

# 监督学习 (Supervised Learning) 第二节

作者：刘益枫 (Lewis Yik-Fung Lau)

## 目录

### 1. 凸优化

1.0 回顾 1.1 凸函数 1.2 李普希兹连续和L-光滑 1.3 强凸函数

1.4 二阶优化 1.5 其他加速方法 (重球法、NAG)

### 2. 神经网络梯度调节实践技巧

2.0 回顾 2.1 随机梯度下降 2.2 小批量梯度下降

2.3 其他梯度下降优化算法 (AdaGrad, RMSProp, AdaDelta, Adam)

### 3. 神经网络分类训练实践技巧

3.1 AlexNet中的技巧 3.2 其他技巧 (提前终止、初始化、神经网络的设计原则、梯度裁剪、批标准化、层标准化等)

### 4. 部分高级神经网络架构介绍

4.1 残差网络 4.2 密集连接网络 4.3 逆卷积和全连接网络

### 5. 总结

## 附录：参考资料

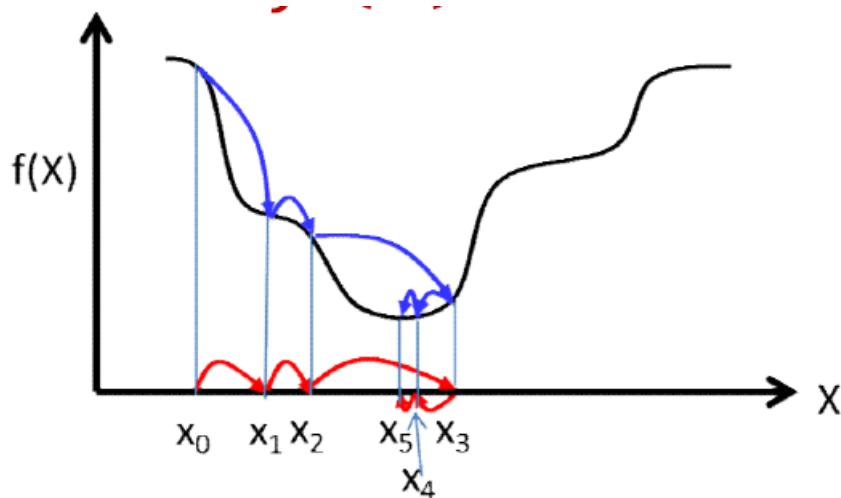
## 1. 凸优化 (Convex Optimization)

### 1.0 回顾

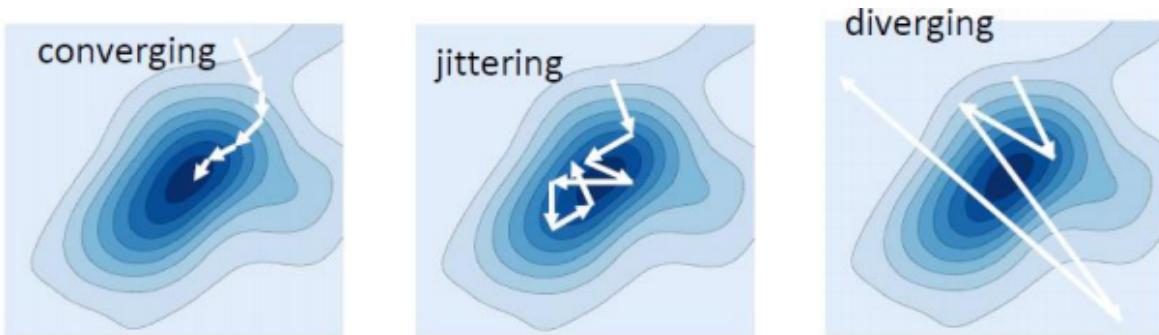
上一节课我们介绍了梯度下降 (**Gradient descent, GD**) 算法：

- 选择初始点  $X^0$
- 对于每一个  $k \geq 0$ , 利用梯度信息进行更新  $X^{k+1} = X^k - \eta^k \nabla_X f(X^k)$ , 其中  $\eta^k$  是第  $k$  步的学习率(**learning rate**)

- 较好情况：能收敛到（局部/全局）最优解：当训练次数N足够多时， $X$ 及其函数值 $f(X)$ 变化复读很小，即当  $k < N$  时， $|f(X^{k+1}) - f(X^k)| < \epsilon$



对于梯度下降算法，每一步的学习率 $\eta^k$ 至关重要，好的学习率可以收敛(**converge**)，而学习率设置不当可能会使网络在极小值附近振动(**jitter**)，还有可能会发散(**diverge**)（见下图）。



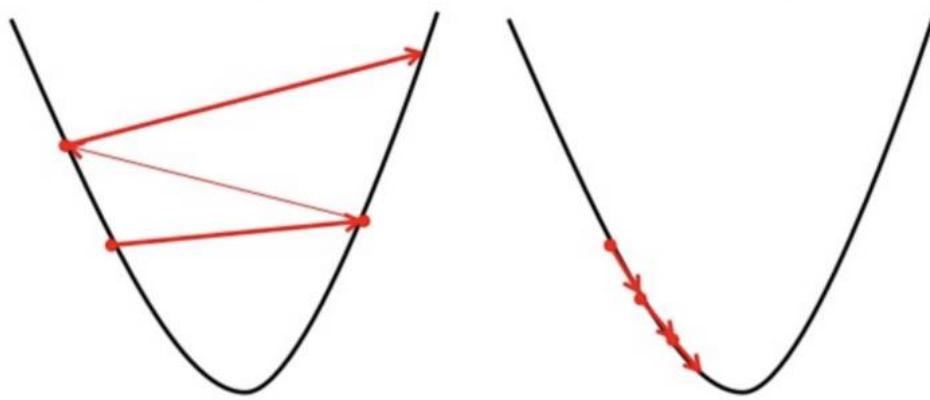
## 1.1 凸函数

在微积分课堂上，我们了解到凸函数（**convex function**, 也叫下凸函数）有一些性质：

- 全局最优解（最大最小值）处梯度为0:  $\nabla f(x) = 0$
- （由定义） $f\left(\frac{x+y}{2}\right) \leq \frac{1}{2}(f(x) + f(y))$
- 海塞矩阵(**Hessian matrix**)半正定 $\nabla^2 f(x) \succeq 0$

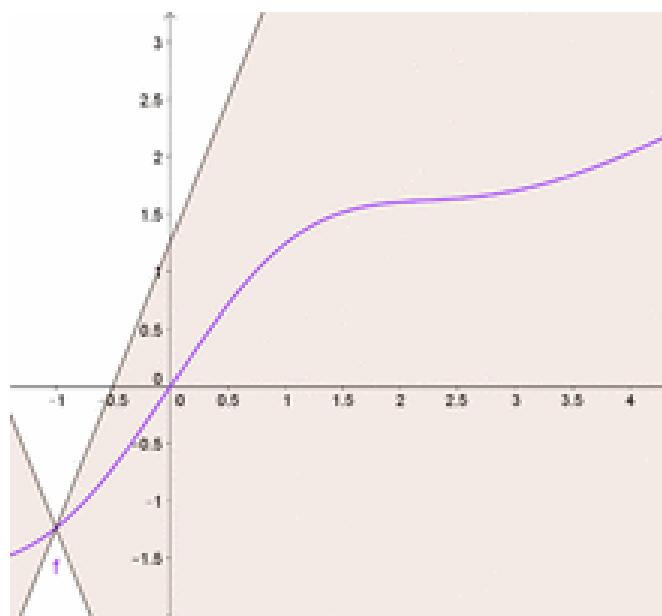
从回顾中我们看到，过大的学习率会导致发散，而适当小的学习率可以逐渐收敛到极小值处（见下图），但过小的学习率收敛速度太慢，我们应该选择一个恰当的学习率。

## Big learning rate      Small learning rate



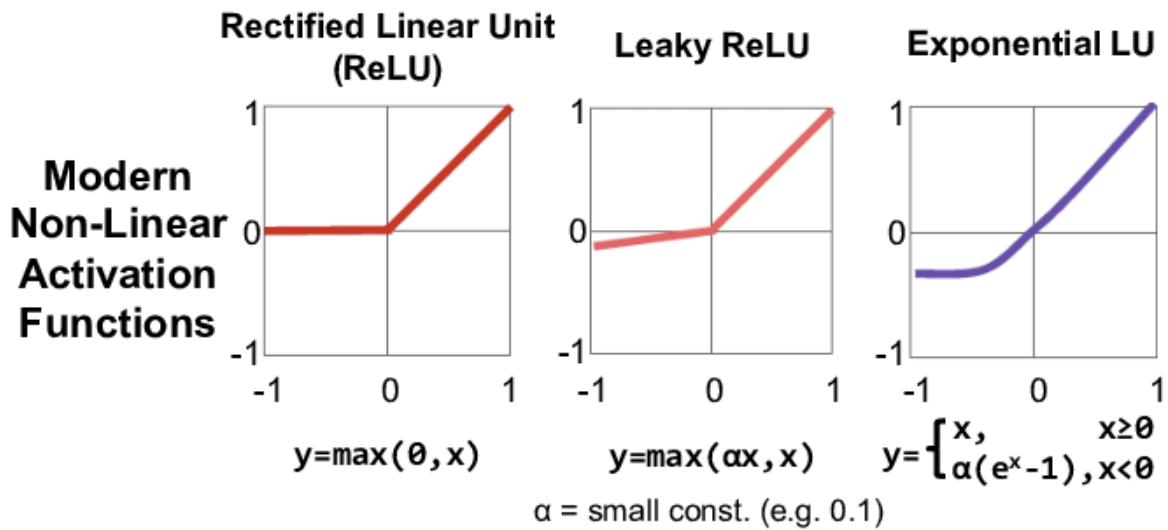
## 1.2 李普希兹连续(Lipschitz continuous)和L-光滑 (L-smoothness)

定义：如果存在一个非负实数 $L$ ，使得 $|g(x) - g(y)| \leq L|x - y|$ ，那么我们就称函数 $g(x)$ 是李普希兹连续的。如下图，在函数上任何一个点处，画两条斜率为 $\pm L$ 的斜线，函数其他点都分布在这两条斜线的夹角内（粉色部分），也即函数不能过于陡峭。



我们假设，神经网络函数 $f(x)$ 是个光滑的凸函数

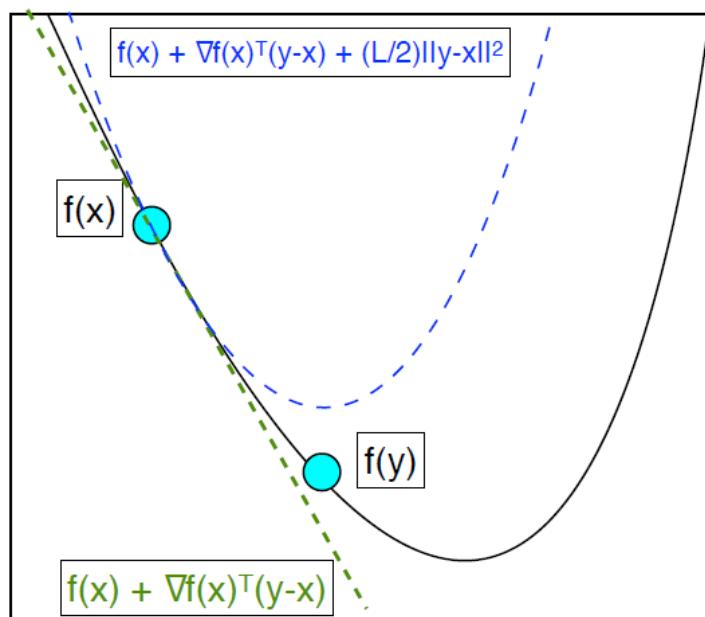
- “光滑”： $f(x)$ 的梯度是李普希兹连续的： $|\nabla f(x) - \nabla f(y)| \leq L|x - y|$ ，那么我们就说 $f(x)$ 是L-光滑的 (L-smoothness)
- 也就是函数的梯度（不是函数本身）变化较平稳、缓慢： $\nabla^2 f(x) \preceq LI$ , 其中 $I$ 是单位矩阵
- 这是一个在机器学习和神经网络实践中普遍采用的假设之一，因为实际情况下遇到的神经网络几乎都是平稳的凸函数，或者大部分地方都是平稳的和凸的。
- 例外：激活函数ReLU在0处有一个突然的转折，导致梯度不连续，因此有时会采用Leaky ReLU或者Exponential LU(ELU)函数缓解这种状况（见下图）
  - 注意，当输入值精确等于0时，ReLU函数不可导，在此时，我们默认导数为左导数，即当输入为0时导数为0。我们可以忽略这种情况，因为输入可能永远都不会为0。这里引用一句谚语，“如果微妙的边界条件很重要，我们很可能是在研究数学而非工程”，这个观点正好适用于这里。（引用自[Dive into Deep Learning — Dive into Deep Learning 0.17.4 documentation \(d2l.ai\)](#)）



