



深度学习

Deep Learning

简介

人工智能 Artificial Intelligence

机器学习 Machine Learning

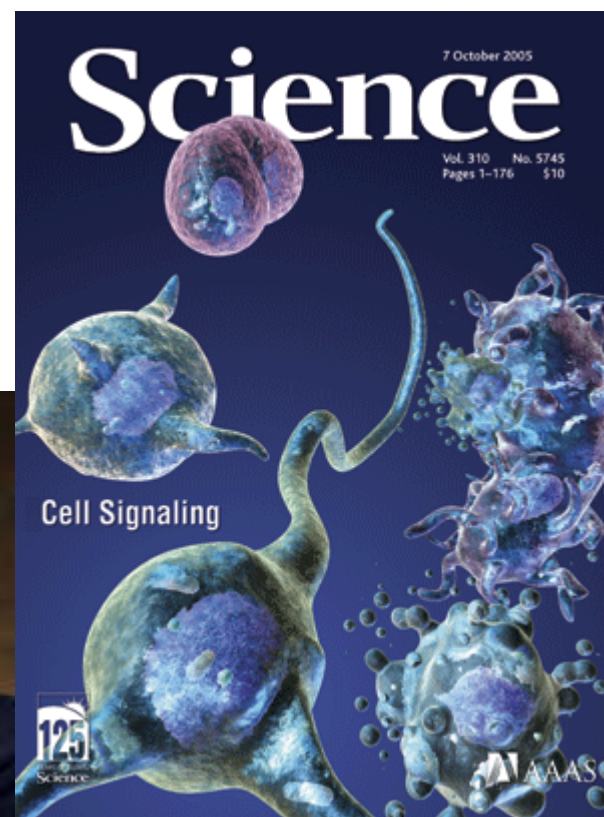
神经网络 Neural Network

深度学习 Deep Learning

深度学习是神经网络的一个大分支，深度学习的基本结构是深度神经网络。

深度学习的提出

- 其概念由著名科学家 Geoffrey Hinton 等人在2006年和2007年在《Sciences》等上发表的文章被提出和兴起。





深度学习

“深度学习”和“多层神经网络”的区别？

没什么区别，为了把学术界的目光重新转到神经网络上提出的新名词。

当然，最近有一些深度学习背景下产生的新技巧新设计，但是本质没变



新设计新技巧？

新的网络结构，如：CNN ? LSTM? ResNet

新的激活函数，如：ReLU

新的权重初始化方法，如：逐层初始化

新的防止过拟合方法，如：Dropout, BN



损失函数

损失函数 (Loss function) 是用来估量你模型的预测值 $f(x)$ 与真实值 Y 的不一致程度，它是一个非负实值函数，通常用 $L(Y, f(x))$ 来表示。

损失函数是经验风险函数的核心部分，也是结构风险函数的重要组成部分。模型的风险结构包括了风险项和正则项，通常如下所示：

$$\theta^* = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i; \theta)) + \lambda \Phi(\theta)$$

L 代表的是损失函数，后面的 Φ 是正则化项



几种损失函数

1. **铰链损失** (Hinge Loss) : 支持向量机 (SVM)
2. **互熵损失** (Cross Entropy Loss, Softmax Loss) :
Logistic 回归与Softmax 分类
3. **平方损失** (Square Loss) : 最小二乘法
4. **指数损失** (Exponential Loss) : Adaboost 集成学习算法
5. **其他损失** (如0-1损失, 绝对值损失)



深度学习中的常用损失函数

SVM Loss (Hinge Loss)

Softmax Loss (Cross Entropy Loss)

SVM Loss

简单例子：线性分类器得分函数



[32x32x3] 的输入矩阵

图像数据 权重/参数

$$f(\mathbf{x}, \mathbf{W})$$

CIFAR-10 中的10个类别的得分向量

SVM Loss

简单例子：线性分类器得分函数



[32x32x3] 的输入矩阵

$$\boxed{f(x, W)} = \boxed{W} \boxed{x} \quad \begin{matrix} 3072 \times 1 \\ 10 \times 3072 \end{matrix}$$

