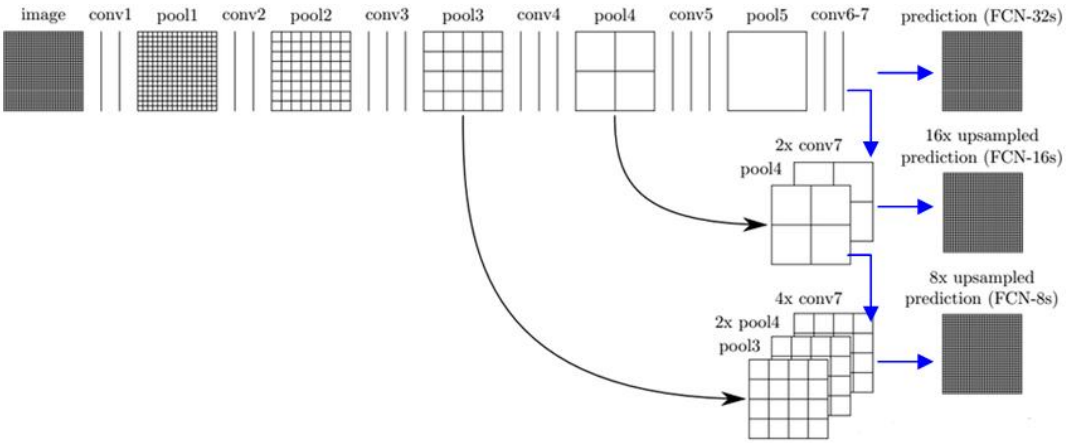
4.2 全卷积网络（FCN）

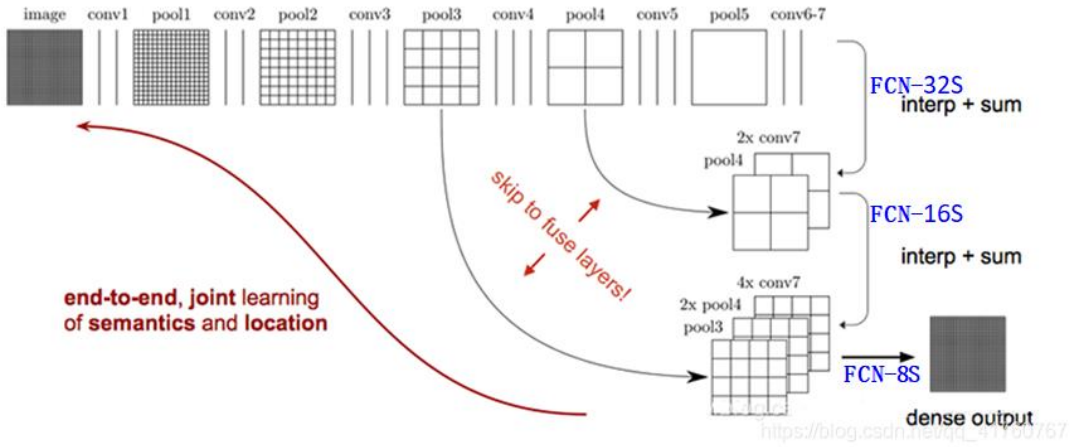
1. 什么是 FCN？

全卷积网络（FCNN）是一种专门用于图像分割的深度学习模型，它的核心思想是**用卷积层替代传统神经网络中的全连接层**，从而可以直接处理任意大小的输入图像，并输出与输入尺寸相同的像素级分类结果。简单来说，传统 CNN（如 VGG、ResNet）主要用于图像分类（判断图片是猫还是狗），而 FCN 可以精确到每个像素的类别（例如在自动驾驶中区分道路、行人、车辆）。

---

2. FCN 的核心原理





（1）全卷积化：抛弃全连接层

传统 CNN 的末尾会用全连接层将特征图“压扁”成一个固定长度的向量（例如 4096 维），再通过分类器输出结果。而 FCN 用**卷积层替代全连接层**，例如用 1x1 的卷积核覆盖整个特征图，这样网络可以接受任意大小的输入。
为什么这么做？

- 全连接层会丢失空间信息（比如物体的位置），而卷积层能保留特征图的位置关系。
- 输出不再是单一标签，而是每个像素的类别概率图（例如每个像素属于“猫”或“背景”）。

（2）转置卷积：恢复图像尺寸

卷积和池化会逐步缩小特征图的尺寸（例如从 256x256 缩小到 16x16），而分割任务需要输出与原图同样大小的结果。FCN 通过 **转置卷积（反卷积）** 将小特征图“放大”到原图尺寸。
举个计算例子：

- 假设经过池化后的特征图是 2x2，需要放大到 4x4。
- 转置卷积的核大小为 3x3，步长为 2，填充为 1。
- 计算过程类似反向的卷积操作，通过插值和参数学习恢复细节（具体公式见下文案例）。

（3）跳跃连接：融合深浅层特征

浅层网络（靠近输入）捕捉细节（如边缘），深层网络（靠近输出）捕捉语义（如物体类别）。FCN 通过**跳跃连接**将浅层和深层的特征图融合，提升分割精度。例如：

- 将第 3 层的特征图（尺寸大、细节多）与第 5 层的特征图（尺寸小、语义强）相加，再上采样输出。

---

3. 完整计算案例

假设输入是一张 **3x3 的灰度图**（数值简化），目标是为每个像素分类（例如 0 或 1）。

步骤 1：特征提取（卷积+池化）

- **输入：** 3x3 矩阵

```
[[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]
```

- **卷积层**：使用 1 个 2x2 卷积核，权重为[[1]][[0]][[0]][0]，步长 1，无填充。
  输出特征图：2x2

```
[[1*1 + 2*0 + 4*0 + 5*1 = 6,   2*1 + 3*0 + 5*0 + 6*1 = 8],

 [4*1 + 5*0 + 7*0 + 8*1 = 12, 5*1 + 6*0 + 8*0 + 9*1 = 14]]
```

- **池化层**：最大池化 2x2，步长 2。
  输出：1x1

```
[[14]]  # 取最大值
```

步骤 2：转置卷积（上采样）

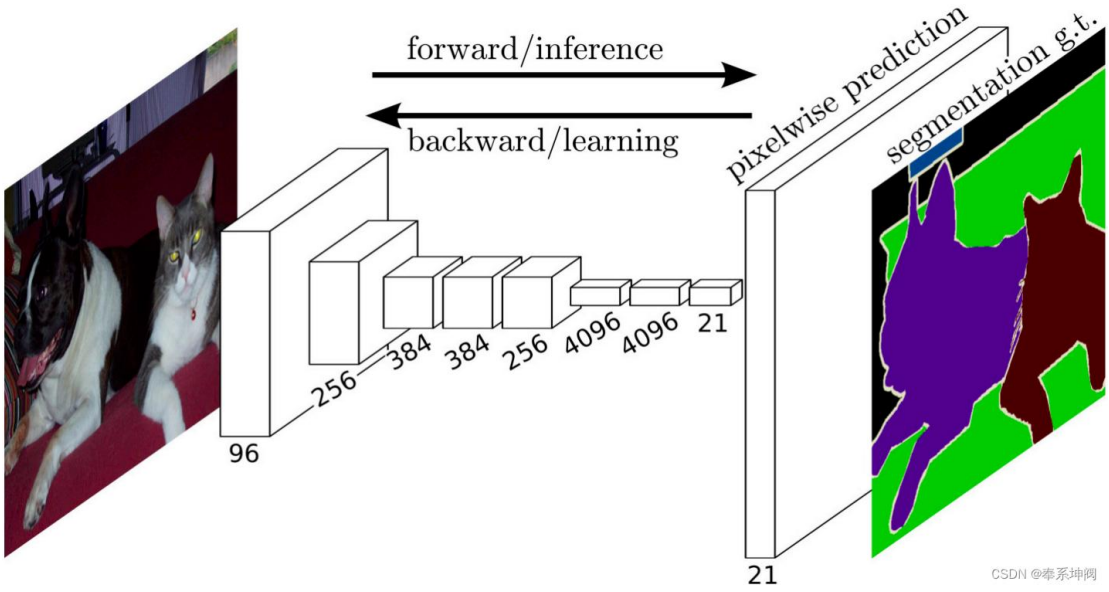
- **输入**：1x1 特征图 [[14]]
- **转置卷积核**：3x3，步长 2，填充 1。
  计算逻辑：将输入值 14“散布”到输出矩阵的对应位置，再与卷积核相乘。
  输出：3x3 矩阵

```
[[14*1, 14*0, 14*1],

 [14*0, 14*1, 14*0],

 [14*1, 14*0, 14*1]]

= [[14, 0, 14],

   [0, 14, 0],

   [14, 0, 14]]
```