## L-光滑函数的性质

### 下降引理 (Descent Lemma)

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}|x - y|^2$$



- 直观上由泰勒展开可以得到:

- $$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$$

- 证明过程:

- $$f(y) - f(x) - \langle \nabla f(x), y - x \rangle = \int_0^1 \langle \nabla f(x + t(y - x)) - \nabla f(x), y - x \rangle dt$$

- 因此,

- $$f(y) - f(x) - \langle \nabla f(x), y - x \rangle = \int_0^1 \langle \nabla f(x + t(y - x)) - \nabla f(x), y - x \rangle dt$$

- 故

$$\begin{aligned}
|f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle| &= \left| \int_0^1 \langle \nabla f(\mathbf{x} + t(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle dt \right| \\
&\leq \int_0^1 |\langle \nabla f(\mathbf{x} + t(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle| dt \\
&\leq \int_0^1 \|\nabla f(\mathbf{x} + t(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x})\|_* \cdot \|\mathbf{y} - \mathbf{x}\| dt \\
&\leq \int_0^1 tL\|\mathbf{y} - \mathbf{x}\|^2 dt \\
&= \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2
\end{aligned}$$

- 是一个二次函数形式的凸上界（见上图），因此我们可以粗略设置学习率  $\eta = \frac{1}{L}$
- 并且对于任意的  $0 < \eta < \frac{2}{L}$ ，每一步梯度更新都会使得  $f(x)$  降低
- 可以凭此推导出当学习率  $\eta = \frac{1}{L}$  时，收敛速率是亚线性的，即更新  $k = O(\frac{1}{\epsilon})$  次后， $|\nabla f(X^k)|^2 \leq \epsilon$ 。推导过程见下

- $X^{k+1} = X^k - \frac{1}{L}\nabla_X f(X^k)$
- $f(X^{k+1}) \leq f(X^k) - \frac{1}{2L}|\nabla f(X^k)|^2$  (decent lemma)
- $|\nabla f(X^k)|^2 \leq 2L(f(X^k) - f(X^{k+1}))$  (progress bound)
- $k \min_{i=0 \dots k} \{|\nabla f(X^i)|^2\} \leq 2L(f(X^0) - f(X^*))$

- 对非凸函数也适用，不过可能会陷入鞍点(saddle point)或者局部最优解
- 事实上，下降引理和L-光滑的定义，以及下面三个式子都是等价的

- $f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle \geq \frac{1}{2L}\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_*^2$
- $\langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \leq L\|\mathbf{x} - \mathbf{y}\|^2$
- $\langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq \frac{1}{L}\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_*^2$

## 通过调节学习率 $\eta$ 来估计 L

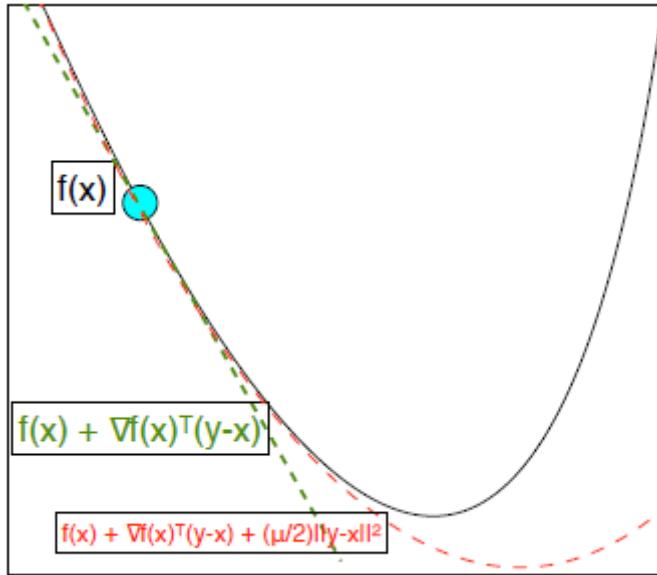
- 可以通过自动调节进行线性搜索(Adaptive learning rate with Line Search)
  - 从一个很大的  $\eta$  开始，当某些条件不满足时，降低  $\eta$ ：
    - 朴素法：如果更新的时候函数值并未下降，就降低学习率
    - 改进：Armijo 条件：对于某个  $\gamma \in (0, 1/2]$ ，如果条件  $f(w - \eta \nabla f(w^k)) \leq f(w^k) - \eta \cdot \gamma |\nabla f(w^k)|^2$  不满足就降低学习率。（又叫回溯直线搜索，Backtracking line-search）
    - 其他：例如 Wolfe 条件等（参见凸优化课）
- 实际情况：如果在验证集 (validation set) 上性能 (performance) 不再上升，就降低学习率至  $\alpha$  倍，即  $\eta \rightarrow \alpha \cdot \eta$

## 1.3 强凸 (Strongly convex) 函数

对于凸函数我们有比较好的结论，那对更特别的凸函数，我们是否有更好的结论呢？下面我们引入强凸函数的概念。

如果对于某个正实数 $\mu > 0$ , 有 $f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2}|y - x|^2$ , 我们就说函数 $f(x)$ 是 $\mu$ -强凸的 ( $\mu$ -strongly convex)。

- 与L-光滑函数类似,  $\mu$ -强凸的可以看做一个二次函数形式的凸下界 (见下图)



- 类比于L-光滑函数, 我们有以下5种等价结论:

$$\begin{aligned}
 f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle &\geq \frac{\mu}{2} \|\mathbf{x} - \mathbf{y}\|^2 \\
 \langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle &\geq \mu \|\mathbf{x} - \mathbf{y}\|^2 \\
 \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_* &\geq \mu \|\mathbf{x} - \mathbf{y}\| \\
 f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle &\leq \frac{1}{2\mu} \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_*^2 \\
 \langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle &\leq \frac{1}{\mu} \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_*^2
 \end{aligned}$$

- 梯度下降中的性质: L-光滑的 $\mu$ -强凸函数是线性收敛的, 即更新 $k = O(\frac{L}{\mu} \log \frac{1}{\epsilon})$ 次后,  $|f(X^k) - f(x_{optimal})| \leq \epsilon$ ; 且我们有 $L\mathbf{I} \succeq \nabla^2 f(x) \succeq 0$ 。
- 证明过程如下:

- 假设 $x^*$ 是最优点, 最开始时处于 $x^0$ 处, 且 $\|x^0 - x^*\| = R$ 。由L-光滑函数和 $\mu$ -强凸函数的性质, 我们有

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2L} \|\nabla f(x^k)\|^2$$

◦

$$f(x^{k+1}) \geq f(x^k) - \frac{1}{2\mu} \|\nabla f(x^k)\|^2$$

- 因为 $x^*$ 是最优点, 则有 $\nabla f(x^*) = 0$ , 且

$$f(x^k) - f(x^*) \geq \frac{1}{2L} \|\nabla f(x^k)\|^2$$

◦

$$f(x^k) - f(x^*) \leq \frac{1}{2\mu} \|\nabla f(x^k)\|^2$$

◦ 那么

$$f(x^{k+1}) - f(x^*) \leq f(x^k) - f(x^*) - \frac{1}{2L} \|\nabla f(x^k)\|^2$$

◦

$$\leq f(x^k) - f(x^*) - \frac{\mu}{L} (f(x^k) - f(x^*))$$

$$= (1 - \frac{\mu}{L}) (f(x^k) - f(x^*))$$

◦ 通过叠合 (telescoping) , 我们可以得出

$$◦ f(x^k) - f(x^*) \leq (1 - \frac{\mu}{L})^k (f(x^0) - f(x^*))$$

◦ 又根据L-光滑函数和 $\mu$ -强凸函数的性质,

$$f(x^k) - f(x^*) \geq \frac{\mu}{2} \|x^k - x^*\|^2 = \frac{\mu}{2} \epsilon^2$$

◦

$$f(x^0) - f(x^*) \leq \frac{L}{2} \|x^0 - x^*\|^2 = \frac{L}{2} R^2$$

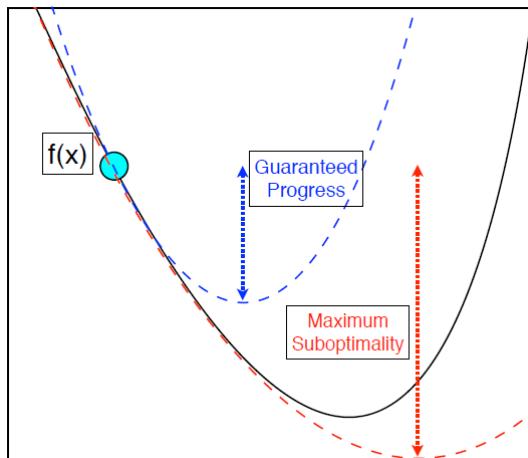
◦ 则

$$◦ \mu \epsilon^2 \leq (1 - \frac{\mu}{L})^k L R^2$$

$$◦ \text{也即 } k = O(\frac{L}{\mu} \log \frac{1}{\epsilon})$$

• 直观上:

◦ 有界性: L-光滑和强凸性把函数限定在两个二次函数之间, 使得函数在梯度下降的过程中在一个有限的范围内变化。



- 次优性 (sub-optimality) : 虽然不能保证达到最优点, 但每一次更新都能至少把到最优点的距离缩小几分之一。

- 我们可以用权值衰减 (weight decay) 来使凸函数变成强凸函数，从而得到更好的性质

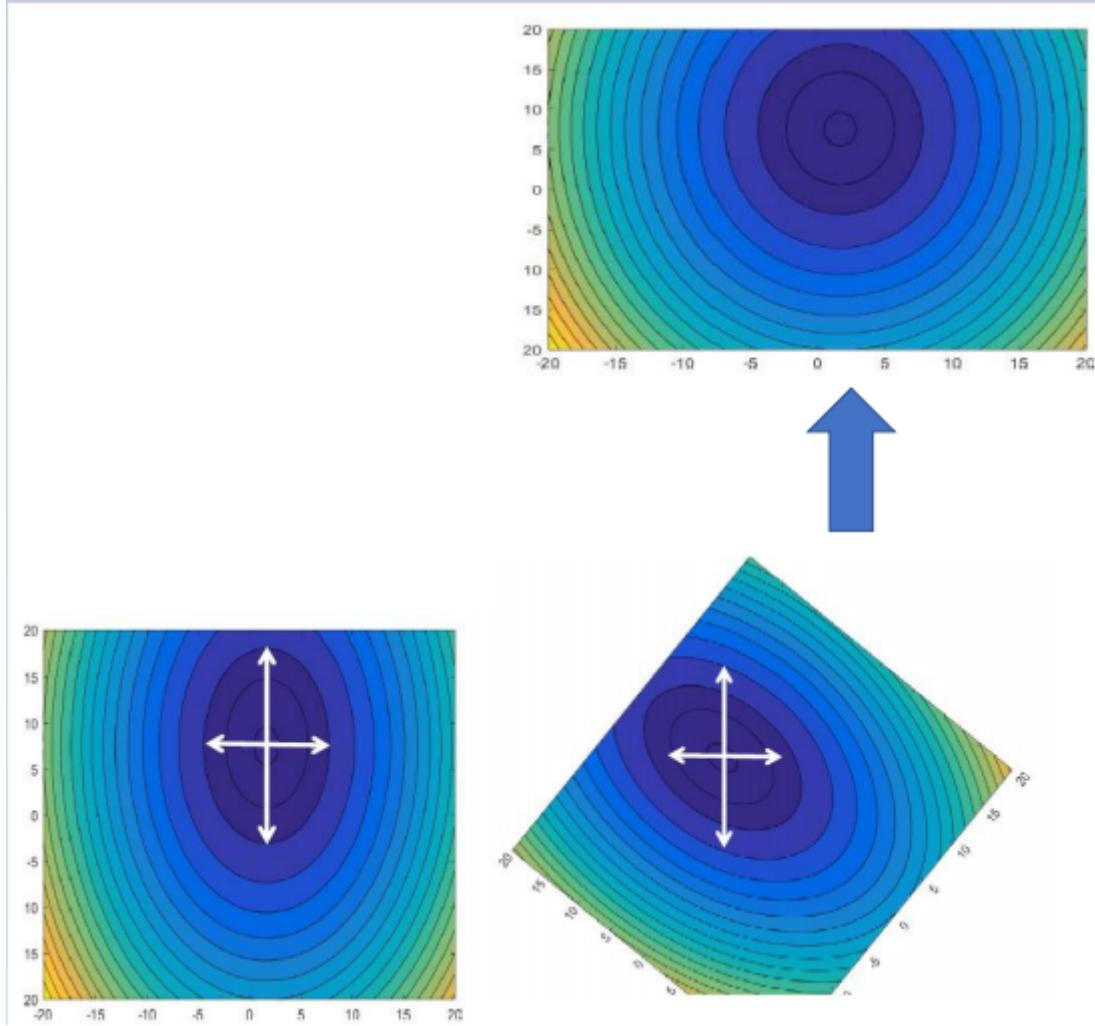
◦ 例如L2-正则化 (L2-regularization) :  $f(x) \rightarrow f(x) + \alpha|x|^2$ 。