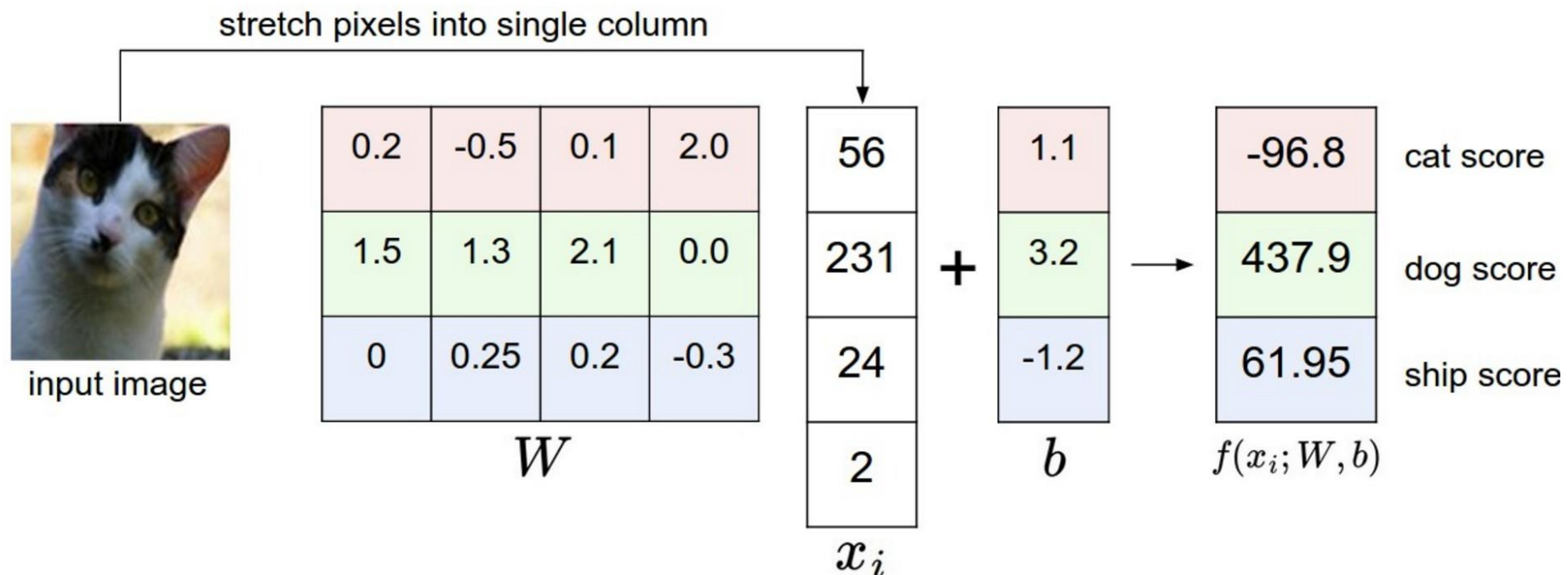
10x1 **10x3072**

CIFAR-10 中的
10个类别的
得分向量

参数，或者叫做权重

SVM Loss

简单例子：线性分类器得分函数





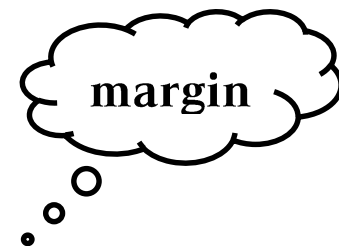
SVM Loss

对于训练集中的第*i*张图片数据 x_i

在 W 下会有一个得分结果向量 $f(x_i, W)$

第*j*类的得分我们记作 $f(x_i, W)_j$

则在該样本上的损失我们由下列公式计算得到:



$$L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$$

假如我们现在有三个类别，而得分函数计算某张图片的得分为 $f(x_i, W) = [13, -7, 11]$ ，而实际的结果是第一类($y_i = 0$)。假设 $\Delta = 10$ (这个参数一会儿会介绍)。上面的公式把错误类别($j \neq y_i$)都遍历了一遍，求值加和：

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

SVM Loss

因为是线性模型，因此可以简化成

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

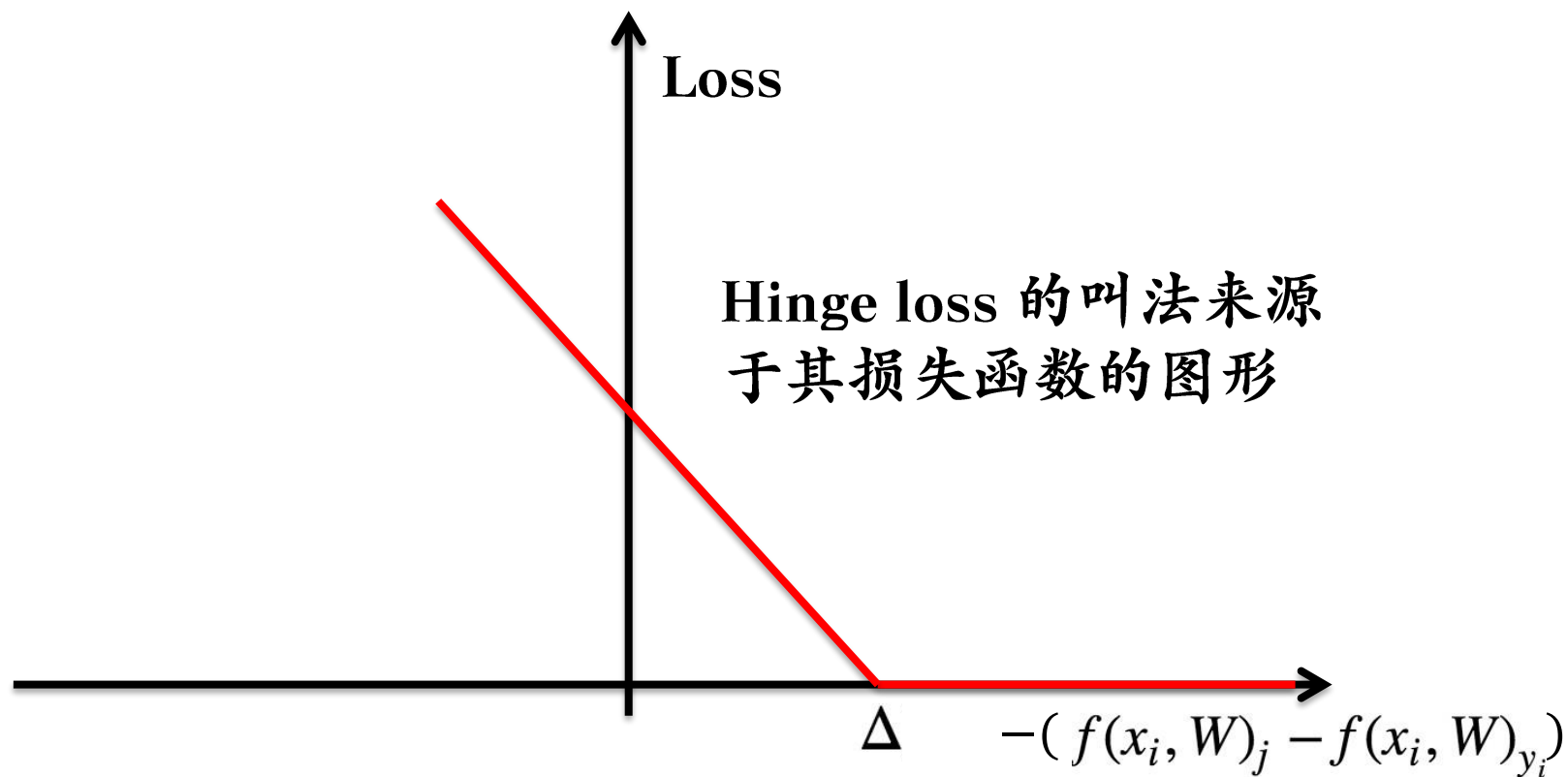


加正则化项

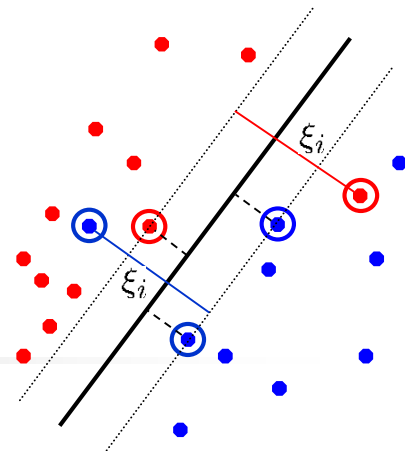
$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l$$

Why Hinge Loss?



Why SVM Loss?



