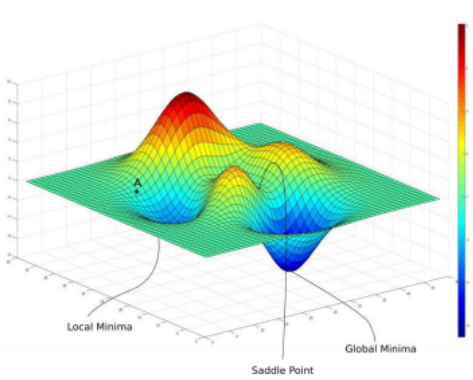
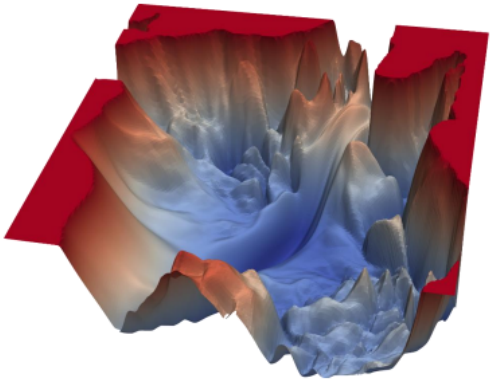
Lecture4-2 Optimization

寻找神经网络上的一组参数 $\boldsymbol{\theta}$，它能显著地降低代价函数 $J(\boldsymbol{\theta})$，该代价函数通常包括整个训练集上的性能评估和额外的正则化项

1. 神经网络优化中的挑战

局部极小值





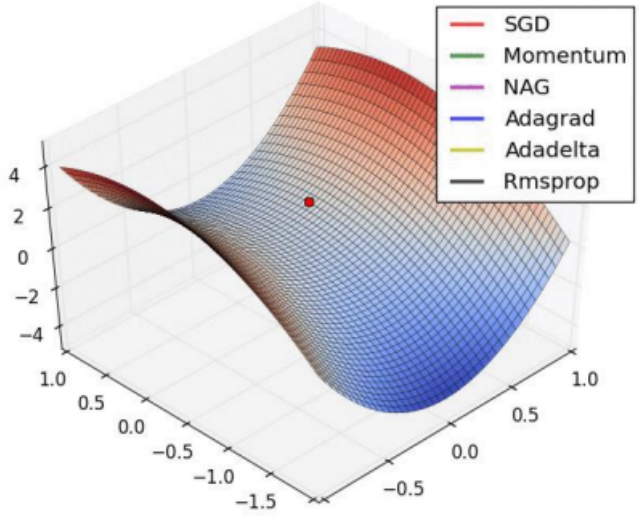
3-D representation for loss contour of a VGG-56 deep network's loss function on the CIFAR-10 dataset.

对于非凸函数时，如神经网络，有可能会存在多个**局部极小值**，事实上，**几乎所有的深度模型基本上都会有非常多的局部极小值**

对于实际中感兴趣的网络，是否存在大量代价很高的局部极小值，优化算法是否会碰到这些局部极小值，都是尚未解决的公开问题

但是学者们现在猜想，对于足够大的神经网络而言，**大部分局部极小值都具有很小的代价函数**，我们能不能找到**真正的全局最小点并不重要**，而是需要在参数空间中找到一个**代价很小（但不是最小）的点**

高原、鞍点和其他平坦区域



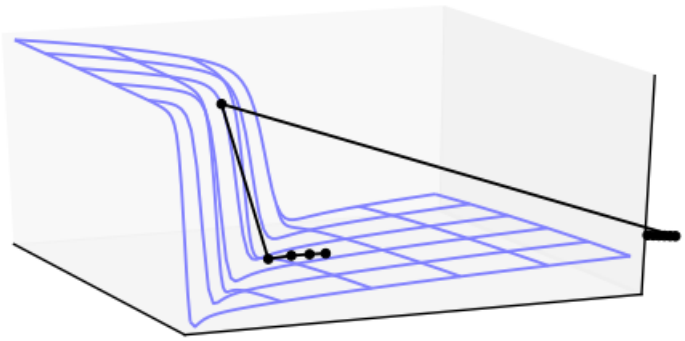
对于很多高维非凸函数而言，局部极小值（以及极大值）事实上都远少于另一类梯度为零的点：**鞍点**