◦ Tips: 正则化有时可以给你一些更好的性质，例如增加强凸性或者防止过拟合。

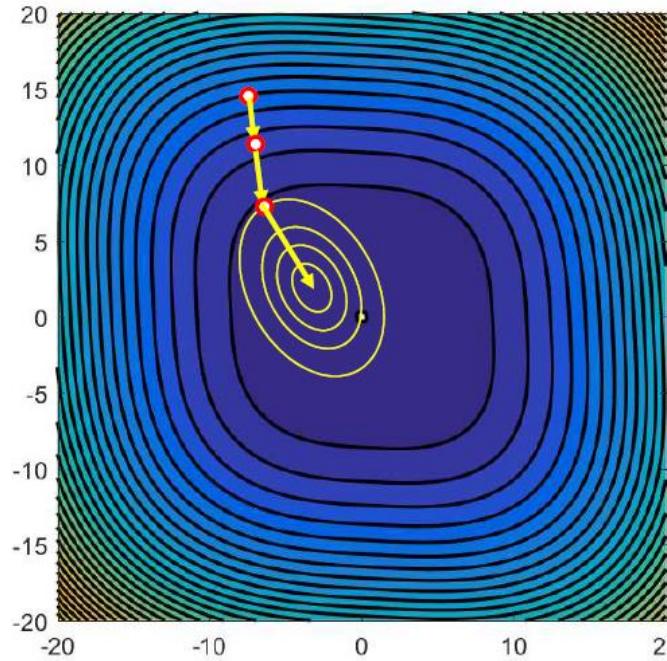
## 1.4 二阶优化 (Second-Order Optimization)

对于L-光滑的强凸函数的一阶优化， $X^{k+1} = X^k - \frac{1}{L} \nabla_X f(X^k)$ ，每次更新时的学习率一样，但还能做得更好。

- 考虑一个二次函数  $y_1 = ax_1^2 + bx_2^2$ ，对  $x_1$  和  $x_2$  有不同的最佳学习率。
- 引进交叉项，即  $y_2 = ax_1^2 + bx_2^2 + cx_1x_2$ ，我们可以旋转和放缩坐标轴以消去交叉项。



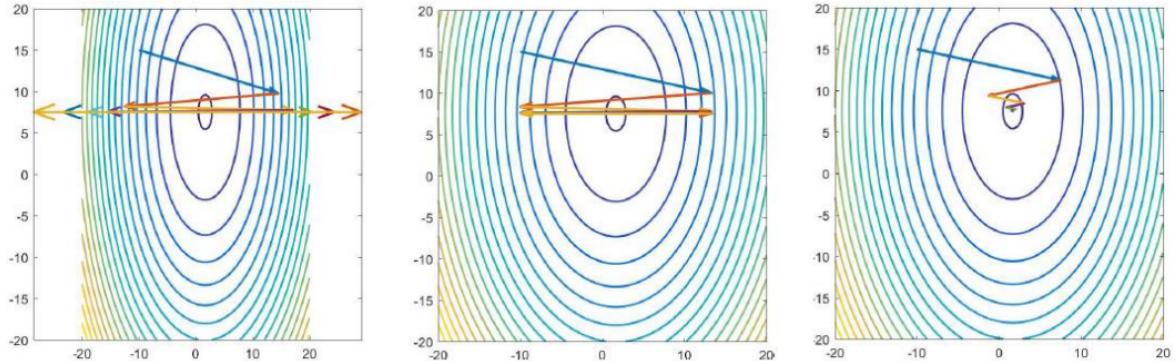
- 再考虑多个变量，写成矩阵形式，即  $y = \frac{1}{2} X^T A X + b X + c$ ，我们可以用  $\hat{X} = A^{0.5} X$  替换  $X$  (其中  $A$  正定)，从而有  $y = \frac{1}{2} \hat{X}^T \hat{X} + b A^{-0.5} \hat{X} + c$ 。
- 对  $\hat{X}$  采取梯度下降  $\hat{X}^{k+1} = \hat{X}^k - \eta \nabla f(\hat{X}^k)$ ，还原后有  $X^{k+1} = X^k - \eta A^{-1} \nabla_X f(X^k)$ 。
- 考虑泰勒展开  $f(y) \leq f(x) + \nabla f(x)^T (y - x) + (y - x)^T \nabla^2 f(x) (y - x)$  的二阶项，可以构造二阶优化： $X^{k+1} = X^k - \eta^k \nabla^2 f(X^k)^{-1} \nabla_X f(X^k)$ 。而根据强凸性和其海塞矩阵的L-光滑性，二阶优化法可以达到二次收敛速率（见下图）。



局限性：计算海塞矩阵本身代价就高，计算海塞矩阵的逆更加困难；并且对于非凸函数，因为有负的奇异值，可能会导致发散。而发散问题常常是由我们强行使每一个参数的学习率相同导致的。

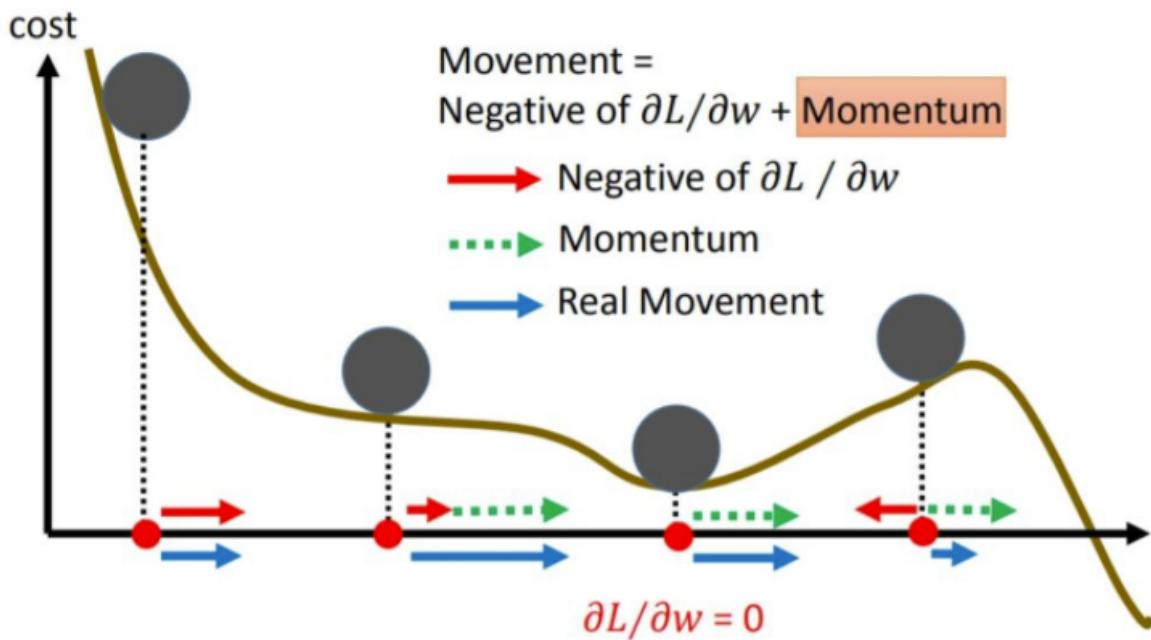
## 1.5 其他加速方法

根据上面的分析，我们知道不同的维度用相同学习率可能导致发散等问题：有时候函数在一些维度上收敛，而在另一些维度上不停震荡，甚至发散（见下图，在纵向上收敛，在横向上可能收敛、震荡或者发散）。因此我们可以设计一些方法增大第一种维度上的步长，同时减小在第二种维度上的步长。



### 1.5.1 重球法 (The heavy-ball method) 或者Polyak动量法

类比于一个重球在高低起伏的地形上能够下降到低处的物理模型，我们不仅考虑地形的坡度（梯度），还考虑下降过程中的速度或者动量（**momentum**）。

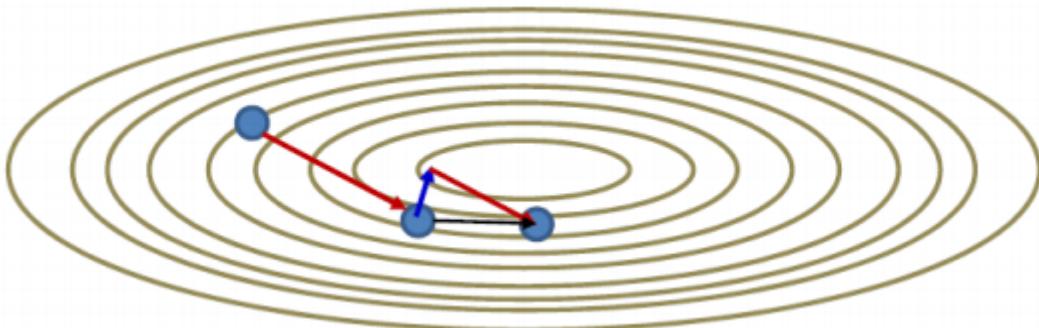


- 根据前一时刻的动量和当前所处位置的梯度计算当前时刻的动量：  

$$\Delta X^k = \beta \Delta X^{k-1} - \eta \nabla f(X^k)$$
, 通常情况取 $\beta = 0.9$ 。
- 根据当前时刻的动量计算下一时刻的位置：  

$$X^{k+1} = X^k + \Delta X^k$$
。
- 总的表达式：  

$$X^{k+1} = X^k - \eta \nabla f(X^k) + \beta(X^k - X^{k-1})$$
。也就是先梯度下降，再（根据惯性）平移上一步的 $\beta$ 倍。（如下图）

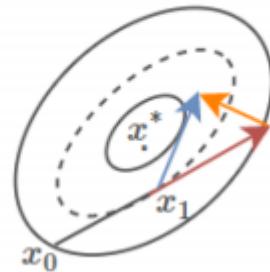
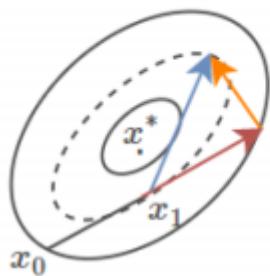


- 直观理解：
  - 当梯度（坡度）和当前动量方向一致的时候，更新时的步长大；
  - 当梯度（坡度）和当前动量方向相反的时候，更新时的步长小。
- 然而，重球法并不一定收敛.....

### 1.5.2 Nesterov加速梯度下降法 (简称NAG)

$$X^{k+1} = X^k - \eta \nabla f(X^k + \beta(X^k - X^{k-1})) + \beta(X^k - X^{k-1})$$

- 原论文 (Nesterov, 1983) 并不是关于随机梯度下降的，也没有明确使用梯度下降方程；而 Nesterov 加速梯度下降法主要是由 Sutskever 等人在 2013 年提出（详见原论文 [On the importance of initialization and momentum in deep learning](#)）。
- 与重球法不同之处：先（根据惯性）平移上一步的 $\beta$ 倍（最后一项）；再根据平移后的梯度信息进行梯度下降。（见下图）



- 收敛速率对比 (其中 $\kappa = \frac{L}{\mu}$ ,  $L \gg \mu$ ) :