间距最大化的问题
SVM 优化函数:

$$\begin{aligned} \underset{w, \zeta}{\operatorname{argmin}} \quad & \frac{1}{2} \|w\|^2 + C \sum_i \zeta_i \\ \text{st.} \quad & \forall y_i w^T x_i \geq 1 - \zeta_i \\ & \zeta_i \geq 0 \end{aligned}$$

将约束项进行变形, 则为: $\zeta_i \geq 1 - y_i w^T x_i$
则损失函数可以进一步写为:

$$\begin{aligned} J(w) &= \frac{1}{2} \|w\|^2 + C \sum_i \max(0, 1 - y_i w^T x_i) \\ &= \frac{1}{2} \|w\|^2 + C \sum_i \max(0, 1 - m_i(w)) \\ &= \frac{1}{2} \|w\|^2 + C \sum_i L_{\text{Hinge}}(m_i) \end{aligned}$$

因此, SVM 的损失函数可以看作是
L2-norm 和 Hinge loss 之和。



深度学习中的常用损失函数

SVM Loss (Hinge Loss)

Softmax Loss (Cross Entropy Loss)



交叉熵

熵的本质是香农信息量的期望： $H(X) = - \sum_x p(x) \log_2 p(x)$

现有关于样本集的2个概率分布 p 和 q ，其中 p 为真实分布， q 非真实分布。按照真实分布 p 来衡量识别一个样本的熵为：

$$H(p) = \sum_x p(x) \log_2 \frac{1}{p(x)}$$

如果使用错误分布 q 来表示来自真实分布 p 的熵：

$$H(p, q) = \sum_x p(x) \log_2 \frac{1}{q(x)} \quad \text{交叉熵}$$



Softmax Loss

对于训练集中的第*i*张图片数据 x_i
在 W 下会有一个得分结果向量 f_{y_i}
则损失函数记作

$$L_i = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \quad \text{或者} \quad L_i = -f_{y_i} + \log \sum_j e^{f_j}$$

实际工程中一般这么算，提高数值稳定性：

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{C e^{f_{y_i}}}{C \sum_j e^{f_j}} = \frac{e^{f_{y_i} + \log C}}{\sum_j e^{f_j + \log C}}$$

一个最常见的 C 取值为： $\log C = -\max_j f_j$

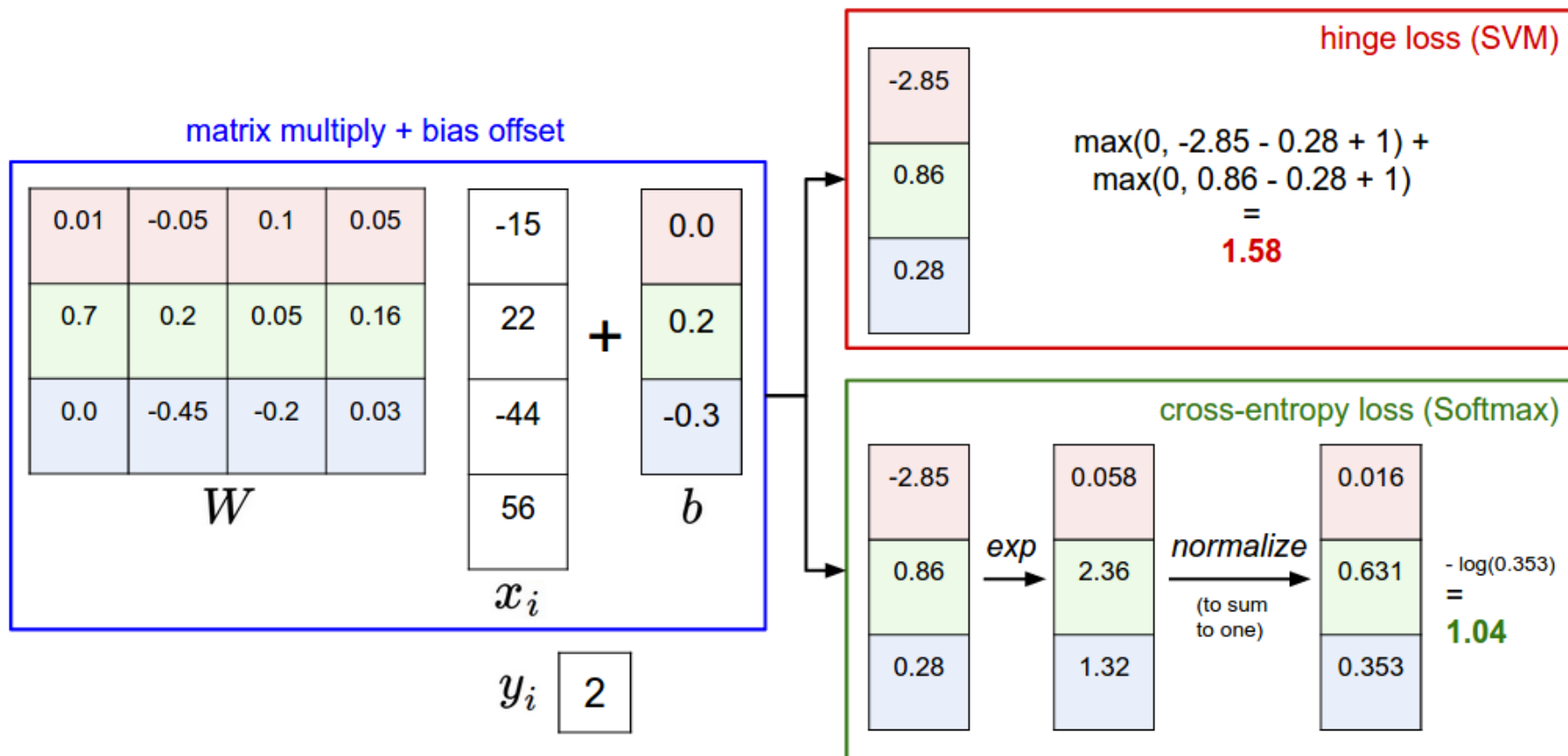
应该平移向量 f 中的值使得最大值为0



深度学习中的常用损失函数

SVM Loss VS Softmax Loss

SVM Loss VS Softmax Loss





SVM Loss VS Softmax Loss

- SVM下，我们能完成类别的判定，但是实际上我们得到的类别得分，大小顺序表示着所属类别的排序，但是得分的绝对值大小并没有特别明显的物理含义。
- Softmax分类器中，结果的数值大小表征属于该类别的概率。



