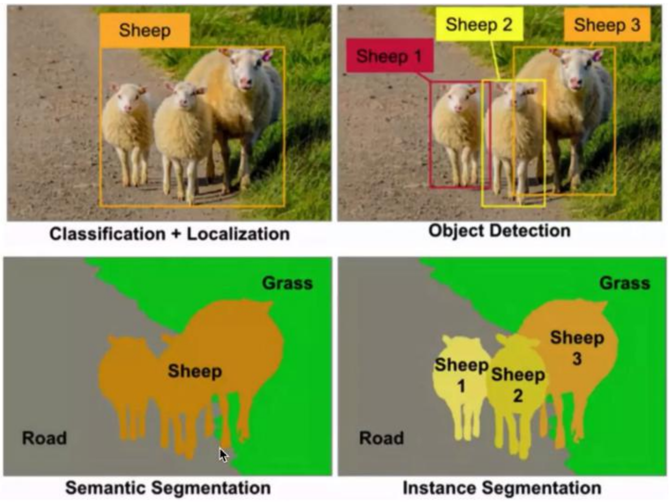
图像识别四大任务

分类、目标检测、语义分割、实例分割



分类：是不是羊

语义分割：哪些像素是羊

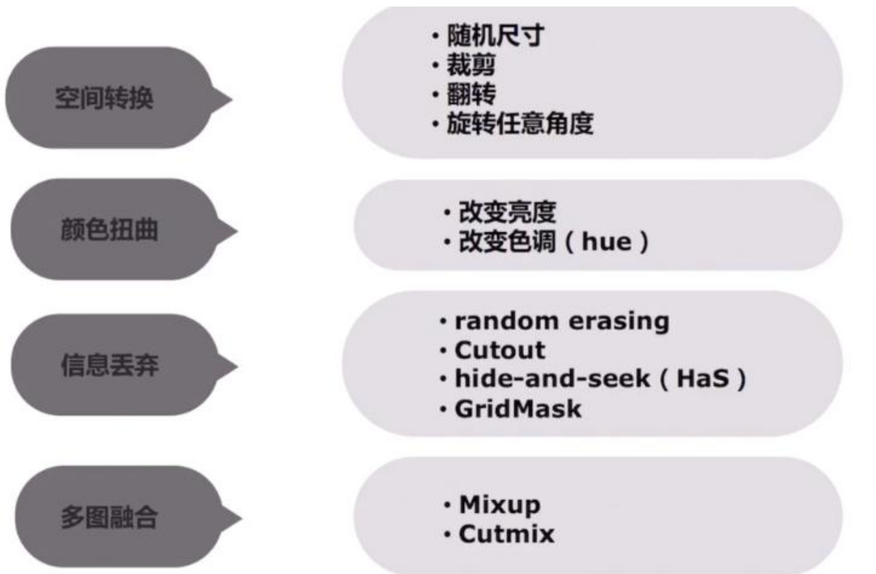
实例分割：哪些像素是哪只羊

目标检测：把羊框起来

数据处理

在送入网络学习以前，我们首先要对拿到的图片数据做一些基本的处理。其目的是为了让训练模型的数据分布尽量跟现实的数据分布情况一致。

数据增强



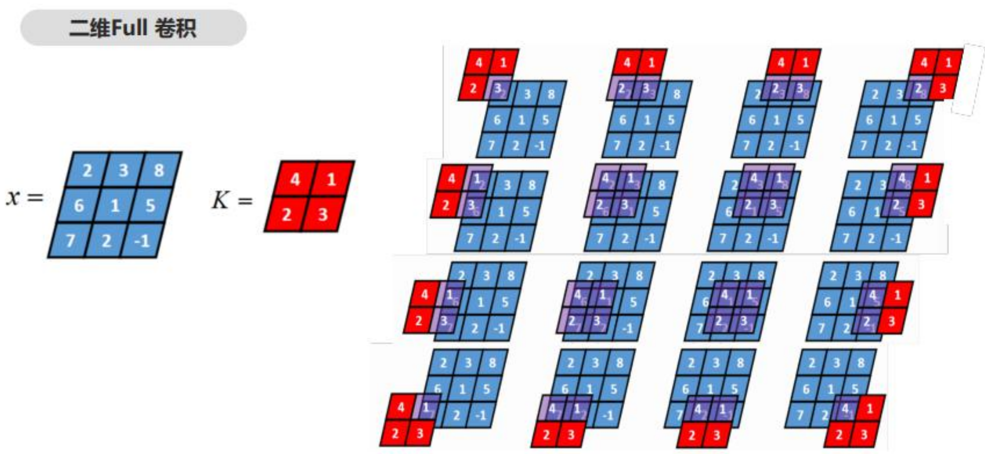
标签处理

我们需要对标签进行处理，按照学习目的的不同。比如分类，只需要标注图像所包含的目标是什么种类，牛还是羊。如果是语义识别，就需要对前景背景的像素做不同的标注。比如前景标注为 1，背景标注为 2。如果是实例分割，还需要对每一个实例的像素单独标注。比如第一只羊标注为 1，第二只羊标注为 2，等等。

