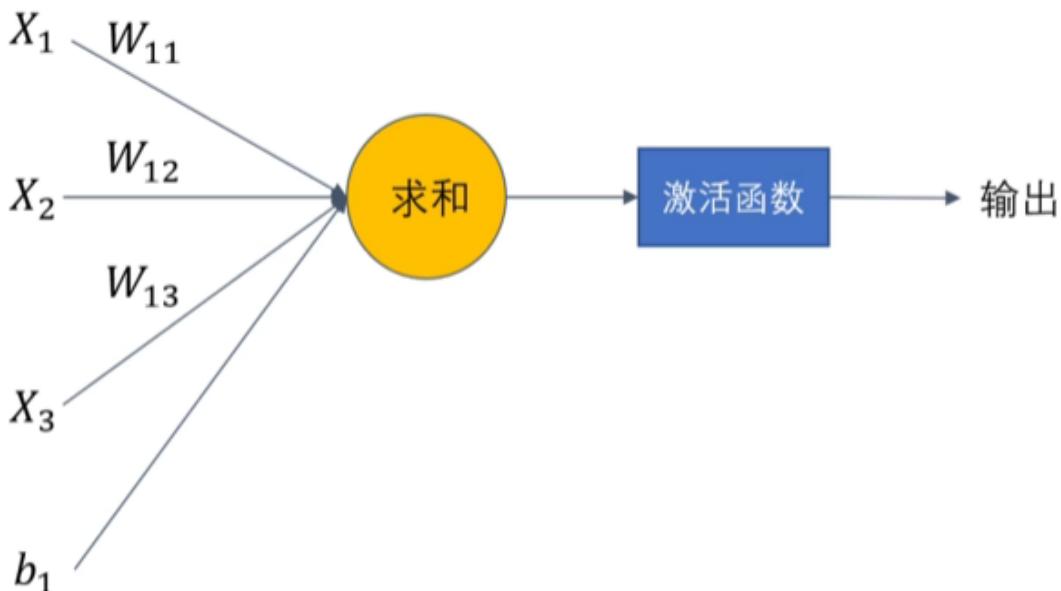


# 全连接神经网络



```
a1 = h(w1*x + b1)
a2 = h(a1*x + b2)
...
an = h(a(n-1)*x + bn)
```

## 激活函数

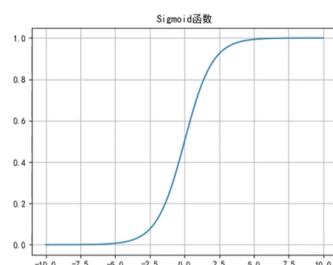
- 激活函数作用：让神经网络突破线性模型的限制，能够拟合任意复杂的非线性关系

`sigmoid`

## 激活函数---Sigmoid函数

Sigmoid函数的公式和导数如式所示：

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

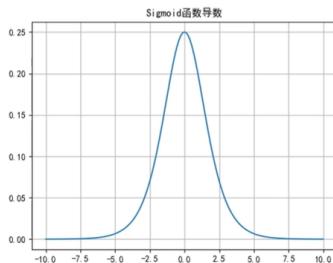


### Sigmoid函数优点：

- 简单、非常适用分类任务；

### Sigmoid函数缺点：

- 反向传播训练时有梯度消失的问题；
- 输出值区间为(0,1)，关于0不对称；
- 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



# 1. w、b 和激活函数导数的关系

以一个最简单的神经元为例：

$$z = wx + b, \quad y = \sigma(z) \quad (\sigma = \text{sigmoid})$$

有一个损失函数  $L$ 。

反向传播时，对参数的梯度是：|

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w} = \frac{\partial L}{\partial y} \cdot \sigma'(z) \cdot x$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial b} = \frac{\partial L}{\partial y} \cdot \sigma'(z)$$

所以：

w 和 b 的梯度里，一定乘着激活函数的导数  $\sigma'(z)$ 。

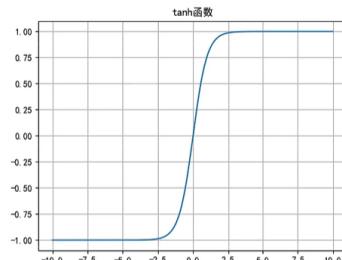
$\sigma'(z)$  一旦很小， $\frac{\partial L}{\partial w}$ ,  $\frac{\partial L}{\partial b}$  就都会很小，更新几乎停了。

Tanh

## 激活函数---Tanh函数

Tanh函数的公式和导数如式所示：

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

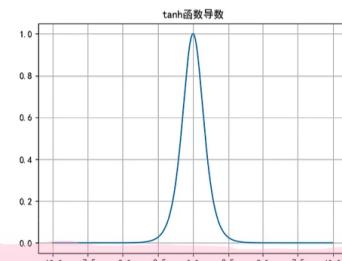


Tanh函数优点：

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

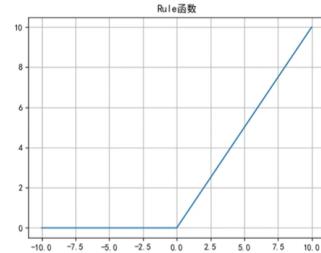


ReLU

## 激活函数---ReLU函数

ReLU函数的公式和导数如式所示：

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

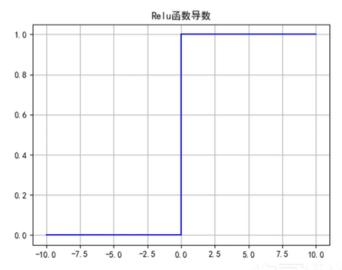


ReLU函数优点：

- 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；

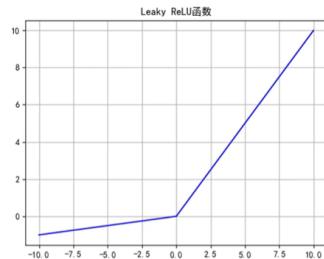


Leaky ReLU

## 激活函数---Leaky ReLU函数

Leaky ReLU函数的公式和导数如式所示：

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

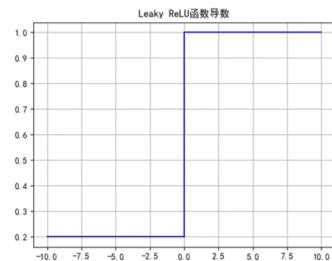


Leaky ReLU函数优点：

- 1、解决了ReLU的神经元死亡问题；

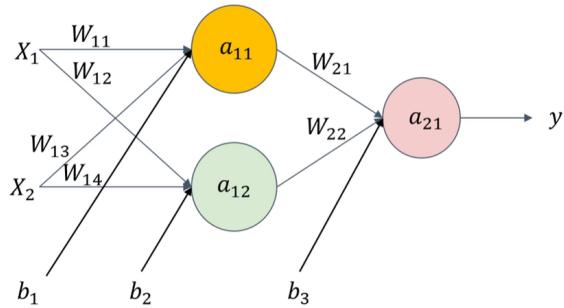
Leaky ReLU函数缺点：

- 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）



## 前向传播

## 全连接神经网络前向传播



前向传播计算过程：

$$\begin{aligned}a_{11} &= \text{sigmoid}(x_1 w_{11} + x_2 w_{13} + b_1) \\a_{12} &= \text{sigmoid}(x_1 w_{12} + x_2 w_{14} + b_2) \\a_{21} &= \text{relu}(a_{11} w_{21} + a_{12} w_{22} + b_3) \\y &= a_{21}\end{aligned}$$

- 目的：算输出 $y$ 预测值，得到损失 Loss

## 反向传播

- 目的：用链式法则算出所有参数的梯度： $\frac{\partial L}{\partial W^{(l)}}, \frac{\partial L}{\partial b^{(l)}}$

## 梯度下降

- 用这些梯度更新参数

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}}$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}}$$