学习率

举例: $J(x) = x^2$

假设学习率: $\alpha = 1$

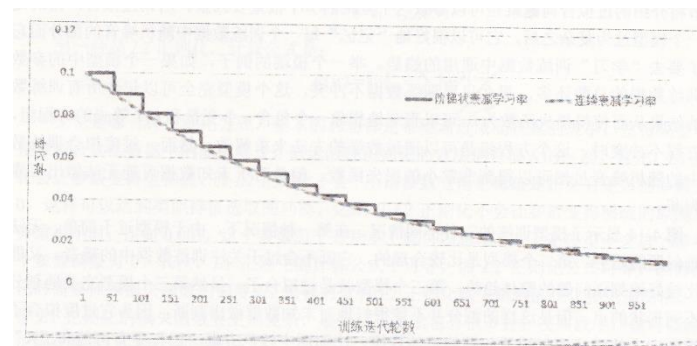
更新: $x = x - \alpha \times 2x$

初始值: $x = 5$

轮次	当前参数值	更新后参数值
1	5	$5 - 1 * 2 * 5 = -5$
2	-5	$-5 - 1 * 2 * (-5) = 5$
3	5	$5 - 1 * 2 * 5 = -5$
...

学习率

- 1、基于经验的手动调整
- 2、基于策略的调整



例如：TensorFlow中的指数衰减法

```
# 设置指数衰减的学习率。
learning_rate = tf.train.exponential_decay(
    LEARNING_RATE_BASE, #初始学习率
    global_step, #总次数
    mnist.train.num_examples / BATCH_SIZE, #每训练多少次修改学习率
    LEARNING_RATE_DECAY, #衰减系数
    staircase=True) #是否开启阶梯衰减
```



滑动平均模型

TensorFlow中的滑动平均模型

```
variable_averages = tf.train.ExponentialMovingAverage(  
    MOVING_AVERAGE_DECAY, global_step)
```

衰减率用于控制模型更新的速度。对每一个变量都会维护一个影子变量。影子变量的初始值就是这个变量的初始值

$$shadow_variable = decay \times shadow_variable + (1 - decay) \times variable$$

实际运用中，decay 一般会设置为十分接近 1 的常数（0.99或0.999）。



滑动平均模型

TensorFlow中的滑动平均模型

为了使得模型在训练的初始阶段更新得更快，还提供了 `num_updates` 参数来动态设置 `decay` 的大小：

$$decay = \min\left\{decay, \frac{1 + num_updates}{10 + num_updates}\right\}$$

Momentum 动量

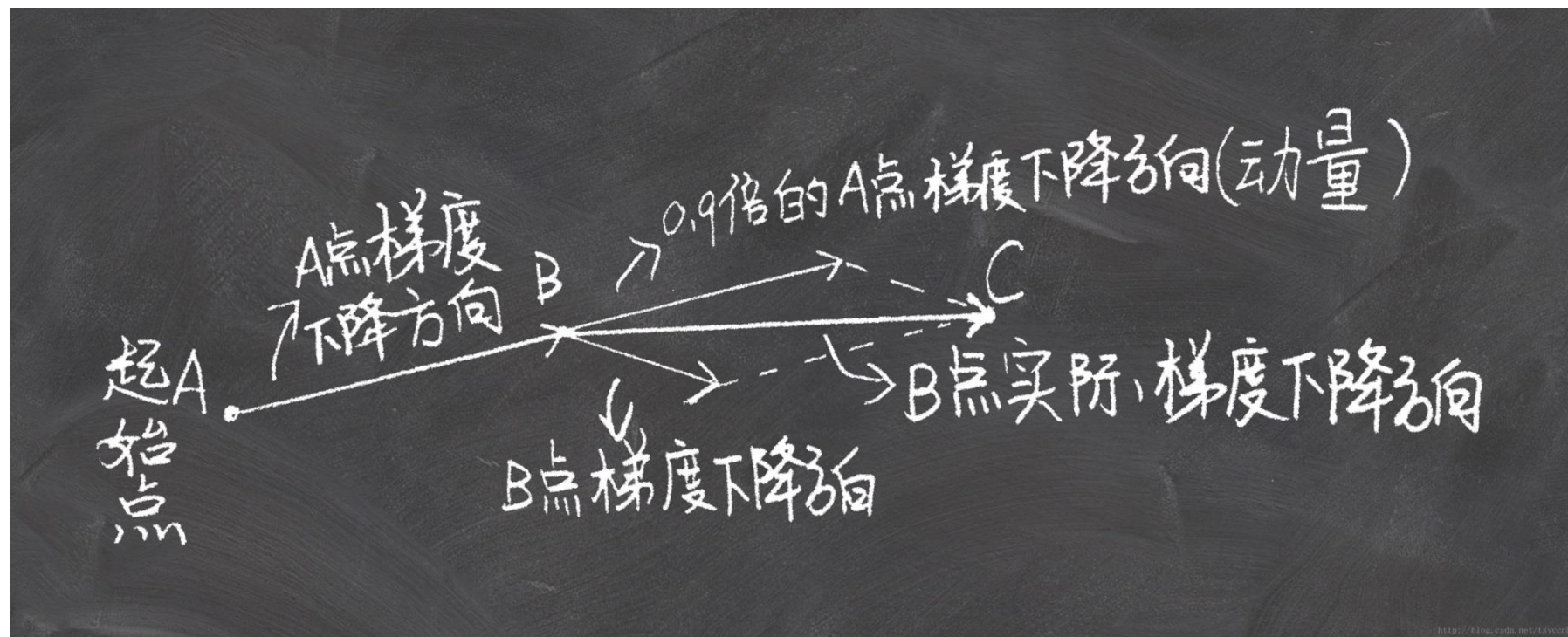
盲人下山 VS 小球下山

用拐杖来回试探
四周的路

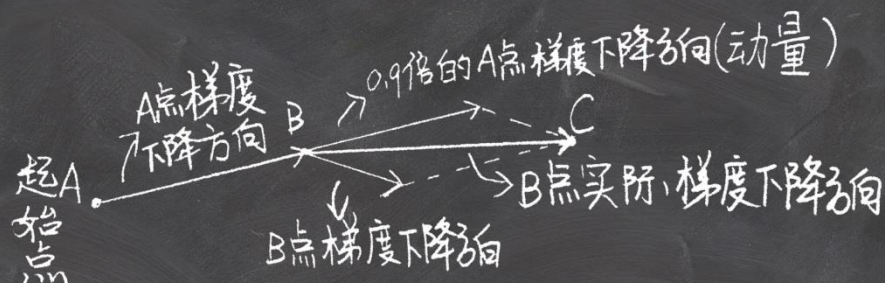


在当前初速度下继续加速下降，小球
会越滚越快

Momentum 动量



Momentum 动量



A为起始点，首先计算A点的梯度 ∇a

然后下降到B点 $\theta_{new} = \theta - \alpha \nabla a$

θ 为参数， α 为学习率

到了B点需要加上A点的梯度，这里梯度需要有一个衰减值 γ ，推荐取0.9。这样的做法可以让早期的梯度对当前梯度的影响越来越小，如果没有衰减值，模型往往会震荡难以收敛，甚至发散。所以B点的参数更新公式是这样的：