- **最终输出**：每个像素的类别概率（例如通过 softmax 激活）。

---

4. 示意图



FCN 结构图：可参考中的示意图，输入图像经过卷积和池化缩小尺寸，再通过转置卷积和跳跃连接恢复细节。

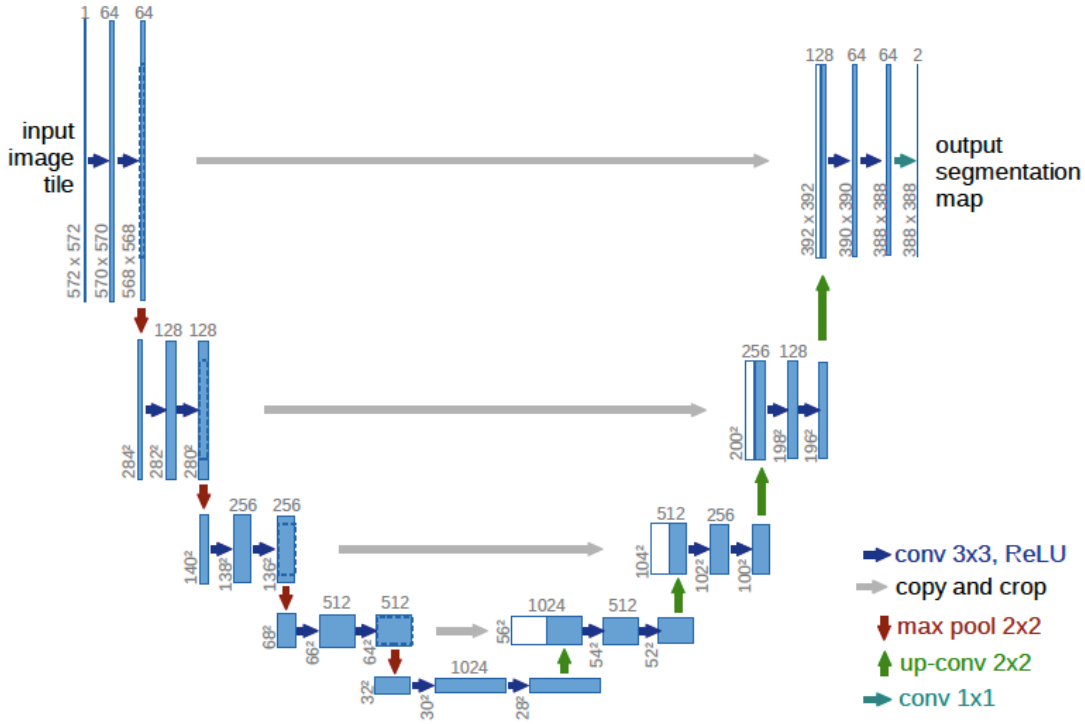
---

5. 总结

FCN 通过全卷积化、转置卷积和跳跃连接，实现了端到端的像素级分类。它的优势在于灵活处理任意尺寸图像，并保留空间信息，因此在自动驾驶、医学图像分割等领域广泛应用。对于中学生来说，只需记住：**FCN 像一支“智能画笔”，能精确描出图像中每个像素的类别**。

4.3 UNet ：像“U 型积木”一样处理图像

UNet 是一种专门用于图像分割的神经网络，可以将图片中的每个像素分类（比如区分细胞和背景）。它的结构像一个对称的英文字母 **U**，因此得名。下面用积木拼接的比喻来解释：



下采样（压缩信息）

就像用不同大小的积木块逐步压缩图片，提取特征。每一层通过 **卷积（提取特征）** 和 **池化（缩小尺寸）** 操作，逐步将图片变小，但保留重要信息。

示例计算：假设输入图片尺寸是 572×572×1（宽×高×通道），经过一次 3×3 卷积（无填充）后，尺寸变为 570×570×64。计算公式：

新尺寸=（原尺寸+2×填充−卷积核大小）/ 步长 +1

这里填充=0，步长=1，所以 570 = (572 + 0 - 3)/1 + 1。

上采样（恢复细节）

像反向拼接积木，逐步放大图片。通过 **反卷积**（或插值）操作扩大尺寸，同时结合下采样时保存的细节（跳跃连接），让分割更精确。

跳跃连接（关键！）

在下采样和上采样对应的层之间，用“通道拼接”传递细节（类似把压缩前的

积木块直接复制到放大后的位置）。例如，下采样到 28×28×512 的特征图会与上采样的 28×28×256 拼接，得到 28×28×(512+256) 的特征。

4.3.2 完整计算案例：从输入到输出的全过程

假设输入是一张 572×572 的灰度图（通道数=1），目标是对每个像素分类（如细胞/非细胞）。

下采样（左侧 U 型）

1. 第 1 层：3×3 卷积 → 570×570×64 → 2×2 最大池化 → 285×285×64
2. 第 2 层：同样操作 → 140×140×128 → 池化 → 70×70×128
3. 重复直到最底层：28×28×512。

上采样（右侧 U 型）

1. 最底层：反卷积将 28×28×512 放大到 56×56×256
2. 跳跃连接：与下采样的 56×56×256 拼接 → 56×56×(256+256)=512
3. 重复卷积和上采样，直到恢复原尺寸。

输出层

最后通过 1×1 卷积将通道数变为分类数（如 2 类），得到 388×388×2（原图边缘因无填充被裁剪）。

4.3.2 UNet 与 FCN 的区别：细节决定成败

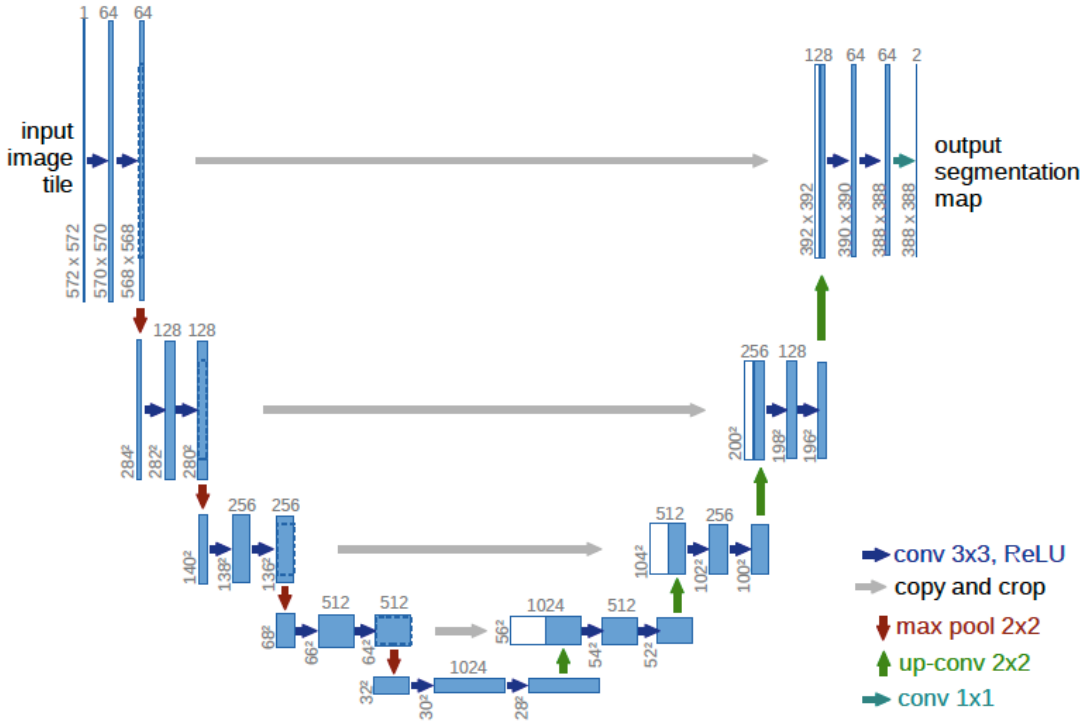
特性	UNet	FCN
结构对称性	完全对称的 U 型，上下采样层数相同	非对称，解码器较简单
跳跃连接方式	通道拼接（Concatenate）	数值相加（Add）
适用场景	医学图像（小数据、细节要求高）	通用分割（如街景、物体）

核心差异解释：

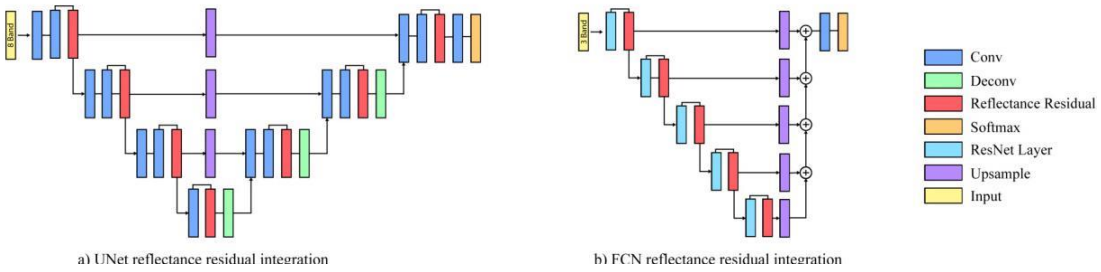
- **拼接 vs 相加**：UNet 的拼接保留了更多原始细节（如细胞边缘），而 FCN 的相加可能模糊细节。
- **对称结构**：UNet 的上采样每一步都结合对应下采层的细节，FCN 仅融合部分层。

4.3.3 图示

UNet 结构图：



UNet vs FCN



a) UNet reflectance residual integration

b) FCN reflectance residual integration

4.3.4 总结：为什么 UNet 适合医学图像？

UNet 的跳跃连接像“记忆面包”，在放大时直接使用压缩前的细节，避免信息丢失。这对需要精确分割细胞边界的医学图像至关重要。而 FCN 更注重全局语义，适合大场景的分割（如道路、天空）

4.3 图像分割的评价指标

图像分割的任务是让计算机把图片中的每个像素分类成不同的物体或区域（比如“猫”“背景”“天空”）。为了判断算法效果，科学家们设计了以下指标：

---

1. 像素精度（Pixel Accuracy, PA）

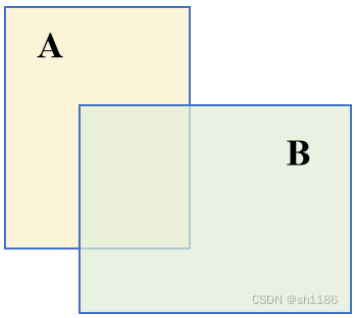
- **原理**：就像考试算正确率一样，统计所有像素中分类正确的比例。
- **公式**：

$$PA = \frac{\text{正确分类的像素数}}{\text{总像素数}}$$

- **例子**：
  假设一张图有 100 个像素，真实标签中“猫”占 20 像素，“背景”占 80 像素。
  模型预测中，正确分类了 18 个“猫”像素和 75 个“背景”像素。
  $PA = \frac{18+75}{100} = 93\%$

---

2. 交并比（IoU）



- **原理**：比较预测区域和真实区域的重叠程度，像两片拼图的重合度。
- **公式**：

$$IoU = \frac{\text{预测区域} \cap \text{真实区域}}{\text{预测区域} \cup \text{真实区域}}$$

- **例子：**
  真实“猫”区域有 30 像素，预测“猫”区域有 25 像素，其中两者重叠 20 像素。
  $IoU = \frac{20}{30+25-20} = \frac{20}{35} \approx 57.1\%$

---

3. 平均交并比（mIoU）

- **原理：**对每个类别的 IoU 取平均值，更全面地评估整体效果。
- **例子：**类别 1（猫）的 IoU 是 50%，类别 2（背景）的 IoU 是 90%。

$$mIoU = \frac{50\%+90\%}{2} = 70\%$$

---

4. Dice 系数

- **原理：**与 IoU 类似，但公式不同，常用于医学图像分割。
- **公式：**

$$Dice = \frac{2 \times \text{预测区域} \cap \text{真实区域}}{\text{预测区域} + \text{真实区域}}$$