- $\eta$ : 学习率
- “当前值 - 学习率  $\times$  梯度” = 朝“让损失变小”的方向迈一步

## Dropout正则化

- 在训练阶段，对某一层的神经元随机丢弃一部分为0，达到降低过拟合，提升泛化能力的效果。
- 使用原因：全连接层参数多时，容易过拟合
- 一般放置位置

Linear --> Activation(ReLU...) --> Dropout --> 下一层 Linear  
Linear --> Dropout --> Activation  
更常见、更稳定是放在激活函数之后

## 图像增强

### 水平翻转 (Horizontal Flip)

- 沿着垂直中轴左右镜像图片

- 适用于场景分类、目标检测、分割任务等

## 随机裁剪 (Random Crop)

- 从原图中随机截取一个子区域，可以再resize回目标尺寸
- 常见形式：`RandCrop`、`RandomResizedCrop`
- 分类任务里常见`RandomResizedCrop(224)`

## PCA颜色增强

- 模拟光照和整体色调变化，能显著提高模型对各种真实场景变化的鲁棒性
- 适合自然图像分类任务

# 卷积神经网络

- 卷积神经网络CNN相比于全连接神经网络FCNN的优势
  - 参数大幅减少** → 减少过拟合、更易训练；
  - 局部连接** → 利用图像“邻近像素相关”的先验；
  - 参数共享** → 对平移更鲁棒，且进一步减少参数；
  - 特征层次化** → 低层学边缘，高层学语义，特别适合视觉任务；
  - 计算效率高** → 对高维输入（图片、视频）更友好。

## 补充：通道channel

- 灰度图-->单通道 ( $H \times W$ )

0=完全黑、255=完全白、中间就是不同程度的灰

- 彩色图-->三通道 ( $H \times W \times 3$ )

常见彩色图用RGB表示：R-->红色、G-->绿色、B-->蓝色

- 一般表示

```
PyTorch: (C, H, W)
TensorFlow: (H, W, C)
灰度图C=1, 彩色图C=3
```

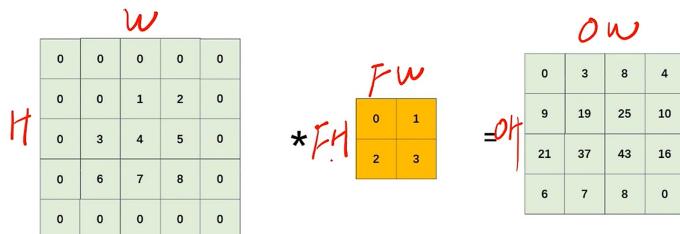
- 在 CNN 里，通道表示 在同一个空间位置上，有多少种不同的特征。
- 每个 **卷积核 (filter)** 会学到一种“特征”
  - 比如：竖直边缘、水平边缘、角点、纹理、某种形状.....
- 输出通道数 = 卷积核个数 = 这层学到的特征种类数**

# 卷积

- 卷积：增加通道。卷积核可以认为是w
- 输入特征图和输出特征图的计算公式！！！

## 卷积层---经过卷积运算后的特征图大小

计算公式：



$$OH = \frac{H + 2P - FH}{S} + 1 \leftarrow$$

$$OW = \frac{W + 2P - FW}{S} + 1 \leftarrow$$

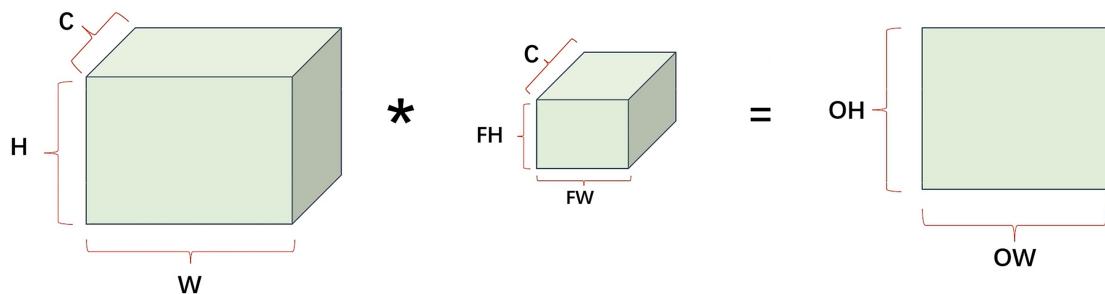
$$OH = \frac{3 + 2 * 1 - 2}{1} + 1 = 4 \leftarrow$$

$$OW = \frac{3 + 2 * 1 - 2}{1} + 1 = 4 \leftarrow$$

其中P为填充， $P=1$ ； $S$ 为步幅， $S=1$ ；输入 $H \times W = 3 \times 3$ ； $FH \times HW$ 为卷积核大小， $2 \times 2$ ；基于此公式求出输出 $OH \times OW = 4 \times 4$

