Fully connected networks, optimization, initialization

从自微分的角度重新看待神经网络

Fully Connected Networks

L-layer 神经网络的标准形式：

$$\begin{aligned} z_{i+1} &= \sigma_i(W_i^T z_i + b_i),\ i = 1, \dots, L \\ h_\theta(x) &= z_{L+1} \\ z_1 &= x \end{aligned} \tag{2}$$

参数 $\theta = \{W_{1:L}, b_{1:L}\}$, $\sigma_i(x)$ 是非线性函数

特别地，$\sigma_L(x) = x$

Matrix Broadcasting

矩阵形式的迭代公式：

$$Z_{i+1} = \sigma_i(Z_i W_i + 1 b_i^T) \tag{3}$$

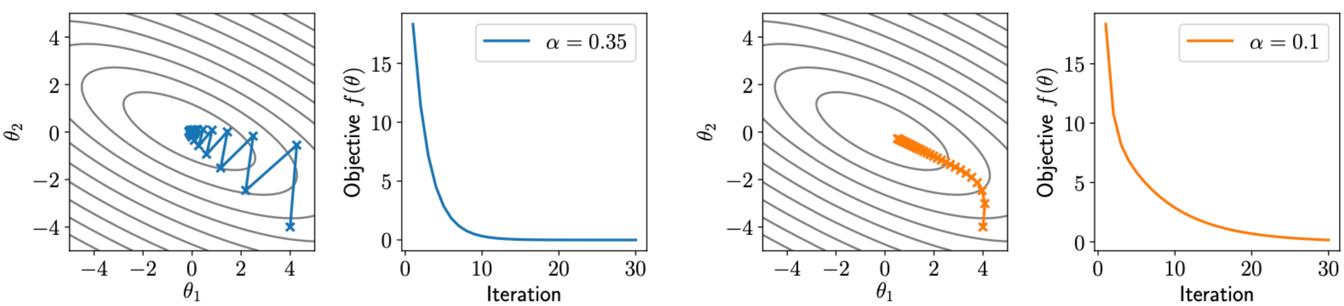
实际中不需要构建 $1b_i^T$ 这个矩阵，可以通过 broadcasting 操作完成

例如，对于一个 $n \times 1$ 的向量，broadcasting 会将其视为 $n \times p$ 的矩阵，将第一列重复 p 次。但实际中 broadcasting 并没有复制任何数据

Optimization

优化的核心问题是让平均 loss 最小化

传统梯度下降法的问题在于 α 的选择对迭代的影响（振荡 or 速度太慢）



本节主要介绍一些实际应用的优化算法，样例图示对应的函数为

$$f(\theta) = \frac{1}{2}\theta^T P \theta + q^T \theta \tag{4}$$

其中 $\theta \in R^2$

Newton's Method

在深度学习中不常用

核心思想：将全局结构考虑到优化中

$$\theta_{t+1} = \theta_t - \alpha(\nabla_\theta^2 f(\theta_t))^{-1} \nabla_\theta f(\theta_t) \tag{5}$$

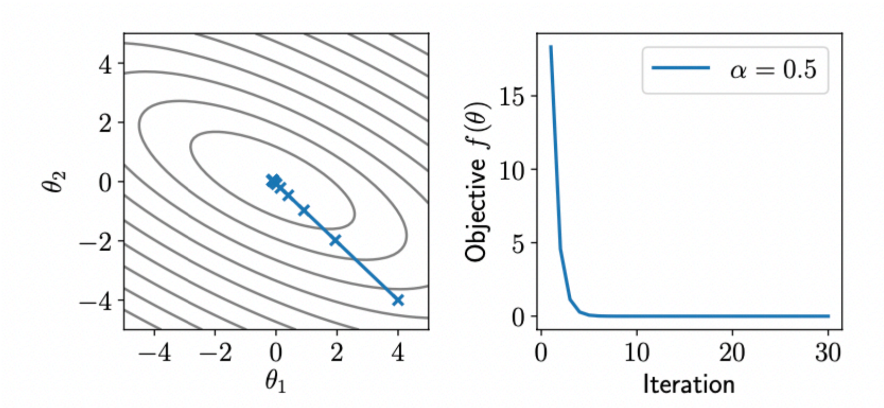
其中 $\nabla_\theta^2 f(\theta_t)$ 是 Hessian 矩阵，即 $n \times n$ 的由二阶偏导组成的矩阵

$$H(F)(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^F}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^F}{\partial x_1 \partial x_n} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^F}{\partial x_2^2} & \cdots & \frac{\partial^F}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_n \partial x_1} & \frac{\partial^F}{\partial x_n \partial x_2} & \cdots & \frac{\partial^F}{\partial x_n^2} \end{bmatrix}$$

二阶导数的值判断梯度下降的速率。Hessian 矩阵能对步长进行调整。

本质上是采用泰勒展开的二阶近似

如果 $\alpha = 1$, 则称为 Full step Newton method; 否则称为 damped Newton method



缺点：

- Hessian 矩阵是 n * n 的，占用内存资源大。
- 即使用自微分也很难求解。
- 对于非凸优化问题，无法确定是否真的要使用 Newton 法得到的方向。

