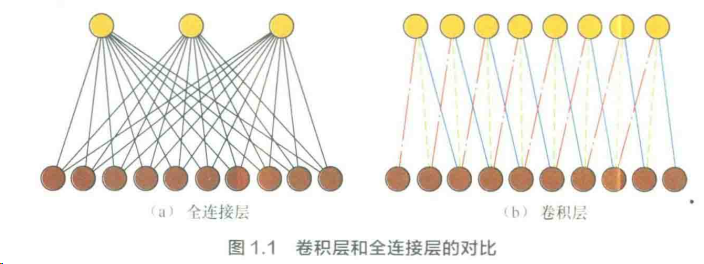
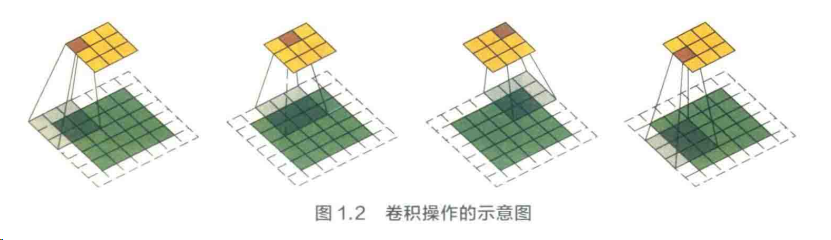
深度学习

* 简述卷积的基本操作，并分析其与全连接层的区别：

区别：卷积神经网络主要由卷积层构成，它具有局部连接和权值共享的特性。

基本操作：通过特定数目的卷积核对输入的多通道特征图进行扫描和运算，从而得到多个拥有更高层语义信息的输出特征图（通道数等于卷积核个数）。

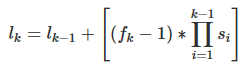


局部连接：输出层上的每个节点都只与输入层的部分节点连接；

权值共享：卷积核的滑动窗机制使得输出层上不同位置的节点与输入层的连接权值都是一样的；

输入/输出数据的结构化：卷积操作能够再输出数据中保持输入数据的结构信息，例如二维图像中不同节点仍然保持着与原始图像基本一致的空间对应关系；而如果将二维图像输入进全连接层，其输出数据会被展平成一维数组，从而丧失输入数据和输出数据在结构上的对应关系；

* 如何计算各层的感受野：



F为卷积核大小，s为步长，l为感受野。

* 卷积层的输出尺寸、参数量和计算量：

Feature map大小计算公式：N=(W-F+2P)/S+1（如果S>1时，可能出现非整数情况，许多深度学习框架采取向下取整的方法，放弃输入特征图的一部分边界数据，一般向下取整），F为卷积核大小，P为padding大小，S为步长；

卷积层参数量：params=channels\_out\*(kernel\*kernel\*channels\_in)+bias，其中bias=channels\_out;

池化层参数量：池化主要用于特征降维，压缩数据和参数的数量，减小过拟合，同时提高模型的容错性。池化不需要参数控制；

全连接层参数计算：可以看作kernerl size为特征图大小的卷积层；

卷积层的计算量：channel\_in\*channel\_out\*kernel\*kernel\*N\_w\*N\_h

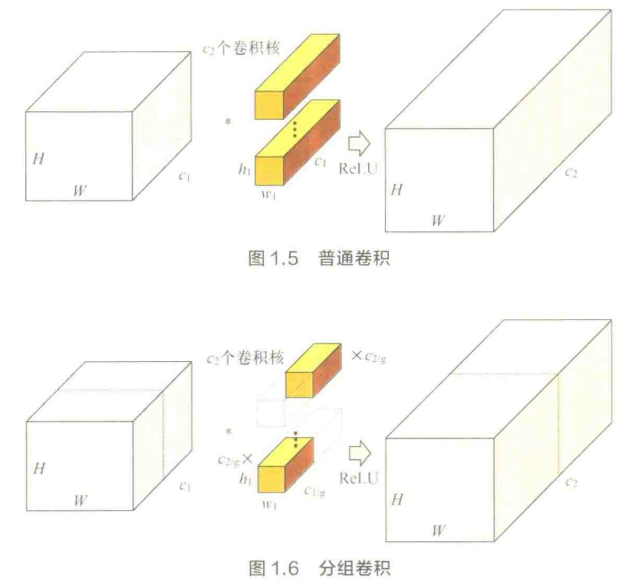
减小网络参数应主要针对全连接层，进行计算量优化时，重点应放在卷积层；

* 分组卷积

普通卷积的卷积核通道数等于输入特征图的通道数，因此最终输出特征图的每个通道都与输入特征图的所有通道相连接，因此最终输出特征图的每个通道都与输入特征图的所有通道相连接，在通道维度上是全连接的。

分组卷积就是将输入通道和输出通道都划分为相同的组数，然后仅让处于相同组号的输入通道和输出通道进行全连接。如果g为输入/输出通道所分的组数，则分组卷积能够将卷积操作的参数量和计算量都降低为普通卷积的1/g；