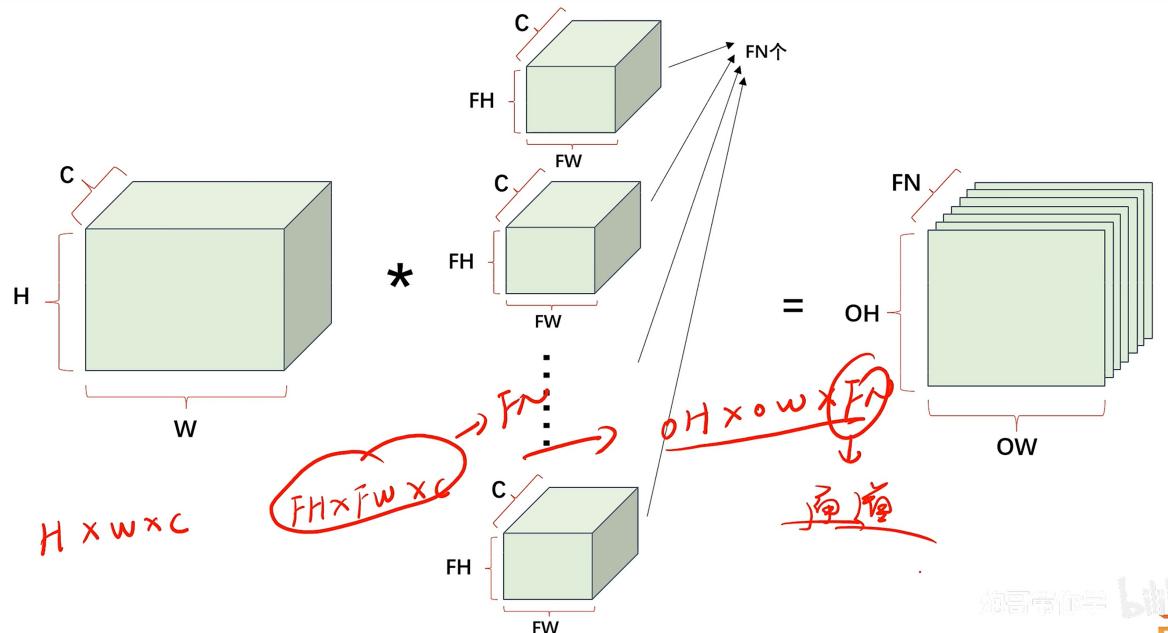
## 多通道卷积

### 卷积层---利用立体图来表示卷积运算



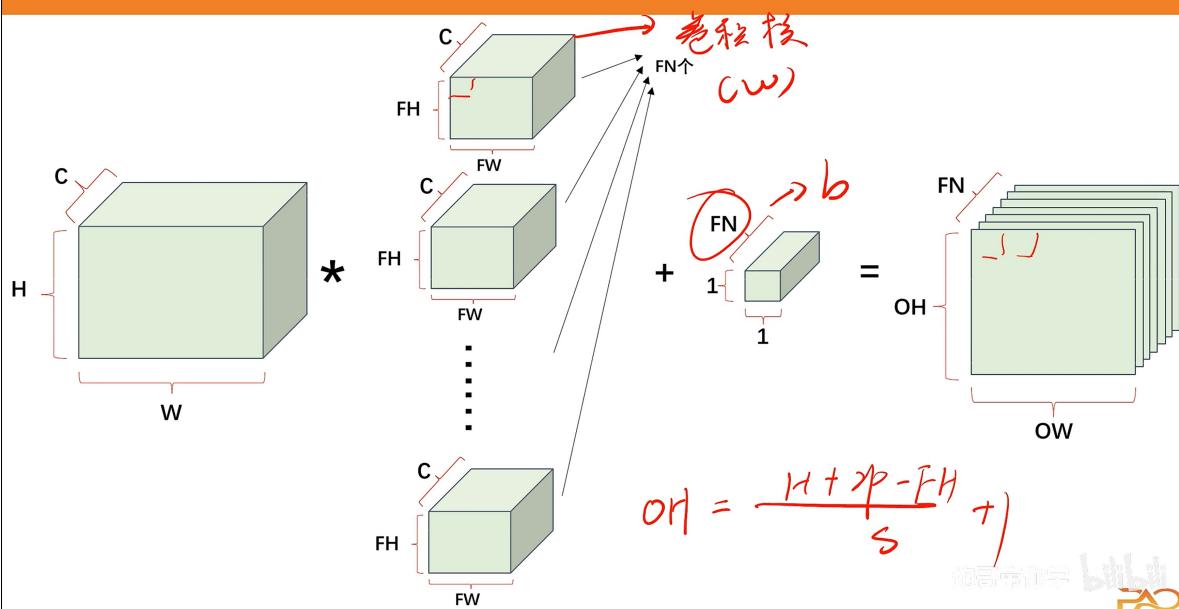
输入层通道数=卷积核通道数

## 卷积层---利用立体图来表示卷积运算



卷积核个数=输出层通道数

## 卷积层---利用立体图来表示卷积运算



输入层通道数=卷积核( $w$ )通道数，卷积核个数=偏置( $b$ )通道数=输出层通道数

## 池化

- 池化：本质是降采样 (downsampling)、压缩空间尺寸，不改变通道，增加模型鲁棒性

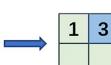
## 池化层---经过池化层后的特征图大小

计算公式：

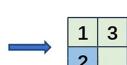
1	2	3	3
0	1	1	5
3	0	1	0
1	4	1	2



1	2	3	3
0	1	1	5
3	0	1	0
1	4	1	2



1	2	3	3
0	1	1	5
3	0	1	0
1	4	1	2



1	2	3	3
0	1	1	5
3	0	1	0
1	4	1	2



$$OH = \frac{H + 2P - FH}{S} + 1 \leftarrow$$

$$OW = \frac{W + 2P - FW}{S} + 1 \leftarrow$$



$$OH = \frac{4 + 2 * 0 - 2}{2} + 1 = 2 \leftarrow$$

$$OW = \frac{4 + 2 * 0 - 2}{2} + 1 = 2 \leftarrow$$

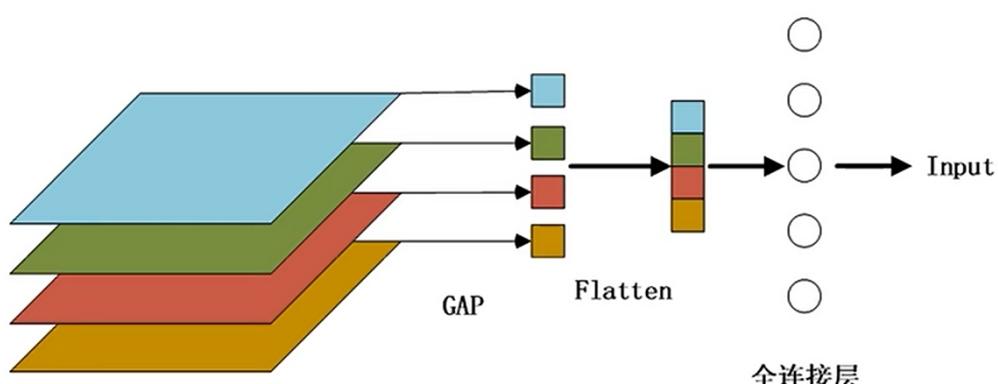
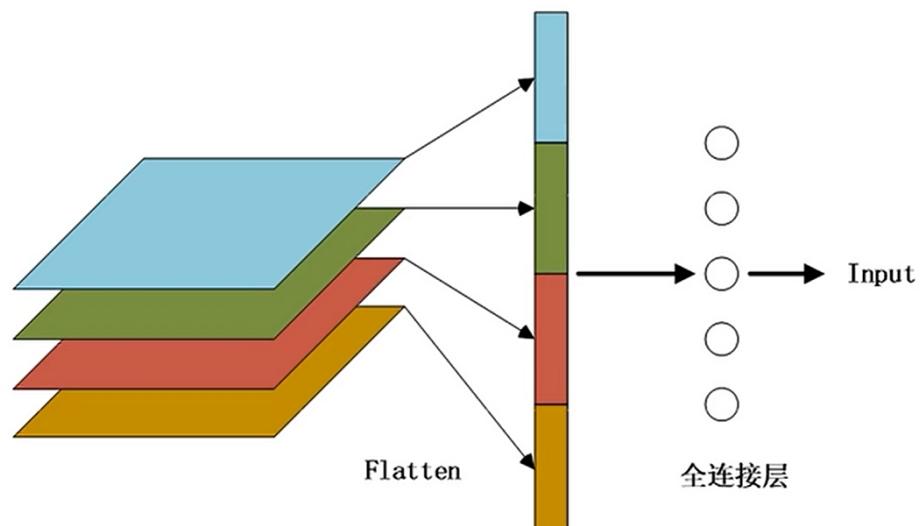
- 这里的FH和FW并不是卷积核的大小，池化没有池化核的说法，这里叫感受野，即每部池化的区域大小

### 平均池化

### 最大池化

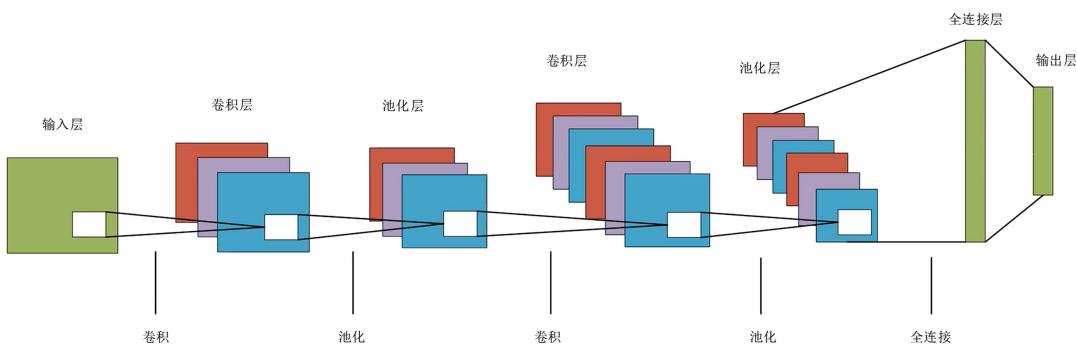
### 全局平均池化 (GAP)

- 优势：
  - 抑制过拟合：直接flatten做全连接层会有很多参数，经GAP后再flatten可减少参数，防止过拟合
  - GAP使特征图输入尺寸更加灵活
- 缺点：
  - 信息丢失
  - 特征丰富性：会丢失细节和特征
  - 会导致梯度消失或梯度爆炸，影响优化过程
  - 不适用于复杂任务



## CNN整体结构

### 卷积神经网络整体结构



- 第一次卷积过程中，输出通道为3，则第一次卷积核个数为3
- 池化不改变通道数，它只改变输出层的高和宽，降维
- 卷积和池化完成特征提取过程
- 卷积和池化结束后先把最后的特征图  $C \times H \times W = 6 \times 6 \times 6$  展平(Flatten)成一个长度为  $C \times H \times W = 216$  的一维向量，再将这个向量送入若干全连接层，最后由输出层根据任务给出分类或回归结果

## 前向传播

- 将输入数据传入模型，得到模型结果

```
outputs = net(imgs)
```

## 计算损失

- 比较模型输出和真实标签，得到损失值
- 回归一般用均方误差，分类一般用交叉熵损失

```
loss = loss_fn(outputs, targets)
```