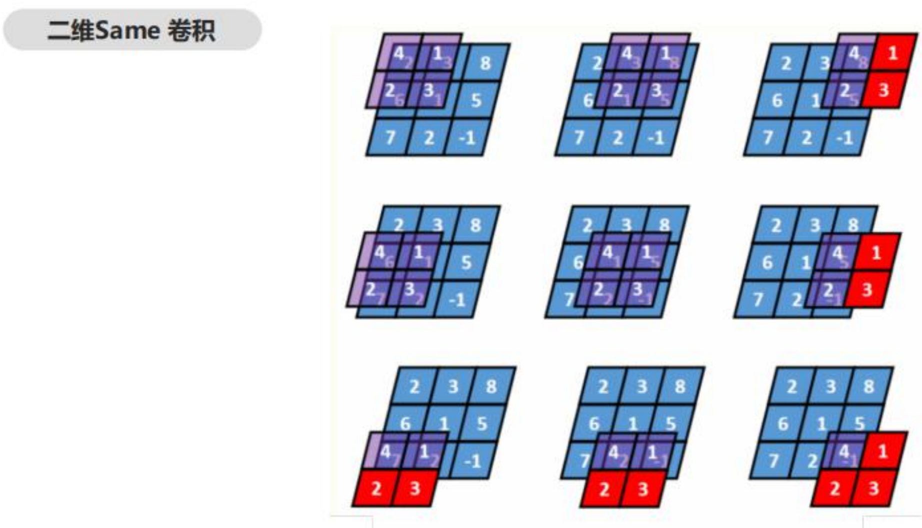
网络

卷积

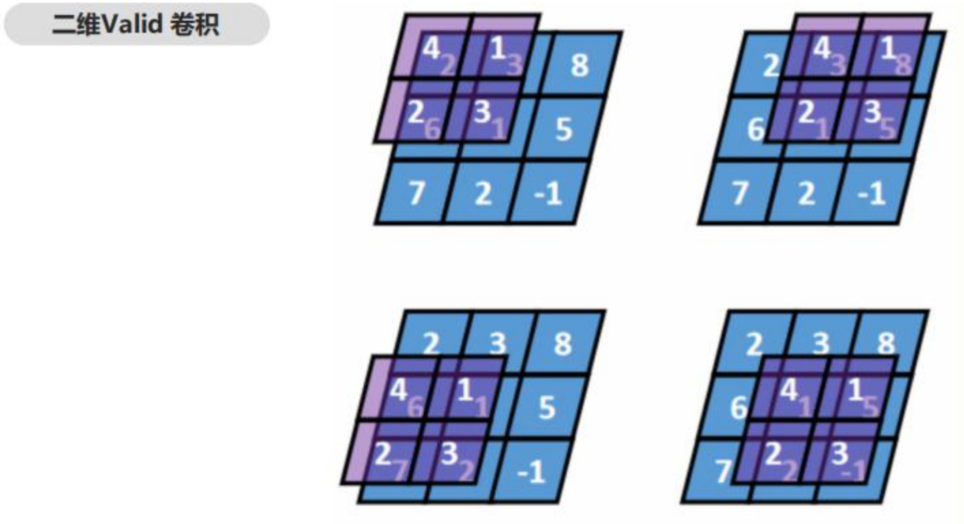
Full 卷积



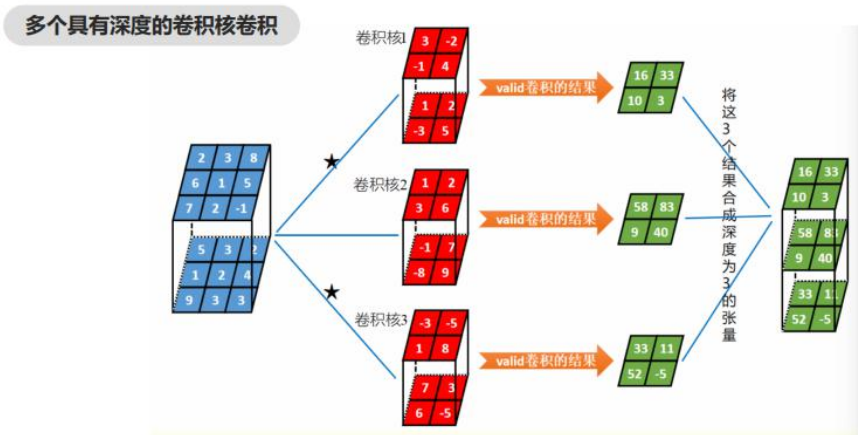
Same 卷积



Valid 卷积



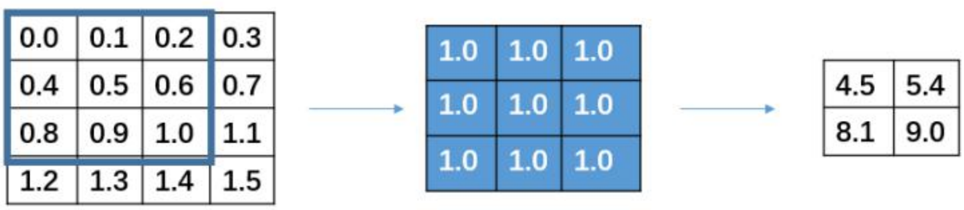
有深度的多卷积核卷积



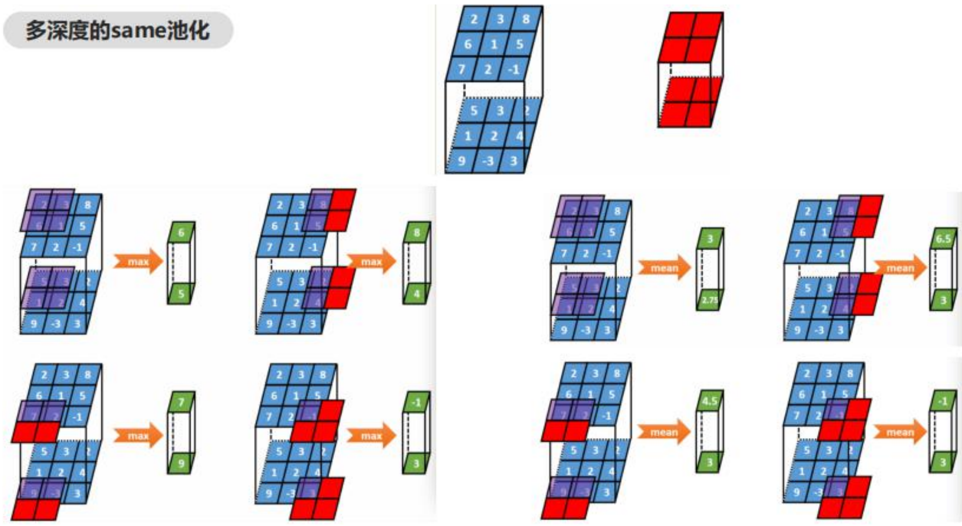
卷积的计算

- I (input) 输入的尺寸
- O (output) 输出的尺寸
- K (kernel) 卷积核的大小
- S (stride) 卷积核的步长
- P (padding) padding的大小

$$o = \frac{i - k + 2 * p}{s} + 1$$

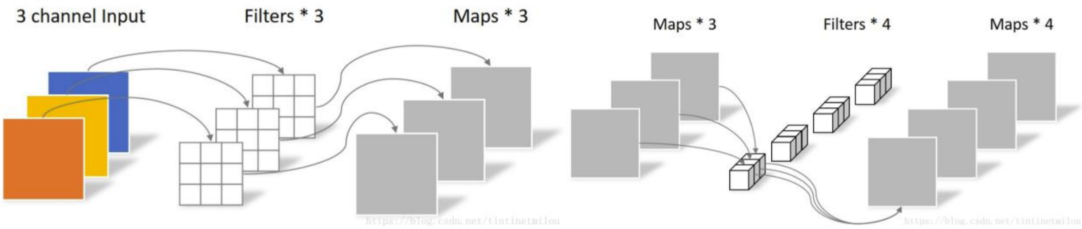


池化



分离卷积（separable 卷积）

首先 Depthwise 卷积-每个深度分别卷积，然后 Pointwise 卷积



将传统的卷积改为深度可分离卷积，首先用和输入的深度层数相同的个数的 Filter 对每一层进行卷积。再用 1*1*层数的 Filter 来卷积。主要作用是为了减少参数量。用于 MobileNet.