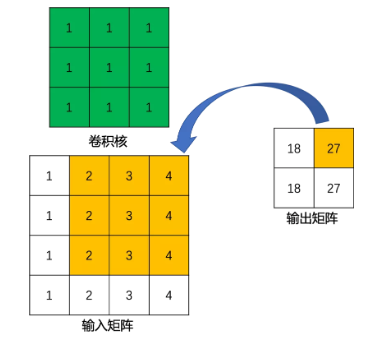
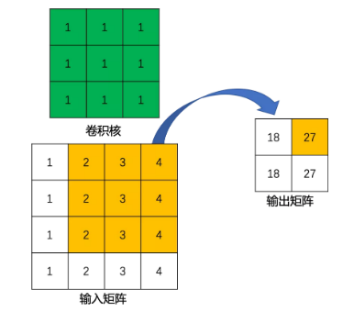
因此分组卷积主要被用来构建用于移动设备的小型网络模型；



* 转置卷积（反卷积）

标准卷积的运算操作 其实是对卷积核中的元素 与输入矩阵上对应位置的元素 进行逐像素的乘积并求和。然后，卷积核在输入矩阵上以步长为单位进行滑动，直到遍历完输入矩阵的所有位置。

对转置卷积而言，我们实际上是想建立一个逆向操作，即 **一对多的映射关系**。



标准卷积 转置卷积

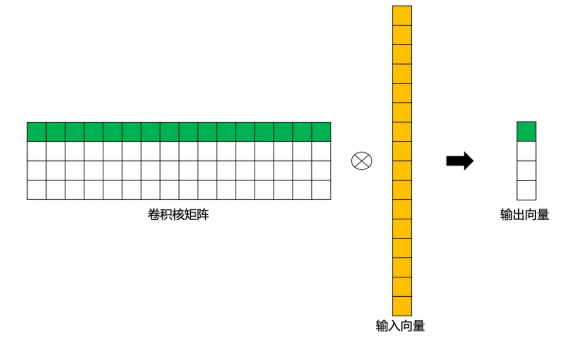
推导过程：

定义一个**4×4 输入矩阵 input**：

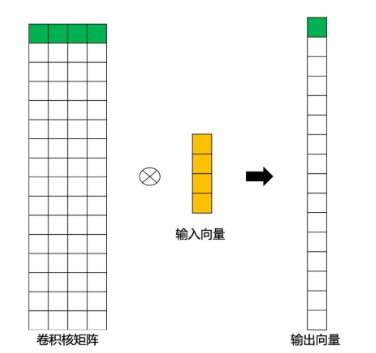
再定义一个 **3×3 标准卷积核 kernel**：

**将输入矩阵 input 和输出矩阵 output 展开成 16×1 列向量 X 和 4×1 列向量 Y，可分别表示为：**

经推到，可得到4×16的稀疏矩阵C：



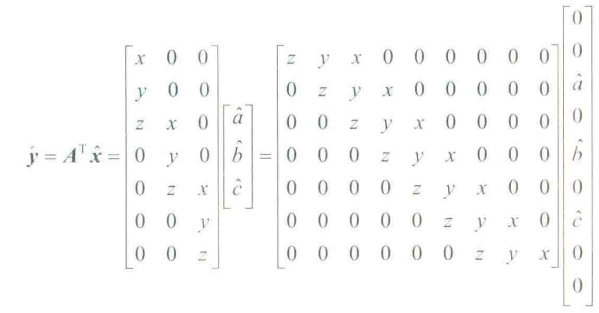
而转置卷积其实就是要对这个过程进行逆运算，即 **通过 C 和 Y 得到 X**：



转置卷积能将普通卷积中输入到输出的尺寸变换回来，但这只是回复了形状，并不能复原具体的取值。

转置卷积的信息正向传播与普通卷积的误差反向传播所用到的矩阵相同。

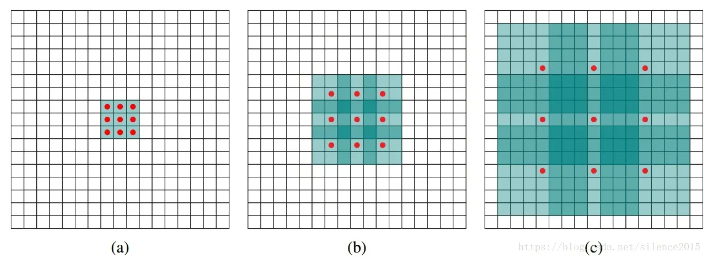
因此转置卷积本质上是对一个输入数据进行适当的变换（补零/上采样）的普通卷积操作。



普通卷积主要用来做特征提取，倾向于压缩特征图尺寸；转置卷积用于对特征图进行扩张或者上采样：语义分割/实例分割，物体检测（热力图），AE，VAE，GAN

* 空洞卷积

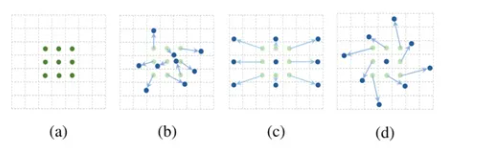
在语义分割任务中，一般采用池化操作来扩大特征图的感受野，但这同时会降低特征图的分辨率，丢失一些信息，导致后续的上采样无法还原一些细节，从而限制最终分割精度。



空洞卷积就是在卷积核中引入扩张率这个超参数，扩张率为r的空洞卷积，卷积核上相邻的数据点之间有r-1个空洞，如上图中扩张率分别为1，2，4；

* 可变性卷积要解决的问题

普通卷积操作是在固定的，规则的网格点上进行数据采样，这束缚了网络的感受野形状，限制了网络对几何形变的适应能力。为了克服这个限制，可变性卷积在卷积核的每个采样点上添加一个可学习的偏移量，让采样点不再局限于规则的网格点。