## 梯度清零

- 每次反向传播前都要清楚上一次的梯度

```
optimizer.zero_grad()
```

## 反向传播

- 根据损失值计算模型参数的梯度

```
loss.backward()
```

## 优化器更新参数

- 根据计算得到的梯度，调整模型参数

```
optimizer.step()
```

## CNN代表

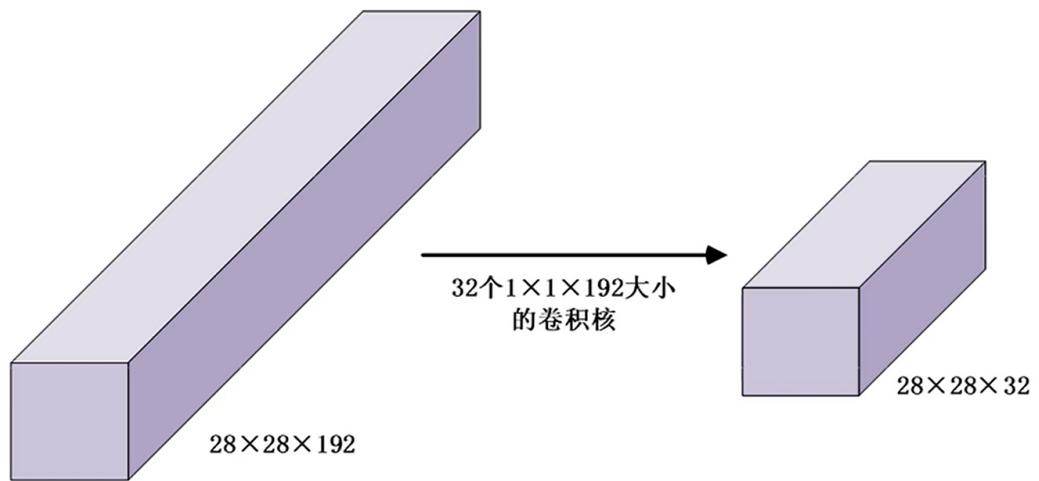
### LeNet

### AlexNet

### VGG

### GoogLeNet

- 1x1卷积作用：
  - 实现跨通道的交互和信息整合；
  - 卷积核通道数的降维和升维，减少网络参数



- inception块：不改变图片的W和H,只改变Channel。通过并行的卷积核filter将输入特征提取并在通道维上进行串联合并，构成下一层输入

## ResNet

- Batch Normalization(BN)

## 1. BN 是干嘛的？

对一个卷积层输出的特征图  $x$  (形状  $[N, C, H, W]$ ) :

BatchNorm2d 会对**每个通道 C 独立做**:

$$\mu_c = \text{mean of } x_c \quad \sigma_c^2 = \text{var of } x_c$$

$$\hat{x}_c = \frac{x_c - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

然后再学两个参数  $\gamma$ 、 $\beta$ :

$$y_c = \gamma_c \hat{x}_c + \beta_c$$

- **N**: batch 维
- **C**: 通道维
- **H, W**: 空间维
- $\gamma$ 、 $\beta$  是可学习的 scale / shift

好处:

- 让每层输出的分布比较稳定 (大致 0 均值、单位方差)
- 可以用更大学习率
- 减少梯度爆炸 / 消失
- 深层网络更容易训练

## 循环神经网络

---

**RNN**

---

**LSTM**

---

**GRU**

---

**Transformer**

---

**算法背景**

---

- RNN

RNN是一种经典序列模型，它通过循环的方式将序列中的信息逐个输入到网络中，并在网络内部使用循环结构来捕捉序列中的时间依赖关系。RNN读句子必须从左往右，一个字一个字地读（串行计算）。若句子很长，第一个字信息就变得模糊不清（局部观察视野），即存在**长距离依赖问题**

- Transformer

transformer是一种基于**自注意力机制**的神经网络架构，可以一眼看完整个句子（并行处理），所有词同时计算，不分先后。让大规模并行计算成为可能。

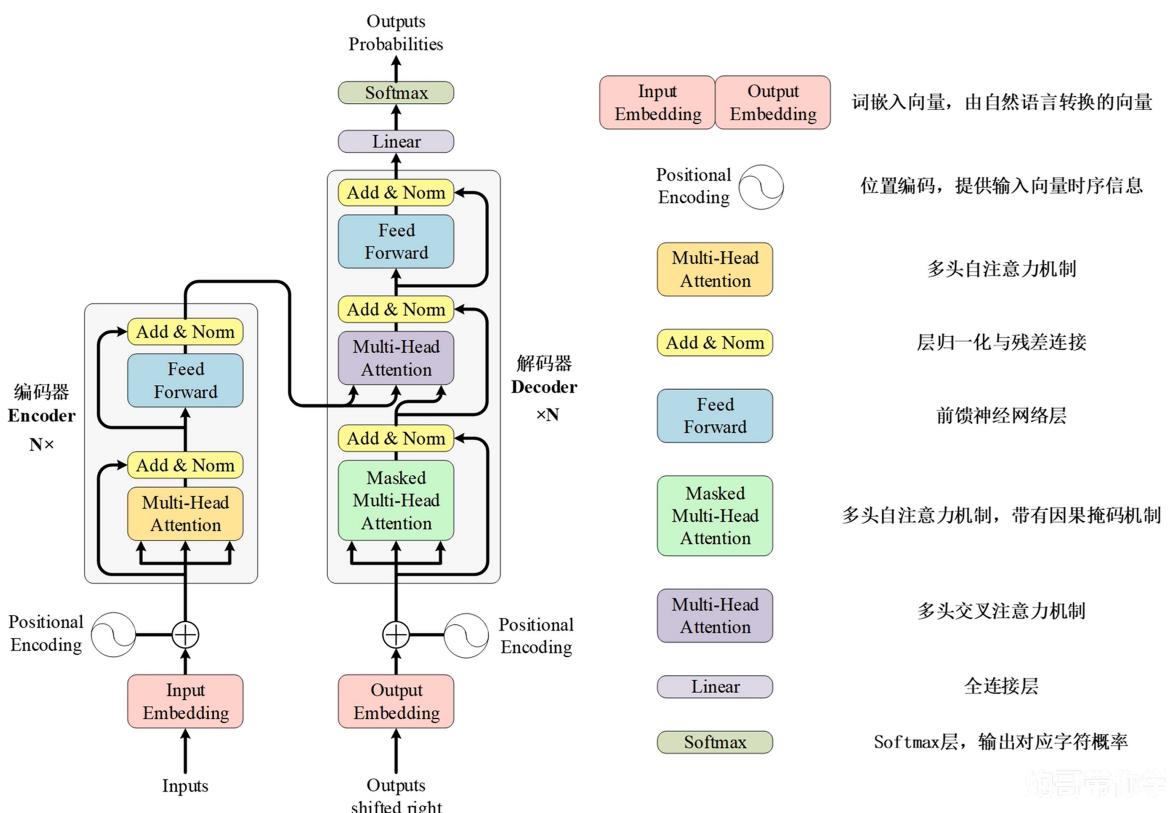
- transformer影像

- NLP（自然语言处理）基石：BERT、大模型系列
- 跨模态发展：ViT（图像）、多模态大模型

- 结构：利用编码器与解码器结构

- 三种注意力：编码器多头自注意力、交叉注意力、解码器多头自注意力（含因果掩码）
- 位置信息：用位置编码赋予词向量序列信息
- 残差连接+层归一化+前馈网络（FFN）：形成标准层块，稳定深层训练

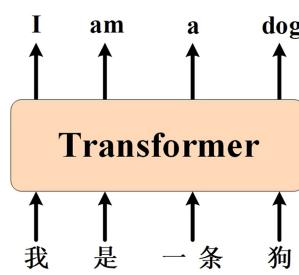
## Transformer架构



## Transformer模型的输入

### 独热向量 (One-Hot Encoding) -->ID

- 原理：假设词表里有 10,000 个词。独热向量就是一个长度为 10,000 的向量。如果是第 3 个词（比如“苹果”），那么向量只有第 3 位是 1，其他全是 0。数学表示： $X_{apple} = [0, 0, 0, 1, \dots, 0]$
- transformer不用它计算原因
  - 维度太大：若词表有10万个词，输入向量长度就是10万，计算量太大
  - 没有语义：模型看不出苹果和香蕉是水果，Attention机制依赖相似度计算，如果是独热向量，无法计算相似度。



**注意：**

- 1、Transformer里的计算都是数值
  - 2、文本只是进来前要被“数字化”，出去后再把数字“还原成文字”。

如何处理自然语言数据

- 1、收集大量的文本数据
  - 2、把字符串切成token，具体如下所示：

[我是一条狗篮球鸡.....]