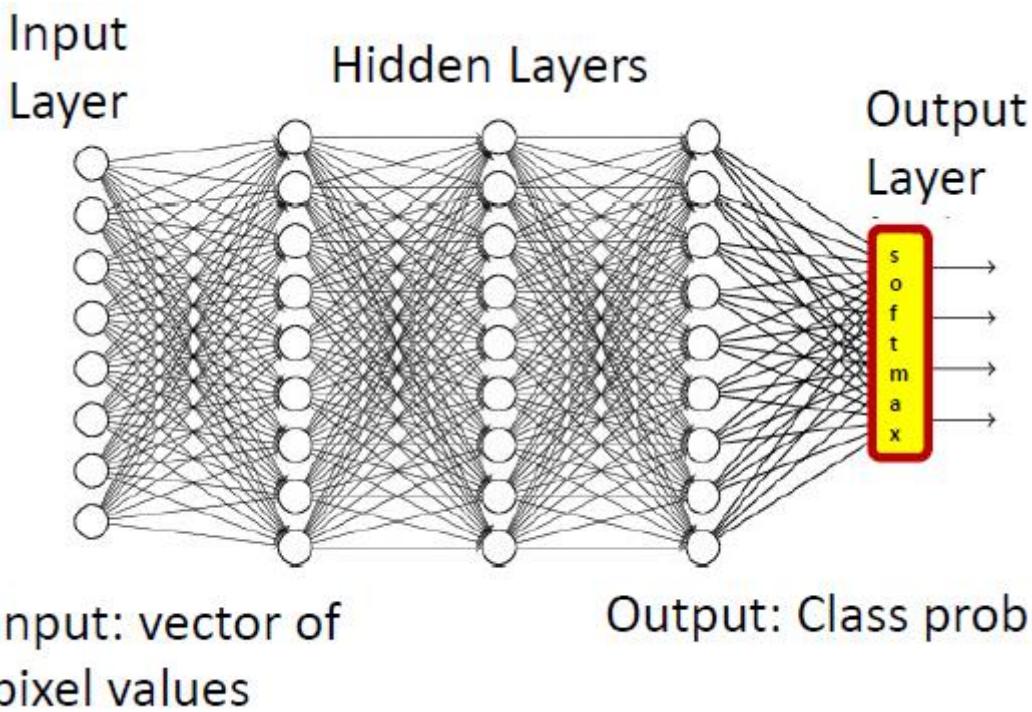
Class of Function	GD	NAG
Smooth	$O(1/T)$	$O(1/T^2)$
Smooth & Strongly-Convex	$O(\exp(-\frac{T}{\kappa}))$	$O\left(\exp\left(-\frac{T}{\sqrt{\kappa}}\right)\right)$

## 2. 神经网络梯度调节实践技巧

尽管大家都是CNN，为什么有些人训练效率比别人高，有些人网络虽然比较小巧，但效果却更好？这就需要用到一些神经网络的实践技巧了。

### 2.0 回顾

上一节课我们还了解了训练神经网络的基本架构：



- 给定训练数据 (集)  $X = \{(x^i, y^i)\}$
- 需要设计 (并训练) 一个神经网络  $y = f(x; \theta)$
- 还要定义损失函数 (**loss function**)  $L(\theta) = \frac{1}{N} \sum_i err(f(x^i; \theta); y^i)$  以评判神经网络的质量

- 目标是找到一个最优的  $\theta$  使得损失函数  $L(\theta)$  最小

然而，这个神经网络非常复杂，且很多情况下都是非凸函数，局部最优点通常不是全局最优点，我们需要用上一部分所讲的方法对一般的梯度下降法进行优化。

我们考虑传统的梯度下降法：

- 对每一个样例进行反向传播 (**Backpropagation**) :  $L_i \theta = \nabla_{\theta} err(f(x^i; \theta); y^i)$
- 计算平均梯度:  $\nabla L(\theta) = \frac{1}{N} \sum_i \nabla_{\theta} L_i(\theta)$
- 进行梯度更新:  $\theta^{k+1} \leftarrow \theta^k - \eta^k \nabla L(\theta^k)$

但如果样例数量  $N$  特别大怎么办？

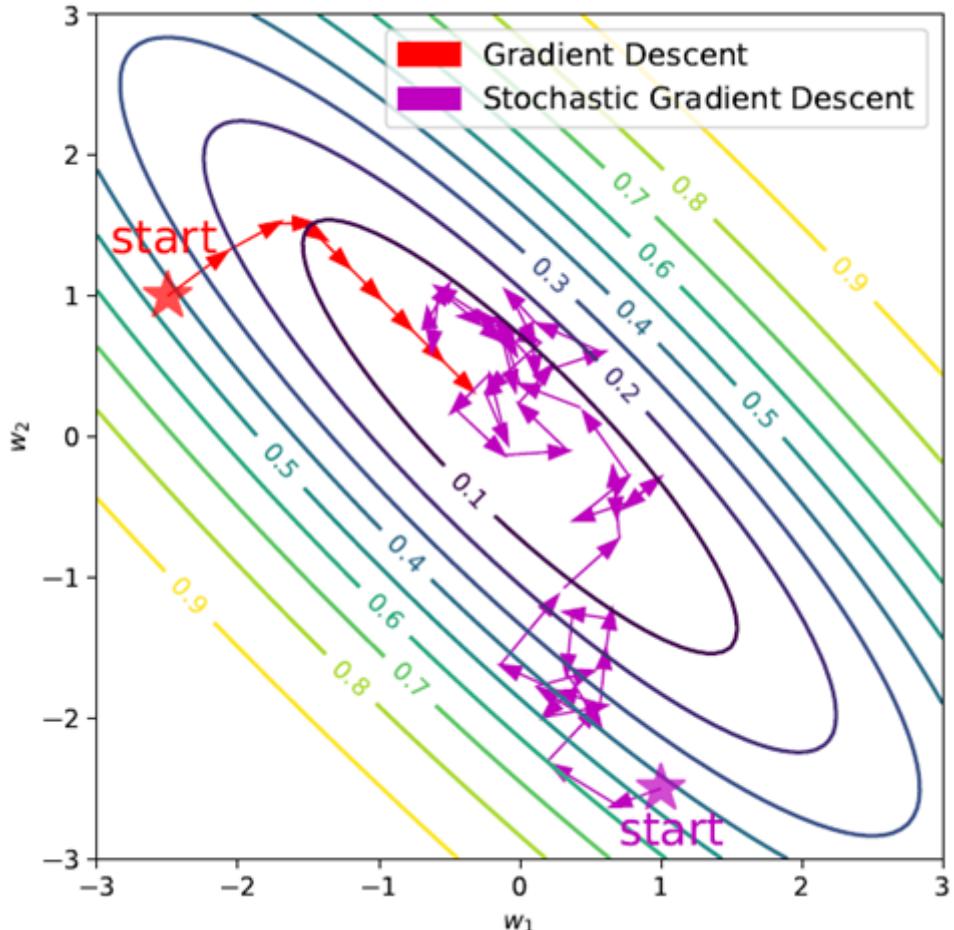
## 2.1 随机梯度下降 (Stochastic Gradient Descent, SGD)

### 步骤

- 给定训练数据  $\mathcal{X} = \{(x^i, y^i)\}$
- 在其中随机选取一个样例点  $(x^j, y^j) \in \mathcal{X}$
- 根据这个样例点的梯度信息进行更新:  $\theta^{k+1} \leftarrow \theta^k - \eta^k \nabla L_j(\theta^k)$

### 注记

- 平均梯度  $\nabla L(\theta)$  是对样例点梯度的无偏估计:  $\nabla L(\theta) = E_j[\nabla L_j(\theta)]$
- 随机梯度下降效率有时较高，但方差较大，不太稳定（见下图）
- 可以按顺序循环选取样例点，使得样例点被均匀采取到



## 收敛条件

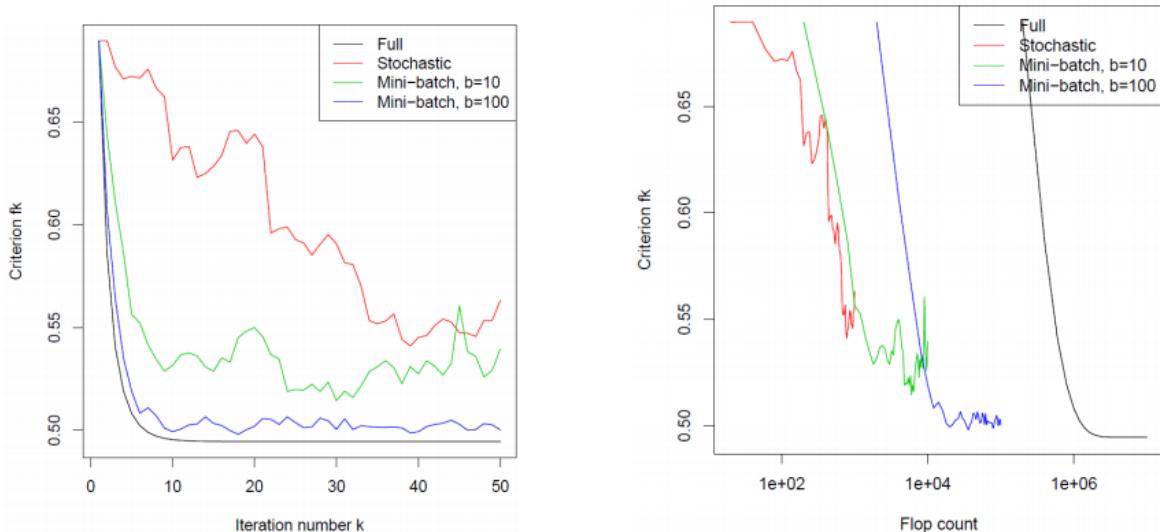
- 需要不断减小的学习率:  $\sum_k \eta^k = \infty$  以及  $\sum_k \eta^{k^2} < \infty$
- 通常选取  $\eta^k \propto \frac{1}{k}$

## 收敛速率

- 对于凸函数:  $E[f(X^k)] - f^* = O(\frac{1}{\sqrt{k}})$ 。 (但L-光滑函数并没有更好的性质)
- 强凸函数:  $E[f(X^k)] - f^* = O(\frac{1}{k})$ 
  - 对比梯度下降: L-光滑的凸函数收敛速率为  $O(1/k)$ , 强凸函数线性收敛
  - 可以看出SGD稍微比GD收敛速率慢一点

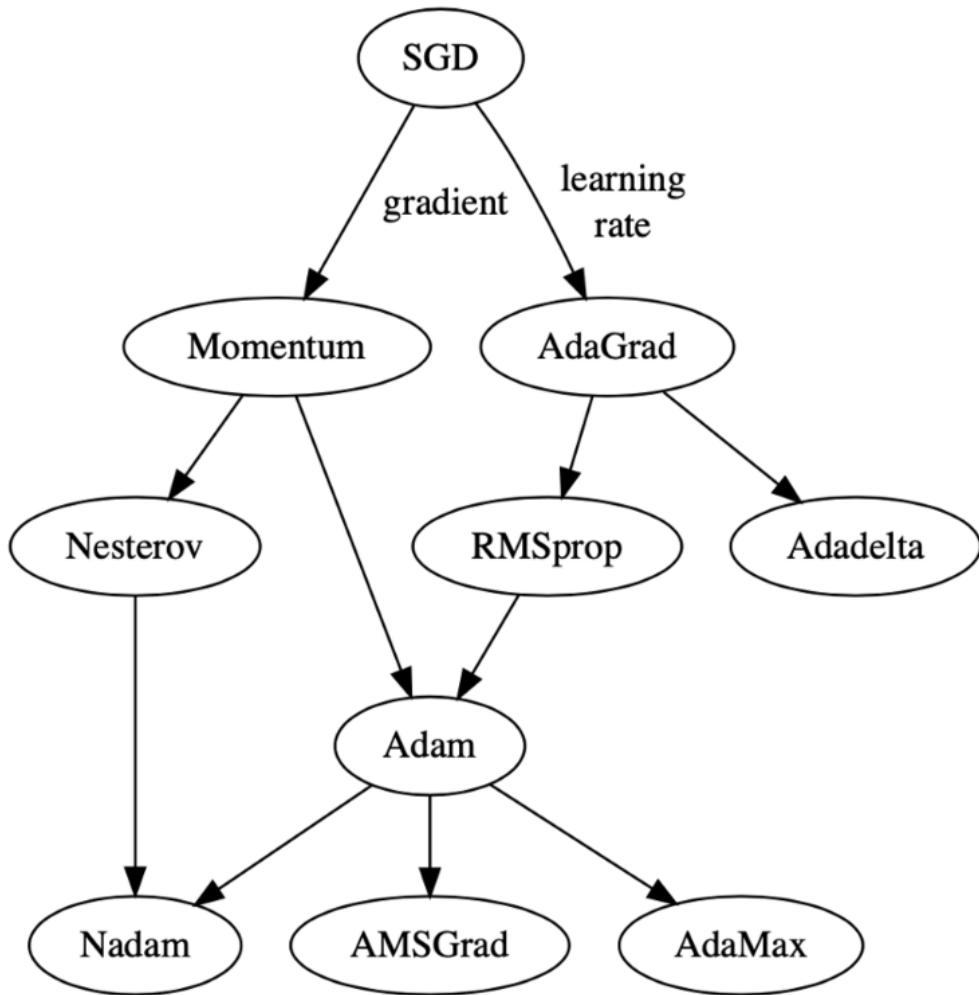
## 2.2 小批量 (Mini-batch) 梯度下降

- 在  $\{1, \dots, N\}$  中随机选取  $b$  个 (称为批大小, **batch-size**) 编号组成集合  $I_j$
- 用小批量的平均梯度估计整体平均梯度  $\nabla L_{I_j}(\theta) = \frac{1}{b} \sum_{i \in I_j} \nabla L_i(\theta)$  (仍然是无偏估计, 并且把梯度方差降低至  $\frac{1}{b}$ )
- $\theta^{k+1} \leftarrow \theta^k - \eta^k \nabla L_{I_j}(\theta^k)$
- 优点: 计算速度比GD快的同时, 方差比SGD小 (当然较大的方差/噪声可能可以帮助函数逃离鞍点和局部最优解)
  - 通常情况下, 假设鞍点 (**saddle point**, 梯度为0但海塞矩阵既不正定也不负定) 比局部最优解多得多, 且绝大多数局部最优解都是相同的且接近全局最优解, 但对于小的网络或者深度强化学习等领域不太适用。
- 实践中, 我们随机将数据集分成  $\frac{N}{b}$  个小批, 每个训练回合 (**epoch**) 用一个小批训练; 并且如果计算性能允许, 我们从较大的批大小开始尝试训练。
- 实例: 逻辑斯蒂回归 (**Logistic regression**, 有强凸性), 有20个维度,  $10^4$  个数据点, 学习率固定, 实验结果如下图:

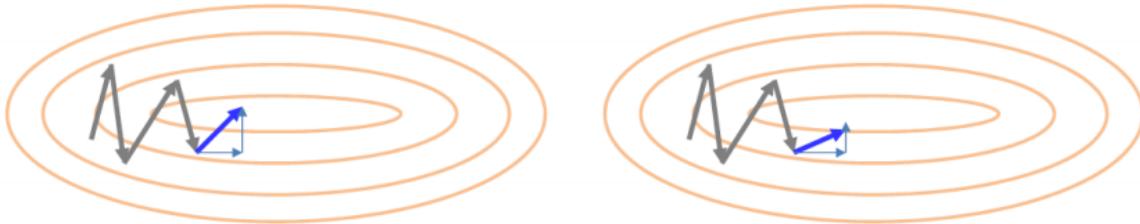


## 2.3 其他梯度下降优化算法

我们还能再改进吗? 从凸优化学习中我们有一些想法, 例如设置可以自适应的学习率, 不同的维度可以有不同的学习率, 二阶优化方法和动量法或许也有帮助。实际上, 有一些模型在SGD的基础上通过调节梯度和学习率等进行了进一步的改进, 例如AdaGrad, RMSProp, AdaDelta, Adam等等 (如下图, R. Karim: [Gradient Descent Optimization Algorithms - Cheat Sheet](#))。



从另一个角度看，标准的加速方法仍然会产生很大的振动。由下图，我们可以发现，在振动的方向上步长较大，而在真正的梯度方向上步长较小，我们可以在振动较大的方向上降低学习率。



### 2.3.1 AdaGrad (自适应学习率) 算法 ([Duchi et al, 2011](#))

#### 步骤

- $g^k = \nabla_{\theta} f(\theta^k)$ : 计算梯度
- $G_i^k = G_i^{k-1} + |g_i^k|^2$ : 对每个参数计算历史梯度的平方和
- $\theta_i^{k+1} = \theta_i^k - \frac{\eta}{\sqrt{G_i^k + \epsilon}} g_i^k$ : 对每个参数进行更新。

#### 注记

- AdaGrad算法不需要手动调节学习率，因为根据第三个步骤，更新速率会自动降低（实际上我们固定学习率 $\eta = 0.01, \epsilon = 10^{-8}$ ）
- 每个参数有不同的学习率，且用历史梯度信息进行正则化，并加上一个少量防止分母为0。
  - GloVe算法也采用AdaGrad算法做词嵌入，因此更低频的词语有更高的学习率。
- 缺点：学习率下降太快。

### 2.3.2 RMSProp (均方根传递) 算法 (Hinton, ~ 2012)

#### 步骤

- $g^k = \nabla_{\theta} f(\theta^k)$ : 计算梯度
- 改进:  $G_i^k = \gamma G_i^{k-1} + (1 - \gamma)|g_i^k|^2$ : 估计梯度的方均根
- $\theta_i^{k+1} = \theta_i^k - \frac{\eta}{\sqrt{G_i^k + \epsilon}} g_i^k$ : 对每个参数进行更新。

#### 注记

- RMSProp算法改良了AdaGrad学习率下降太快的问题，尤其适用于RNN。

### 2.3.3 AdaDelta (自适应增量) 算法 (Zeiler, 2012, 和RMSProp同时)

#### 步骤

- $g^k = \nabla_{\theta} f(\theta^k)$ : 计算梯度
- $G_i^k = \gamma G_i^{k-1} + (1 - \gamma)|g_i^k|^2$ : 估计梯度平方的平均值
- 改进1:  $E[\Delta\theta^2]_i^k = \gamma E[\Delta\theta^2]_i^{k-1} + (1 - \gamma)(\theta_i^k - \theta_i^{k-1})^2$ : 估计梯度变化大小的方均根
- 改进2: 更新公式:  $\theta_i^{k+1} = \theta_i^k - \frac{\sqrt{E[\Delta\theta^2]_i^k + \epsilon}}{\sqrt{G_i^k + \epsilon}} g_i^k$ 。

#### 注记

- AdaDelta算法不再需要学习率：用 $\Delta\theta$ 的方均根动态更新学习率
- 从量纲的角度进行分析：
  - 在SGD中， $\Delta x$ 正比于梯度 $\frac{\partial f}{\partial x}$ ，即单位为 $\frac{1}{x}$ 的单位；
  - 在牛顿梯度下降法中， $\Delta x$ 正比于 $\frac{\frac{\partial f}{\partial x}}{\frac{\partial^2 f}{\partial x^2}}$ ，即单位和x单位相同；
  - AdaDelta用近似海塞矩阵纠正了量纲问题： $\Delta x = \frac{\frac{\partial f}{\partial x}}{\frac{\partial^2 f}{\partial x^2}}$  得到 $\frac{1}{\frac{\partial^2 f}{\partial x^2}} = \frac{\Delta x}{\frac{\partial f}{\partial x}}$ ，从而有  
$$\Delta x_t = -\frac{RMS[\Delta x]_{t-1}}{RMS[g]_t} g_t$$
。

### 2.3.4 Adam (适应性矩估计) 算法(Diederik P. Kingma & Jimmy Ba, 2014)

——有动量的RMSProp，最广泛使用的优化器 (optimizer)，引用量高达6.7万！

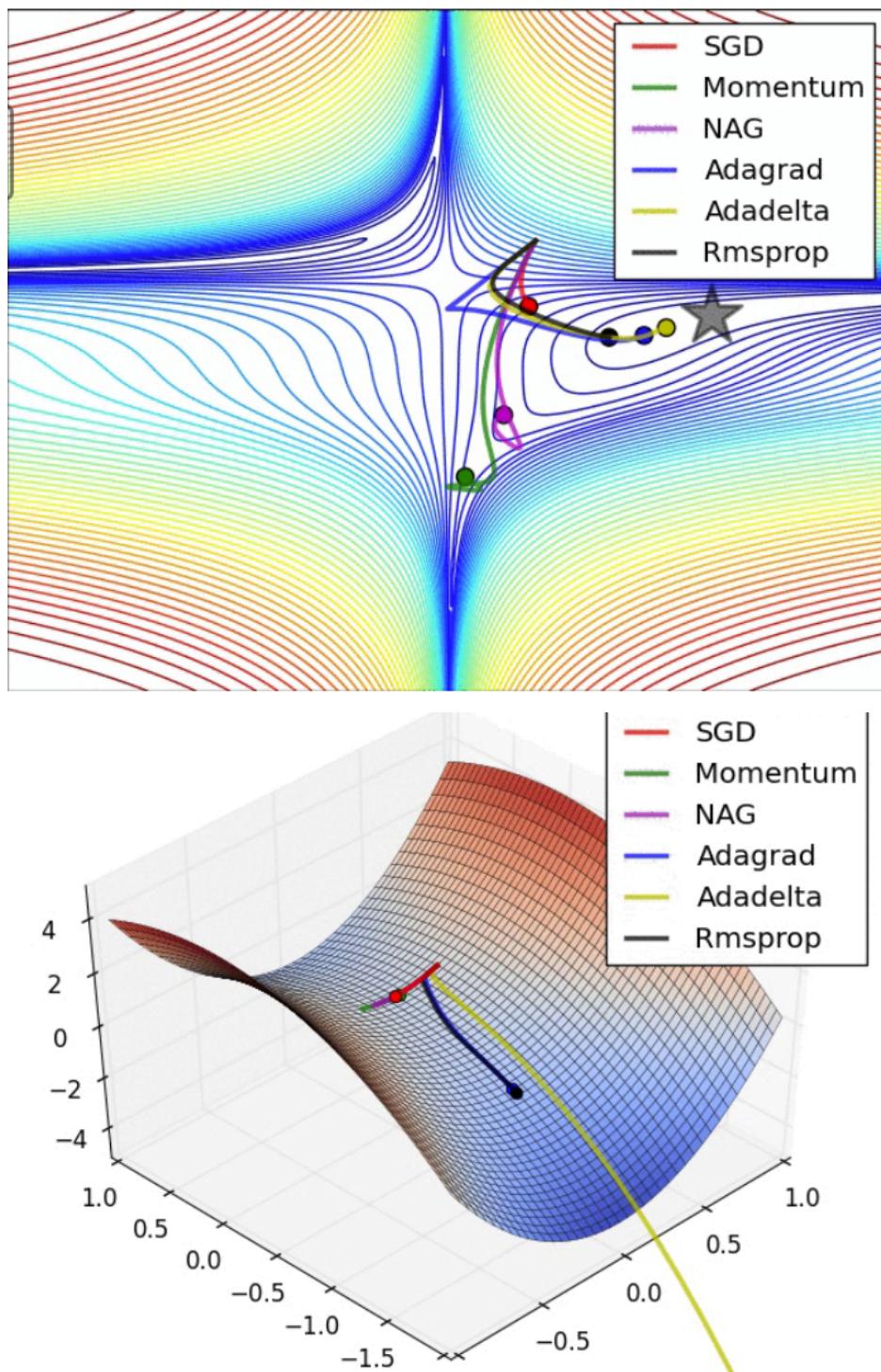
#### 步骤

- $g^k = \nabla_{\theta} f(\theta^k)$ : 计算梯度
- 改进1:  $M_i^k = \delta M_i^{k-1} + (1 - \delta)g_i^k$ : 动量
- $G_i^k = \gamma G_i^{k-1} + (1 - \gamma)|g_i^k|^2$ : 估计梯度的方均根
- 改进2:  $\hat{M}^k = \frac{\hat{M}^k}{1 - \delta^k}$ ,  $\hat{G}^k = \frac{G^k}{1 - \gamma^k}$ : 使得 $\gamma$ 和 $\delta$ 在早期的时候不会占据主导地位。
- 更新公式:  $\theta_i^{k+1} = \theta_i^k - \frac{\eta}{\sqrt{\hat{G}_i^k + \epsilon}} \hat{M}_i^k$ 。

#### 注记

- Adam算法对RNN、生成模型和强化学习尤其适用。

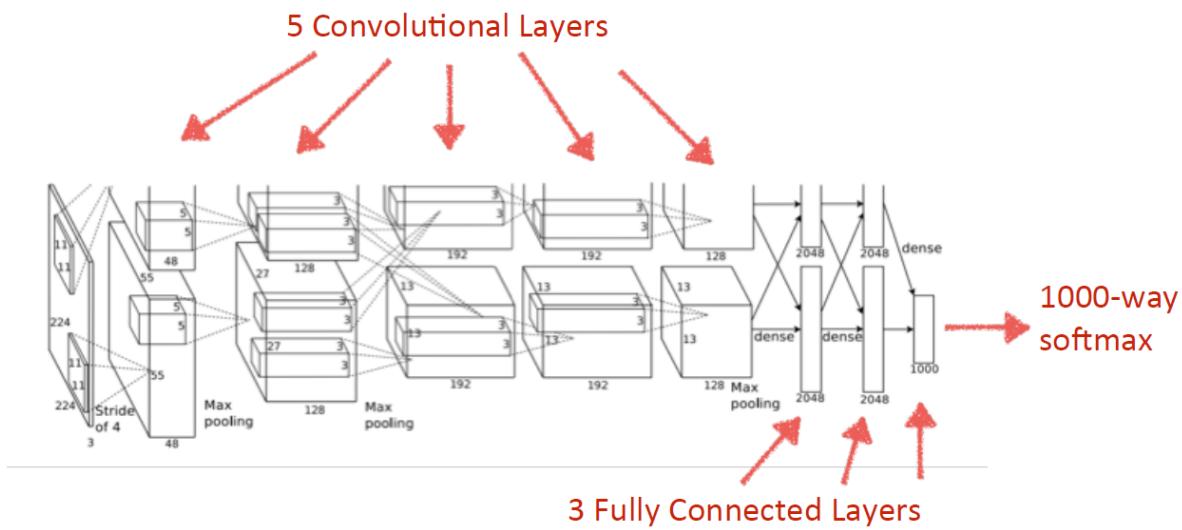
通过可视化直观理解各优化法机制 (Alec Radford, [OpenAI: Github](#))