传统的卷积操作： VS **深度可分离卷积：**

传统的卷积操作：

操作前 feature size : H_D*W_D*M

Kernel size: H_K*W_K*M*N

卷积后 feature size: H_A*W_A*N

其中 $H_A=(H_D-H_K+2*Paddingstride) / stride +1$

W_A与 H_A类似

深度可分离卷积：

➢ Depthwise Convolution :

Kernal size： H_K*W_K*1*M

卷积后 feature size ： H_A*W_A*M

➢ Pointwise Convolution:

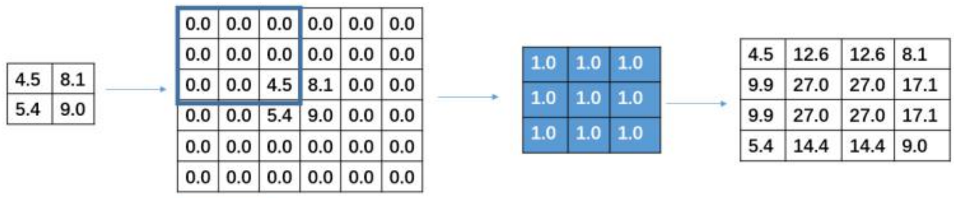
Kernal size： $1*1*M*N$

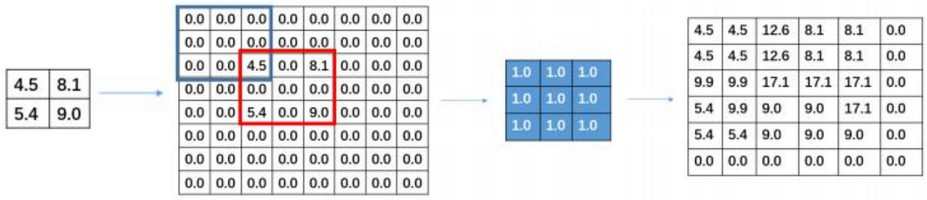
卷积后 feature size ： H_A*W_A*N

名字	参数量	计算量
标准卷积	$Cal_s = H_K*W_K*M*N$	$Para_s = H_A*W_A*N*H_K*W_K*M$
深度可分卷积	$Cal_d = H_K*W_K*1*M+1*1*M*N$	$Para_d = H_A*W_A*M*H_K*W_K+H_A*W_A*N*1*1*M$

$$\frac{Cal_d}{Cal_s} = \frac{Para_d}{Para_s} = \frac{1}{N} + \frac{1}{H_K * W_K}$$

转置卷积-主要用于上采样





在图像语义分割网络FCN-32s中，上采样反卷积操作的输入每张的尺寸是 7 X 7，希望一次上采样后能恢复成原始图像的尺寸224 X 224，代入公式：

O = (I – 1) * s + k – 2p

O = 224, I = 7

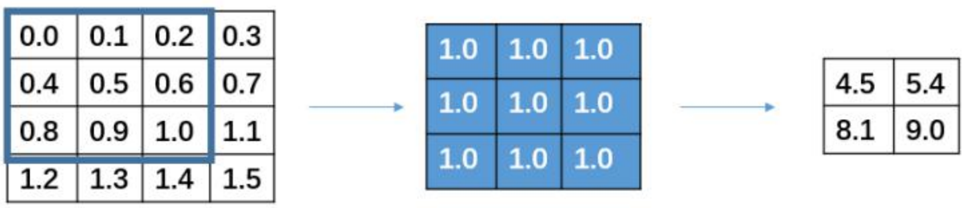
6 s + k – 2p = 224

通过实验，最终找出了最合适的一组数据：

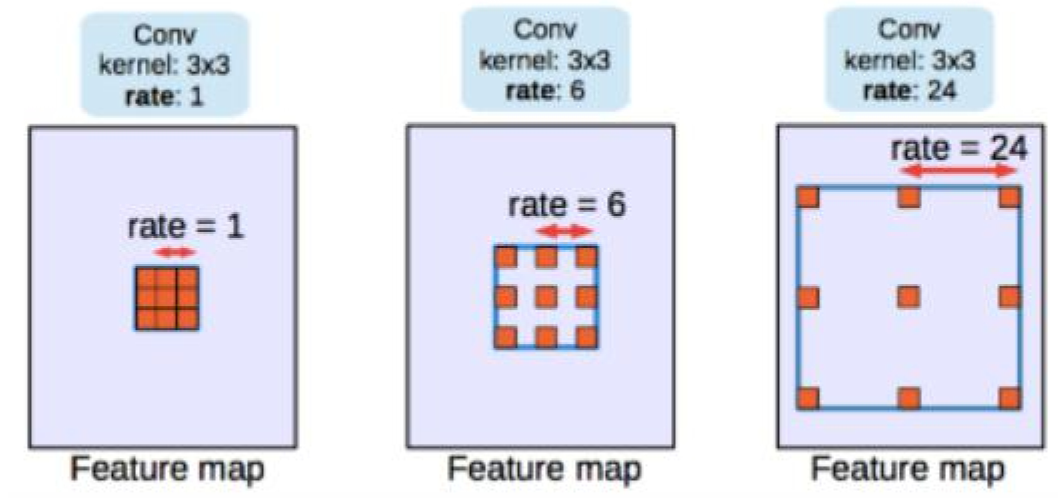
s = 32, k = 64, p = 16

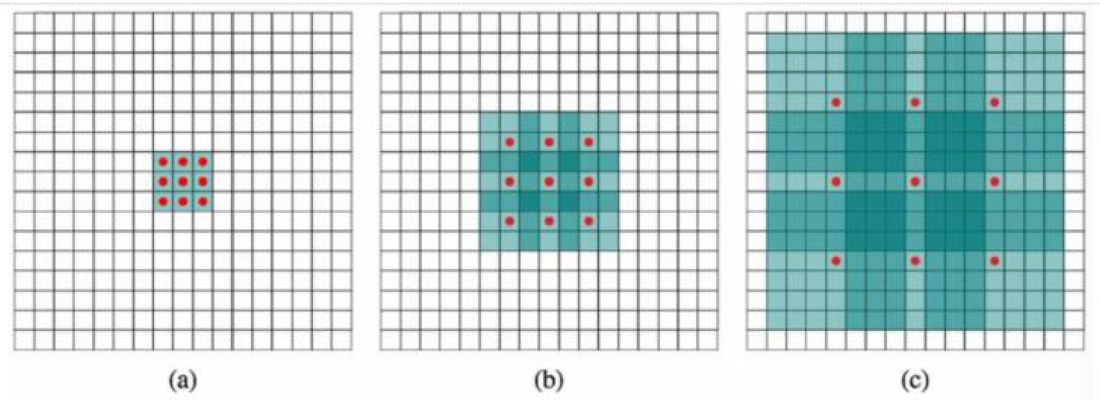
- **I (input) 输入的尺寸**
- **O (output) 输出的尺寸**
- **K (kernel) 卷积核的大小**
- **S (stride) 卷积核的步长**
- **P (padding) padding的大小**

$$o = \frac{i - k + 2 * p}{s} + 1$$



扩张/空洞卷积(dilated/atrous convolutions)