- 3、统计收集的样本中token的出现频次，并进行排序。

4、将token转换为独热向量。

我 → [1 0 0 0 0 0 0 .....]

是  $\rightarrow [010000 \dots]$

一条  $\rightarrow [0\ 0\ 1\ 0\ 0\ 0\ \dots\dots]$

狗 → [0 0 0 1 0 0 0 .....]

篮球 → [0 0 0 0 1 0 0 .....]

鷄 → [0 0 0 0 1 0 .....]

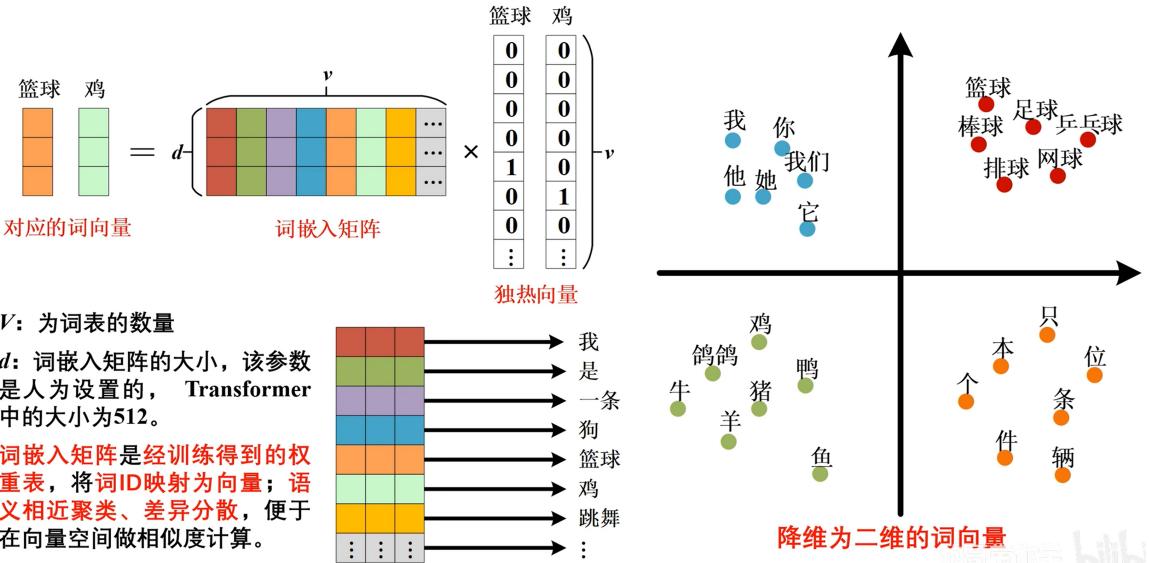
• • • • •

独热向量处理数据

强烈推荐使用数据

**词嵌入 (word Embedding) -->模型实际输入部分**

- 原理：将长长的稀疏向量，压缩成一个低维（例如512维）的稠密向量。每一维都是一个实数。数学： $X_{apple} = [0.12, -0.45, \dots, 0.03]$ ，这样的优势是苹果和香蕉的向量在空间中距离很近。



- Embedding就是把one-hot通过一个可学习的线性层变成低维稠密向量，这个Embedding向量就是进入self-Attention的X

## 位置编码

Transformer的位置编码公式：

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \quad PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

其中：

$d$ : 词嵌入  $i = 0, 1, \dots, \frac{d}{2}-1$   $pos$ 指当前token在序矩阵的大小  $\frac{d}{2}$  列中的第几个位置

例如：

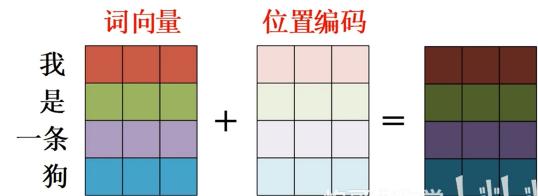
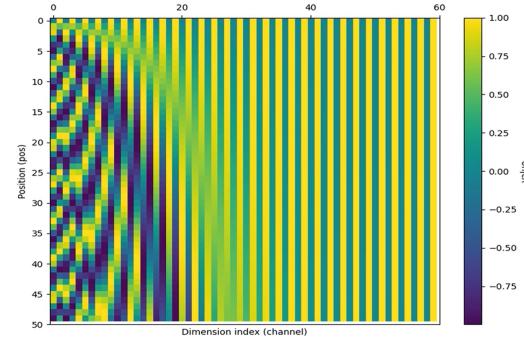
"我是一条狗"令位置索引 $pos = 0, 1, 2, 3$ ，维度  $d=3$  时：

$pos = 0$ ("我") : 0.000000, 1.000000, 0.000000

$pos = 1$ ("是") : 0.841471, 0.540302, 0.002154

$pos = 2$ ("一条") : 0.909297, -0.416147, 0.004309

$pos = 3$ ("狗") : 0.141120, -0.989992, 0.006463



## 自注意力机制

模型参数解释：

$L$ : 序列长度,  $d$ 每个token的维度;  
 $d_k$ : 权重向量与不同输出向量维度;

投影权重与线性投影向量：

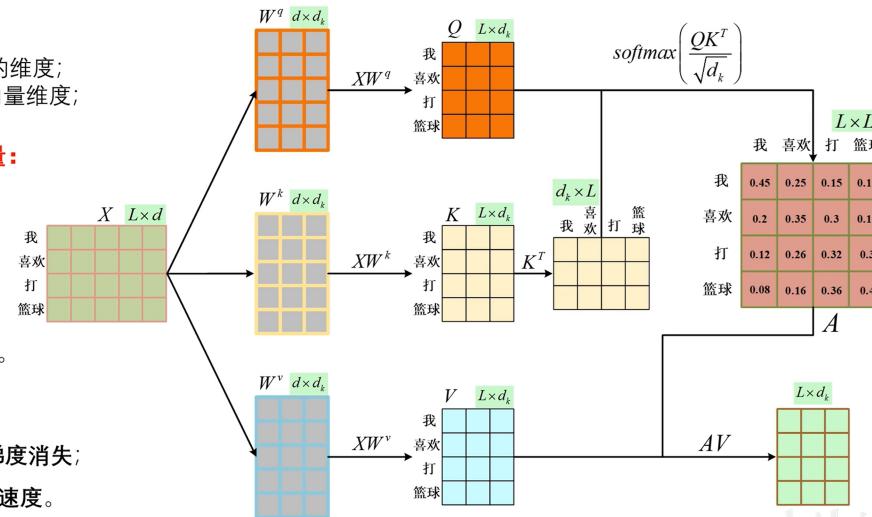
$W^q, W^k, W^v; Q, K, V$

注意力权重：

$A$ : 注意力权重矩阵, 用来把各位置信息按重要性加权汇聚, 生成新的上下文表示。

为什么要除以 $\sqrt{d_k}$ :