多类随机函数表现出以下性质：**低维空间中，局部极小值很普遍，在更高维空间中，局部极小值很罕见，而鞍点则很常见**

也可能存在**恒值的、宽且平坦的区域**，在这些区域，梯度和 Hessian 矩阵都是零，在凸问题中，一个宽而平坦的区间肯定包含全局极小值，但是对于一般的优化问题而言，**这样的区域可能会对应着目标函数中一个较高的值**

悬崖和梯度爆炸

多层神经网络通常存在像悬崖一样的斜率较大区域



- 高度非线性的深度神经网络或循环神经网络的目标函数通常包含由**几个参数连乘**而导致的参数空间中**尖锐非线性**，这些非线性在某些区域会产生非常大的**导数**

长期依赖

当**计算图变得极深**时，神经网络优化算法会面临的另外一个难题就是**长期依赖**问题，由于变深的结构使模型丧失了学习到先前信息的能力，让优化变得极其困难

假设某个计算图中包含一条反复与矩阵 $\boldsymbol{W}$ 相乘的路径，那么 t 步后，相当于乘以 $\boldsymbol{W}^t$，假设 $\boldsymbol{W}$ 又特征值分解 $\boldsymbol{V}diag(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$，那么在这种情况下

$$\boldsymbol{W}^t = (\boldsymbol{V}diag(\boldsymbol{\lambda})\boldsymbol{V}^{-1})^t = \boldsymbol{V}diag(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}$$

当特征值 λ_i 不在 1 附近时，若在量级上大于 1 则会爆炸；若小于 1 时则会消失

梯度消失使得我们难以知道参数朝哪个方向移动能够改进代价函数，而梯度爆炸会使得学习不稳定

> 循环网络在各时间步上使用相同的矩阵 $\boldsymbol{W}$，而前馈网络并没有
>
> 所以即使使用非常深层的前馈网络，也能很大程度上有效地避免梯度消失与爆炸问题

2. 梯度的优化 $\nabla_\theta J(\theta)$

批量梯度下降 Batch Gradient Descent

如果我们在在整个训练集训练一次后，进行一次梯度下降的话，那么叫做**批量（batch）梯度下降**，或者叫**确定性（deterministic）**梯度算法

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\nabla_\theta J(\theta; x_i, y_i)$$

- m：训练集中训练样本 (x_i, y_i) 的总个数
- 所有样本样本的损失函数加和求平均
    - 减少噪音的影响
- 但注意，这只是一个整体样本梯度下降效果，这个估计可能与真实梯度不同
- 在实践中，我们将损失计算为所有训练样本的平均损失，然后我们计算这个数的梯度

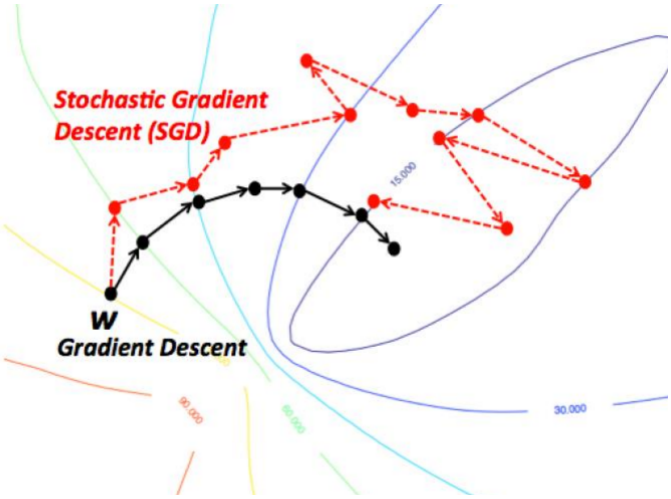
优点

- 可以使用加速度技术基于二阶导数（Hessian）
- 我们不仅可以测量梯度，也可以测量损失函数的**曲率**
- 可以对**收敛速度**做一个简单的理论分析

缺点

- **数据集可能太大了**，无法进行完整的梯度计算
- 在每次参数更新之前，多次为**数据很接近的样本**重新计算梯度（冗余）
- 损失表面是**非凸的和高维的**