空洞卷积：在卷积核中间填充0，有两种实现方式，第一，卷积核填充0，第二，输入等间隔采样。

用空洞卷积扩大感受野，提取不同感受野上的特征，避免了下采样再上采样，节约了计算量。

用不同的卷积核膨胀系数对输入进行特征提取，尺寸没改变，再将结果 concat 到一起。

假设输入 $(N, C_{in}, H_{in}, W_{in})$,卷积操作后输出 $(N, C_{out}, H_{out}, W_{out})$ ，计算公式为

$$H_{out} = \frac{H_{in} + 2 * pad - F}{stride} + 1$$

膨胀后卷积核尺寸 = 膨胀系数 * （原始卷积核尺寸 - 1） + 1

膨胀卷积的Feature Map尺寸：

$$H_{out} = \frac{H_{in} + 2 * pad - r * (F - 1) - 1}{stride} + 1$$

感受野计算，普通卷积：

$$r_n = r_{n-1} + (k-1) * \prod_{I=1}^{n-1} s_i$$

r_n 为本层感受野

r_{n-1} 上层感受野

k 卷积核大小

s_i 是第i层卷积或池化的步长

3×3 卷积(stride=1)： $r = 1 + (3-1) = 3$ 感受野为 3×3

2×2 池化(stride=2)： $r = 3 + (2-1) * 1 = 4$ 感受野为 4×4

3×3 卷积(stride=3)： $r = 4 + (3-1) * 2 * 1 = 8$ 感受野为 8×8

3×3 卷积(stride=2)： $r = 8 + (3-1) * 3 * 2 * 1 = 20$ 感受野为 20×20

空洞卷积：

1-dilated conv：rate=1的卷积其实就是普通 3×3 因此 $r = 1 + (3-1) = 3$

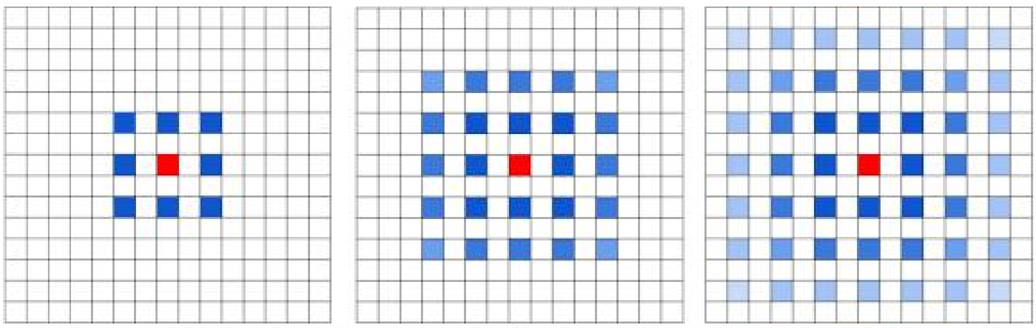
2-dilated conv：rate=2可以理解为将卷积核变成了 5×5 ，因此 $r = 3 + (5-1) * 1 = 7$

4-dilated conv：rate=4可以理解为将卷积核变成了 9×9 因此 $r = 7 + (9-1) * 1 * 1 = 15$

感受野公式为 $r = 2^{log_2 rate + 2} - 1$

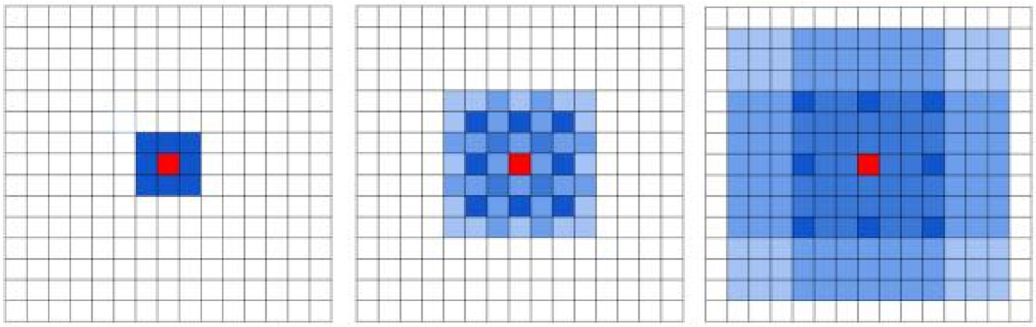
空洞卷积有感受域不连续的问题。如果一个区域包含了大量的细节信息，那么卷积核 get 到的特征可能是紊乱的，甚至是错误的。

HDC 结构，Hybrid Dilated Convolution，即混合空洞卷积。通俗的解释即避免在所有层中采用相同间隔的空洞卷积。



(a)

如果连续做 rate=2 的空洞卷积



(b)

一开始就保留了完整连续的 3x3 区域，之后几层的 rate 设置又刚好保证拼起来感受域的连贯性，即使有所重叠，也密不透风。

卷积核

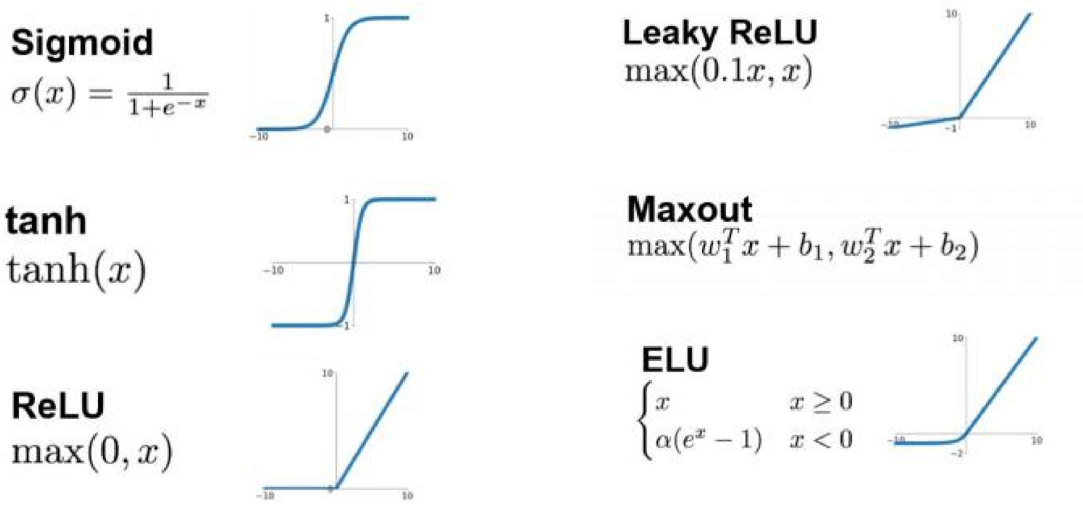
大核是为了大的感受野，但是参数量和计算量也大。一般用 3*3 的卷积核，两层 3*3=5*5 的感受野。或者用 1*1*小层数的卷积来降维，再用 3*3*小层数的卷积学习，再用 1*1*大层数升维，为了减小计算量。或者将 3*3 用 1*3 和 3*1 卷积核分别卷积，再 Concat，都是为了减少参数量，减少计算量。

全连接

在分类任务中，卷积最后接一层全连接层。Dropout 一般用于全连接层。后来多用卷积层替代全连接层。即用 h*w*n(类别数)的卷积核来替代。这样输入的图像尺寸可以是任意的。全连接层神经元个数是固定的，如果输入尺寸有变化，那么后面全连接层就接不上。但是如果是接卷积层，那么 h 和 w 的大小可以无所谓，只要是 n 个通道就可以。