- 1、防止 softmax 饱和  $\rightarrow$  梯度消失;
- 2、提升训练稳定性与收敛速度。

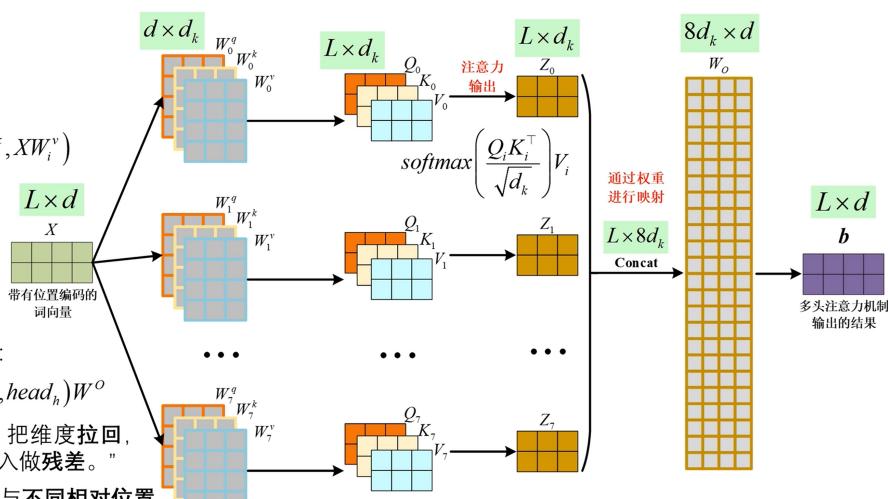


## 多头注意力机制

多头注意力机制定义：

做  $h$  组独立的注意力机制

$$head_i = \text{Attention}(XW_i^q, XW_i^k, XW_i^v)$$



拼接并线性融合回模型维度:

$$MHA(X) = \text{Concat}(head_1, \dots, head_h)W^O$$

$W^O$ “把多头拼接变成融合, 把维度拉回, 让后续网络能用、还能和输入做残差。”

直觉：不同头在不同子空间与不同相对位置上做对齐；有人看短依赖，有人看长依赖，有人关注实体，有人关注标点、句法等。

通过上图可以得出一个结论，就是注意力机制的输入可以是不同序列长度，这也符合机器翻译等时间序列任务。

## 掩码注意力机制

## 填充掩码

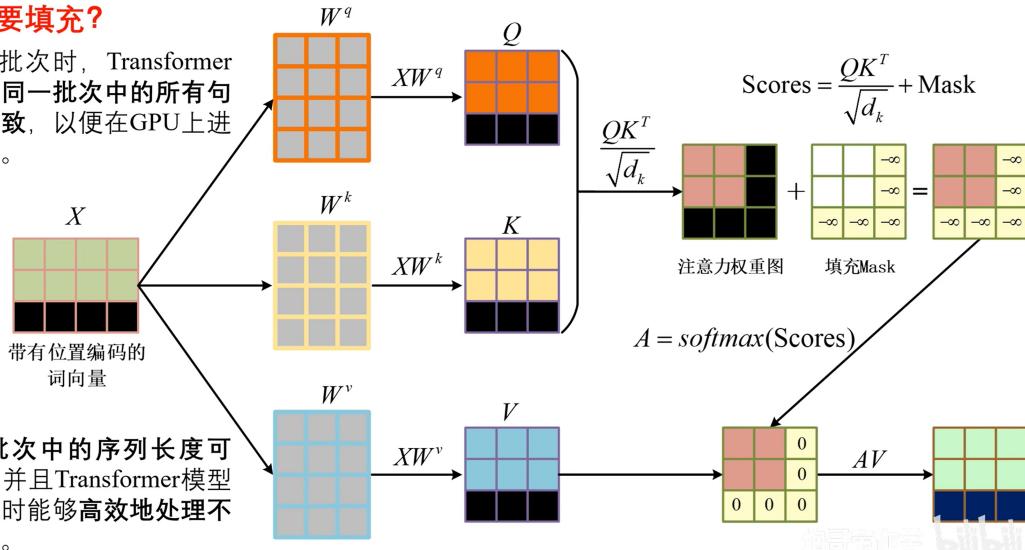
- 解决长短不一的问题

### 为什么需要填充？

在处理批次时，Transformer的输入要求同一批次中的所有句子的长度一致，以便在GPU上进行并行计算。

**注意：**

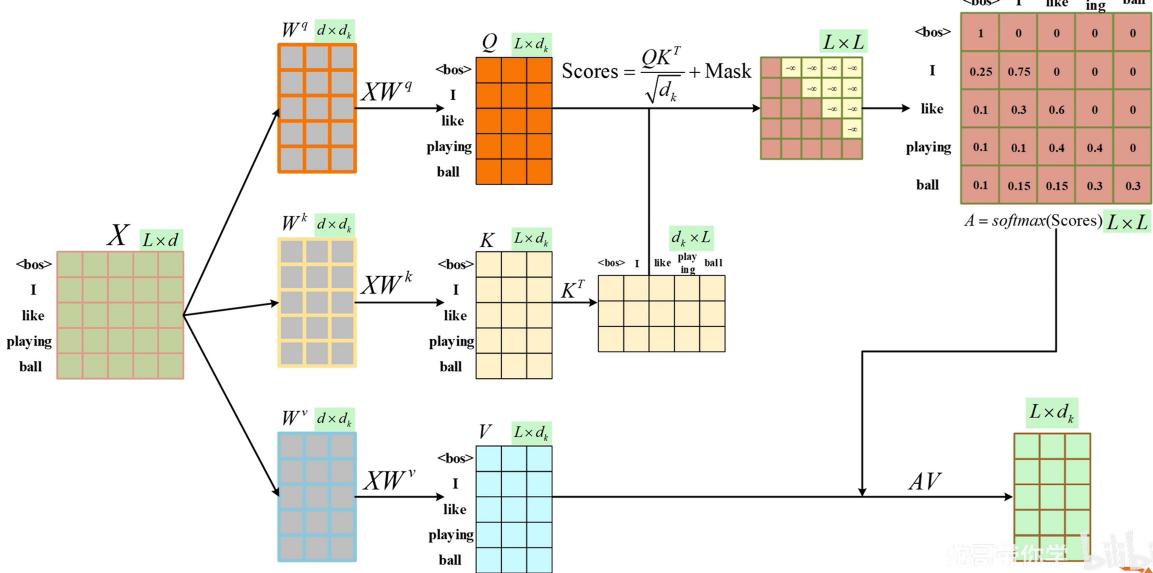
不同批次中的序列长度可以不一样，并且Transformer模型通常在处理时能够高效地处理不等长的序列。