随机梯度下降 Stochastic Gradient Descent SGD

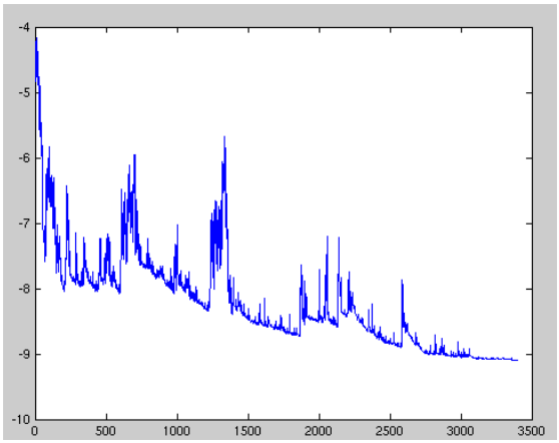


每次只使用**单个**样本的优化算法有时被称**随机（stochastic）**或者**在线（online）**算法，我们在**逐个**输入训练样本时便**计算梯度**，使用它来更新权值

优点

- 比梯度下降快
  - 从第一个样本开始更新梯度，而不是等待，此外，在考虑整个训练数据时，可能存在冗余
- 随机性有助于避免过拟合，从而提高准确性
- 适用于随时间变化的数据集

缺点



- 大多数情况下，**它是近似值的近似值**，所以它注定是不完美的
  - SGD 执行频繁的更新与高方差，导致目标函数波动很大
  - 但实际上这不是问题，事实上这是一个优势（噪声有助于防止过拟合）
- 主要问题是，对于大小为 1 的样本，**无法利用大规模并行性**

小批量梯度下降 Minibatch Gradient Descent

使用一个以上，而又不是全部的训练样本，这种训练方法**小批量（minibatch）**或**小批量随机（minibatch stochastic）**方法

SGD 算法

下面的算法虽然是应用于 Minibatch 的，但是我么也把它称为 SGD 算法

Input

- ϵ_k：学习率
- $\boldsymbol{\theta}$：初始参数

Algorithm

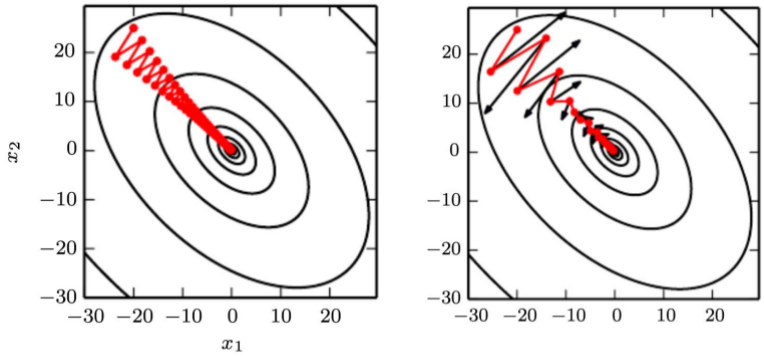
1. $\textbf{while}$ 停止准则未满足 $\textbf{do}$
2. 　　从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\}$ 的小批量，其中 $\boldsymbol{x}^{(i)}$ 对应目标为 $\boldsymbol{y}^{(i)}$
3. 　　计算梯度更新：$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
4. 　　应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
5. $\textbf{end while}$