激活函数

1. 非线性
2. 几乎处处可微
3. 计算简单
4. 非饱和性
5. 单调性
6. 输出范围有限
7. 参数少
8. 归一化

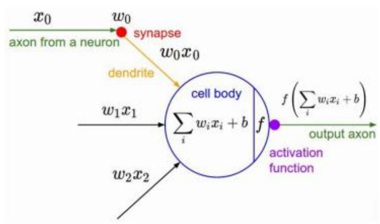


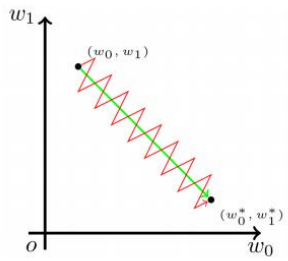
Sigmoid：计算复杂，y 没有关于 0 对称，有平台，会出现梯度消失

Tanh：计算复杂，有平台，会出现梯度消失

Relu：计算简单，将小于零的置为 0，让部分神经元失活，但也让模型稀疏化避免过拟合

为什么需要对 0 对称：对每个 wi，导数是 $\alpha\frac{\partial L}{\partial y}\frac{\partial y}{\partial z}$*xi，对每个 xi，$\alpha\frac{\partial L}{\partial y}\frac{\partial y}{\partial z}$ 都一样。而 xi 都是正数。因此 $\alpha\frac{\partial L}{\partial y}\frac{\partial y}{\partial z}$xi 都为正或者都为负。就会造成 Z 字形下降而不是直线下降。



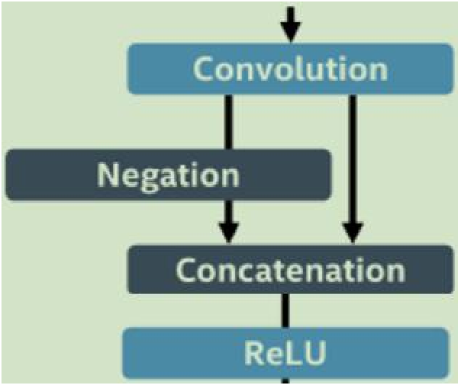


$$y = f(z) = f(\sum_i w_i x_i) \qquad w_i = w_i - \alpha\frac{\partial L}{\partial w_i}$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z}x_i \qquad w_i = w_i - \alpha\frac{\partial L}{\partial w_i} = w_i - \alpha\frac{\partial L}{\partial y}\frac{\partial y}{\partial z}x_i$$

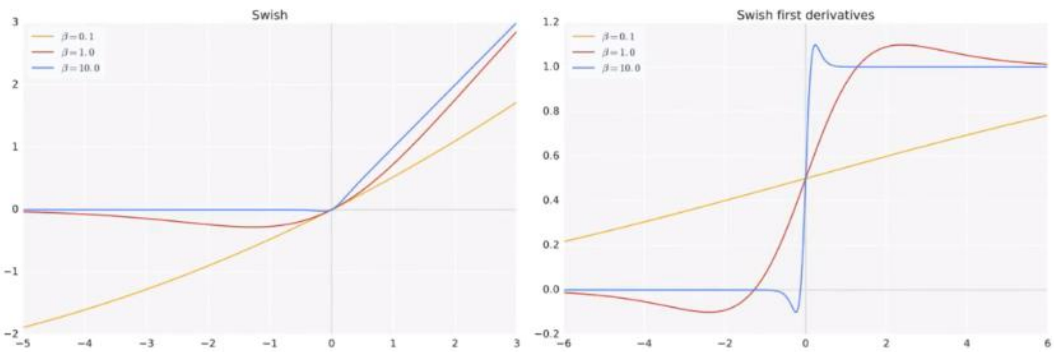
CRelu：Relu 会过滤掉一半的神经元，但是有的特征，比如向左斜的边和向右斜的边，用一个卷积核去卷积，就会出现同样的结果但是正负相反的情况。Relu 过滤了结果为负的特征，意味着要再学习一个卷积核来提取同样只是方向不同的特征。这是一种冗余。为了解决这个问题，提出了 CRelu。将卷积得到结果取反，和原结果 concat 再通过 Relu。

$$CReLU(x) = [ReLU(x), ReLU(-x)]$$



Swish：大于 0 不存在梯度消失，小于 0 不存在神经元失活，但是计算复杂

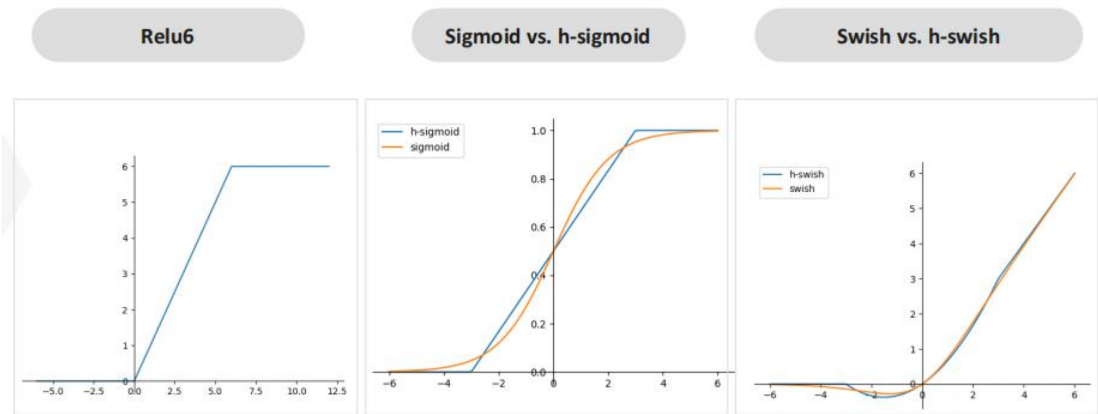
$f(x) = x \cdot \sigma(\beta x)$ $\sigma(x)$就是sigmoid函数



Rule6：6 以上置为 0，主要用于手机端，精度控制

Sigmoid vs. h-sigmoid：将 Relu6 向左平移三个单位，再除以 6，近似模拟 sigmoid

Swish vs. h-Swish：Swish=x*sigmoid，也就可以用 h-swish 模拟 swish

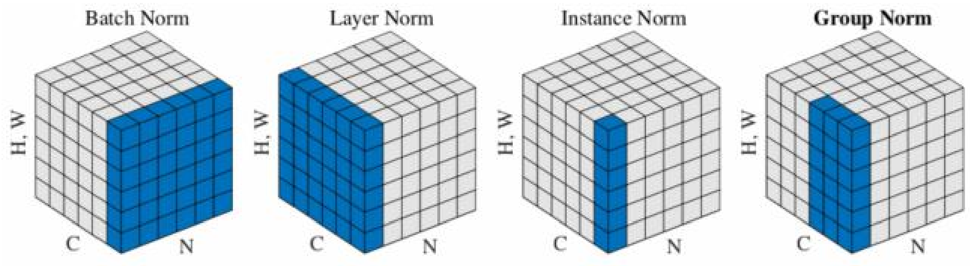


$$\text{h-swish}[x] = x\frac{\text{ReLU6}(x+3)}{6}$$

BN，IN，LN，GN

解决 Internal Covariate Shift 的和梯度消失，梯度爆炸的问题

BN 受 batch size 影响较大，LN 主要用于 RNN，IN 用于风格迁徙。



BatchNorm：batch方向做归一化，算N*H*W的均值，对小batchsize效果不好；
LayerNorm：channel方向做归一化，算C*H*W的均值，主要对RNN作用明显；
InstanceNorm：一个channel内做归一化，算H*W的均值，用在风格化迁移；
GroupNorm：将channel方向分group，然后每个group内做归一化，算(C//G)*H*W的均值；这样与batchsize无关，不受其约束。
SwitchableNorm是将BN、LN、IN结合，赋予权重，让网络自己去学习归一化层应该使用什么方法。