- **例子：**
  沿用 IoU 的例子，

$$Dice = \frac{2\times20}{25+30} = \frac{40}{55} \approx 72.7\%$$

4.4 图像分割数据集

数据集是用于训练和测试模型的图片集合，每张图都有精确的像素级标签。

---

1. 常用数据集

1. **PASCAL VOC**：包含 20 类物体（如人、车、猫），用于语义分割 。
2. **COCO**：80 类物体，支持实例分割（区分同一类物体的不同个体）。
3. **Cityscapes**：街景图片，标注了道路、车辆、行人等，用于自动驾驶 。
4. **医学数据集（如 BraTS）** ：脑肿瘤分割，使用 Dice 系数评估 。

---

2. 数据集的作用

- **训练模型**：像“题库”一样让 AI 学习不同物体的特征。
- **测试模型**：用未见过的新图片检验模型的真实能力。

4.4.1 完整计算案例

假设一张图分为“猫”和“背景”两类，真实标签和预测结果如下：

真实标签	预测结果
猫（10 像素）	预测为猫：8 像素，预测为背景：2 像素
背景（90 像素）	预测为背景：88 像素，预测为猫：2 像素

计算 IoU：

- 猫的IoU：交集8，并集 $10+10-8=12 \rightarrow 8/12 \approx 66.7\%$。
- 背景的IoU：交集88，并集 $90+90-88=92 \rightarrow 88/92 \approx 95.6\%$。
- mIoU = $(66.7\% + 95.6\%)/2 \approx 81.2\%$ [8]。

计算 PA：

- 正确像素数 = 8（猫） + 88（背景） = 96。
- PA = $96/100 = 96\%$ [6]。

5 姿态估计案例

待更新：以仰卧起坐计数为例，重点讲清楚人体姿态估计的原理

6 人群智能分析案例

要求：
（1）实现监控相机下行人检测、行人跟踪、行人属性等基本功能。
（2）输入一个景区、店铺等场景的监控视频，统计一段时间内一共出现了多少人，男女老幼各多少人？
（3）行人属性要自己设计神经网络，考虑如何提高识别的准确率。
（4）能否对监控相机的数据进行实时获取和分析。
注：题目来源于 2023 年中国大学生服务外包竞赛赛题：景区室内外游览任务调度与群组推荐平台。本案例是这个题目的部分功能。

实现思路：
（1）行人检测算法
可采用 yolov5，22 智能在第 1 学期《人工智能创新实践》课实践过。
（2）行人属性分析
我们第 1 学期都实践过是否戴口罩识别。行人性别识别，跟是否戴口罩识别，是类似的。
（3）行人跟踪
可以采用算法 deepsort
https://github.com/mikel-brostrom/Yolov5_DeepSort_Pytorch
下载代码运行。
运行步骤可以自己摸索， 或参考《目标检测与跟踪指导书》

实验步骤：
杨力老师录制了 4 个视频，包含详细操作步骤。

7 时间序列分析

时间序列预测问题在我们生活中非常常见，例如根据过去一周的销量数据，预测接下来几天的销量；根据过去一段时间的股票市场，预测未来一段时间股票会达到多少点；根据过去一周的天气，预测未来三天的天气。这些问题都是时间序列预测问题。

简单来说，就是基于某一类变量过去按时间顺序排列的数据，也就是历史数据，来推测变量在未来某一时间或某一段时间里的取值或变化趋势。

时间序列预测通常更适合做短期预测，因为预测的时间越长，预测值的准确性往往会越低。例如预测明天的天气，比预测未来一个月的天气，会准得多。

7.1 基于 LSTM 的产品销量预测

LSTM（长短期记忆网络）是一种特殊的循环神经网络（RNN），能够学习长期依赖信息，是一种经典的时间序列预测算法。这里用一个简单的案例，即根据某产品过去 7 天销量预测未来 3 天销量，来说明 LSTM 的基本原理。

假设原始数据是长度为 20 的销量序列，代表了产品在某 20 天时间内的每天销量。

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

我们的任务是根据过去 7 天销量预测未来 3 天销量，因此输入输出数据定义如下。

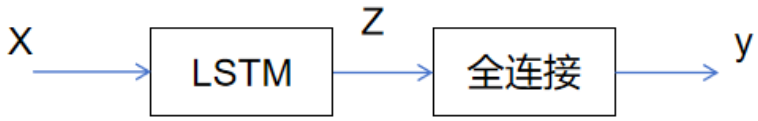
输入样本 X: [1,2,3,4,5,6,7] → 输出 y: [8,9,10]

下一个样本 X: [2,3,4,5,6,7,8] → y: [9,10,11]

以此类推，以滑动窗口生成训练数据。

请读者算一下，一共有多少组训练样本？答案是 11 组训练样本。

整个计算过程包括了两个步骤，首先输入 X 经过 LSTM 模块，得到中间结果 Z，再通过全连接，得到最后的输出 y。



输入数据 X

形状: (batch_size, sequence_length, input_size)

batch_size: 一次计算所用的样本数量，例如 10 个样本

sequence_length: 时间步长，这里是 7 天

input_size: 每个时刻数据的特征数，这里只有销量，所以是 1

输出数据 y

形状: (batch_size, output_size)

batch_size: 与上面相同，为 10

output_size: 预测的时间步长，如未来 3 天，所以为 3

中间结果 Z

形状：（batch_size, sequence_length, hidden_size）

batch_size: 与上面相同，为 10

sequence_length: 与上面相同，为 7

hidden_size: LSTM 的隐藏层节点数，例如 50。

在 LSTM 模型中，hidden_size 是一个重要的超参数，表示 LSTM 单元隐藏层的神经元数量。它决定了 LSTM 网络的记忆容量和特征提取能力。它和我们前面讲到的神经网络中的隐藏层的数量是类似的概念。

还需要说明的是，全连接层的输入维度是（batch_size, hidden_size），输出维度是（batch_size, output_size）。全连接层是用了 LSTM 层的最后一个时刻的输出作为它的输入。

7.2 LSTM 算法：以光伏电力负荷预测为例

光伏电力负荷预测，简单来说就是通过天气数据（比如温度、风速、云量等），预测未来一段时间太阳能发电站能发多少电。就像根据今天的天气，预测明天太阳能热水器里的水有多热。

为什么要做这个预测？简单说其中一个原因。太阳能发电靠天吃饭，但电网必须保证供给，如果提前知道明天太阳能发电少，电网可以提前通过火电厂补缺口。

我们的案例的任务目标是根据一段历史时间的 GHI（太阳辐照度）预测未来的 GHI。采用 LSTM 算法来实现。

案例的数据如下图所示。它表示的是时间间隔为 1 分钟的辐照度和天气数据记录。

timeStamp	ghi	air_temp	windsp	winddir
2014/1/2 15:22	3.08	2.9	1.76	187.8
2014/1/2 15:23	3.08	2.9	1.9	197.8
2014/1/2 15:24	3.18	2.9	1.82	199.4
2014/1/2 15:25	3.8	2.9	2.22	197.2
2014/1/2 15:26	4	2.9	2.3	212.3
2014/1/2 15:27	4.62	2.86	2.14	248.7
2014/1/2 15:28	4.87	2.8	2	199.6
2014/1/2 15:29	5.59	2.7	1.74	190.4
2014/1/2 15:30	1.23	2.7	1.78	193.6
2014/1/2 15:31	1.62	2.7	1.72	192.2
2014/1/2 15:32	2.04	2.62	1.66	188.2
2014/1/2 15:33	2.52	2.6	1.7	195.5
2014/1/2 15:34	3.17	2.56	1.7	183.1
2014/1/2 15:35	3.9	2.5	1.66	184
2014/1/2 15:36	4.64	2.44	1.56	179.6
2014/1/2 15:37	5.36	2.4	1.8	196.7

数据包含几个字段：

timeStamp：时间

ghi：总辐射

air_temp 温度

windsp 风速

winddir 风向

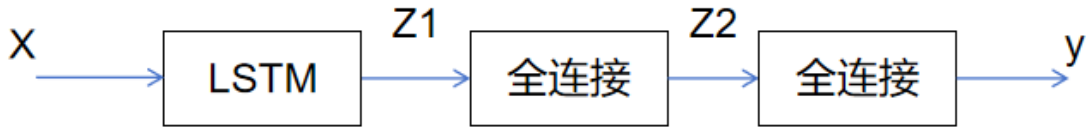
这个案例的代码在 AI_course 代码包的 data_mining/solar 目录下。它是用百度飞桨框架写的，程序结构和 PyTorch 类似。

想象你经营一个太阳能光伏电站，你希望对太阳辐照度进行预测。任务描述如下。

输入：过去 60 分钟的气象数据（温度、风速、风向、历史辐照度）

输出：预测下一分钟的太阳辐照度（GHI）

你的 LSTM 网络分为 3 层，也就是 1 个 LSTM 层和 2 个全连接层三个模块依次计算，如下图所示。



第 1 层，LSTM 层

输入形状：[批量大小, 60 分钟, 4 个特征]

输出形状：[批量大小, 60 分钟, 16]

16 表示隐藏节点数量，可以通俗地理解为用 16 种方式总结天气规律。

第 2 层，全连接层

输入形状：批量大小*16*60,　输出形状：批量大小*120

第 3 层，全连接层

输入形状：批量大小*120,　输出形状：批量大小*1

这里 1 表示输出只有太阳辐照度这 1 个值。

在第 2 层到第 3 层之间，还有激活函数，比如 ReLU，这里略去没有讲。

9 Transformer 案例

想象你是一名影评编辑，需要快速判断观众对某部电影的评价："画面惊艳，但剧情无聊透顶——适合看特效，别期待深度。" 人类能瞬间抓住核心：特效出色，剧情糟糕，整体评价中性偏负面。

但机器如何做到这一点？Transformer 模型正是通过一种"动态划重点"的机制，像人类一样理解文本的深层含义。本章将通过文本分类案例的计算，带你一步步拆解 Transformer 的"思考过程"，并揭示其如何从"随机猜测"进化到"精准判断"。