$$v_t = \gamma v_{t-1} + \alpha \nabla b$$

$$\theta_{new} = \theta - v_t$$

其中 v_{t-1} 表示之前所有步骤所累积的动量和。

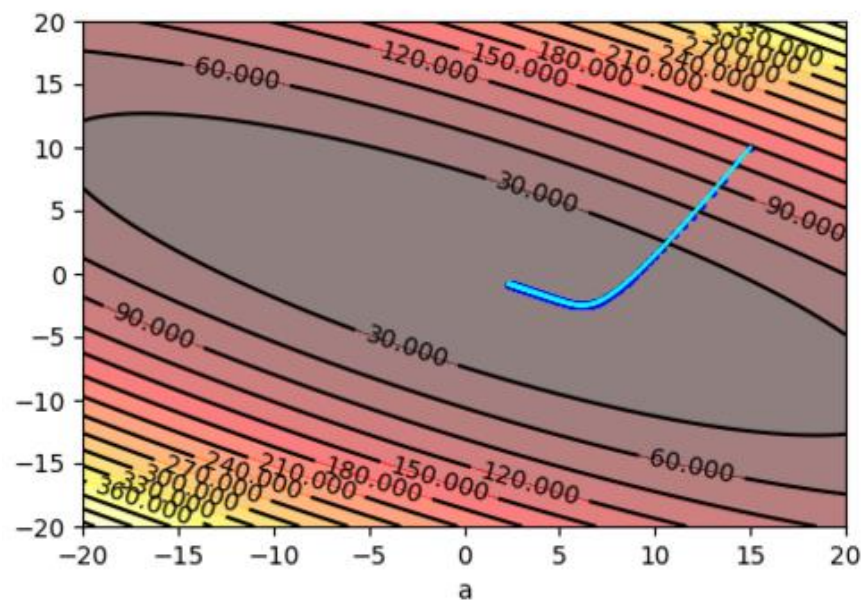
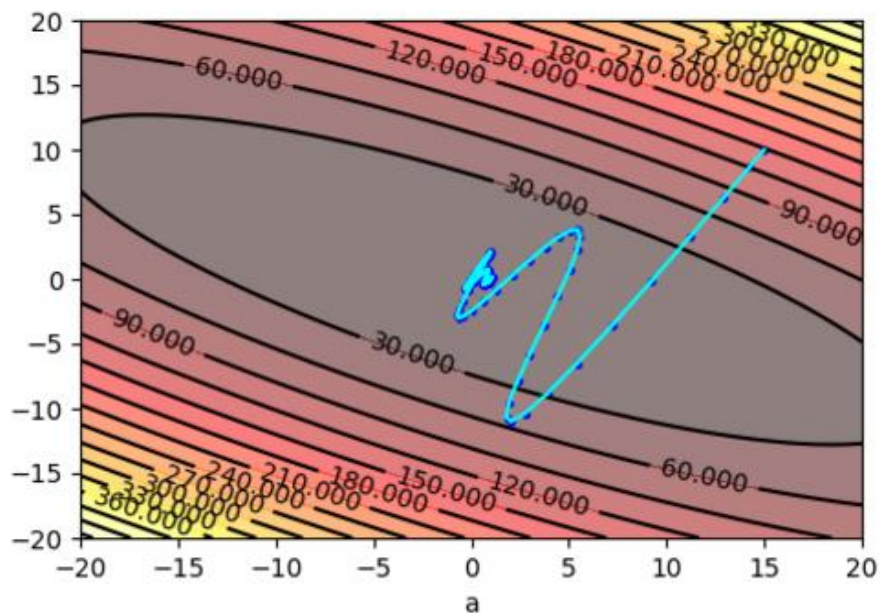
这样一步步下去，带着初速度的小球就会极速的奔向谷底。

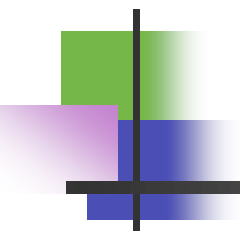
实验对比

+Momentum 动量

VS

梯度下降





卷积

Convolution



什么是卷积?

就是一种数学运算，跟减加乘除没有本质的区别。

连续域： $f(x), g(x)$ 是 R^1 上的两个可积函数，作积分：

$$h(x) = \int_{-\infty}^{+\infty} f(\tau)g(x - \tau)d\tau$$

离散域：卷积的变量是序列 $x(n)$ 和 $h(n)$ ，卷积的结果定义：

$$y(n) = \sum_{i=-\infty}^{+\infty} x(i)h(n - i) = x(n) * h(n)$$

卷积符号



卷积的来源

大学课程：《信号与系统》

如果一个信号是一组历史信号的组合，比如 $a(0), a(1), a(2), \dots, a(n), \dots$ ，其中 $a(i)$ 是 i 时刻信号的量值，我们要计算在某一时刻 n 的信号的组合量值 $f(n)$ ， $f(n)$ 是 $a(0), a(1), a(2), \dots, a(n)$ 的组合。