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

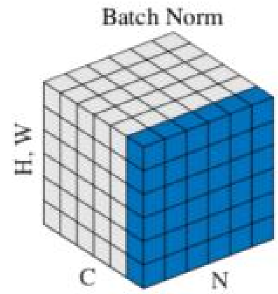
- torch.nn.BatchNorm1d(*num_features*, *eps*=*1e-05*, *momentum*=*0.1*, *affine*=*True*, *track_running_stats*=*True*)
- torch.nn.BatchNorm2d(*num_features*, *eps*=*1e-05*, *momentum*=*0.1*, *affine*=*True*, *track_running_stats*=*True*)
- torch.nn.BatchNorm3d(*num_features*, *eps*=*1e-05*, *momentum*=*0.1*, *affine*=*True*, *track_running_stats*=*True*)

参数说明：
num_features：为 feature map 的 channel 数目
eps： 为保证数值稳定性（分母不能趋近或取0）,给分母加上的值。默认为1e-5。
momentum： 动态均值和动态方差所使用的动量。默认为0.1。
affine： 布尔值，=True，给该层添加可学习的仿射变换参数。=False, 只做归一化，不乘以 gamma 加 beta（通过训练才能确定）
track_running_stats：布尔值，=True，记录训练过程中的均值和方差；=False，求当前 batch 真实平均值和标准差，而不是更新全局平均值和标准差

Note：BN只在训练的时候用，inference的时候不会用到

1. 滑动加权： $\hat{x}_{\text{new}} = (1 - \text{momentum}) \times \hat{x} + \text{momentum} \times x_t$

2. Tranning，track_running_stats



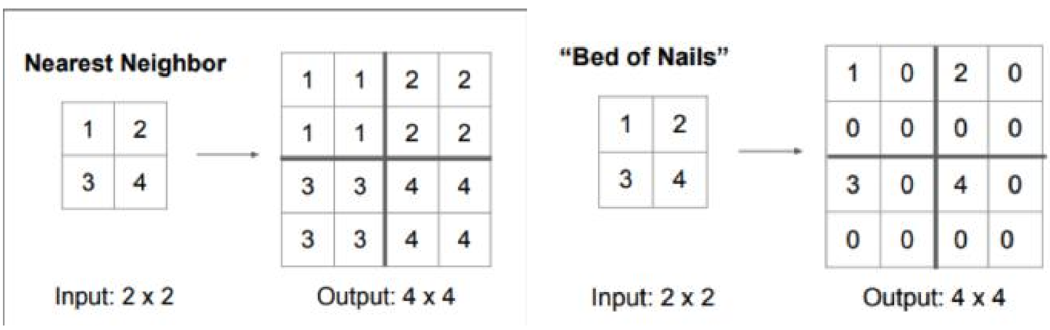
SwitchableNorm

```
import numpy as np
def SwitchableNorm(x, gamma, beta, w_mean, w_var):
    eps = 1e-5
    mean_in = np.mean(x, axis=(2, 3), keepdims=True)
    var_in = np.var(x, axis=(2, 3), keepdims=True)
    mean_ln = np.mean(x, axis=(1, 2, 3), keepdims=True)
    var_ln = np.var(x, axis=(1, 2, 3), keepdims=True)
    mean_bn = np.mean(x, axis=(0, 2, 3), keepdims=True)
    var_bn = np.var(x, axis=(0, 2, 3), keepdims=True)
    mean = w_mean[0] * mean_in + w_mean[1] * mean_ln + w_mean[2] * mean_bn
    var = w_var[0] * var_in + w_var[1] * var_ln + w_var[2] * var_bn
    x_normalized = (x - mean) / np.sqrt(var + eps)
    results = gamma * x_normalized + beta
    return results

x = np.random.rand(5, 10, 3, 3) * 100
y = SwitchableNorm(x, 0.1, 0.1, [0.1, 0.1, 0.8], [0.01, 0.02, 0.05])
```

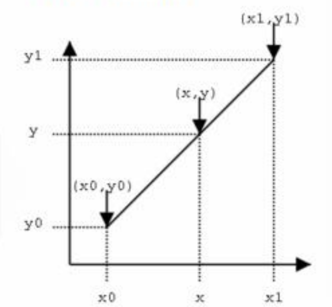
上采样

插值

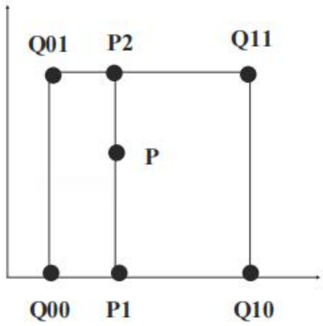


单线性插值，双线性插值

1) 单线性插值



2) 双线性插值



$$\begin{aligned} f(x,y) &= f(x_0) + w(f(x_1) - f(x_0)) \\ &= (1-w)f(x_0) + wf(x_1) \end{aligned}$$

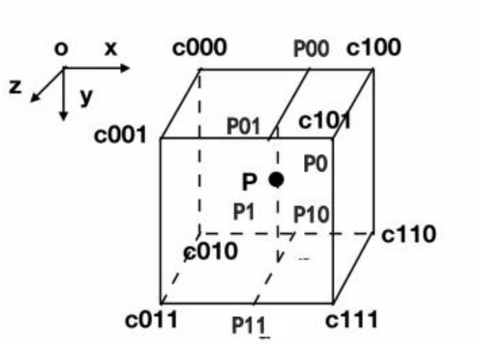
$$w = \frac{y-y_0}{y_1-y_0} = \frac{x-x_0}{x_1-x_0}$$

$$f(P_1) = (1-w_x)f(Q_{00}) + w_x f(Q_{10})$$
$$f(P_2) = (1-w_x)f(Q_{01}) + w_x f(Q_{11})$$

$$w_x = \frac{x-x_0}{x_1-x_0} \qquad w_y = \frac{y-y_0}{y_1-y_0}$$

三线性插值

1) 三线性插值



$$f(x,y,z) = (1-w_z)f(P_0) + w_z f(P_1)$$

$$f(P_0) = (1-w_y)f(P_{00}) + w_y f(P_{10})$$
$$f(P_1) = (1-w_y)f(P_{01}) + w_y f(P_{11})$$