（c）和（d）是可变性卷积的两个特例，一个体现了可变性卷积的尺度变换特性，后者则是对卷积核进行旋转。

实际上，空洞卷积可以看作一种特殊的可变性卷积。

* ResNet

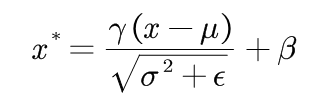
ResNet的提出源于这一种现象：随着网络层数的加深，网络的训练误差和测试误差都会上升，这种现象称为网络的退化。它采用跳跃连接，缩短误差反向传播到各层的路径，有效抑制梯度消失的现象；由于有近道的存在，若在网络层数加深时性能退化，则它可以通过控制近道退化到浅层时的连接状态。

* 批归一化

在机器学习中一般会假设模型的输入数据的分布时稳定的。如果模型输入数据分布发生变换，则成为协变量偏移。由于深度模型前一层的输出与网络参数的取值有很大关系，在训练过程中，如果之前层的参数被更新后，该层的输入数据的分布必然也跟着变化。网络越深这种现象越明显。

这会影响学习效率，并使得学习过程变得不稳定；网络前几层的参数更新很可能使后几层的输入数据变得过大或过小，从而掉进激活函数的饱和区，导致学习过程过早停止；为了减小这种影响会采用较小的学习率，降低收敛速度；

批归一化的作用就是确保网络中的各层，即使参数发生了变化，输入/输出的数据分布也不能产生较大的变化；采用批归一化后，模型的训练过程更加稳定，对初始值不再那么敏感，可以采用更大的学习率来加速收敛。



在批归一化过程中放置了两个可学习参数，他们有如下作用：

保留网络各层在训练过程中的学习成果，如果没有这两个参数，批归一化会退化为普通的标准化，使得输出的分布几乎不变，不能有效地进行学习；

保证激活单元的非线性表达能力，如果采用Sigmoid和Tanh等函数，则经过标准化的数据会落在激活函数的近似线性区域，没利用上他们的非线性区域；

批归一化具有自我关闭能力，可以回复到原始的输入值，当批归一化导致特征分布被破环，网络泛化性能减弱时，可以通过这两个参数将其关闭；

批归一化放置的位置：

在激活层之前，可以有效避免批归一化破坏非线性特征的分布，可以使数据点尽量不落入激活函数的饱和区域，缓解梯度消失问题；

由于ReLU没有那些问题，因此也可以把BN放在激活层之后，避免在激活层之前被转换成相似的模式从而使得非线性特征分布趋于同化；

* 用于分类的CNN最后几层一般是什么层？

网络末端一般是几层全连接层，这是因为卷积层具有局部连接，权值共享的特性，其在不同位置是采用相同的卷积核进行特征提取的，也就是说卷积层的特征是局部的且位置不敏感的。因此，全卷积网络也许能知道图片中不同位置有哪些元素，但无法提取这些元素之间的关联关系。而分类任务要考虑它们之间的关联关系，需要一个全局的、位置敏感的特征提取器。

近几年，分类网络在卷积层之后，最后一层之前通常采用全局平局池化，它与全连接层有着相似的效果（可以提取全局信息），并且减小参数量和计算量，具有较好的可解释新，可以知道特征图上哪些点对最后的分类贡献大。**对整个网路在结构上做正则化防止过拟合。**其直接剔除了全连接层中黑箱的特征，直接赋予了每个channel实际的内别意义。

* 瓶颈结构

瓶颈结构的初衷是为了降低大卷积层的计算量，即在计算比较大的卷积层之前，先用1×1的卷积核来压缩大卷积层输出特征的通道数，再采用一个1×1的卷积核提升通道数（可选）。瓶颈结构可以用更小的计算代价达到与之前相似甚至更好的效果。

* 沙漏结构：

一般包含两个分支：自底向上（bottom-up）分支，利用卷积池化压缩特征图尺寸，类似encoder；自顶向下（top-down）分支，利用反卷积或插值将特征图的尺寸增大，类似decoder。沙漏结构一般是将多尺度信息进行融合，同时，沙漏结构单元中的多个卷积层可以提升感受野，增强模型对小尺寸但又依赖上下文的物体的感知能力。

* Dropout为什么可以缓解过拟合

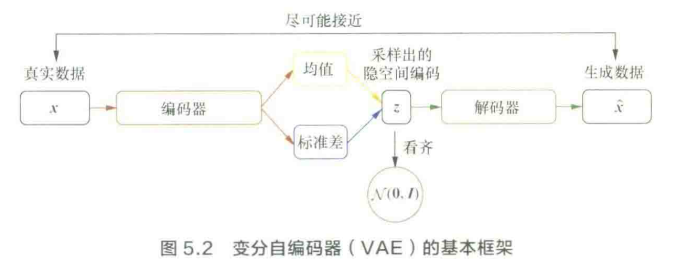
Dropout是指在网络训练阶段，每次迭代时会从基础网络中随机丢弃一定比例的神经元。一方面，Dropout可以看作是集成了大量神经网络的Bagging方法（相同数据训练若干个不同的模型，最终预测结果是这些模型投票得到的）。另一方面，Dropout可以减少神经元之间复杂的共适应关系，网络中的每个神经元不会对另一个特定的神经元的激活非常敏感。