章节学习目标

8.1 拆解 Transformer"黑箱"：

用手动计算揭开自注意力、位置编码等关键模块的数学本质；

理解模型如何通过"词与词的关联强度"提取语义（例如：计算"震撼"与"特效"的注意力权重）。

8.2 掌握 Transformer 框架核心知识点：

自注意力机制（Self-Attention）；

位置编码（Positional Encoding）的设计原理；

8.3 透视 Transformer 的"自我进化"机制：

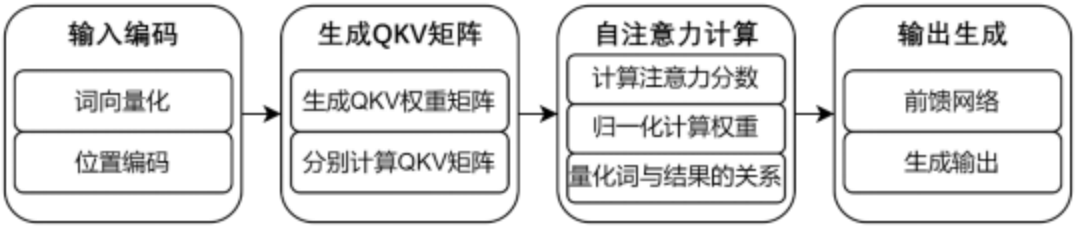
从损失计算、反向传播到参数更新，完整揭示模型如何通过误差反馈调整内部逻辑

8.1 拆解 Transformer "黑箱"

导读：Transformer 模型如同一台神秘的魔术机，总能够通过精准判断后生成合适的语句，究竟模型的背后影藏了什么样的计算流程，这一节我们将亲手拆开"黑箱"，计算每个步骤，理解模型的理解过程。

模型架构总览：

（图片 8.1.1）



完整计算流程剖析（详细解析见第二节）

1.输入编码：

（图片 8.1.2）

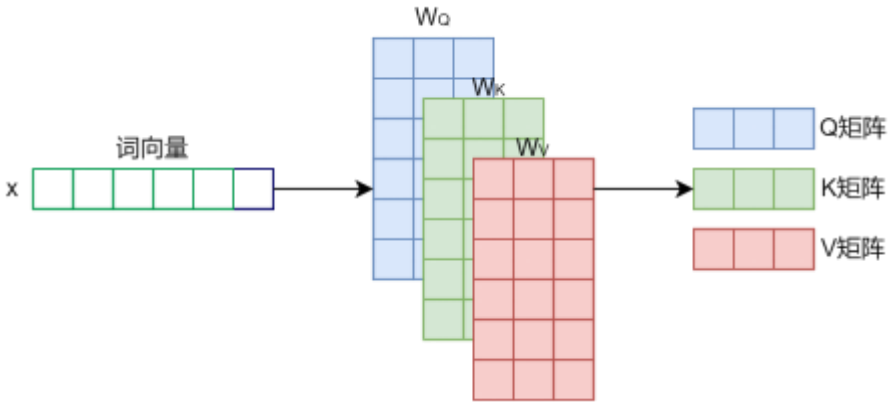
词向量：$\boldsymbol{e}_{词}$　　位置编码：$\boldsymbol{P}_{位置}$

输入向量：$\boldsymbol{x} = \boldsymbol{e} + \boldsymbol{p}$

通过这样的编码，就会给予每个单词一个单独的身份令牌，随着模型的学习，模型给予不同单词的身份令牌也会越来越准确。

2.生成 QKV 矩阵：

（图片 8.1.3）



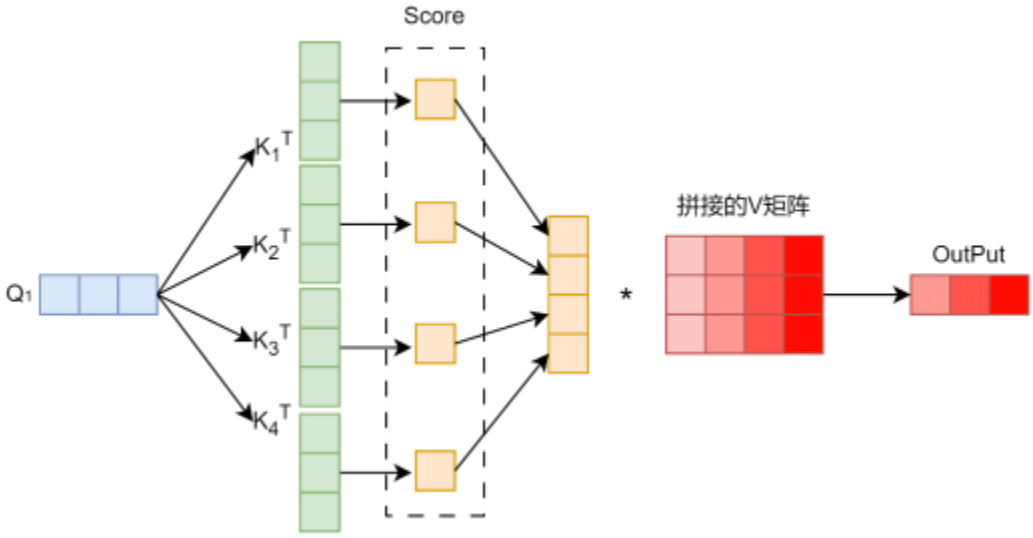
权重矩阵：$\boldsymbol{W_Q}, \boldsymbol{W_K}, \boldsymbol{W_V}$

生成 Q/K/V 向量：$\mathrm{Q} = \boldsymbol{x} \cdot \boldsymbol{W_Q}, \quad K = \boldsymbol{x} \cdot \boldsymbol{W_K}, \quad V = \boldsymbol{x} \cdot \boldsymbol{W_V}$

通过词令牌和权重矩阵相乘计算并拼接，得到完整语句的矩阵，未后续计算做准备（本节暂时不做详细的介绍，会在第二节详细讲解 QKV 矩阵的区别和作用）。

3.自注意力计算：

（图片 8.1.4）



注意力分数：$\text{Score}_{ij} = Q_i \cdot K_j$

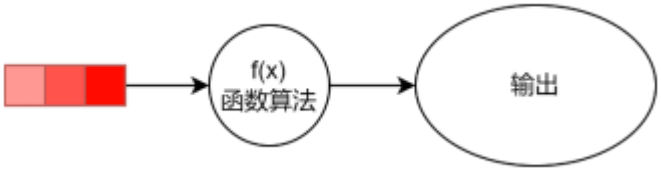
Softmax 归一化：$\text{权重}ij = \frac{e^{\text{Score}ij}}{\sum_k e^{\text{Score}_{ik}}}$

加权计算：$\boldsymbol{z_i} = \sum_j \boldsymbol{权重_{ij}} \cdot \boldsymbol{V_j}$

利用 QKV 矩阵完整计算得到不同词语的权重数值，不同词语在语句中的互相关系引起了重要程度的变化，一个句子也因此有了重点。

4.输出生成：

（图片 8.1.5）



前馈网络（以线性分类模型为例）：$h = f(zW_1 + b_1)W_2 + b_2$

利用前馈网络将抽象的概率分布转化为具体输出：在分类任务中，计算每个类别的比重；在生成任务中，预测最可能的文本内容。本章以分类任务为例。

分类概率：$\text{概率} = \sigma(\boldsymbol{h} \cdot \boldsymbol{w}_{\text{class}} + b_{\text{class}})$

案例任务：判断句子“画面 惊艳 特效”是否为好评（标签：1=好评，0=差评）

本章节我们将从同一个 Transformer 模型的两种形态——**未训练**和**经过训练**来解释为什么模型能够理解语言，因为在了解任何复杂系统之前，我们常常需要先观察它的“初始状态”。

就像拼一幅巨型拼图时，我们会先摊开所有碎片观察它们的原始形状和颜色，再逐步找到拼接规律。Transformer 模型也是如此：

未经训练的模型：所有参数（如词向量、注意力权重）均为**随机初始值**，此

时模型如同“新生儿”，对语言毫无理解能力，所有词语对于模型来说，没有区别；

训练后的模型：通过模型不断的试错，模型逐渐从大量数据中**提取规律**。

通过对比这两种状态，我们能更清晰地看到模型在训练前后发生了什么样的变化。

本节我们将**并行**对比未训练、已训练模型的计算流程，来揭示这个黑箱究竟暗藏什么能力，其关键部位在哪里。

步骤 1：输入编码——词向量与位置编码

未训练模型（初始状态）

词向量初始化编码：

词语	词向量（由于随机生成所以可以认为相同）
画面	[0.1, 0.1]
惊艳	[0.1, 0.1]
特效	[0.1, 0.1]

位置编码叠加（固定公式）：

词语	输入向量
画面	[0.1+0.8, 0.1+0.001]= [0.9, 0.101]
惊艳	[0.1+0.9, 0.1+0.002]= [1.0, 0.102]
特效	[0.1+0.1, 0.1+0.003]= [0.2, 0.103]

已训练模型（优化后）

词向量初始化编码：

词语	词向量
画面	[0.8, -0.2]
惊艳	[1.2, -0.5]
特效	[0.9, -0.3]

位置编码叠加（固定公式）：

词语	输入向量
画面	[0.8+0.8, -0.2+0.001] = [1.6, -0.199]
惊艳	[1.2+0.9, -0.5+0.002] = [2.1, -0.498]
特效	[0.9+0.1, -0.3+0.003] = [1.0, -0.297]

对比模型最后生成的词向量：

词语	训练前	训练后
画面	[0.9, 0.101]	[1.6, -0.199]
惊艳	[1.0, 0.102]	[2.1, -0.498]
特效	[0.2, 0.103]	[1.0, -0.297]

未训练：输入向量仅因位置不同而差异微小。

已训练：词向量已经携带语义信息

步骤 2：自注意力计算——注意力权重的差异

目标：计算每个词对其他词的关注程度。

核心公式：

查询（Q） = 输入向量 × 权重矩阵 W_Q

键（K） = 输入向量 × 权重矩阵 W_K

值（V） = 输入向量 × 权重矩阵 W_V

未训练模型（随机权重矩阵）

生成 Q/K/V 向量：

根据前面计算的输入向量：画面 =[0.9, 0.101]