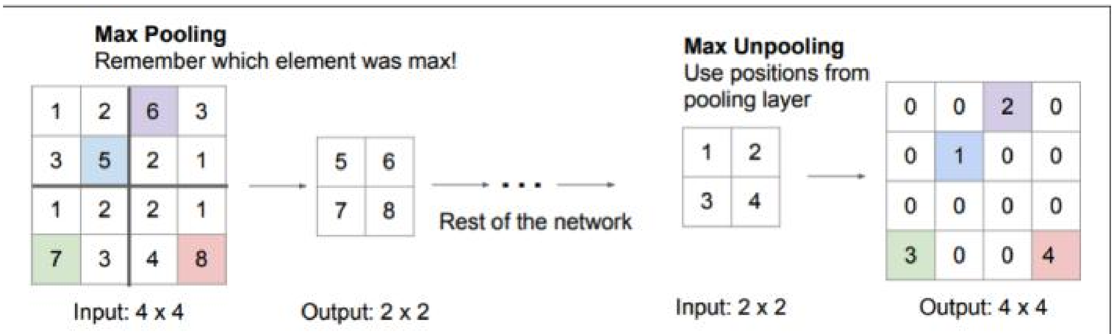
$$f(P_{00}) = (1-w_x)f(C_{000}) + w_x f(C_{100})$$
$$f(P_{10}) = (1-w_x)f(C_{010}) + w_x f(C_{110})$$
$$f(P_{01}) = (1-w_x)f(C_{001}) + w_x f(C_{101})$$
$$f(P_{11}) = (1-w_x)f(C_{011}) + w_x f(C_{111})$$

$$w_x = \frac{x-x_0}{x_1-x_0}$$

$$w_y = \frac{y-y_0}{y_1-y_0} \qquad w_z = \frac{z-z_0}{z_1-z_0}$$

Up-pooling

Max 和平均。Max 记录位置，up-pooling 的时候填充回去，其它置为 0.