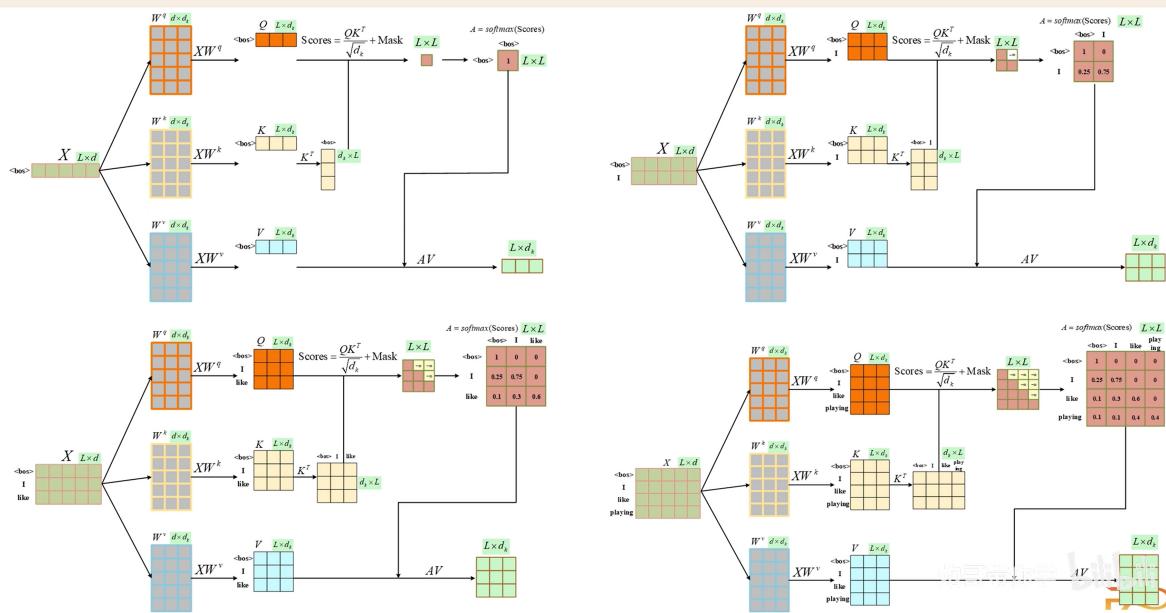
## 因果掩码

- 解决偷看答案问题

### 掩码注意力机制-因果掩码（训练过程中）

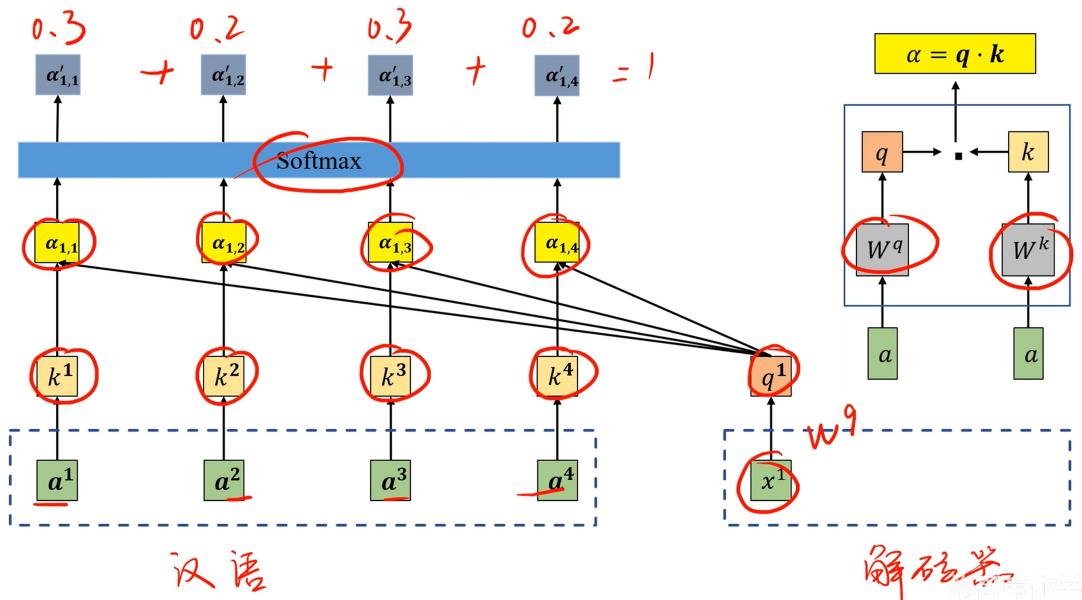


## 掩码注意力机制-因果掩码（推理过程中）

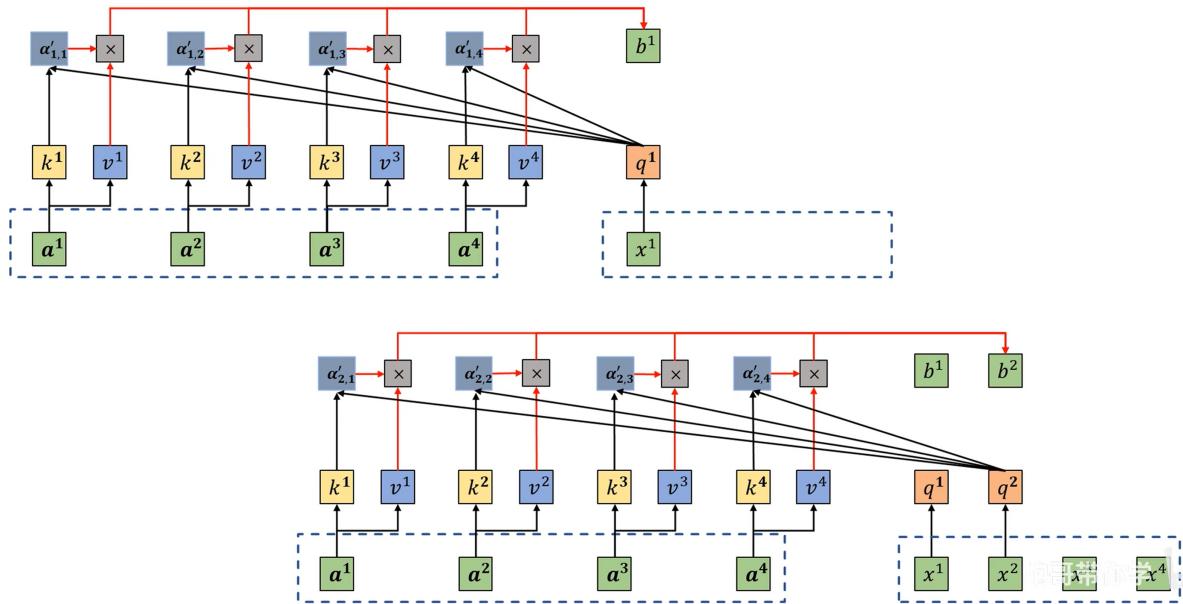


## 交叉注意力机制

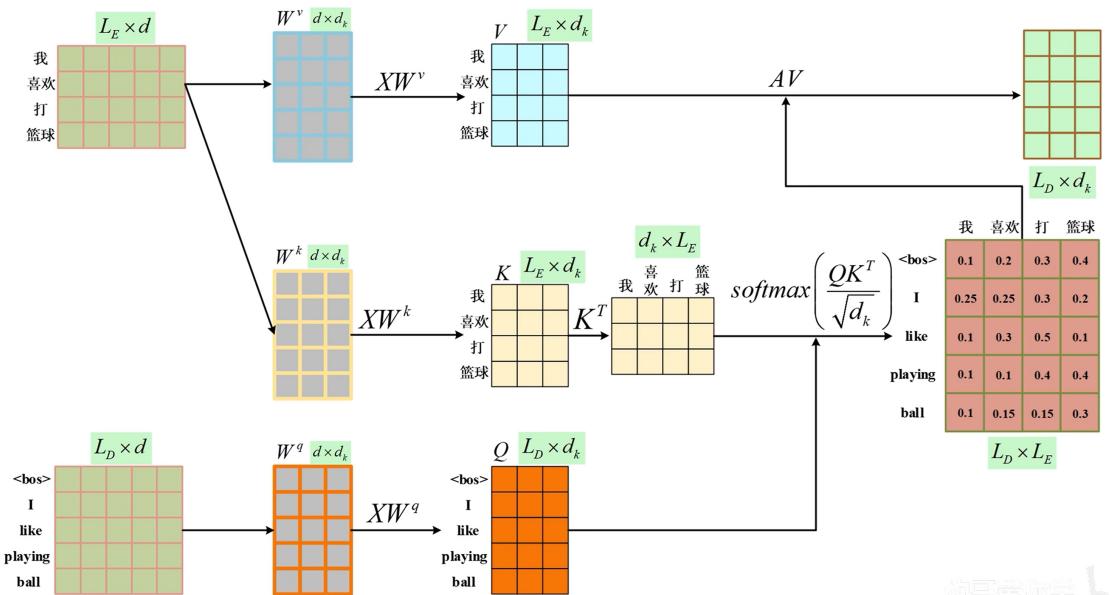
### 交叉注意力机制



## 交叉注意力机制



## 交叉注意力机制（矩阵表达方式）



## 归一化

- 归一化目的是把数据拉回标准正态分布。

假设一个 Batch 的输入数据张量为  $X$ , 其维度为  $(N, L, D)$  (在 Transformer 语境下):

- $N$  (Batch Size): 样本数量。
- $L$  (Sequence Length): 序列长度 (词数)。
- $D$  (Embedding Dimension): 特征维度 (隐藏层大小)。

通用的归一化公式为:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y = \gamma \hat{x} + \beta$$

其中  $\mu$  是均值,  $\sigma^2$  是方差,  $\epsilon$  是防止分母为 0 的微小常数,  $\gamma$  和  $\beta$  是可学习的仿射变换参数 (用于恢复模型的表达能力)。