未训练的随机权重矩阵（完全随机生成，下为示例）：

$$W_Q = \begin{bmatrix} 0.2 & -0.3 \\ 0.4 & 0.5 \end{bmatrix}, \quad W_K = \begin{bmatrix} -0.1 & 0.6 \\ 0.2 & -0.2 \end{bmatrix}, \quad W_V = \begin{bmatrix} 0.3 & 0.3 \\ -0.2 & 0.1 \end{bmatrix}$$

例如，根据**矩阵计算公式**，计算“画面”的 Q 向量(其他三个词计算过程也相同)：

$$Q_{画面} = [0.9 \times 0.2 + 0.101 \times 0.4,\ 0.9 \times (-0.3) + 0.101 \times 0.5] \approx [0.2204, -0.2195]$$

同理计算 K 和 V 向量：

$$K_{画面} = [0.9 \times (-0.1) + 0.101 \times 0.2,\ 0.9 \times 0.6 + 0.101 \times (-0.2)] \approx [-0.0698, 0.5198]$$

“惊艳”和“特效”的 QKV 计算也同理，读者可以自行计算，这里略去过程，仅保留需要的一下计算需要使用的三个词的 K 值计算结果：

$K_{画面}$	$[-0.0698, 0.5198]$
$K_{惊艳}$	$[-0.0796, 0.5796]$
$K_{特效}$	$[0.0006, 0.0994]$

注意力分数计算：

词对	分数计算（Q·K）	原始分数
画面 → 画面	0.2204×(-0.0698) + (-0.2195)×0.5198	-0.130
画面 → 惊艳	0.2204×(-0.0796) + (-0.2195)×0.5796	-0.145
画面 → 特效	0.2204×0.12 + (-0.2195)×0.08	0.009

为了把权重缩放成比值，进行 Softmax 归一化：

词对	原始分数	指数化（e^分数）	Softmax 权重
画面 → 画面	-0.130	$e^{-0.130} \approx 0.8786$	0.8786/2.7527 ≈0.3192
画面 → 惊艳	-0.145	$e^{-0.145} \approx 0.8652$	0.8652/2.7527 ≈0.3143

画面 → 特效	0.009	$e^{0.009} \approx 1.0089$	1.0089/2.7527 ≈0.3665

可以看出，未训练模型关注分散，词之间关系的大小仅仅因位置不同而不同，并且大小接近 **0.33**，即平均概率。

加权和计算（权重×词向量）：

$$\begin{cases} 0.3192 \times 0.2498 + 0.3143 \times 0.2796 + 0.3665 \times 0.0394 \approx 0.18 \\ 0.3192 \times 0.2801 + 0.3143 \times 0.3102 + 0.3665 \times 0.0703 \approx 0.21 \end{cases}$$

最终向量：[0.18, 0.21]

已训练模型（优化权重矩阵）

因为大致的计算流程与上述相同，而真实的模型参数量巨大，所以以下只是一个模拟的结果：

生成 Q/K/V 向量：

Q 向量：[1.38, -0.23]

K 向量：[-0.63, 1.63]

V 向量：[1.5, -0.4]

注意力分数与权重：

词对	原始分数	Softmax 权重
画面 → 画面	0.8	0.20
画面 → 惊艳	2.1	0.65
画面 → 特效	0.6	0.15

加权和计算

$$\begin{cases} 0.20 \times 0.8 + 0.65 \times 1.5 + 0.15 \times 0.6 \approx 1.23 \\ 0.20 \times (-0.2) + 0.65 \times (-0.4) + 0.15 \times (-0.3) \approx -0.35 \end{cases}$$

最终向量：[1.23, -0.35]

并行对比表格

计算步骤	未训练模型（随机权重）	已训练模型（优化后的权重）
注意力权重	画面→惊艳：47%	画面→惊艳：65%
加权和结果	[0.06, 0.19]（无明确语义）	[1.23, -0.35]（正面情感概率更高）

通过对比表格中的关键数据，我们可以直观理解 Transformer 的进化过程：

注意力权重的变化：

未训练模型（47%）：模型对“画面→惊艳”的关注度不到一半，其他注意力分散在“画面”和“特效”上，相当于“每句话都随便划几条线”，没有重点。

已训练模型（65%）：模型像“老师批改作文”，用红笔圈出关键词“惊艳”，并弱化其他词的影响。

加权和结果的差异：

未训练模型（[0.06, 0.19]）：数字接近零，像一张白纸——模型根本不知道这句话在说什么。

已训练模型（[1.23, -0.35]）：第一个数字远大于零（代表好评信号），第二个数字为负（抑制差评信号），模型明确判断这是好评。

章节小结：拆解 Transformer 的“学习密码”

Transformer 并非天生“聪明”，它的能力完全来自训练过程中的参数调整。

Transformer 的核心机制：**动态**划重点

自注意力是模型的“荧光笔”：通过调整权重，学会在句子中划出重要词语。

参数优化是模型的“老师”：反向传播不断纠正错误，让词向量和注意力权重逐渐贴合真实语义。

8.2 掌握 Transformer 核心知识点——注意力机制

导读：

想象你正在读一本小说，眼睛会不自觉地聚焦在关键情节上，同时记住角色的出场顺序——Transformer 的注意力机制正是模仿了这种能力。

本章将通过**生活类比**与**矩阵可视化**，带你理解 Transformer 如何从零开始“学会阅读”。

从文字到数字：分词的奥秘

核心问题：计算机如何理解“画面惊艳特效”这句话？

Transformer 提供的方案：分词——把句子拆成“乐高积木”（图 8.2.1）

人类思维：我们自然地将句子拆解为词语，如“画面 / 惊艳 / 特效”。

模型实现：

使用分词器算法，像剪刀一样裁剪句子。例如：

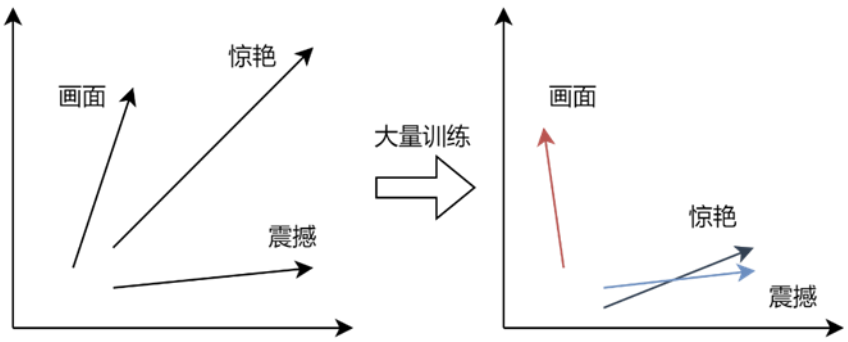
```
#输入为
text = "画面惊艳特效"
# 输出则为
tokens = ["画面", "惊艳", "特效"]
```

有时还会采取子分词，来解决生僻词问题。例如“ChatGPT” → ["Chat", "G", "PT"]。

词向量：词语的“数字身份证”

核心问题：计算机如何知道“惊艳”和“震撼”是近义词？

Transformer 提供的方案：词向量化——词语的数学分身



人类思维：我们理解“惊艳”和“震撼”都代表强烈的正面情感。

模型实现：

词向量表：每个词对应一个数字向量（如 512 维），通过训练学习语义。

"惊艳"的词向量

惊艳 = [0.7, -0.3, 1.2, ..., 0.4]

"震撼"的词向量

震撼 = [0.6, -0.2, 1.1, ..., 0.5]

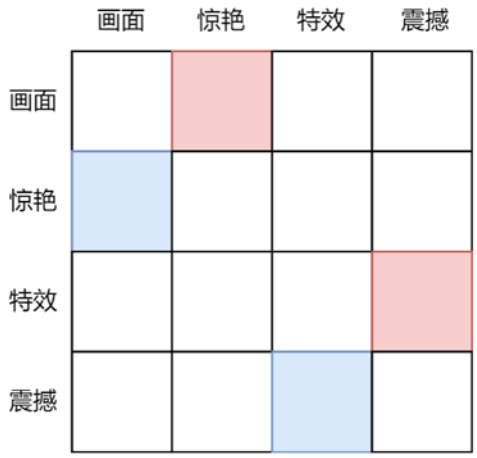
语义相似度：向量夹角越小，词义越接近。

在模型训练初期，词向量通常是随机初始化的。通过充分训练，模型能够逐渐调整词向量，使得语义相似的词语在向量空间中距离更近，从而表现出更强的语义关联

自注意力机制：模型的“思维导图”

核心问题：模型如何知道“特效”一词在“画面惊艳特效”中重要性较低？

Transformer 提供的方案：自注意力机制——全文通读理解词语含义



1. Q/K/V：角色扮演游戏

人类思维：

阅读时，大脑会同时思考“当前重点”（Query）、“已读内容”（Key）和“记忆信息”（Value）。

模型实现：

Q（Query）：当前词的“提问”，如“哪些词与我相关？”

K（Key）：其他词的“回答”，如“我包含情感信息！”

V（Value）：词语的真实“信息”，如“惊艳”的情感强度为+1.2。

2. 注意力分数：词语的“关联热度”

人类思维：

读到“惊艳”时，你会联想到“画面”，而非无关词“特效”。

模型实现：

$Q \cdot K^T$ 计算：向量点积衡量词间相关性，值越大关联越强。

缩放与 Softmax：将分数转化为概率，突出关键联系。

3. 信息聚合：提炼句子精华

人类思维：

读完整个句子后，你记住的是“画面惊艳”，而非每个字。

模型实现：

加权和：根据注意力权重，融合 V 向量生成新表示。

位置编码：词语的“时空坐标”

核心问题：如何让模型知道“狗咬人”和“人咬狗”的区别？

Transformer 提供的方案：位置编码——词语的“座位表”

人类思维：

我们通过词序理解“狗→咬→人”的动作方向。

模型实现：

正弦函数编码：为每个位置生成唯一向量

与词向量相加：将位置信息注入语义表示。

核心启示：

Transformer 的智慧不在于复杂的计算，而在于用矩阵构建了一张“词与词的社交网络”——每个词通过 Q/K/V 提问、回答、传递信息，最终让模型像人类一样理解语言。