转置卷积

见前面

微调和预训练

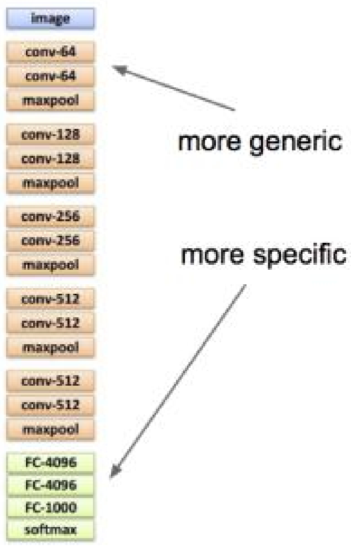
已经完成训练的参数和模型，用于新的任务

不训练直接分类，适用于和训练数据分布差不多的数据，比如用 imageNet 训练的模型，拿来对家居物品进行分类。

训练最后一层，适用于和训练数据分布类似，只有细微差别的数据，比如用 imageNet 训练的模型，拿来对猫进行分类。

完全重新训练，适用于和训练数据分布差别很大的数据，比如用 imageNet 训练的模型，拿来对 X 光片进行分类。

对于数据量大的数据，可以使用第二三种微调方法，对于小量数据，适用一二种微调方法。

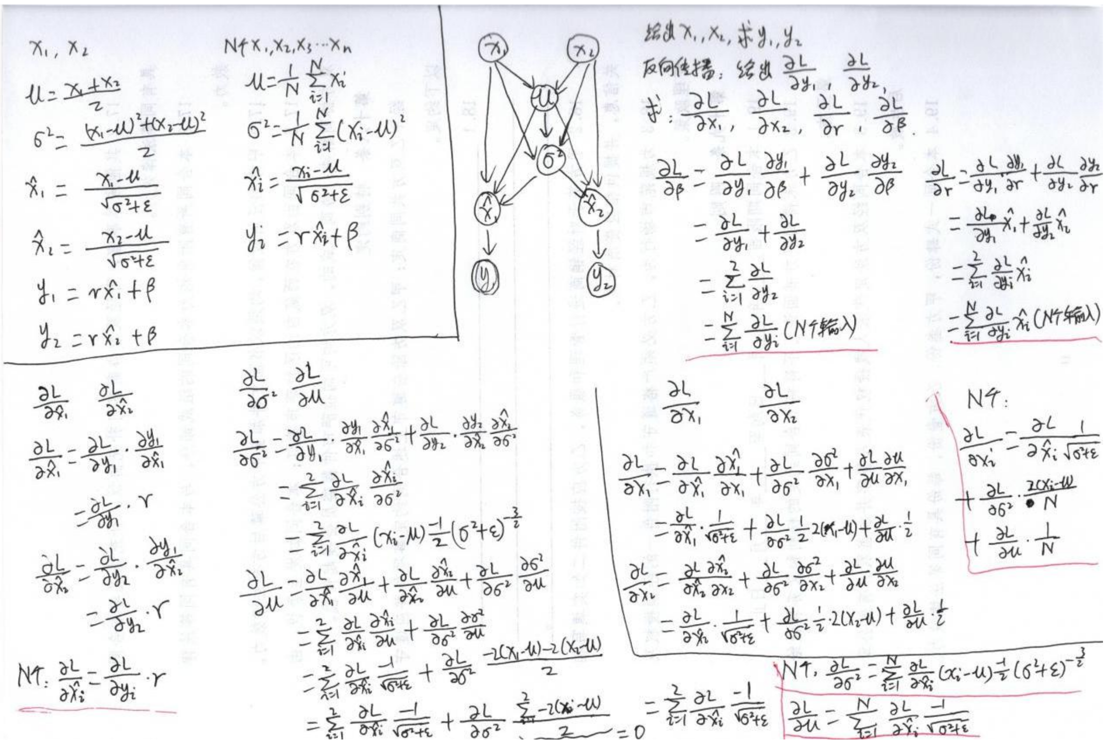


	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

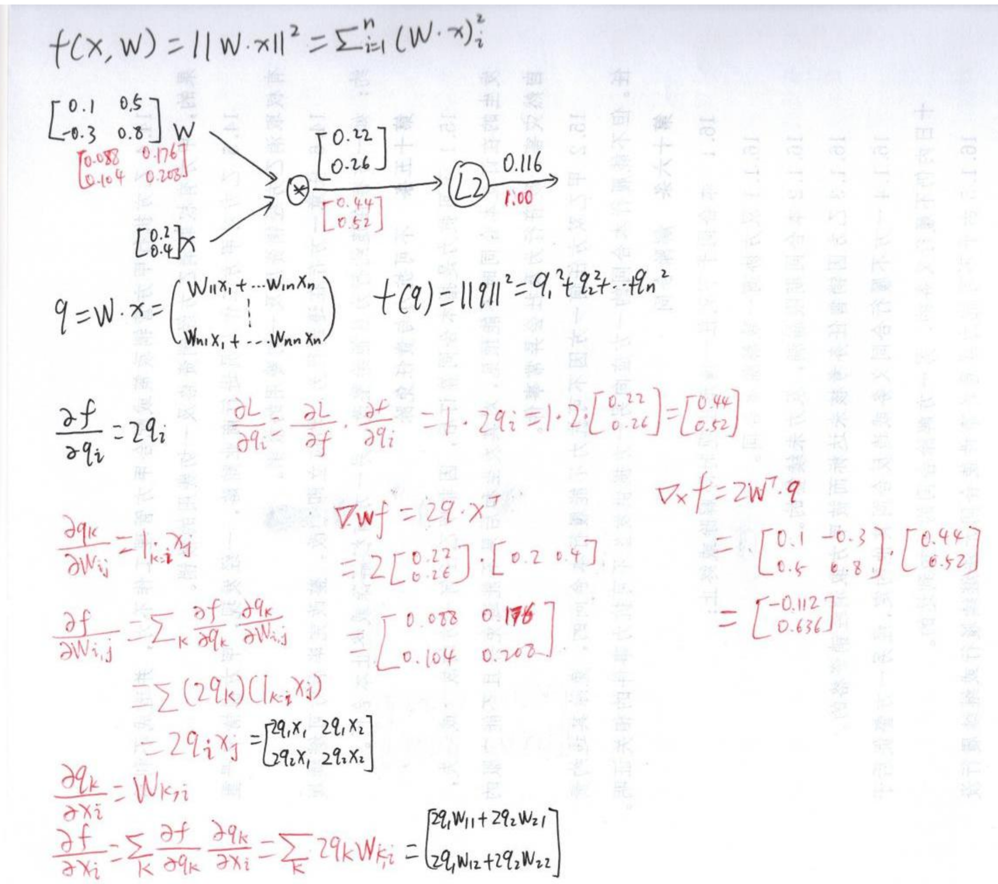
梯度反向传播公式以及常见函数的导数

原函数	导函数
$y=c$	$y'=0$
$y=n^x$	$y'=n^x\ln n$
$y=\log_a x$	$y'=\frac{1}{x\ln a}$
$y=\ln x$	$y'=\frac{1}{x}$
$y=x^n$	$y'=nx^{n-1}$
$y=\sqrt[n]{x}$	$y'=\frac{x^{-\frac{n-1}{n}}}{n}$
$y=\frac{1}{x^n}$	$y'=-\frac{n}{x^{n+1}}$
$y=\sin x$	$y'=\cos x$
$y=\cos x$	$y'=-\sin x$
$y=\tan x$	$y'=\frac{1}{\cos^2 x}=\sec^2 x$
$y=\cot x$	$y'=-\frac{1}{\sin^2 x}=-\csc^2 x$
$y=\sec x$	$y'=\sec x\tan x$
$y=\csc x$	$y'=-\csc x\cot x$
$y=\arcsin x$	$y'=\frac{1}{\sqrt{1-x^2}}$
$y=\arccos x$	$y'=-\frac{1}{\sqrt{1-x^2}}$
$y=\arctan x$	$y'=\frac{1}{1+x^2}$
$y=\mathrm{arccot} x$	$y'=-\frac{1}{1+x^2}$
$y=arcsecx$	$y'=\frac{1}{x\sqrt{x^2-1}}$
$y=arccscx$	$y'=-\frac{1}{x\sqrt{x^2-1}}$
$y=shx=\frac{e^x-e^{-x}}{2}$ （双曲函数）	$y'=chx$ （双曲函数）
$y=chx=\frac{e^x+e^{-x}}{2}$	$y'=shx$
$y=thx=\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$y'=\frac{1}{ch^2x}$
$y=arshx=ln\left(x+\sqrt{x^2+1}\right)$	$y'=\frac{1}{\sqrt{x^2+1}}$
$y=archx=ln\left(x+\sqrt{x^2-1}\right)$	$y'=\frac{1}{\sqrt{x^2-1}}$
$y=arthx=\frac{1}{2}ln\left(\frac{1+x}{1-x}\right)$	$y'=\frac{1}{1-x^2}$

BN反向传播



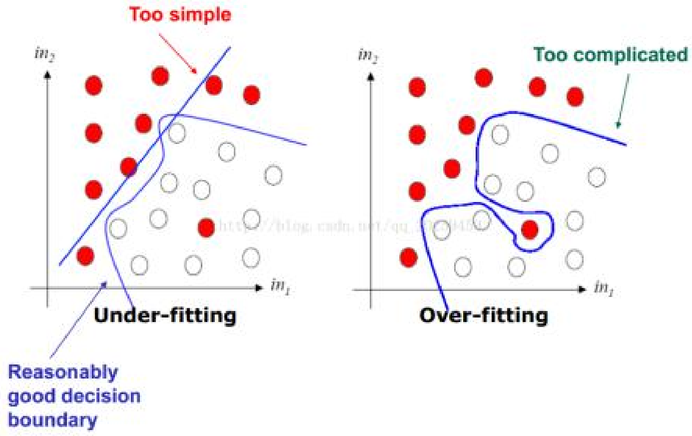
反向传播的矩阵形式

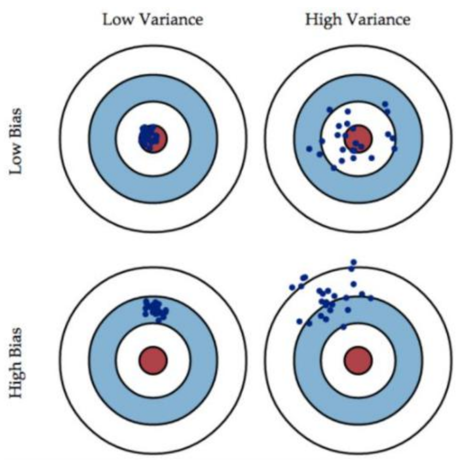


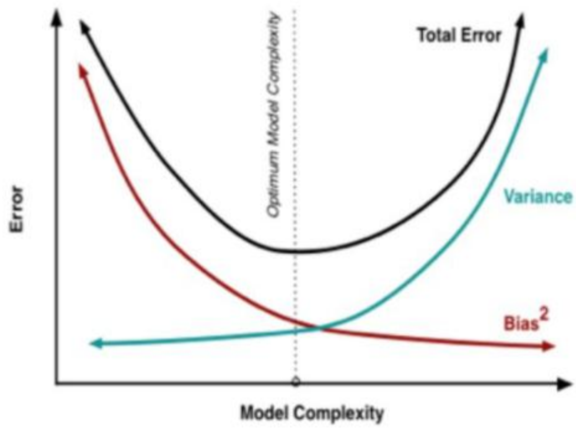
Softmax推导，反向传播

Bias, Variance

欠拟合和过拟合







Bias 是训练出的函数和目标函数在分布上不相等，就像打靶的人散光，瞄准的时候就瞄错了，但是准头很好。Variance 是打靶的人准头太差。

One-hot and Embedding

独热和编码

Normalize and Scalar

标准化（均值，方差）和归一化（缩放）