- BN和LN
  - BN假设同一特征通道在不同样本间具有可比性, 在Batch维度(N)上进行统计
  - LN假设单个样本内部的特征分布具有统计意义, 在特征维度(D)上进行统计

## 层归一化(Layer Normalization)-->RNN、Transformer

### 层归一化 (Layer Normalization)

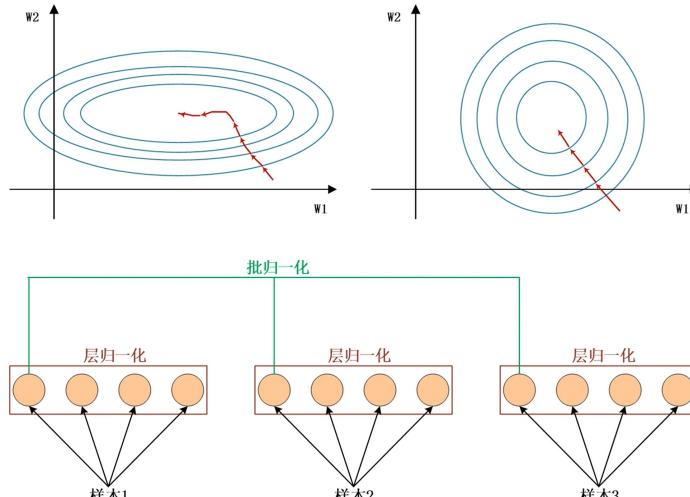
归一化对象: 每个样本的每一层 (特征维度) 进行归一化。

计算方式: 对单个样本的每一层的所有神经元进行归一化。

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$

$\mu$  和  $\sigma$  是当前样本的均值和方差。  
 $\gamma$  和  $\beta$  是学习的缩放因子和偏置。

适用场景: 通常用于 RNN、Transformer 等序列模型中, 因为这些模型处理的是变长的序列, 而批归一化在处理变长序列时会受到限制。



## 批量归一化(Batch Normalization)-->CNN

## 2. 为什么 Transformer 采用 LN 而非 BN?

这涉及到自然语言处理 (NLP) 数据的固有特性。

### A. 序列长度的可变性 (Variable Sequence Length)

- BN 的局限性:** NLP 任务中, 不同 Batch 的序列长度  $L$  往往不同。如果使用 BN, 对于长句子的末尾部分, 很多 Batch 可能因为 Padding (填充) 而只剩下极少数有效样本, 甚至全为 0。这会导致计算出的均值和方差产生极大的统计偏差 (抖动), 破坏模型的训练稳定性。
- LN 的适应性:** LN 在单个 Token 的  $D$  维度上计算, 不依赖序列长度, 也不受 Padding 的影响。

### B. Batch Size 的约束

- BN 的局限性:** BN 的有效性依赖于较大的 Batch Size 以逼近真实的数据分布。Transformer 模型通常参数量巨大 (LLM), 受显存限制, 训练时的 Batch Size 往往很小 (有时甚至为 1 或 2)。此时 BN 计算出的统计量具有极高的噪声, 会导致模型无法收敛。
- LN 的适应性:** LN 的计算仅依赖单条数据, 与 Batch Size 无关, 在 Batch Size = 1 时依然能稳定工作。

### C. 特征维度的物理意义

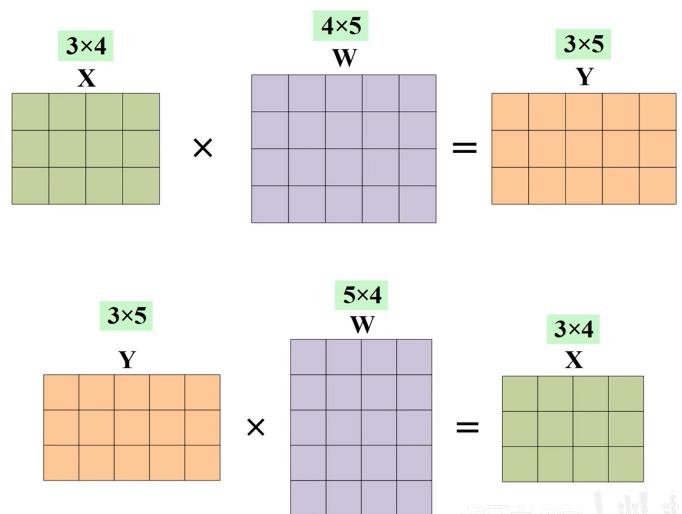
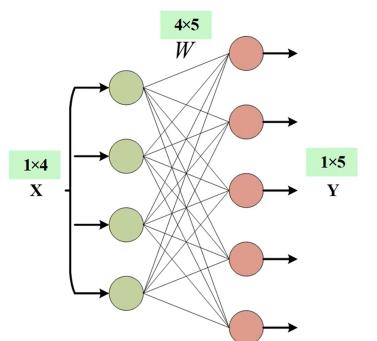
- BN 的假设:** BN 适用于 CNN, 因为图像不同通道 (Feature Map) 之间特征差异大 (如边缘、纹理), 但在整个 Batch 上同一通道的分布相对一致。
- LN 的假设:** 在 NLP 中, 词向量 (Embedding) 的不同维度共同构成了一个词的语义空间。对这组特征向量进行归一化 (使其模长标准化), 类似于对词向量进行缩放, 这有助于点积注意力机制 (Dot-Product Attention) 的数值稳定性。

## 前馈神经网络

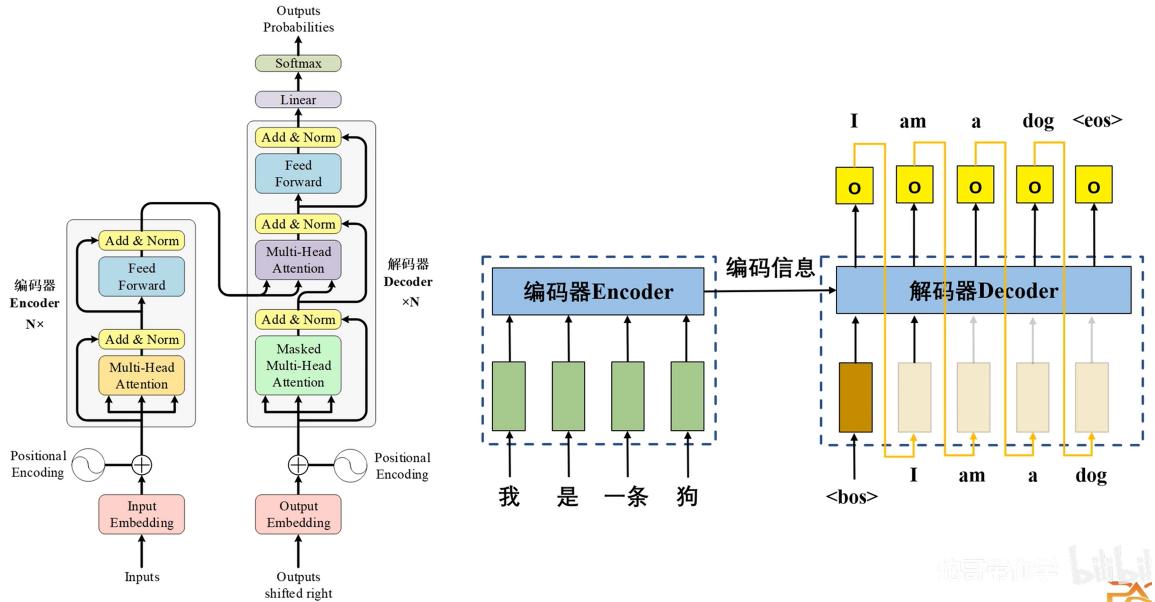
### 前馈神经网络定义:

前馈神经网络是一种无循环的多层次神经网络, 信息从输入到输出单向流动。

“前馈神经网络”是一种结构类型而不是特定网络。



## Transformer推理过程



## Transformer训练过程