简单线性组合：

$$f(n) \propto a(0) + a(1) + a(2) + \dots + a(n)$$



卷积的来源

假设信号的衰减规律符合统一规律函数 $b(n)$, 称为衰减率。
信号0时刻: $b(0)$, 信号1时刻: $b(1)$, 以此类推……

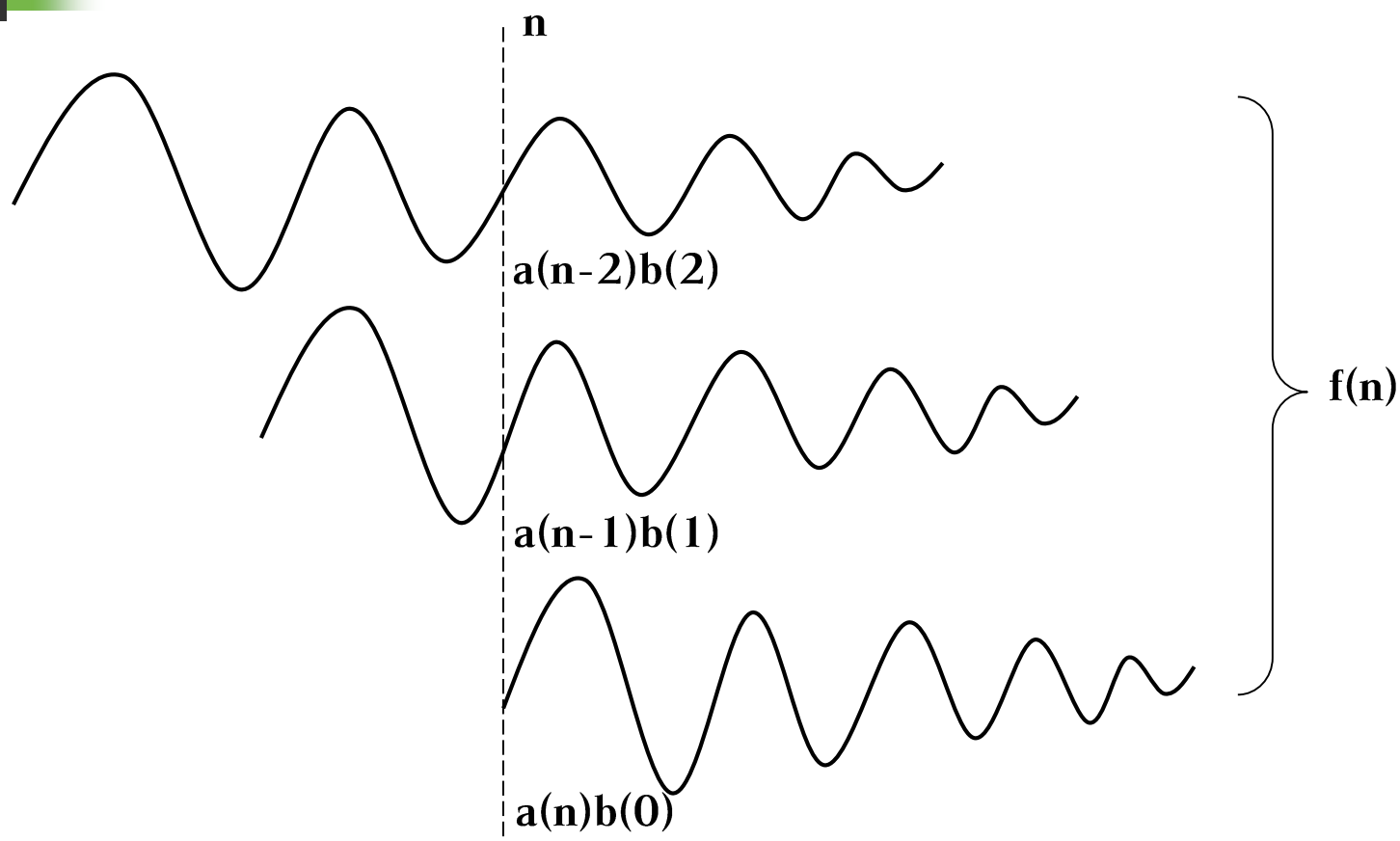
求 n 时刻的信号组合量 $f(n)$?

因为 n 时刻 $a(n)$ 信号刚出来, 它的衰减剩余率应该为 $b(0)$
而 $n-1$ 时刻的信号衰减了一个时间周期了, 它的衰减剩余率是 $b(1)$ ……, 写成式子就是:

$$f(n) = a(0)b(n) + a(1)b(n-1) + a(2)b(n-2) + \dots + a(n)b(0)$$

$$f(n) = \sum_{i=0}^n a(i)b(n-i) = a(n) * b(n)$$

卷积的来源



$$f(n) = a(0)b(n) + a(1)b(n-1) + a(2)b(n-2) + \dots + a(n)b(0)$$



卷积的物理意义

一组值乘以他们相应的“权重”系数的和。

- 以上是一个变量的数列的卷积物理意义解释
- 推广到一元函数
- 推广二元、多元函数



图像处理中的卷积

一幅图像可以看成是一个二维函数，自变量是图片象素的坐标 (x,y) ，函数值是象素的颜色（灰度）取值 $(0\sim 255)$ 。

举例：模糊处理

把一个象素的周围象素颜色（灰度）值乘以一个权重值求和，效果会使得图像效果变得朦胧。

这个过程也符合卷积的物理意义，所以这个处理也被称为卷积。

图像处理中的卷积

实例1：平滑

Input image

Convolution
Kernel

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

Feature map



$$\begin{bmatrix} \text{green} & \text{green} & \text{green} \\ \text{green} & \text{red} & \text{green} \\ \text{green} & \text{green} & \text{green} \end{bmatrix} * \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \text{green} & \text{green} & \text{green} \\ \text{green} & \text{light green} & \text{green} \\ \text{green} & \text{green} & \text{green} \end{bmatrix}$$

图像处理中的卷积

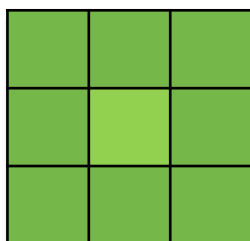
实例2：边缘检测

Input image

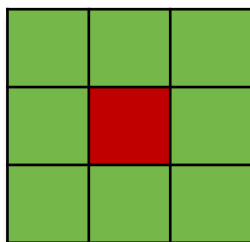
Convolution
Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \text{dark gray square}$$



$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \text{light gray square}$$

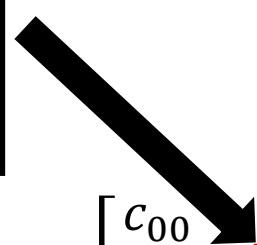
$$f(n) = a(0)b(n) + a(1)b(n-1) + a(2)b(n-2) + \dots + a(n)b(0)$$

信号统一到图像的卷积

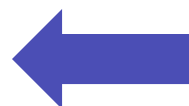
$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & \dots & a_{0n} \\ a_{10} & a_{11} & a_{12} & \dots & a_{1n} \\ a_{20} & a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{m0} & a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

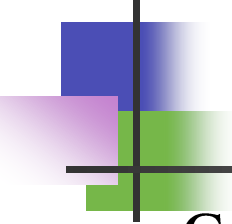


$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$



$$\begin{bmatrix} c_{00} & c_{01} & c_{02} & \dots & c_{0n} \\ c_{10} & c_{11} & c_{12} & \dots & c_{1n} \\ c_{20} & c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ c_{m0} & c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}$$