### 3. 神经网络分类训练实践技巧

神经网络训练中很有可能会过拟合，这是因为神经网络通常是广义上的函数，当权重过大时，会出现过拟合现象，这时我们可以采用L2-正则化等权重衰减（**weight decay**）方法避免。不过我们还有其他技巧。

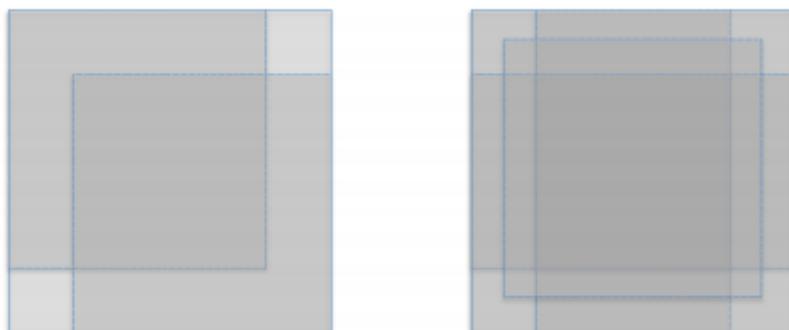
#### 3.1 AlexNet中的技巧([Alex Krizhevsky, et al., 2012](#))



AlexNet是深度学习领域对图像分类问题上第一个突破，采用了很多技巧：

- 利用ReLU作为激活函数并采用交叠的池化函数。
- 进行一些数据预处理和增广：
  - 每个像素都减去训练集中的平均信息：使训练集数据对称分布
  - 将图片（和水平对称像）裁剪成224x224份小块进行训练
  - 在测试时将其中10块的预测值进行平均得到最终预测
  - 改变RGB信道的强度，增加主成分分析的成分。

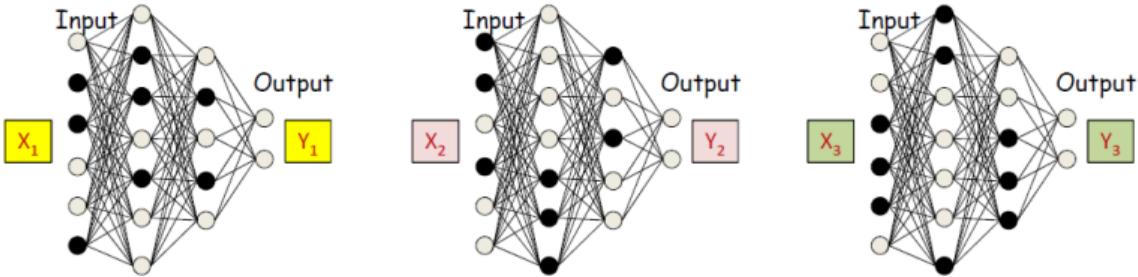
$$[I_R, I_G, I_B] \doteq \alpha [P_R, P_G, P_B] [\lambda_R, \lambda_G, \lambda_B]^T, \alpha \sim N(0, 0.1)$$



- 通过对图片进行旋转、拉伸、翻转等进行数据增广（见下图）

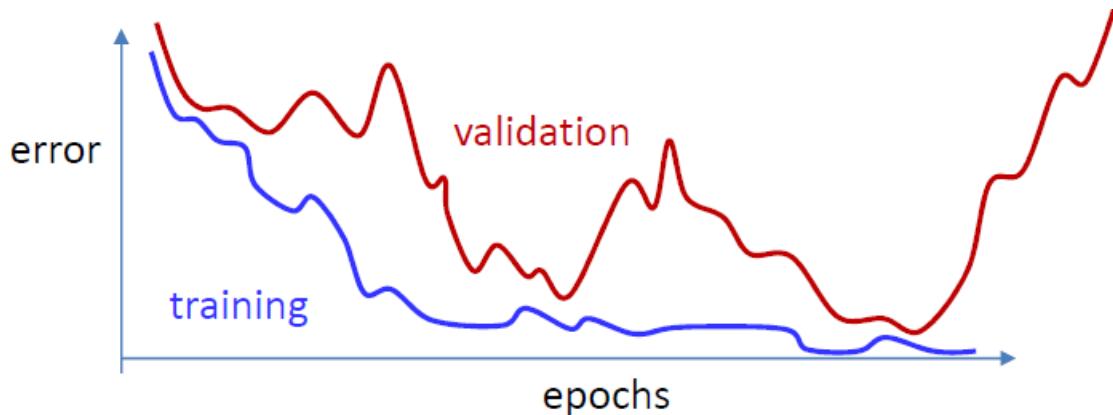


- 随机失活 (Dropout) :
  - 对每个训练样例，在每次迭代过程中，对每个神经元都以 $1 - \alpha$ 的概率随机失活，即随机切断神经元之间的联系。
  - 实践中，每个神经元都遵循 $(1 - \alpha)$ 的伯努利分布；而梯度只在未失活的神经元中传递。
  - 可以有效防止过拟合：强制神经网络学习冗余的信息，并且可以看做隐式的L2-正则化方法。
  - 因为随机失活改变了输出神经元的大小： $y = \text{Dropout}(\sigma(\sum_i w_i x_i + b))$ ，  
 $E[y] = \alpha E[\sigma(\sum_i w_i x_i + b)]$ ；故在测试中，没有神经元被失活时，对输出神经元乘上 $\alpha$ 作为补偿： $y = \alpha \sigma(\sum_i w_i x_i + b)$



## 3.2 其他技巧

- 提前终止 (Early Stopping)
  - 过度训练可能导致过拟合
  - 在验证集上追踪性能的变化

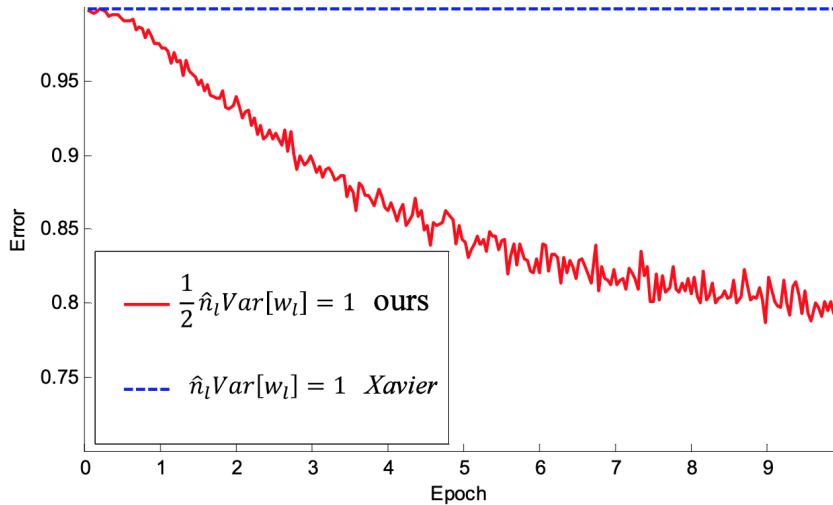


- 初始化
  - 随机初始化（因为0初始化使得所有神经元学到同样的东西），通常遵循高斯分布
  - 初始一个少量（太大的初始化参数可能带来梯度爆炸）
  - 更好的初始化方法：Xavier初始化法 (Xavier Glorot & Yoshua Bengio, AISTATS10)
    - $b^{(k)} \leftarrow 0$ : 偏差 (bias) 置为0
    - $W^{(k)} \sim \text{unif} \pm \frac{\sqrt{6}}{\sqrt{n_k + n_{k+1}}}$ :  $n_k$  为第 $k$ 个隐藏层的大小 (扇入大小, fan-in)， $n_{k+1}$  为第 $k+1$ 个隐藏层的大小 (扇出大小, fan-out) ,
      - 假设第 $k$ 层权重的分布具有零均值和方差 $\sigma^2$ 输入 $x_k$ 具有零均值和方差 $\gamma^2$ ，并且它们独立于此层权重 $w_{ij}$ 且彼此独立，则可以算出此层输出 $o_i$ 方差 $\text{Var}[o_i] = n_k \sigma^2 \gamma^2$
      - 保持方差不变的一种方法为 $n_{in} \sigma^2 = 1$ ，但反向传播时如果还要保持方差不变，则需要有 $n_{k+1} \sigma^2 = 1$ ，进行折中， $\sigma = \sqrt{\frac{2}{n_k + n_{k+1}}}$ 。
      - 如果将正态分布改为均匀分布，则初始化的范围为  

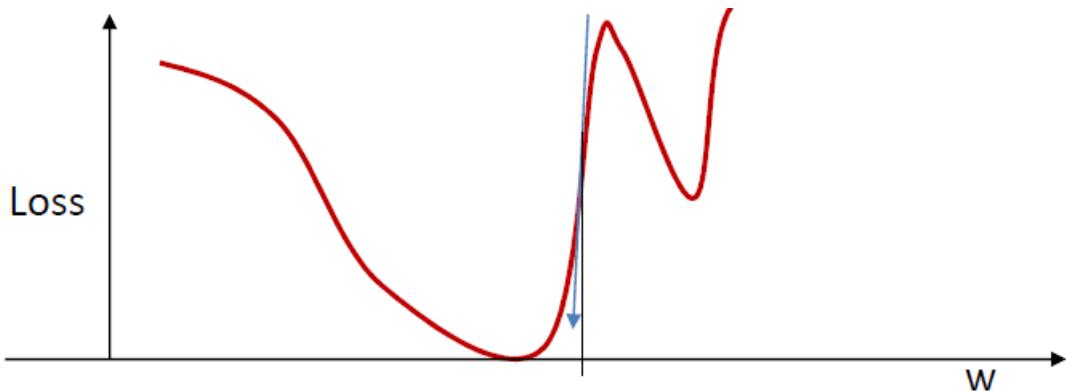
$$U \sim \left(-\sqrt{\frac{6}{n_k + n_{k+1}}}, \sqrt{\frac{6}{n_k + n_{k+1}}}\right)$$

- 或者Kaiming初始化法(Kaiming He et al., 2015)

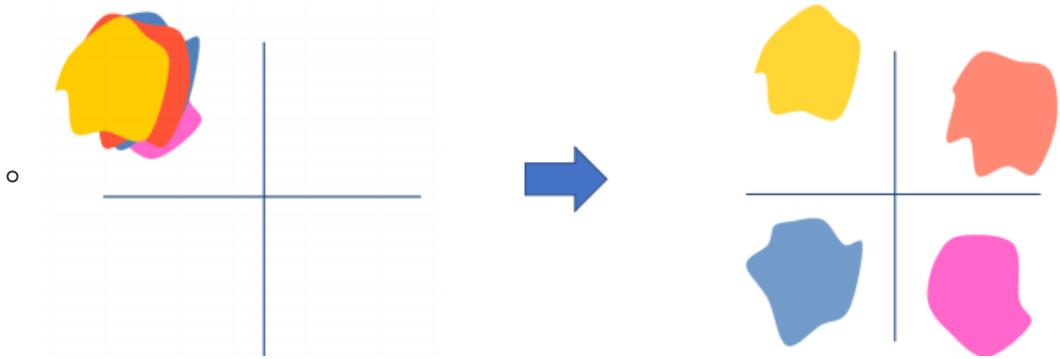
- $b^{(k)} \leftarrow 0$ : 偏差 (**bias**) 置为0
- $W^{(k)} \sim unif \pm \frac{\sqrt{2}}{\sqrt{n_k}}$ : 只有扇入
- 为ReLU专门设计的初始化法
- 实验结果 (30层神经网络, 与Xavier对比) :