* AE,VAE,GAN的联系与区别：

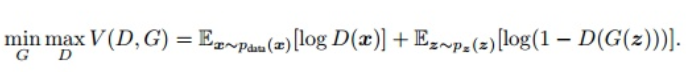
AE，VAE，GAN同属于可微生成网络，他们常常用神经网络来表示一个可谓函数G，用这个函数来刻画隐变量z到样本分布的映射关系。

AE：由encoder和decoder组成，首先将真实数据x压缩为低维隐空间中的一个向量z，然后decoder将z解压，得到生成样本。最终目的是期望由真是样本x得到的隐向量能够尽可能地抓住输入的精髓，使重建的数据尽可能接近真实样本。可应用于数据去噪，可视化降维以及数据生成等方向。

VAE：的主要优势在于能够产生新的隐向量z，进而生成有效的新样本。VAE能够生成新样本的原因是，VAE在编码过程中加入了一些限制，迫使编码器产生的隐向量的后验分布q(z|x)尽量接近某个特定的分布。VAE的编码器输出的不再是隐向量，而是所属正态分布的均值和标准差，然后再根据均值和标准差来采样出隐向量z。由于采样操作存在随机性，因此生成的图像不再是唯一的。



GAN：解决的是生成模型的一大难点，如果度量生成分布与真实分布的相似度。GAN把度量任务交给一个神经网络来做，这个网络称为判别器。



AE与VAE的联系与区别：都属于有向图模型，模型的目的都是对隐变量空间进行建模，但AE只会模仿不会创造。AE的优化目标是最小化真实样本与生成样本的重构误差，VAE除了考虑重构误差，还加入了对隐变量空间的约束目标。

GAN与AE/VAE的区别：VAE的重构损失目的和迫使隐变量后验分布接近正态分布的这两种损失相互对立，这与GAN一样，他们内部都存在对抗思想，只不过VAE是将两部分同步优化，而GAN是交替优化的。与AE相同，GAN的优化目标只涉及到生成样本与真实样本之间的比较，没有VAE中对后验分布的约束，但是GAN利用对抗训练绕过了对分布间距离的度量，且判断样本真假时不需要真实样本与生成样本一一对应。GAN没有像AE那样从学习后验分布q(z|x)中获得生成样本的能力，所以存在模式坍塌，训练不稳定的问题。此外，AE/VAE可以通过对数据的操控实现数据在语义层次上的操控。

* 物体检测中单步模型和两步模型的性能差异：

单步模型是指没有独立的，显示地提取候选区域（region proposal），直接由输入图像得到其中存在的物体的类别和位置信息的模型，典型的单步模型有SSD，YOLO。

两步模型先在输入图像上筛选出一些可能存在物体的候选区域，然后针对每个候选区域判断是否存在物体，如果存在，就给出物体的类别和位置修正信息。典型的两步模型有R-CNN,Fast R-CNN,Faster R-CNN, Mask R-CNN。

单步模型在计算效率上有优势，两步模型在检测精度上有优势。多数单步模型是利用预设的Anchor Box来捕捉存在于图像中各个位置的物体，因此单步模型会对数量庞大的Anchor Box进行分类，由于一幅图像中实际含有的物体数目远小于Anchor Box的数量，因此在训练这个分类器的时候正负样本是极不均衡的，导致分类器训练效果不佳。

两步模型在候选区域提取的过程中会对候选框位置和大小进行修正，因此在进入第二部之前，候选区域的特征已被对齐，这样有利于第二步的分类提供质量更高的特征。另外两步模型在第二步中会再次修正候选框，这带来了更高的定位精度，但增加了模型复杂度。但不模型没有候选框提取过程，Anchor Box的预测基于该层上每个特征点的感受野，输入特征未被对齐，质量较差，因而精度较低。

两步模型在第二步对候选区域进行分类和位置回归时，是针对每个候选区域独立进行的，因此该部分的算法复杂度线性正比于预设的候选区域数目。

* 有哪些措施可以增强模型对小物体的检测效果：

在模型设计方面，可以采用特征金字塔，沙漏结构等特征子结构，增强网络对小尺度特征的感知和处理能力，尽可能提高网络的感受野，使得网络能够利用更多的上下文信息，同时减少网络总的下采样比，使得最后用于检测的特征分辨率更高；

在训练方面，可以提高小物体样本在总体样本中的比例，或者利用数据增强手段，将图像缩小以生成小物体样本；

在计算允许的范围内，可以尝试使用更大的输入图像尺寸；

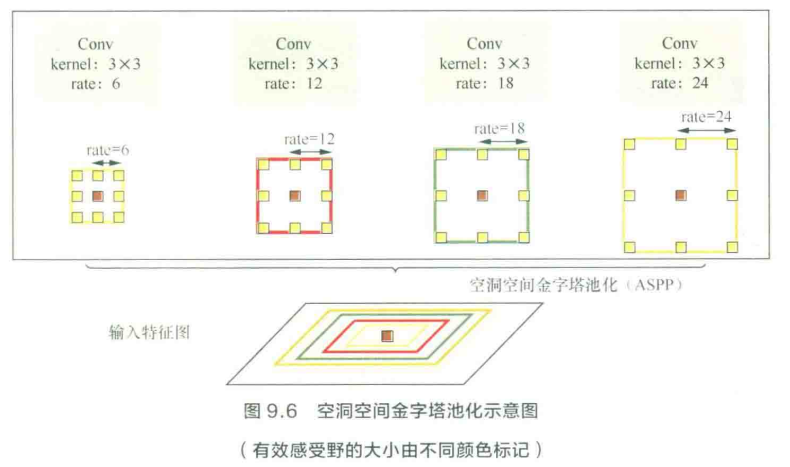
* 编码器解码器结构的设计

图像分割中的编码器可视为特征提取网络，通常使用池化层来逐渐缩减输入数据的空间维度；而解码器则通过上采样/反卷积等网络层来逐步恢复目标的细节和相应的空间维度。由于引入池化层可以增加后续卷积层的感受野，并能使特征提取聚焦在重要信息中，降低背景干扰。然而池化操作使位置信息大量流失，不足以对像素进行精确的分割，因此提出建立快捷连接，使高分辨率的特征信息参与到后续的解码环节。