注意

- 小批量要保证**随机抽取**，因此两个连续的小批量样本应该是彼此**独立**的
- 很有必要在抽取小批量样本前打乱样本顺序

带动量计算的 SGD

动量方法旨在加速学习，特别是处理高曲率、小但一致的梯度，或是带噪声的梯度



- 动量的主要目的是解决两个问题：**Hessian 矩阵的病态条件**和**随机梯度的方差**

Input

- ϵ：学习率
- $\alpha \in [0, 1)$：动量参数

Algorithm

1. $\textbf{while}$ 停止准则未满足 $\textbf{do}$
2. 从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\}$ 的小批量，其中 $\boldsymbol{x}^{(i)}$ 对应目标为 $\boldsymbol{y}^{(i)}$
3. 计算梯度更新：$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
4. 计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \hat{\boldsymbol{g}}$
5. 应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
6. $\textbf{end while}$

- 动量算法引入变量 $\boldsymbol{v}$ 充当速度角色，它代表参数在参数空间移动的方向和速率
- 超参数 α 决定了之前梯度的贡献衰减得有多快，α 越大，之前梯度对现在方向的影响也越大
  - 继续考虑过去的梯度，但让它们的**贡献随时间呈指数衰减**
  - 在实践中，α 的一般取值为 0.5，0.9 和 0.99
- 这抑制了振荡，产生了更稳健的梯度，进而导致更快的收敛

Nesterov 动量 SGD

就像标准的动量计算一样，但要使用未来梯度（这会产生更好的收敛）

Input

- ϵ：学习率

- $\alpha \in [0, 1)$：动量参数

Algorithm

1. $\textbf{while}$ 停止准则未满足 $\textbf{do}$
2. 　从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\}$ 的小批量，其中 $\boldsymbol{x}^{(i)}$ 对应目标为 $\boldsymbol{y}^{(i)}$
3. 　应用临时更新：$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
4. 　计算梯度更新：$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(\boldsymbol{x^{(i)}}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
5. 　计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \hat{\boldsymbol{g}}$
6. 　应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
7. $\textbf{end while}$

3. 学习率/步长的优化 ϵ

SGD 中学习率的衰减

SGD 算法中的一个关键参数是**学习率**，在实践中，有必要随着时间的推移逐渐降低学习率

我们将第 k 步迭代的学习率记作 ϵ_k

在实践中，一般会线性衰减学习率直到第 τ 次迭代

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

- $\alpha = \frac{k}{\tau}$

Delta-bar-Delta

是一个早期的在训练时适应模型参数各自学习率的**启发式方法**

- 如果损失对于某个给定模型参数的偏导保持相同的符号，那么学习率应该增加
- 如果对于该参数的偏导变化了符号，那么学习率应减小

它**只适用于批量梯度下降**

AdaGrad

AdaGrad 算法独立地适应所有模型参数的学习率

算法 8.4 AdaGrad 算法

Require: 全局学习率 ϵ

Require: 初始参数 $\boldsymbol{\theta}$

Require: 小常数 δ，为了数值稳定大约设为 10^{-7}

初始化梯度累积变量 $\boldsymbol{r} = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

累积平方梯度：$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ （逐元素地应用除和求平方根）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

- 调整模型参数的学习速率，方法是将它们缩放成与梯度的所有过去的平方和的平方根成反比
- loss 的偏导数值越大的参数学习率越低
  - 在缓慢倾斜的方向上更快的进展
- 但长期的梯度历史会减慢速度
- 优点：它消除了手动调整学习速率的需要
- 缺点：它在分母中积累了梯度的平方，由于每增加一项都是正的，所以在训练过程中积累的总和不断增加，这反过来又会导致学习速率下降，最终变得无穷小

RMSProp

RMSProp 已被证明是一种有效且实用的深度神经网络优化算法，目前它是深度学习从业者经常采用的优化方法之一

- RMSProp 使用**指数衰减平均**以丢弃遥远过去的历史，使其能够在找到凸碗状结构后快速收敛
- RMSprop 是一个（未发布的）修改使用指数移动平均线累积过去的梯度的 AdaGrad
- AdaGrad 设计用于凸函数，而 RMSprop 在非凸设置(典型的深度学习)中工作得更好