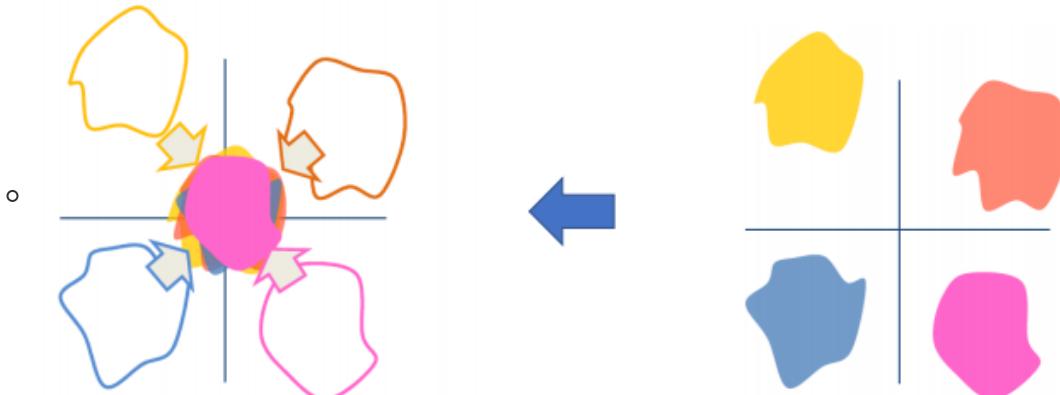
- 或者从预训练模型开始进行微调 (**fine-tuning**)
- 神经网络的设计原则
  - 神经网络隐藏层通常是零均值: 在0附近的激活函数 (如tanh、sigmoid等) 通常有较大的梯度
  - 每一个隐藏层的方差通常保持相同, 从而避免梯度消失 (**gradient vanishing**) 或者梯度爆炸 (**gradient explosion**) 现象
- 梯度裁剪 (**Gradient Clipping**)
  - 有时候损失函数会导致大的梯度下降, 从而使得整个神经网络很不稳定, 因此我们有必要对每一步的步长进行限定, 例如, 可以规定如果  $\nabla \theta_i > 5$ , 就设  $\nabla \theta_i = 5$ 。当然, 也可以设置其他限定条件, 比如对  $|\nabla \theta|$  进行限定等等。



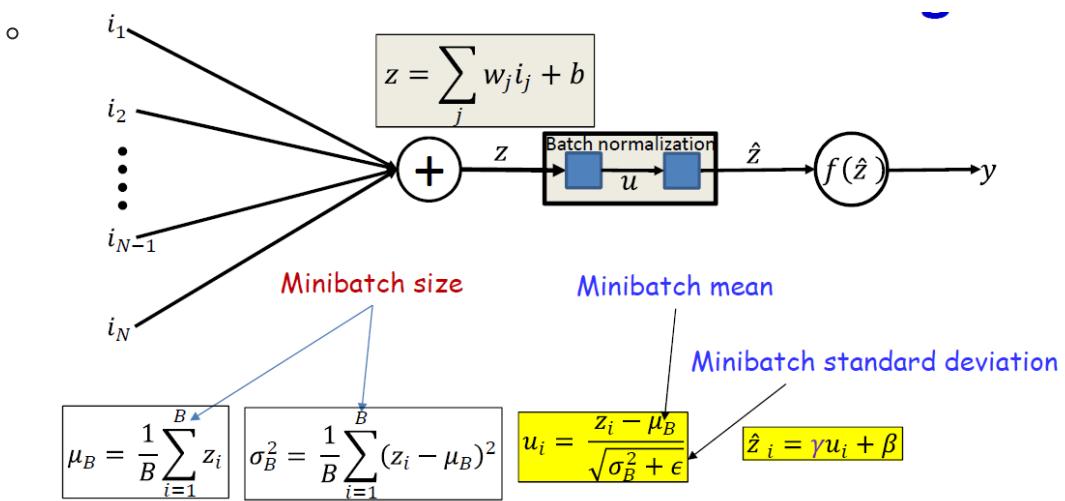
- 协变量偏移 (**Covariance Shift**) 的改善
  - 设计神经网络的时候, 我们假设各个小批量的数据都遵循同样或者相近的分布
  - 但事实上, 每个小批量的样例都可能有各自不同的分布情况, 这就是协变量偏移的问题



- 协变量偏移使得网络为了拟合各个不同的样例小批量，可能会推翻自己的重复劳动，导致训练的低效；并且协变量偏移还会在不同层之间传播。
- 解决策略：使每个小批量的均值和方差都相同



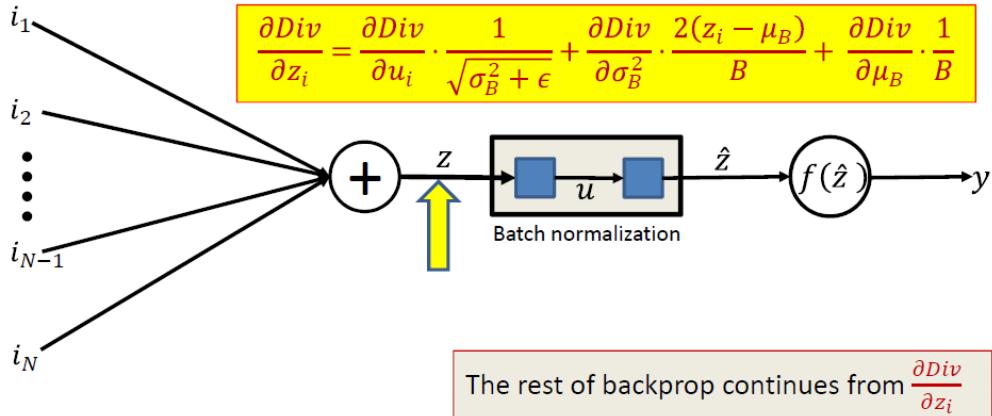
- 批标准化 (Batch Normalization, BN) ([Sergey Ioffe & Christian Szegedy](#), 2015)
  - 对于每个神经元的输出  $z = \sum_j w_j i_j + b$ , 计算均值  $\mu_B = \frac{1}{B} \sum_{i=1}^B z_i$  和方差  $\sigma_B^2 = \frac{1}{B} \sum_{i=1}^B (z_i - \mu_B)^2$ , 其中  $B$  为小批量的大小;
  - 然后将神经元的输出调整到均值为0和标准方差的状态:  $u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$ , 其中  $\epsilon$  为一个小量, 防止分母为0;
  - 最后将输出转移到新的分布上:  $\hat{z}_i = \gamma u_i + \beta$ , 其中  $\gamma$  和  $\beta$  都是参数。



- 不同于标准的后向传播，在训练的时候，批标准化后的后向传播会对整个小批量进行。
-

$$\frac{\partial \text{Div}}{\partial \sigma_B^2} = \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2} \sum_{i=1}^B \frac{\partial \text{Div}}{\partial u_i}$$

$$\frac{\partial \text{Div}}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \sum_{i=1}^B \frac{\partial \text{Div}}{\partial u_i}$$

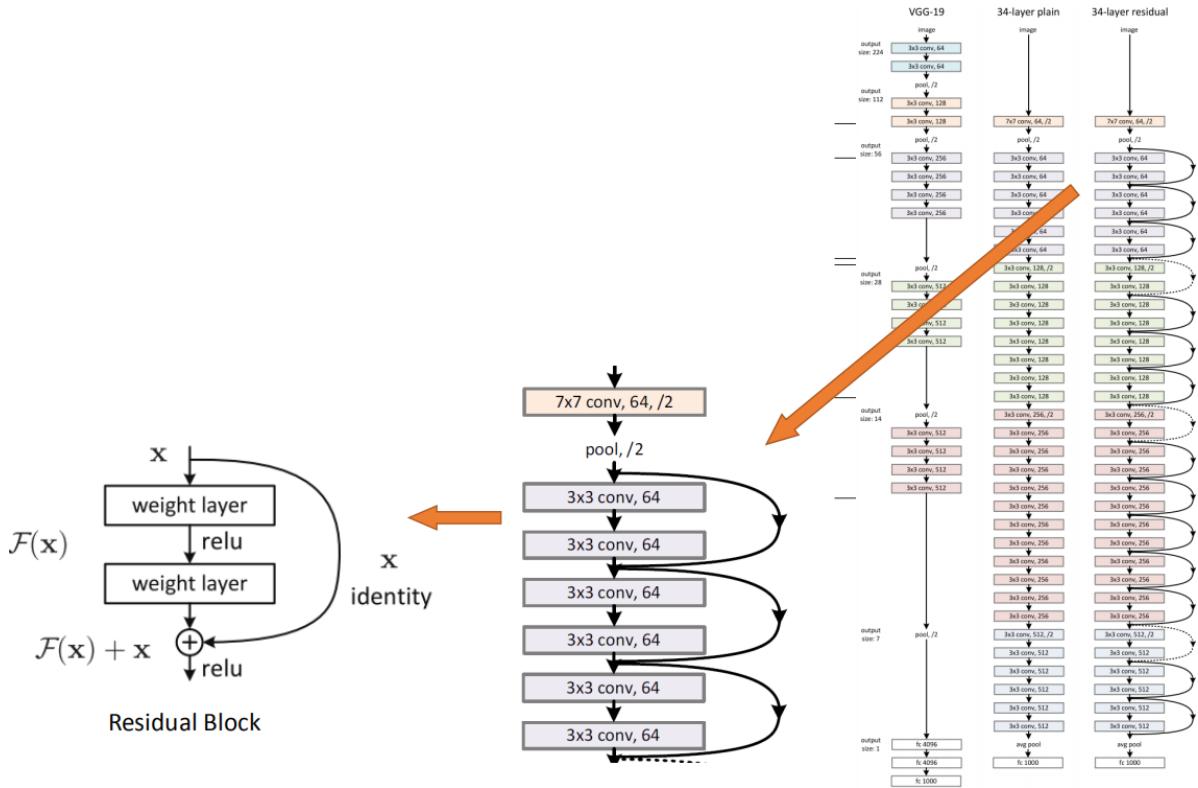


- 但在测试时，我们需要去估计小批量的均值和方差
  - 均值估计:  $\mu_B = \frac{1}{N\_batch} \sum_{batch} \mu_B(batch)$
  - (无偏)方差估计:  $\sigma_B^2 = \frac{1}{N\_batch} \frac{B}{B-1} \sum_{batch} \sigma_B^2(batch)$
- 显然，对于批标准化，我们无需使用随机失活的策略，或者只需要很小的失活率就可以满足需求；并且需要较大的学习率和更快的学习率衰减，这是因为数据总是分布在梯度较大的区域内。然而，批标准化只适用于一些特别的层内，例如多应用于卷积层上。
- 层标准化 (Layer Normalization, LN) ([Jimmy Ba., Jamie Kiros, Hinton, 2016](#))
  - 批标准化并不适用于RNN等动态网络，且在批大小较小的时候效果不好，LN能有效解决这个问题，且LN尤其适用于多层感知机。
  - 调节隐藏层的均值和方差:  $h^t = f[\frac{g}{\sigma^t} \odot (a^t - \mu^t) + b]$
  - 其中  $\mu^t = \frac{1}{H} \sum_{i=1}^H a_i^t$ ,  $\sigma^t = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^t - \mu_t)^2}$
  - 可以看出，层标准化是对批数据特征无关的 (**batch-independent**)，因此能够有效解决批标准化在批大小较小的时候效果不好的问题。
- 其他正则化限制方法：
  - 权值标准化 (WeightNorm, [Tim Salimans & Diederik P. Kingma, 2016](#)) 适用于元学习 (meta-learning) 等需要计算高阶梯度的场合
  - 样例标准化 (InstanceNorm, [Dmitry Ulyanov & Andrea Vedaldi, 2017](#)) 对批数据特征无关，适用于泛化 (generation) 问题
  - 群标准化 (GroupNorm, [Yuxin Wu & Kaiming He, 2018](#)) 对批数据特征无关，能够显著改善批标准化在批大小较小的时候的效果

## 4. 部分高级神经网络架构介绍

### 4.1 残差网络 (Residual Network, ResNet)

- 何恺明等人于2015年提出，获得ImageNet2015年冠军奖
- 是第一个有超过100层的深度神经网络
- 网络结构见下图，右边是宏观结构，有许多构成单元（中间）组成，每个构成单元都由一些残差块（左边）构成。

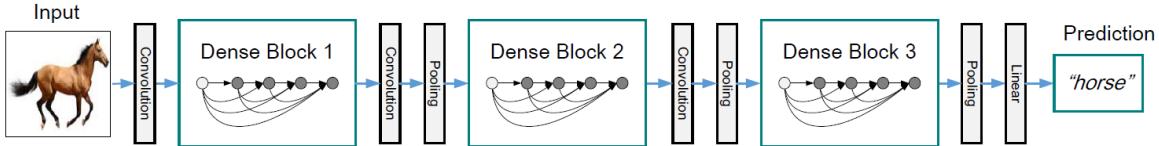


- 输入残差块的数据 $x$ , 经过两层卷积层后得到 $\mathcal{F}(x)$ , 与原数据 $x$ 进行加和后经过ReLU激活函数得到本残差块的输出:  $z = \sigma(\mathcal{F}(x) + x)$ 。
- 然而, 对于论文中的残差网络, 我们可以发现对任意深的残差网络都有一个平凡解 $W^{(k)} = I$ , 因此此残差网络虽然容易学习零变换, 但很难去学习等价变换。为了解决此问题, 我们可以固定残差函数 $H(x) = \mathcal{F}(x) - x$ 的大小。