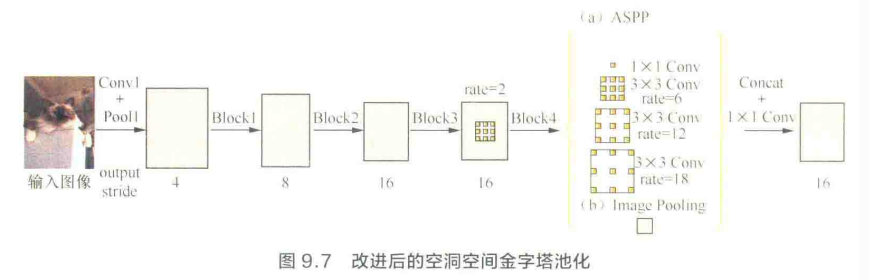
* DeepLab

DeepLab v1的创新分别是空洞卷积和全连接条件随机场。空洞卷积是为了解决编码过程中信号不断被下采样，细节丢失的问题。由于卷积层提取的特征具有平移不变性，这就限制了定位精度，所以引入了全连接条件随机场来提高模型捕获局部结构信息的能力。

DeepLab v2提出了空洞空间金字塔池化（ASPP），并将VGG网络替换成了更深的ResNet网络。ASPP可用于解决不同检测目标大小差异的问题：通过在给定的特征层上使用不同扩张率的空洞卷积，ASPP可以有效地进行重采样，最后将ASPP各个空洞卷积分支采样后地结果融合到一起。

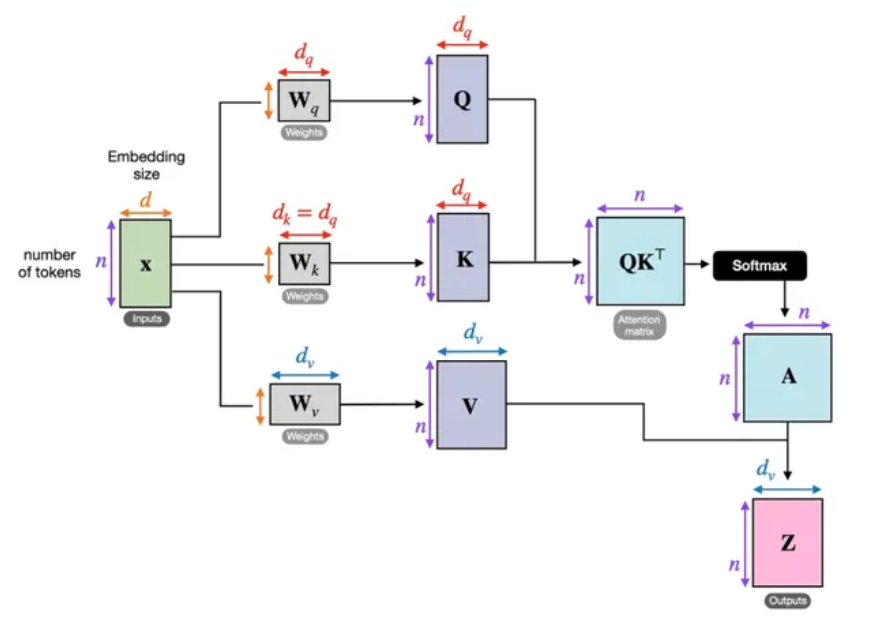


DeepLab v3：加入了批归一化（BN）层；其次将ASPP中尺寸为3×3，空洞大小为24地卷积替换成一个普通的1×1卷积，以保留滤波器中间部分的有效权重。因为，随着空洞卷积扩张率的增大，滤波器中有效权重的个数在减小，为了克服长距离下有效权重减小的问题，在ASPP最后增加了全局平均池化以便更好地捕捉全图信息。



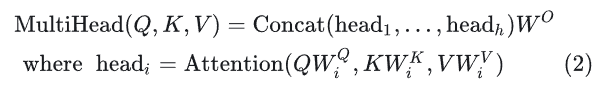
Transformer

* Self-Attention机制



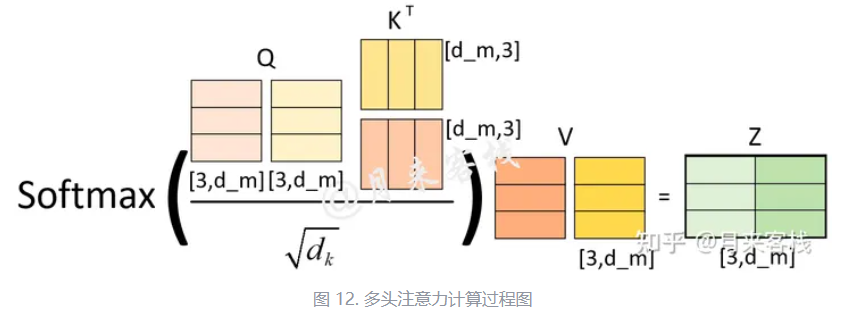
* Transformer为何使用多头注意力机制：**多头保证了transformer可以注意到不同子空间的信息，捕捉到更加丰富的特征信息。**