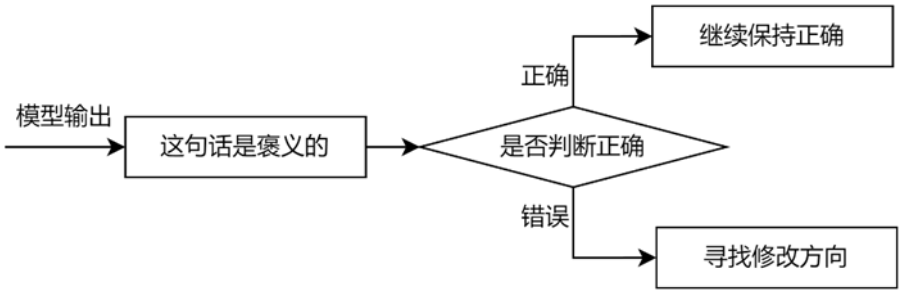
8.3 透视 Transformer 的“自我进化”机制

导读：

训练 Transformer 就像教一个新手画家作画——初始时他胡乱涂抹（随机参数），但每画完一幅作品，你都会指出哪里画得不好（计算损失），于是他默默调整手法（参数更新）。本章将揭示 Transformer 如何通过反向传播完成这场“自我进化”。

损失函数：模型的“成绩单”

核心问题：如何告诉模型它是否理解对了句子？



损失函数的作用

人类类比：老师批改试卷，用分数衡量学生表现。

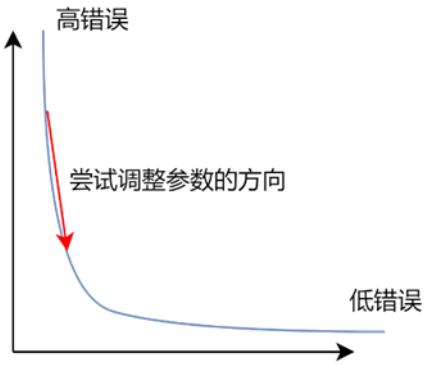
模型实现：

分类任务（如句子分类）：交叉熵损失，直接衡量预测概率与真实标签的差距；

生成任务（如翻译）：困惑度（Perplexity），结合大量真实语句，衡量生成句子的流畅度、真实度。

反向传播：误差的“溯因之旅”

核心问题：模型如何知道该调整哪些参数以减少损失？



梯度：参数的“调整指南针”

人类类比：画家发现天空画得太暗，于是决定“下次多用亮蓝色”。

模型实现：

梯度计算：通过一步一步回溯，计算损失对每个参数的调整方向。

参数更新：模型的“微调手术”

核心问题：如何根据梯度调整参数？

优化器：参数的“智能教练”

利用算法，合理根据参数的调整方向对模型参数进行微调，避免“一刀切”导致的震荡或停滞。

章节总结

损失函数：模型的“错误指示器”；

反向传播：沿计算图逆向传递误差，定位责任参数；

参数更新：优化器根据梯度调整参数，逐步逼近正确答案。

学习启示：

Transformer 的“聪明”并非天生，而是通过动态划重点的 自注意力机制 与参数优化 逐步习得。本章通过拆解“输入编码→生成 Q/K/V→自注意力计算→输出生成”四步流程，揭示模型从“随机混乱”到“精准理解”的进化本质：未训练时，模型如同无头苍蝇般胡乱关联词语；训练后，注意力权重聚焦关键信息（如“惊艳”权重 65%），词向量携带语义特征，最终输出明确表达语义逻辑。理解这一过程，即可穿透黑箱迷雾，掌握 Transformer 如何用数学构建语言的智慧。

10 大模型基础案例

10.1 概述：大模型是什么

大模型，顾名思义，就是规模庞大的人工智能模型。在人工智能领域，模型可以被理解为一种复杂的数学公式或算法集合，用于处理和分析数据，进而做出预测或决策。大模型通常指的是参数数量巨大（通常在数十亿到数万亿不等）的深度学习模型。

10.2 大模型工作原理

大模型的核心是**神经网络**。神经网络是一种模仿人脑结构和工作方式的计算模型。想象一下，如果你的大脑是由无数的神经元通过复杂的连接方式组成的，那么人工神经网络就是用数学方程来模拟这些神经元和连接。
在神经网络中，信息通过多个**层**进行传递和处理。每一层都由多个**节点**（相当于神经元）组成。信息从输入层开始，经过多个隐藏层的处理，最后到达输出层产生结果。

数学原理简介

让我们来用数学知识来理解大模型内部运作的基本原理：
1.线性代数：大模型大量使用矩阵运算。例如，一层神经网络的基本操作可以表示为：

$$y = Wx + b$$

（x 是输入向量，W 是权重矩阵，b 是偏置向量，y 是输出向量）
2.激活函数：为了引入非线性，每个节点的输出通常会通过一个激活函数。常见的激活函数包括 sigmoid 函数、tanh 函数和 ReLU 函数。
3.反向传播：这是训练神经网络的关键算法。它通过计算预测值与实际值之间的误差，然后从输出层向输入层逐层调整权重，以减小误差。

10.3 从案例中了解

10.3.1 GPT（生成预训练转换器）

GPT 系列模型是目前最著名的大模型之一。它主要用于自然语言处理任务，如文本生成、问答系统等。

它的主要工作原理包含有：1.预训练：GPT 首先在大量文本数据上进行预训练，学习语言的基本规律和知识。2.自注意力机制：这允许模型在处理一个词时考虑上下文中的其他词，从而理解语境。3.微调：预训练后，模型可以针对特定任务进行微调，如翻译、摘要生成等。

举例：

假设我们要让 GPT 模型续写一个故事的开头：

"从前，在一个遥远的王国里..."

GPT 模型会基于它学到的模式，生成合理的后续内容，比如：

"... 住着一位美丽的公主。她有着金色的长发和蓝宝石般的眼睛。然而，公主并不快乐，因为她被一个邪恶的巫师囚禁在高塔之中..."

10.3.2 BERT（基于转换器的双向编码器表示）

BERT 是一个广泛使用的大模型，主要用于理解和分析文本。

它的主要工作原理包含有双向学习：1.与 GPT 不同，BERT 可以同时考虑一个词前后的上下文，这使得它在理解语言方面更加强大。2.掩码语言模型：BERT 在训练时会随机遮蔽一些词，然后尝试预测这些被遮蔽的词，这有助于它学习词与词之间的关系。

举例：

假设我们有这样一个句子需要 BERT 模型来理解：

"小明 [MASK] 了一本有趣的 [MASK]。"

BERT 模型会尝试填补这些空缺，可能的结果是：

"小明读了一本有趣的书。"

展示了 BERT 理解句子结构和语境的能力。

10.3.3 预测下一个词语

假设我们有一个非常简单的语言模型，给定一个句子：“今天天气很”，模型需要预测下一个词。

模型输入：输入句子：“今天天气很”

模型输出：预测下一个词的概率分布，例如：“好”、“坏”、“热”等。

我们使用一个基于 Transformer 的语言模型（如 GPT），模型的核心是自注意力机制和前馈神经网络。模型的输入是词嵌入（word embeddings），输出是每个可能词的概率分布。

计算步骤：

假设我们有一个非常简化的模型，词汇表只有 3 个词：“好”、“坏”、“热”。模型的输出层会生成一个 3 维的概率分布。

1. **输入嵌入**：输入句子“今天天气很”被转换为词嵌入，假设每个词嵌入是 2 维向量：

   今天：[0.1, 0.2]

   天气：[0.3, 0.4]

   很：[0.5, 0.6]

2. **自注意力机制**：模型计算每个词与其他词的关系，生成新的表示。假设经过自注意力机制后，每个词的表示变为：

   今天：[0.2, 0.3]

   天气：[0.4, 0.5]

   很：[0.6, 0.7]

3. **前馈神经网络**：经过前馈神经网络后，每个词的表示变为：

   今天：[0.3, 0.4]

   天气：[0.5, 0.6]

   很：[0.7, 0.8]

4. **输出层**：模型输出一个 3 维的概率分布，假设为：

   好：0.6

   坏：0.3

   热：0.1

5. **预测**：模型选择概率最高的词“好”作为预测结果。

6. **最终输出**：模型预测下一个词是“好”，因此完整的句子是：“今天天气很好”。

10.4 大模型现状和发展

10.4.1 大模型的重要性

1. 突破性的性能

大模型在多个领域都展现出了超越传统方法的性能。例如，在自然语言处理、图像识别、语音识别等任务中，大模型常常能够达到或超越人类水平的表现。

2. 迁移学习能力

大模型的一个重要特点是它们具有强大的**迁移学习能力**。这意味着在一个大规模数据集上预训练的模型，可以相对容易地适应到其他相关任务上，即使这些新任务的训练数据相对较少。

3. 多模态学习

最新的大模型正在打破不同数据类型之间的界限。例如，有些模型可以同时处理文本、图像和音频数据，这为创建更加智能和通用的 AI 系统开辟了道路。

4. 推动技术创新

大模型的发展正在推动整个 AI 领域的技术创新。为了训练和部署这些模型，研究人员不得不开发新的算法、硬件和软件架构，这些创新又反过来推动了其他技术领域的发展。

10.4.2 大模型面临的挑战

尽管大模型展现出了巨大的潜力，但它们也面临着一些重要的挑战：

1. 计算资源需求：训练和运行大模型需要大量的计算资源。这不仅带来了高昂的经济成本，还引发了关于能源消耗和环境影响的担忧。
2. 数据隐私和安全：大模型需要海量数据进行训练，这引发了关于数据收集、使用和保护的伦理问题。确保用户隐私和数据安全是一个重要的挑战。
3. 偏见和公平性：大模型可能会继承训练数据中存在的偏见，导致在某些任务上表现出不公平或歧视性的行为。解决这个问题需要在数据收集、模型设计和评估等多个环节进行改进。
4. 解释性和可控性：由于大模型的复杂性，理解它们为什么做出某些决策 often 很困难。提高模型的可解释性和可控性是当前研究的一个重要方向。