卷积定理

Convolution Theorem

它将时域和空域上的复杂卷积对应到了频域中的元素间简单的乘积

连续域:

$$h(x) = \int_{-\infty}^{+\infty} f(\tau)g(x - \tau)d\tau = \mathcal{F}^{-1}(\sqrt{2\pi}\mathcal{F}[f]\mathcal{F}[g])$$

离散域:

$$f(n) = \sum_{i=0}^n a(i)b(n - i) = \mathcal{F}^{-1}(\sqrt{2\pi}\mathcal{F}[a]\mathcal{F}[b])$$

卷积定理也是快速傅里叶变换算法被称为20世纪最重要的算法之一的一个原因。

傅里叶级数

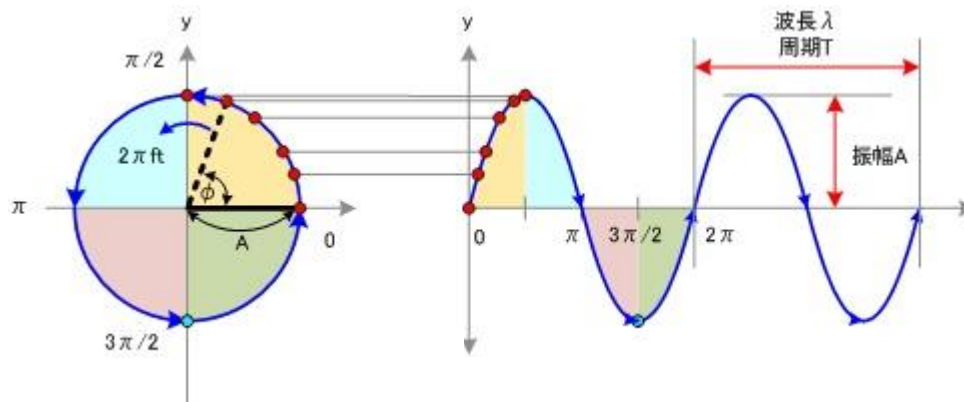


$$y = A \sin(\omega x + \theta)$$

概念：
频率、幅值、相位

在时域是一个周期且连续的函数，
而在频域是一个非周期离散的函数

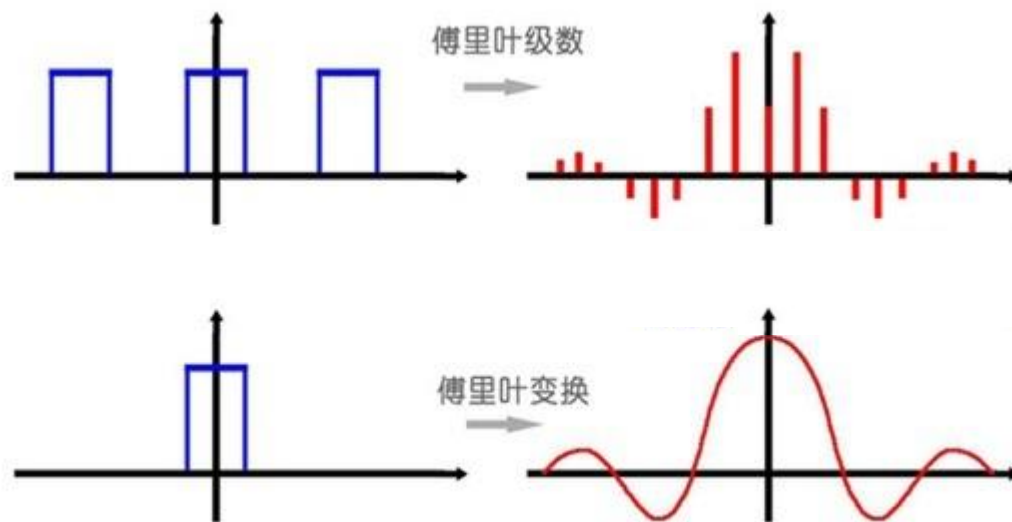
傅里叶级数



傅里叶变换

傅里叶级数的本质是将一个周期的信号分解成无限多分开的（离散的）正弦波，但是宇宙似乎并不是周期的。

傅里叶变换实际上是对一个周期无限大的函数进行变换。



傅里叶变换将时域非周期的连续信号，转换为在频域非周期的连续信号

$$y = A \sin(\omega x + \theta)$$

图像中的傅里叶变换

将图像的亮度变化brightness variation作为正弦变量

频率



在空间域上可由亮度调节，例如左图的频率比右图的频率低

幅值

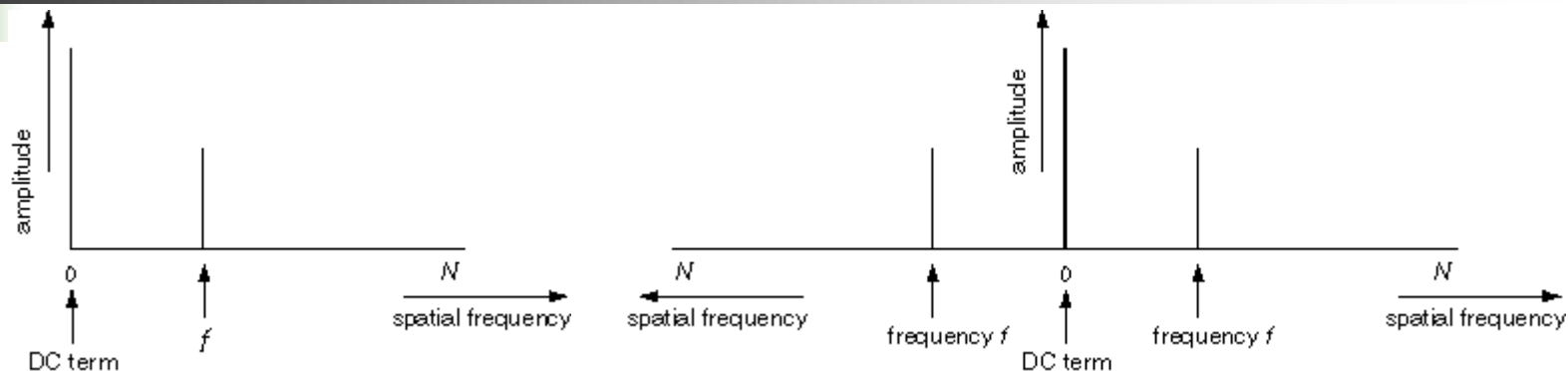
图像中最明和最暗的峰值之间的差，一个负幅值表示一个对比逆转

相位