所谓的多头注意力机制其实就是将原始的输入序列进行多组的自注意力处理过程；然后再将每一组自注意力的结果拼接起来进行一次线性变换得到最终的输出结果。具体的，其计算公式为：

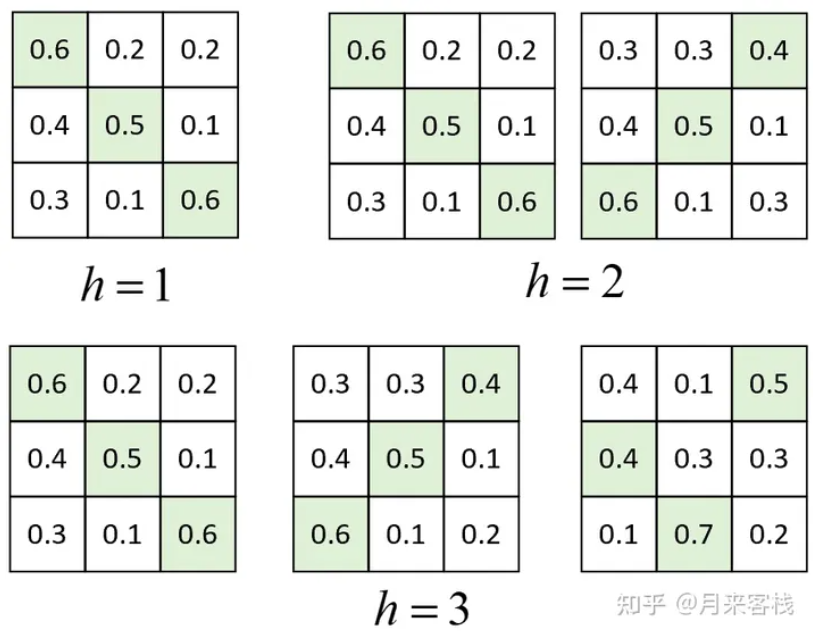




自注意力机制的缺陷就是：模型在对当前位置的信息进行编码时，会过度的将注意力集中于自身的位置，因此作者提出了通过多头注意力机制来解决这一问题。同时，使用多头注意力机制还能够给予注意力层的输出包含有不同子空间中的编码表示信息，从而增强模型的表达能力。



在dm固定的情况下，不管是使用单头还是多头的方式，在实际的处理过程中直到进行注意力权重矩阵计算前，两者之前没有任何区别。当进行进行注意力权重矩阵计算时，h越大那么Q,K,V就会被切分得越小，进而得到的注意力权重分配方式越多，



* Transformer为什么Q和K使用不同的权重矩阵生成，为何不能使用同一个值进行自身的点乘？

点乘的物理意义，两个向量的点乘表示两个向量的相似度。K和Q的点乘是为了计算一个句子中每个token相对于句子中其他token的相似度，这个相似度可以理解为attetnion score，关注度得分。经过与attention score相乘后，V中每个token的向量，在每一列上，都会对其他token做出调整（关注度不同）。与V相乘这一步，相当于提纯，让每个单词关注该关注的部分。

但是直接拿K和K点乘的话，你会发现attention score 矩阵是一个对称矩阵。因为是同样一个矩阵，都投影到了同样一个空间，所以泛化能力很差。

**使用Q/K/V不相同可以保证在不同空间进行投影，增强了表达能力，提高了泛化能力。**

* Transformer计算attention的时候为何选择点乘而不是加法？两者计算复杂度和效果上有什么区别

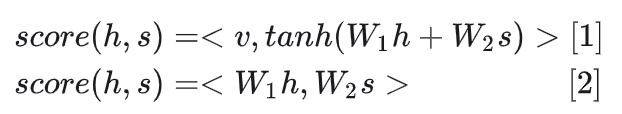
**计算效率：** 点乘（也称为内积）计算的复杂度较低。在点乘的注意力机制中，将查询向量和键向量进行点乘操作，然后对结果进行缩放（通过除以sqrt(d\_k)）以控制梯度的稳定性，最后通过softmax函数进行归一化。而加法注意力机制需要将查询向量和键向量相加，然后再进行softmax归一化。相比之下，点乘的计算量更小，尤其是在硬件加速器如GPU上更加高效。

**梯度传播：** 点乘的归一化（除以sqrt(d\_k)）可以控制梯度的大小，使得梯度更加稳定地传播。这有助于避免在深度网络中出现梯度爆炸或梯度消失的问题。

**模型性能：** 实践中发现，点乘的注意力机制在大多数情况下能够取得良好的效果，特别是在自然语言处理任务中，如机器翻译、文本生成等。

* 为什么在进行softmax之前需要对attention进行scaled（为什么除以dk的平方根）

Self-attention的两种基本形式：



<>代表矩阵点积，使用点乘的计算速度更快，整体计算复杂度二者相当，但是点乘有更多的硬件加速。随着attention dim的增大，加法形式的self-attention的性能强于点乘。点乘的性能更弱可能是由于极大的点积值将整个softmax推向梯度平缓区，使得收敛困难，导致梯度消失。如果s和h分布在[0,1]上，那么经过点乘之后整体的分布会在[0,dk]上。而除以根号dk是为了让QK的分布方差纠正回1。