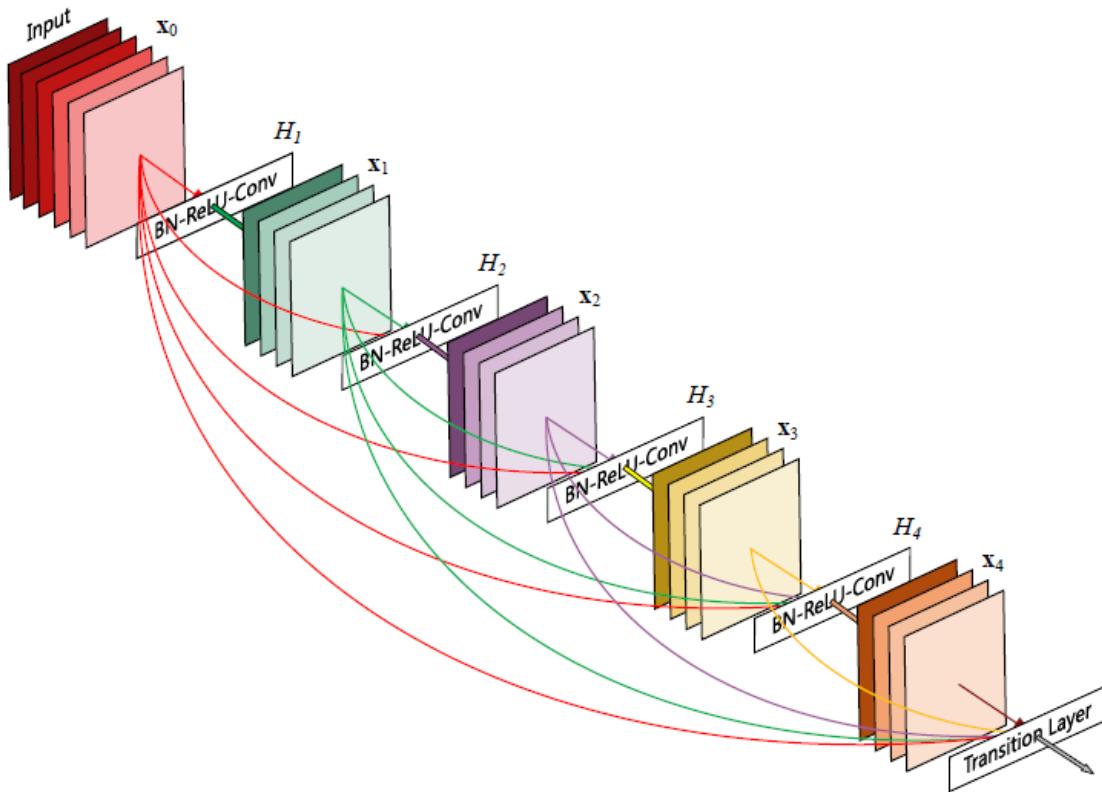
• 题外话:

- 何恺明是2003年清华大学本科班学生, 在博士3年级才发表第一篇论文, 此论文直接获得了CVPR 2009年最佳论文奖;
- 并且他之后曾获得CVPR 2016年和ICCV 2017年最佳论文奖以及ECCV 2018年最佳论文奖提名奖等等。
- 因此, 对于大学生, 更应该做一些有意义的工作而不是急于求成。

## 4.2 密集连接网络 (Densely Connected Network, DenseNet)

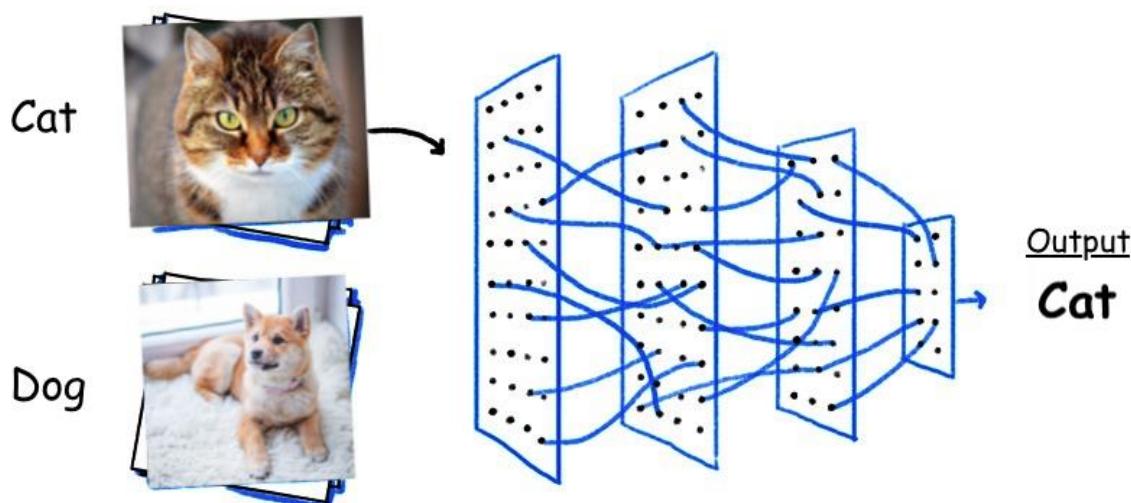


- 黄高 & 刘壮等人于2016提出, 获得CVPR17年最佳论文奖
- 密集连接网络用较浅的网络便能达到与残差网络相同的性能: 既然残差神经网络可以利用本残差块之前的数据, 那么为什么不更进一步, 利用之前所有层的信息呢? 密集连接网络直接将所有层连接起来, 减轻了梯度消失现象, 更有效地利用了特征信息; 并一定程度上减少了参数数量, 使得网络更窄、更紧凑。

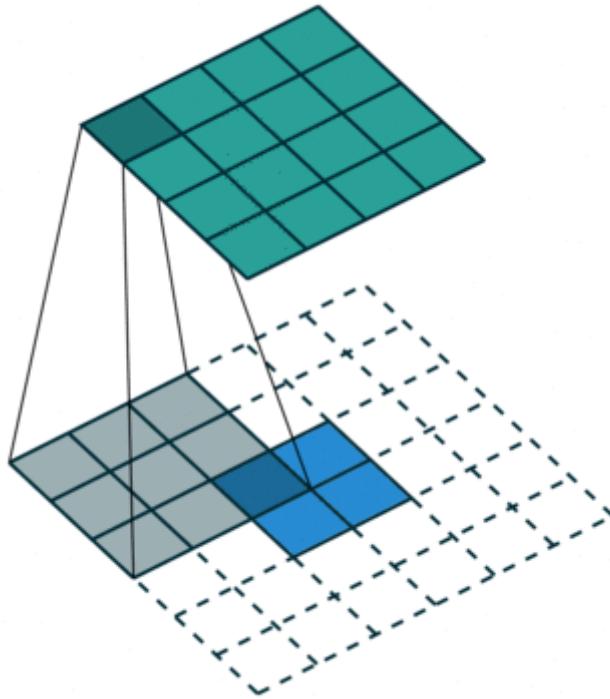


## 4.3 逆卷积 (Deconvolution)

- 通常的图像分类网络都是从高维图像信息出发得到低维标签等输出信息，其中的卷积层和池化层使得图像缩小，造成下采样（down-sampling）现象。

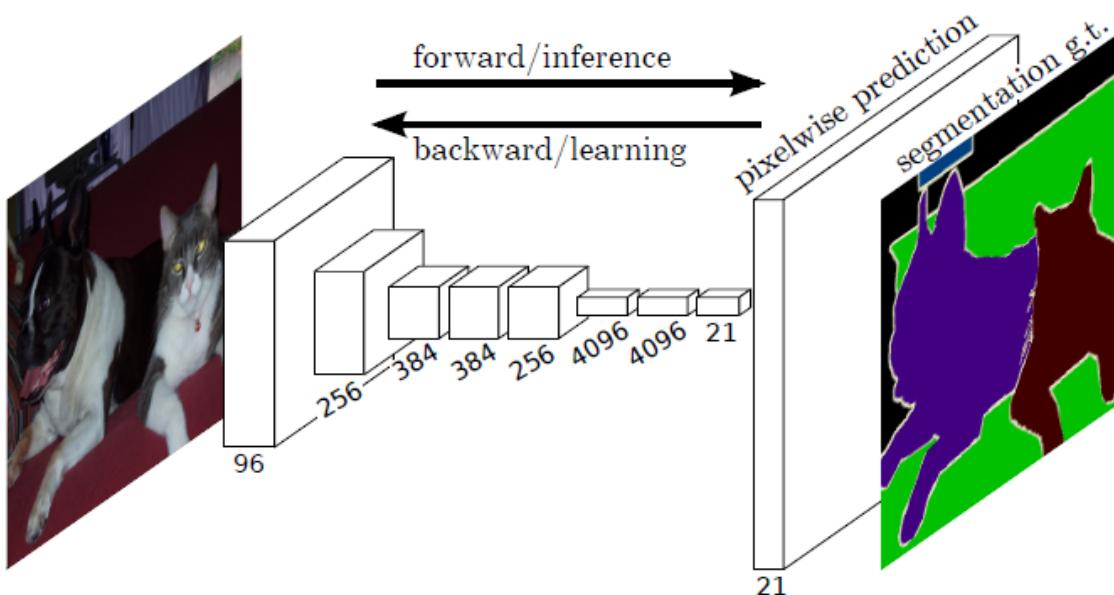


- 我们其实可以反过来，从标签信息出发生成图像，或者从图像出发生成另一张图像，这就需要反卷积层将图像放大：如下图，对于 $2 \times 2$ 的输入数据，我们可以扩大卷积核进行卷积，生成 $4 \times 4$ 的更大的输出数据。通过多个反卷积层，我们就能从标签等低维信息生成图片等等。



## 全卷积网络 (Fully Convolutional Network, FCN)

- [J. Long, et al.](#)于2014年提出用于图像的语义分割 (**Semantic Segmentation**, 即对图像的各个部分分割出来并判定各部分是什么类型的物体) 任务 (将图像) 的网络是第一个全卷积网络的例子 (如下图)



- 他们便用到了反卷积层用来放大图像信息 (即上采样, **up-sampling**)
- 现在更多地用在生成模型里面通过标签等低维信息生成图像等高维输出结果。

## 5. 总结

我们学习了

- 李普希兹连续、L-光滑、强凸函数等凸优化理论；
- 二阶优化法、动量法、SGD、小批量梯度下降法、AdaGrad、RMSProp、AdaDelta、Adam等优化算法；
- 提前终止、初始化、梯度裁剪、数据预处理等训练技巧；

- 随机失活、批标准化、层标准化等正则化层的运用；

此外，我们还学习了残差网络、密集连接网络、全卷积网络等高级神经网络架构。我们还会在今后的课程中以及计算机视觉等课程中学到更多的架构和技巧。

现在你们已经具备作为调参专家的基本知识了！赶快行动起来，多练练手吧！



## 附录：参考资料

- [Dive into Deep Learning — Dive into Deep Learning 0.17.4 documentation \(d2l.ai\)](#).
- [Jaewan-Yun/optimizer-visualization: Visualize Tensorflow's optimizers. \(github.com\)](#), OpenAI, 2018
- [\(PDF\) Some methods of speeding up the convergence of iteration methods \(researchgate.net\)](#), Boris T. Polyak, 1964
- [On the importance of initialization and momentum in deep learning](#), Ilya Sutskever等, 2013
- [Gradient Descent Optimization Algorithms - Cheat Sheet](#), Raimi Karim in Towards Data Science
- [Adaptive Subgradient Methods for Online Learning and Stochastic Optimization](#), Duchi, et al., 2011
- T. Tieleman, and G. Hinton. RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. Technical report, 2012.
- [ADADELTA: An Adaptive Learning Rate Method](#), Matthew D. Zeiler, 2012
- [Adam: A Method for Stochastic Optimization \(arxiv.org\)](#), Diederik P. Kingma & Jimmy Ba, 2014
- [ImageNet Classification with Deep Convolutional Neural Networks \(nips.cc\)](#), Alex Krizhevsky, et al., 2012
- [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift \(mlr.press\)](#), Sergey Ioffe & Christian Szegedy, 2015
- [LayerNormalization.pdf \(toronto.edu\)](#), Jimmy Ba, Jamie Kiros, Hinton, 2016
- [Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks \(nips.cc\)](#), Tim Salimans & Diederik P. Kingma, 2016

- [Instance Normalization: The Missing Ingredient for Fast Stylization](#) Dmitry Ulyanov & Andrea Vedaldi, 2017
- [Group Normalization](#), Yuxin Wu & Kaiming He, 2018
- [Convolutional Neural Networks at Constrained Time Cost \(cv-foundation.org\)](#), Kaiming He & Jian Sun, 2015
- [Densely Connected Convolutional Networks](#), Gao Huang, et al., 2016.
- [Fully Convolutional Networks for Semantic Segmentation \(cv-foundation.org\)](#), J. Long, et al. 2014