10.4.3 大模型的未来展望

随着大模型技术日新月异改变着我们的未来，它将有 1.更高效的架构：研究人员正在探索如何设计更加高效的模型架构，以减少计算资源的需求。2.领域特化：虽然通用大模型很强大，但针对特定领域（如医疗、金融、教育等）优化的模型可能会带来更好的性能，Agent。3.多模态融合：未来的大模型可能会更好地整合文本、图像、视频和音频等多种数据类型，创造出真正的 ″通用智能″。

边缘计算：将大模型部署到移动设备和其他边缘设备上，实现更快速、更私密的 AI 应用。5.人机协作：大模型将不仅仅是工具，而是人类的智能伙伴，在各种复杂任务中与人类协作。

大模型正在重塑我们与技术互动的方式，为人工智能带来前所未有的可能性。虽然它们面临着诸多挑战，但其潜力是巨大的。

11 基于预训练大模型的本地知识库案例

1.为什么需要本地知识库

虽然大模型本身包含大量知识，但这些知识可能:

1.1 时效性有限

模型的知识更新需要依赖于模型的更新，而模型的更新需要依赖于大量的数据和计算资源，所以模型的知识更新是一个非常缓慢的过程，图 10.1 就是 GPT-4o 模型的训练截止日期 sim

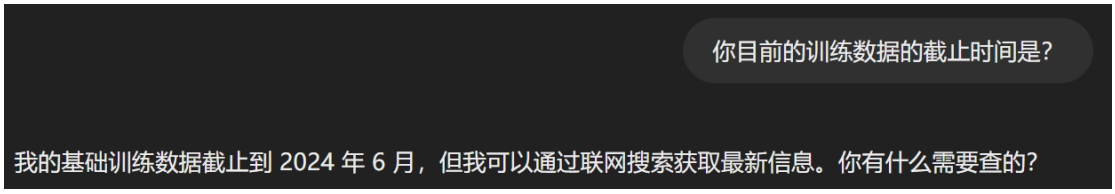


图 10.1(询问时间为 2025.2.8)

1.2 领域专业性不足

虽然大模型本身包含大量知识，但是这些知识是通用的，对于特定领域和任务，可能需要更专业的知识

拿中医领域的例子举例：考虑中医诊断的复杂性，比如同病异治、异病同治等，模型可能无法准确辨证论治

1.3. 无法涵盖企业/个人的私有信息

一些较为隐私的、或者模型训练时接触不到的信息，比如企业、学校内的一些规章制度，个人数据（如日记、聊天记录、社交媒体内容）等

1.4 存在事实性幻觉风险

可能生成看似合理但实际错误的内容，但是用户却无法察觉。图 10.2 是 Deepseek 回答的“用人单位在哪些节假日期间应当依法安排劳动者休假？”，然而根据劳动法第 40 条显示，正确的答案应该是:元旦、春节、国际劳动节、国庆节以及法律、法规规定的其他休假节日。这就是大模型产生的幻觉，可见大模型也会犯错，对其回答我们也要有自己的理解与判断。

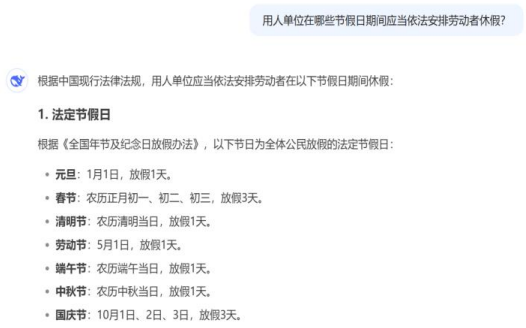


图 10.2(Deepseek 回答)

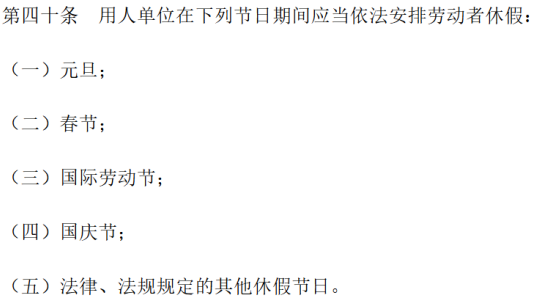


图 10.3(劳动法第 40 条)

这个时候如果有一个本地知识库，就可以尽量避免这些问题，因为知识库是用户自己建立的，所以用户可以随时添加知识信息到知识库中，同时确保知识库中的内容是正确的。那么大模型在在生成内容时，就可以调用知识库中的内容，使其的回答更加准确，更有专业性。

2. 预训练大模型+本地知识库可以做哪些有意思的事情？

2.1 将大模型的通用知识与本地知识库（如公司文档、个人笔记、行业资料）结合，提供　精准的领域内问答服务。

例如：企业内部知识库助手，回答员工关于规章、流程的问题；法律、医疗等垂直领域的专业问答机器人。

2.2 结合用户个人数据（如日记、聊天记录、社交媒体内容），生成定制化内容

例如：基于用户写作风格生成诗歌、故事，或根据兴趣推荐新闻摘要。

2.3 虚拟角色与互动游戏

例如：为角色设计本地知识库（如角色设定、历史对话记录），每次对话时，结合知识库中的角色背景和大模型生成符合人设的回答。

除此之外，预训练大模型+本地知识库还有众多有意思、有价值的应用场景，期待同学们在学完本章节后，能够举一反三，创造出更多有趣的应用。

那么最重要最独特的部分应该是： 如何让预训练好的模型能够"记住"和"调用"新的知识？

这让我想到了人类的学习过程的类比 - 我们在学校学到的是通用知识(预训练)，但在工作中还需要学习新的专业知识(本地知识库)。

3. RAG 技术的核心原理：

3.1 什么是 RAG：

RAG（Retrieval-Augmented Generation，检索增强生成）是一种结合了信息检索和文本生成技术的方法。它通过从外部知识库中检索相关信息，并将其作为输入提供给语言生成模型，从而增强模型在处理知识密集型任务时的表现。

3.2 RAG 的核心流程：

（1）文档预处理（非结构化数据->向量数据库）

- 文本分块（chunking）：将长文档分割为语义连贯的文本片段（最重要的部分）
- 向量编码（embedding）：将文本转换为数值向量表示
- 索引构建（indexing）：建立可快速检索的向量数据库

（2）问题向量化

- 问题解析：理解用户意图和关键信息
- 向量转换：将问题转换为与文档相同的向量空间表示

（3）语义检索

- 相似度匹配：在向量空间中查找与问题最相关的文档片段
- 结果排序：按相关性对检索结果进行排序筛选

有时候最合适的答案不在最前面->top_k

（4）增强生成

- 上下文组合：将检索结果与原始问题组合成增强提示（prompt）
- 答案生成：大模型基于增强提示生成最终回答

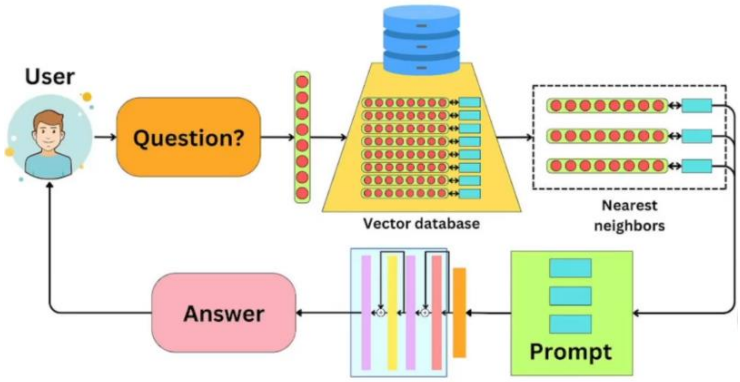


图 10.4 RAG 的流程图解

出处：大模型主流应用 RAG 技术架构深度解析（非常详细）零基础入门到精通，理论+原理+代码深入解析！（附课件教程）_哔哩哔哩_bilibili 8：28 处

我们用一个有趣的例子来方便大家理解：

RAG 就像 AI 客服的糊弄学宝典：

1. 【建话术库】把《敷衍客户 100 句》撕成小纸条
   - “亲的反馈很重要”,“将在 xx 个工作日给您回复，请您耐心等待”(敷衍话术向量化)
   - 塞进自动回复机器人（组成向量数据库）
2. 【收投诉】用户咆哮：“快递找不到了！”
   - 把咆哮转成翻译成机器可理解的语言（问题向量化）
3. 【搜话术】启动糊弄模式：
   - 匹配到：（语义检索到最相关的话术，星级表示相似度）
   ① 物流异常模板 ★★★★
   ② 不可抗力话术 ★★
   ③ 卖萌装死指南 ☆
4. 【回复】开启废话文学：
   ″亲亲，根据 xx 快递条例，您的情况已记录...(生成毫无卵用的安慰)″

3.3 文本向量化处理核心技术：

3.3.1 文本分块（chunk）策略

文本的分块可以说是整个 RAG 流程中的最困难的一部分。因为文本如果分得密度太大，那么单个文本片段中有太多信息，这样就会使后续的文本检索的难度提高，导致检索不精准；反之，如果分得密度太小，那么单个片段中的信息就不完全，可能想找的答案跨了两个甚至更多个文本片段，导致文本检索不全面。 常见的文本分块的策略有两种：固定长度分块和语义分块。

3.3.1.1 固定长度分块

固定长度分块顾名思义，就是按照固定的长度来对文本进行分块，但是固定的长度就意味着，大部分情况上下文是有割裂的，有时候关键信息可能恰好被切割在两个块的边界处。于是我们在处理文本的时候采用了块间重复的方法，具体来说就是，在固定分块长度的基础上，使每一个块含有重叠的部分。

示例：

原文：人工智能正在改变我们的生活。它让我们的工作更加高效，使我们的生活更加便捷。

分块(固定分块长度 20 字，重叠 5 字)：

块 1：人工智能正在改变我们的生活。它让我们的工

块 2：让我们的工作更加高效，使我们的生活更加便

块 3：生活更加便捷。

这样重叠设计能增加关键信息出现在检索结果中的概率，从而提高向量匹配的相关性。可见设置重叠部分的优势有:**1.防止语义断裂，2.提升检索准确性。**

但是需要注意的是，重叠比例需根据文本类型和任务需求调整。过大的重叠会增加存储冗余，而过小则可能无法有效保留上下文。

3.3.1.2 语义分块

语义分块不同于固定长度分块，它的目的是将文本划分成有意义的片段，而不是简单地基于固定长度或字符数进行分割。它通过分析文本的语义内容，识别出自然的语义边界，从而生成语义上完整的块。例如：