算法 8.5 RMSProp 算法

Require: 全局学习率 ϵ，衰减速率 ρ

Require: 初始参数 $\boldsymbol{\theta}$

Require: 小常数 δ，通常设为 10^{-6}（用于被小数除时的数值稳定）

初始化累积变量 $\boldsymbol{r} = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

累积平方梯度：$\boldsymbol{r} \leftarrow \rho\boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$

计算参数更新：$\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$ （$\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ 逐元素应用）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Adam

Adam 通常被认为对超参数的选择相当鲁棒，尽管学习率有时需要从建议的默认修改

- 最好理解为将 RMSprop 与 SGD + 动量结合起来的一种方式
- 使用平方梯度来缩放学习速率 RMSprop +动量梯度的移动平均
- 对超参数的选择是相当稳健的

算法 8.7 Adam 算法

Require: 步长 ϵ （建议默认为：0.001）

Require: 矩估计的指数衰减速率，ρ_1 和 ρ_2 在区间 $[0, 1)$ 内。（建议默认为：分别为 0.9 和 0.999）

Require: 用于数值稳定的小常数 δ （建议默认为：10^{-8}）

Require: 初始参数 $\boldsymbol{\theta}$

初始化一阶和二阶矩变量 $\boldsymbol{s} = 0, \boldsymbol{r} = 0$

初始化时间步 $t = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

更新有偏一阶矩估计：$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

更新有偏二阶矩估计：$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

修正一阶矩的偏差：$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

修正二阶矩的偏差：$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

计算更新：$\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ （逐元素应用操作）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

4. 预处理和参数的初始化

深度学习模型的训练算法通常是迭代的，因此要求使用者指定一些**开始迭代的初始点**

训练深度模型是一个足够困难的问题，以致于大多数算法都很大程度地受到**初始化选择**的影响

- 偏置：以为每个单元的偏置设置**启发式挑选的常数**，仅随机初始化权重
- 权重：初始化模型的权重为**高斯或均匀分布**中随机抽取的值

权重初始化

- 深度学习训练是迭代的，并且强烈依赖于初始化点（即，我们如何初始化网络参数）
- 重要的原则：**权重的非对称性**
  - 为什么？因为如果两个单元共享相同的激活、相同的权重和相同的输入，它们将以相同的方式更新（没有学习）
  - 所以，不要给所有权重都赋予相同的值（例如 0）
  - 权重由高斯或均匀抽样得出
- 但是，需要注意的是

- **较大的权重**具有较强的对称破坏效应，会在正向和反向传播时传播较强的信号
- 然而，它们也可能导致爆炸值，单位的饱和
- 但是我们也不想要较小的权重
- 此外，我们还希望保持输入和输出的方差相同（因为输出是下一层的输入）

标准初始化/均匀分布初始化(tanh)

$$W_{ij} \sim U(-\sqrt{(\frac{6}{m+n})}, \sqrt{(\frac{6}{m+n})})$$

- 初始化 m 个输入和 n 个输出的全连接层权重的启发式方法

Xavier 初始化 (tanh)

$$W_{ij} \sim N(0, \sqrt{\frac{1}{m}})$$

- 其中 m 是输入的个数，n 是输出的个数

均匀分布初始化 (sigmoid)

$$W_{ij} \sim U(-4\sqrt{(\frac{6}{m+n})}, 4\sqrt{(\frac{6}{m+n})})$$

- 其中 m 是输入的个数，n 是输出的个数

Xavier 初始化(ReLU)