* 在计算attention score的时候如何对padding做mask操作？

将padding位置取值为负无穷（如-1000），softmax的函数会导致为padding的值占全局一定概率，mask就是让这部分值取无穷小，让他在softmax之后基本也为0，不去影响非attention socore的分布。

* 为什么在进行多头注意力的时候需要对每个head进行降维？

计算效率： 注意力机制涉及对所有序列位置进行加权计算，这可能需要大量的计算资源，尤其是在深度学习模型中，如Transformer。通过降维，可以减少每个头的计算复杂度，从而提高整体模型的计算效率。

参数数量： 每个注意力头都具有自己的参数矩阵，降维可以减少每个头中参数的数量。这对于减少模型的参数量以及控制模型的过拟合有帮助。

更好的特征表达： 通过降维，可以将高维特征空间转换为低维空间，从而可能提取出更加抽象和有意义的特征表示。这有助于模型更好地学习输入序列之间的关系和语义信息。

减少过拟合： 随着模型深度的增加，过拟合成为一个常见的问题。通过降维，可以降低每个头的自由度，减少过拟合的风险。

更好的模型解释性： 降维可以使得每个头所关注的信息更加清晰，有助于理解模型在处理输入时的决策过程。

* MSA的代码

**import** torch

**import** torch.nn **as** nn

**import** torch.nn.functional **as** F

**import** math

**class** **MultiHeadAttention**(nn**.**Module):

**def** \_\_init\_\_(self, d\_model, num\_heads):

super(MultiHeadAttention, self)**.**\_\_init\_\_()

self**.**num\_heads **=** num\_heads

self**.**d\_model **=** d\_model

**assert** d\_model **%** self**.**num\_heads **==** 0

*# Define the dimension of each head or subspace*

self**.**d\_k **=** d\_model **//** self**.**num\_heads

*# These are still of dimension d\_model. They will be split into number of heads*

self**.**W\_q **=** nn**.**Linear(d\_model, d\_model)

self**.**W\_k **=** nn**.**Linear(d\_model, d\_model)

self**.**W\_v **=** nn**.**Linear(d\_model, d\_model)

*# Outputs of all sub-layers need to be of dimension d\_model*

self**.**W\_o **=** nn**.**Linear(d\_model, d\_model)

**def** **scaled\_dot\_product\_attention**(self, Q, K, V, mask**=**None):

batch\_size **=** Q**.**size(0)

K\_length **=** K**.**size(**-**2)

*# Scaling by d\_k so that the soft(arg)max doesn't explode*

QK **=** torch**.**matmul(Q, K**.**transpose(**-**2, **-**1)) **/** math**.**sqrt(self**.**d\_k)

*# Apply the mask*

**if** mask **is** **not** None:

QK **=** QK**.**masked\_fill(mask**.**to(QK**.**dtype) **==** 0, float('-inf'))

*# Calculate the attention weights (softmax over the last dimension)*

weights **=** F**.**softmax(QK, dim**=-**1)

*# Apply the self attention to the values*

attention **=** torch**.**matmul(weights, V)

**return** attention, weights

**def** **split\_heads**(self, x, batch\_size):

"""

The original tensor with dimension batch\_size \* seq\_length \* d\_model is split into num\_heads

so we now have batch\_size \* num\_heads \* seq\_length \* d\_k

"""

**return** x**.**view(batch\_size, **-**1, self**.**num\_heads, self**.**d\_k)**.**transpose(1, 2)

**def** **forward**(self, q, k, v, mask**=**None):

batch\_size **=** q**.**size(0)

*# linear layers*

q **=** self**.**W\_q(q)

k **=** self**.**W\_k(k)

v **=** self**.**W\_v(v)

*# split into multiple heads*

q **=** self**.**split\_heads(q, batch\_size)

k **=** self**.**split\_heads(k, batch\_size)

v **=** self**.**split\_heads(v, batch\_size)

*# self attention*

scores, weights **=** self**.**scaled\_dot\_product\_attention(q, k, v, mask)

*# concatenate heads*

concat **=** scores**.**transpose(1,2)**.**contiguous()**.**view(batch\_size, **-**1, self**.**d\_model)

*# final linear layer*

output **=** self**.**W\_o(concat)

**return** output, weights

* Transformer的Encoder模块
* Transformer的位置编码，有什么意义和优缺点？

(1)对于任何一门语言，单词在句子中的位置以及排列顺序是非常重要的，它们不仅是一个句子的语法结构的组成部分，更是表达语义的重要概念。一个单词在句子的位置或排列顺序不同，可能整个句子的意思就发生了偏差。(2)当抛弃循环神经网络结构，完全采用Attention取而代之，这些词序信息就会丢失，模型就没有办法知道每个词在句子中的相对和绝对的位置信息。因此，有必要把词序信号加到词向量上帮助模型学习这些信息，位置编码（Positional Encoding）就是用来解决这种问题的方法。

优点：**简单有效**： 位置编码是一种简单而有效的机制，可以直接与输入嵌入相加，不需要额外的学习参数。**不受序列长度限制**： 位置编码与序列长度无关，可以应用于任何长度的输入序列。**允许模型学习序列顺序信息**： 通过位置编码，Transformer 能够学习到输入序列中词语的相对位置信息，有助于理解序列中的语义和语法结构。