原文：人工智能正在改变我们的生活。它让我们的工作更加高效，使我们的生活更加便捷。

块 1：人工智能正在改变我们的生活。

块 2：它让我们的工作更加高效，使我们的生活更加便

语义分块的优势是它能够更好地保留文本的上下文信息，避免因简单分割导致的语义丢失。但它通常比基于规则的分块方法更复杂，计算成本也更高。

3.3.2 如何将文本转换为向量

经过上一步文本分块的操作，我们已经成功将长文本分成了多个文本片段，但是对于这种自然语言文本，计算机是很难处理的，所以我们需要将其转换成向量，方便计算机的理解与处理。

3.3.2.1 向量化的基本概念

文本向量化就是将文本转换为数值向量的过程。例如，将句子“我喜欢人工智能”转换为一个数值向量：[0.23, -0.45, 0.12, ...]。这个过程就像是在给文本找到它在高维空间中的坐标位置。

3.3.2.2 文本向量化的计算

（不清楚是否要补充）

3.3.3 构建向量数据库

我们将文本转换为向量后，就需要将这些向量存储起来，但是如果仅将向量存储在普通文件（如 CSV 或二进制文件）中，每次检索都需要遍历所有向量。我们将这种搜索方法成为：当数据量达到百万级时，检索延迟会变得极高。那么怎样才能更好的存储向量数据呢？这时候就需要一种全新的存储形式：**向量数据库**。

3.3.3.2 向量数据库的特点

- **建立映射关系**：向量数据库会建立向量与原始数据的映射关系。例如，　检索到相似向量后，能直接返回对应的文本段落、图片路径或相关元数据。

- **提升检索效率：**向量数据库通过索引优化加速相似性搜索。
- **支持语义匹配：**传统数据库依赖关键词精确匹配（如搜索"小狗"无法返回"柴犬"），而向量数据库通过余弦相似度等算法实现语义匹配。这将会在后面“3.4 相似度计算方法”的内容中提及。
- **动态更新：**如果直接存储为静态文件，新增或删除数据需全量重新生成文件，效率极低，而向量数据库支持实时插入与删除数据。
- **优化资源利用：**向量数据库采用量化技术，可将 1024 维向量压缩至 64 字节，节省 75%存储空间。

3.3.3.1 向量数据库的检索方法

暴力搜索：比较每一个向量的相似度。搜索时间长，但是搜索的质量是完美的，真的比较了每一个向量，如果数据库中的数据规模较小，这也不失为一种好的方法，但是实际应用中的数据规模往往都不会太小，那么有没有其他进行优化过的算法呢？

近似最近邻搜索（Approximate Nearest Neighbor, ANN）：为了提高大规模数据集上的查询效率，向量数据库基本采用 ANN 算法。这些算法可以大幅提高数据检索的速度，但是它不一定能够保证一定能找到最近邻的向量，只能得到一些近似的结果，这也是为什么这种方法称之为近似最近邻搜索。

（这里不清楚是否需要补充一些具体的**近似最近邻搜索**算法）

3.4 相似度计算方法

构建好向量数据库后，我们只需要将问题转换成问题向量，再与向量数据库中的向量进行检索，不就能找到问题最相关的答案了吗？但需要注意的是，这里的检索只能找到一些与问题匹配的文本片段，这是对原来文本片段的一次初筛，但是不能准确判断出最匹配的文本片段，所以我们需要通过计算向量相似度的方法来近一步缩小范围。

3.4.2 相似度计算方法

计算相似度的方法有很多种，这里只举几种比较常见的方法

欧氏距离：

$$d(A,B)=\sqrt{\sum_{i=1}^{n}(a_i-b_i)^2}$$

欧氏距离衡量的是多维空间中两点之间的直线距离，如图 10.5 所示，欧氏距离的值越小，表示两点越相近，则向量的相似度就越高。

余弦相似度：

$$\mathrm{sim}(A,B)=\frac{A\cdot B}{|A||B|}=\cos\theta$$

是余弦相似度衡量的是两个向量在方向上的相似性，通过计算它们的夹角余弦值来评估方向一致性，如图 10.6 所示，余弦相似度的值越大，表示两个向量的夹角越小，则向量的相似度就越高。

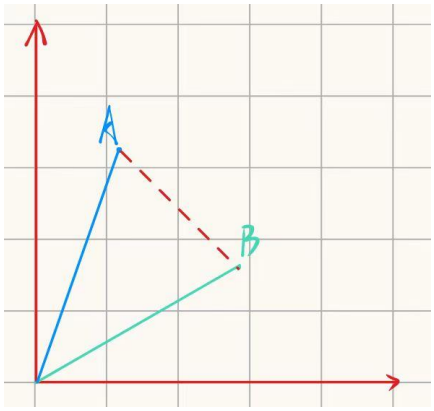


图 10.5 欧氏距离图解

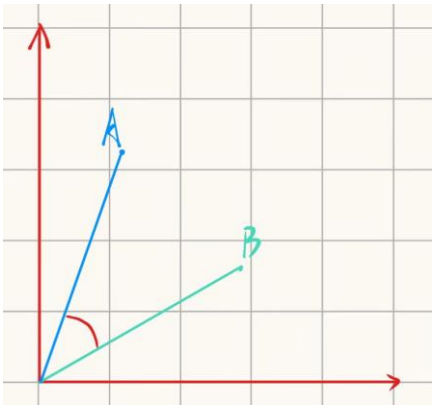


图 10.6 余弦相似度图解

接下来我们举个简单的计算示例

示例：

假设向量维度为 2，数据库中有以下向量：

文本块 1："物流异常模板" → $A(0.6, 0.8)$

文本块 2："不可抗力话术" → $B(0.5, 0.5)$

查询问题："快递延迟" → $Q(0.7, 0.7)$

求问题 Q"快递延迟"最匹配的文本是？

解：

（1）使用欧氏距离求解：

$$d(Q,A) = \sqrt{(0.7-0.6)^2 + (0.7-0.8)^2} = \sqrt{0.02} \approx 0.14$$

$$d(Q,B) = \sqrt{(0.7-0.5)^2 + (0.7-0.5)^2} = \sqrt{0.08} \approx 0.28$$

排序结果：$d(Q,A) < d(Q,B)$，所以问题 Q“快递延迟”匹配到的文本就是 A“物流异常模版”

(2) 使用余弦相似度求解：

$$\text{sim}(Q,A) = \frac{0.7 \times 0.6 + 0.7 \times 0.8}{\sqrt{0.7^2 + 0.7^2} \times \sqrt{0.6^2 + 0.8^2}} = \frac{0.98}{0.9899 \times 1} \approx 0.99$$

$$\text{sim}(Q,B) = \frac{0.7 \times 0.5 + 0.7 \times 0.5}{\sqrt{0.7^2 + 0.7^2} \times \sqrt{0.5^2 + 0.5^2}} = \frac{0.7}{0.9899 \times 0.7071} \approx 0.71$$

排序结果：$sim(Q,A) > sim(Q,B)$，所以问题 Q“快递延迟”匹配到的文本就是 A“物流异常模版”

3.4 增强生成

现在我们已经通过语义检索找到了最符合文本向量，向量数据库会根据当前向量的映射关系，找到原始的数据即：问题对应的文本片段

然后我们将文本片段与原始问题组合成增强提示词（prompt）发送给大模型，大模型就会基于增强提示词生成最终回答，至此就完成了 RAG 的全流程。

4.LangChain+GLM 实现基于本地知识库系统搭建:

本项目旨利用 LangChain+GLM 实现基于本地知识库系统搭建，尤其是问答等应用。本项目将让同学们快速了解当今最先进的大语言模型（LLM）的基础内容，并能掌握 LangChain 这一 LLM 开发的技术框架，并利用其内部集成的组件库可以将 LLM 模型与各种外部数据源（包括但不限于图片、PDF、视频、邮件...）进行连接。项目中 LangChain 作为一个工具链，通过提示（prompt），可以指导 LLM 执行一系列的任务(简单的对于文件查询、复杂的对于行业知识图谱的搜集)。本项目主要任务是通过使用 LangChain 框架和组件，利用开放大模型 ChatGLM，建立可离线部署的检索增强生成(RAG)大模型知识库，重点实现基于知识库的问答应用，并为 LLM 扩展了更多的能力提供参考。

近年来，随着 OpenAI 发布了多款大语言模型的线上应用，掀起了人工智能研究和应用的新高潮，AIGC 日益成为热门话题。其中前年的对话式语言大模型 ChatGPT (chat generative pre-trained transformer)第一次面世，允许允许用户使用自然语言对话形式进行交互, 可实现包括自动问答、文本分类、自动文摘、机器翻译、聊天对话等各种自然语言理解和自然语言生成任务. 以 ChatGPT，ChatGLM 等模型 在开放环境的自然语言理解上展现了出色的性能, 甚至无需调整模型参数, 仅使用极少数示例数据即可在某些任务上超过了传统的模型，传统模型主要为特定任务专门设计并且使用专门的监督数据和算法进行训练的模型。

当面对用户所提出的各种文本生成类任务时，通过 prompt 调试方式，以 ChatGPT 为首的 LLM 在多数情况下可以生成流畅通顺、有逻辑性且多样化的长文本。使用相似模型架构和训练方式，LLM 还可以扩展到其他模态应用上，比如今年年初热门的 SORA 系统，由 OpenAI 发布，可以通过简单语言 prompt 提示，可以生成逼真，与提示相符的长时间的视频，展示了 LLM 的基于预训练+调试的的应用处理新范式。在这背景下，掌握 LLM 的部署和 prompt 提示过程来完

成一系列任务成为必要的人工智能专业人士技能。

视频参考:

https://www.bilibili.com/video/BV13M4y1e7cN/?share_source=copy_web&vd_source=e6c5aafe684f30fbe41925d61ca6d514

项目环境:

Python：3.11

LLM： THUDM/ChatGLM3-6B

Embedding ： BAAI/bge-large-zh

Git：https://github.com/chatchat-space/Langchain-Chatchat

12 大模型+机器人（具身智能）案例

具身智能。