$$W_{ij} \sim N(0, \sqrt{\frac{2}{m}})$$

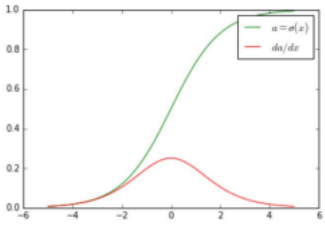
- 其中 m 是输入的个数，n 是输出的个数

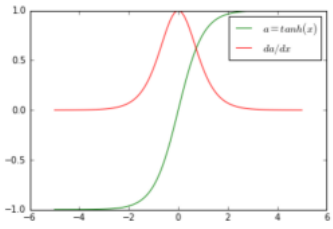
预训练/ fine-tuning 工作

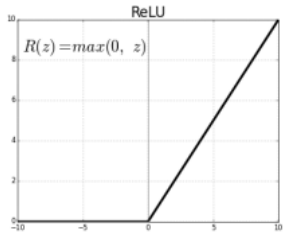
- 你有一个机器学习的模型 m
- 预训练：你有一个数据集 A，在这个数据集上你训练 m 来完成一些特定的任务
- 你有一个数据集 B，在你开始训练模型之前，你用 m 的一些参数来初始化你的模型

数据预处理

- 激活函数通常以零为中心



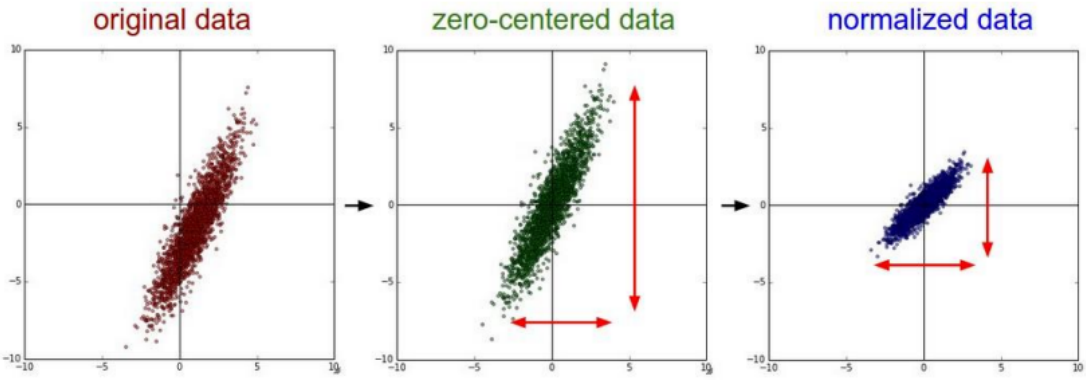




- 这是一件好事，因为它可以帮助我们避免饱和，而饱和度会导致**梯度消失**
- 同时，我们喜欢**单边饱和**，因为它有助于避免由于噪声而产生的方差
- 减去均值，训练数据也以零为中心
    - 否则可能导致梯度消失
- 对输入进行缩放，使其具有类似的对角线协方差
    - 否则，具有非常不同协方差的输入样本会产生非常不同的梯度，使梯度更新更加困难

单元标准化 Unit Normalization

当输入变量是正态分布时，使用单元标准化：数据集减去均值除以标准差



批标准化 Batch Normalization

两个重要的原则，提供给网络各层的数据的分布应该是

- 以零为中心的
    - 我们已经解决了这个问题
- 时间和数据不变（小批量）
    - 每一层的激活都要进行规范化

$$
\begin{aligned}
\boldsymbol{Z} &= \boldsymbol{X}\boldsymbol{W} \\
\tilde{\boldsymbol{Z}} &= \boldsymbol{Z} - \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{Z}_{i,:} \\
\hat{\boldsymbol{Z}} &= \frac{\tilde{\boldsymbol{Z}}}{\sqrt{\epsilon + \frac{1}{m}\sum_{i=1}^{m}\tilde{\boldsymbol{Z}}_{i,:}^2}} \\
\boldsymbol{H} &= \max\{0, \gamma\hat{\boldsymbol{Z}} + \boldsymbol{\beta}\}
\end{aligned}
$$

- 解决了梯度消失的问题

- 模型正则化，提高网络泛化能力
- 解决了梯度消失的问题