相位表示相对于原始波形，这个波形的偏移量（左or右）

将时域非周期的离散信号，转换为在频域非周期的离散信号

图像中的傅里叶变换



$$x = [x_1, x_2, x_3 \dots x_n]$$

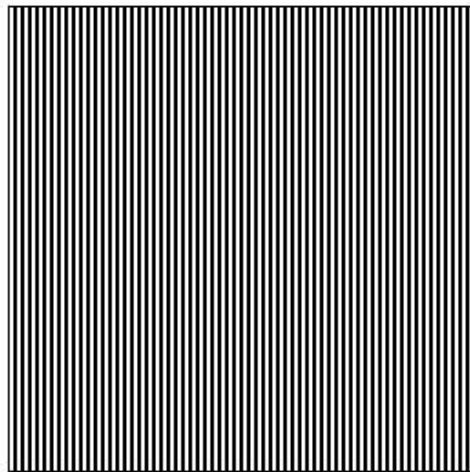
$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix}$$

$$x + y = ?$$

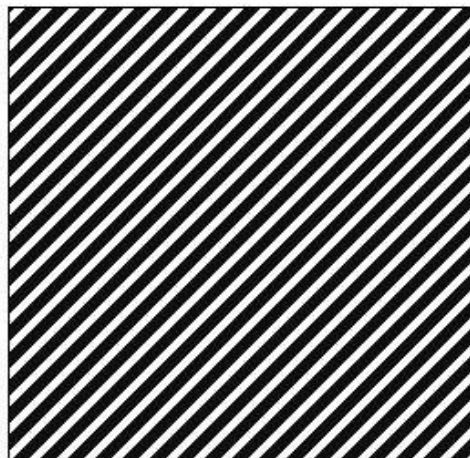
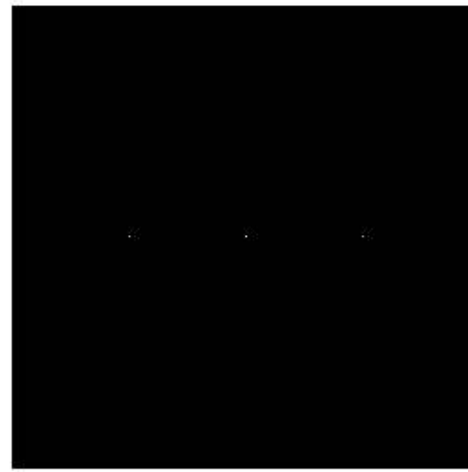
在频域中，低频率更接近中央而高频率更接近边缘。

图像中的傅里叶变换

Original



Fourier transformed (real part)

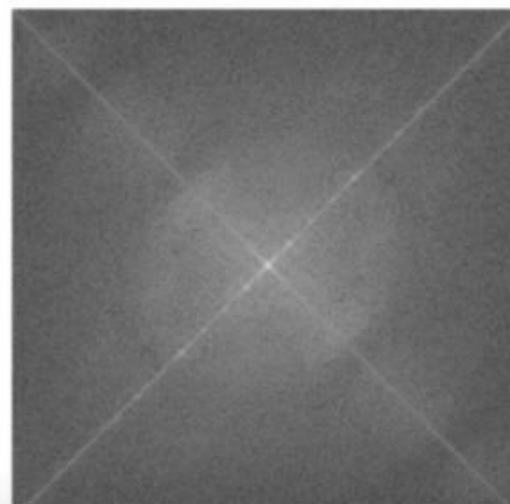
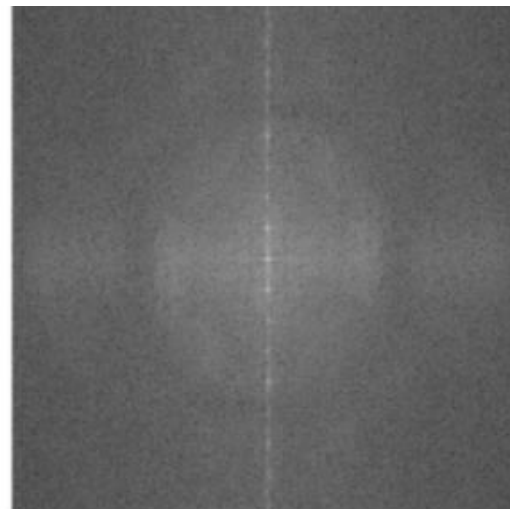


图像判别方向

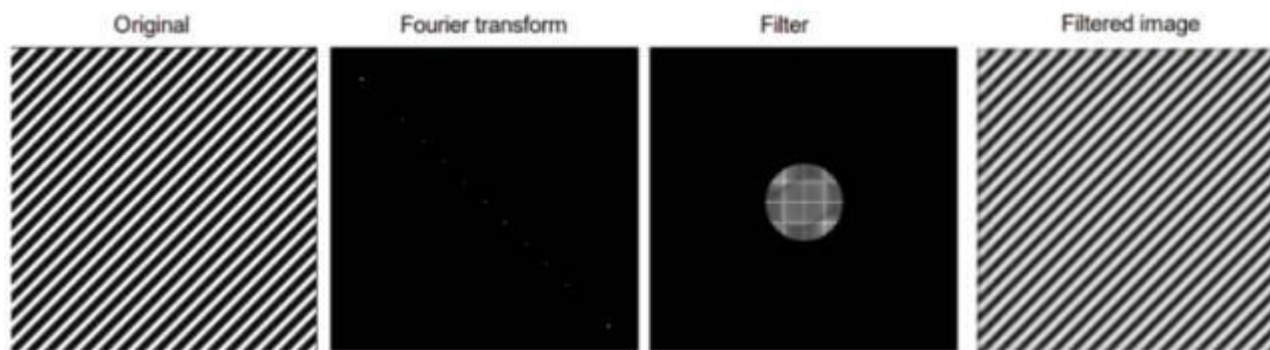
Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I could impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while those setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Cochran



频率过滤与卷积



- 1、对图像执行傅里叶变换
- 2、乘以一个圆形（背景填充黑色，也就是0）

可以过滤掉所有的高频值（它们会成为0）。
注意过滤后的图像依然有条纹模式，但图像质量下降了很多——这就是jpeg压缩算法的工作原理（虽然有些不同但用了类似的变换）