缺点：**固定的位置信息**： 位置编码假设了输入序列中词的位置信息是固定的，这可能不适用于一些上下文动态变化的任务。**位置信息可能不够准确**： 对于较长的序列，位置编码可能无法提供足够精确的位置信息，导致模型难以理解长距离依赖关系。**不适用于变长序列**： 对于变长序列，如机器翻译中的句子长度可能不同，位置编码的设计可能需要额外的处理。

* 还有哪些关于位置编码的技术，各自的优缺点是什么？

**绝对位置编码（Absolute Positional Encoding）：**

常用的绝对位置编码方法包括使用三角函数（如正弦和余弦函数）或者使用固定的矩阵来编码序列位置信息。

优点：简单易实现，不需要额外的学习参数。

缺点：固定的位置编码可能无法捕捉输入序列中的动态模式，尤其是在处理变长序列时可能效果不佳。对于较长的序列，可能无法提供足够精确的位置信息。

**相对位置编码（Relative Positional Encoding）：**

一种实现方法是通过在自注意力计算中引入相对位置偏移量，并根据相对位置偏移量来调整注意力分数，从而捕捉输入序列中词与词之间的相对位置关系。

优点：能够捕捉输入序列中词与词之间的相对位置关系，有助于理解长距离依赖关系。在处理变长序列时可能比绝对位置编码更灵活。

缺点：相对位置编码的设计和实现可能较复杂，需要额外的计算开销。

**学习式位置编码（Learned Positional Encoding）：**

学习式位置编码通常是通过在模型中引入额外的可学习参数来实现的，这些参数可以被模型端到端地训练。

优点：允许模型通过学习来适应不同任务和数据集的位置信息。可以灵活地调整位置编码的参数，以适应不同长度和结构的输入序列。

缺点：需要额外的学习参数，并且可能增加模型的训练和推理成本。学习式位置编码的性能高度依赖于训练数据的质量和规模。

* Transformer中的残差结构以及意义

在 Transformer 的每个子层（如自注意力层和前馈全连接层）中，都会使用残差连接。

**梯度传播：** 残差连接有助于缓解梯度消失或梯度爆炸问题。通过将输入直接传递到输出中，可以确保即使在非常深的网络中，梯度仍然能够从输出反向传播到输入，有助于更有效地训练深层网络。

**减轻训练难度：** 在深度神经网络中，通过将每个层的输出与其输入相加，可以确保网络的训练过程更加稳定，更容易收敛到较好的解决方案。

**网络深度：** 残差连接允许构建非常深的网络，而不会出现梯度消失的问题。这在 Transformer 中尤为重要，因为 Transformer 模型通常包含大量的层，而残差连接确保了每个层都能够有效地学习到输入和输出之间的映射关系。

**网络表示能力：** 残差连接使得网络能够更好地学习数据的高级表示。通过残差连接，网络可以逐层地学习数据的不同抽象层次的表示，从而提高了网络的表示能力。

* 为什么transformer块使用LayerNorm而不是BatchNorm？LayerNorm 在Transformer的位置是哪里？

任何norm的意义都是为了让使用norm的网络的输入的数据分布变得更好，也就是转换为标准正态分布，数值进入敏感度区间，以减缓梯度消失，从而更容易训练。当然，这也意味着舍弃了除此维度之外其他维度的其他信息。首先要明确，如果在一个维度内进行normalization，那么在这个维度内，相对大小有意义的，是可以比较的；但是在normalization后的不同的维度之间，相对大小这是没有意义的。

**独立样本的处理：** BatchNorm 是在一个批次内对每个特征维度进行标准化，因此依赖于批次的统计信息。然而，在 Transformer 模型中，每个位置的输入都可以看作是一个单独的样本，而不是批次中的样本。因此，BatchNorm 不适用于 Transformer，因为它无法有效地对每个位置的输入进行归一化。相比之下，LayerNorm 对每个位置的输入独立进行归一化，更适合 Transformer 的结构。

**稳定性：** LayerNorm 在不同批次和不同输入之间的行为更加稳定，而 BatchNorm 对批次大小和输入分布更敏感。这使得 LayerNorm 更适合应对不同长度和类型的输入序列，保证了模型的稳定性和泛化性能。

**模型并行性**： 在训练大型 Transformer 模型时，常常需要使用模型并行（Model Parallelism）来分布式处理模型的不同部分。LayerNorm 相对于 BatchNorm 更易于实现模型并行，因为每个位置的归一化操作可以独立进行，而不需要跨多个设备共享统计信息。

LN的位置：在 Transformer 中，LayerNorm 通常被应用在每个子层的输入和输出上，包括自注意力层（Self-Attention Layer）和前馈全连接层（Feedforward Layer）。具体来说，LayerNorm 会对每个位置的输入进行归一化，然后再应用残差连接。这样可以确保每个位置的输入都具有相似的分布特性，有助于模型的稳定训练和更好的收敛性能。

* BatchNorm技术，以及它的优缺点。
* Transformer中的前馈神经网络，使用了什么激活函数？
* Encoder端和Decoder端是如何进行交互的
* Decoder阶段的多头自注意力和encoder的多头自注意力有什么区别？
* Transformer的并行化提现在哪个地方？Decoder端可以做并行化吗？
* Transformer训练的时候学习率是如何设定的？Dropout是如何设定的，位置在哪里？Dropout 在测试的需要有什么需要注意的吗？