Momentum

在深度学习中很常用

核心思想：既要像梯度下降那样容易计算，又要像 Newton 法那样考虑全局结构。于是将过去计算过的梯度考虑进来：

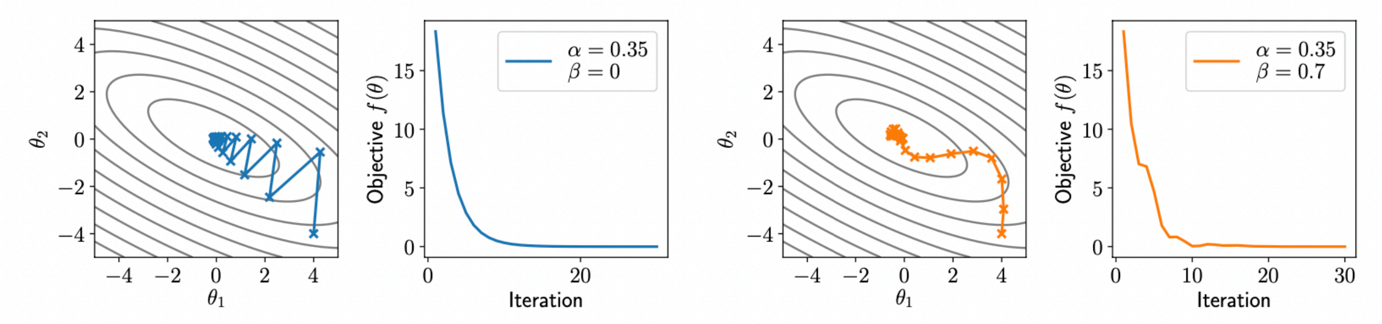
$$\begin{aligned} u_{t+1} &= \beta u_t + (1-\beta)\nabla_\theta f(\theta_t) \\ \theta_{t+1} &= \theta_t - \alpha u_{t+1} \end{aligned} \tag{6}$$

其中 α, β 均为超参数

$$u_{t+1} = (1-\beta)\nabla_\theta f(\theta_t) + \beta(1-\beta)\nabla_\theta f(\theta_{t-1}) + \cdots \tag{7}$$

也有其他形式，例如 $u_{t+1} = \beta u_t + \nabla_\theta f(\theta_t)$ 或 $u_{t+1} = \beta u_t + \alpha \nabla_\theta f(\theta_t)$

Momentum 能让梯度下降过程更为平滑，但也会引入其他形式的振荡以及梯度不下降的行为

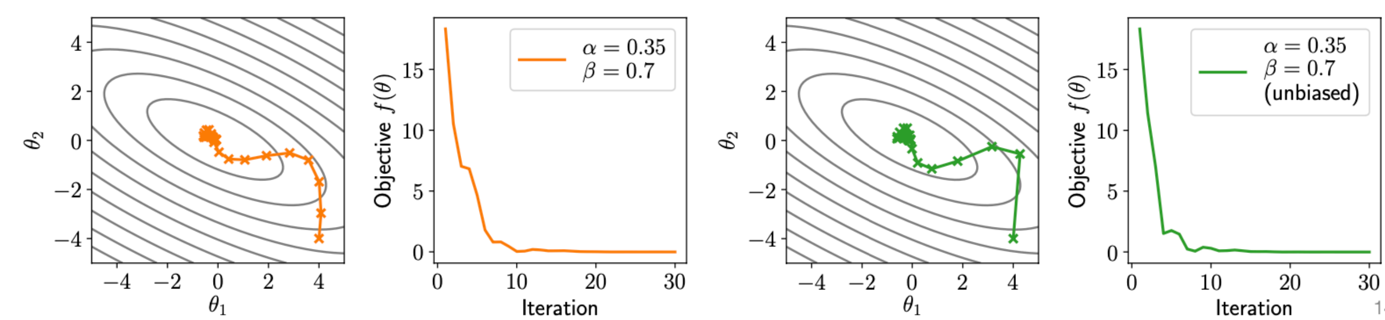


Unbiasing Momentum

通常而言，u_t 初值为 0. 在迭代初期会比 f 的梯度值更小 (乘了 $1-\beta$)

因此为了 unbias 以保证相同的量级，采用

$$\theta_{t+1} = \theta_t - \alpha u_t / (1-\beta^{t+1}) \tag{8}$$

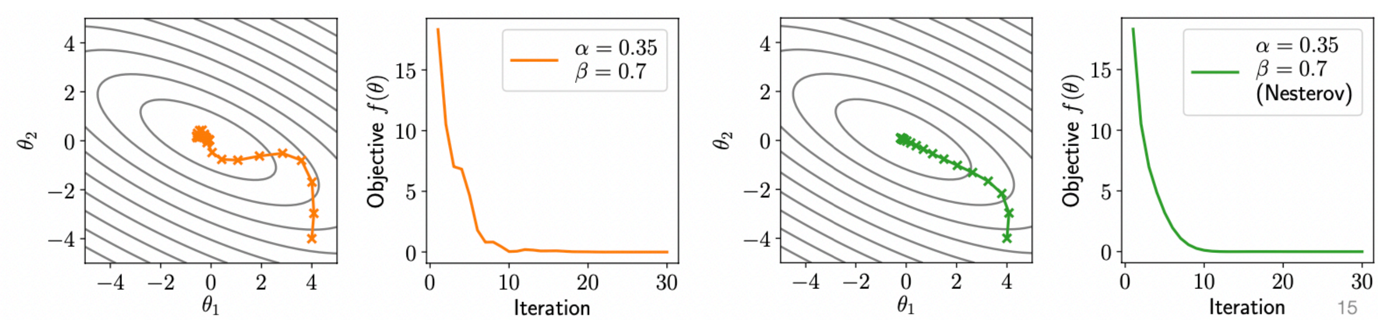


Nesterov Momentum

核心思想：在计算 Momentum 时采用下一个参数点

$$\begin{aligned} u_{t+1} &= \beta u_t + (1-\beta)\nabla_\theta f(\theta_t) \\ \theta_{t+1} &= \theta_t - \alpha u_t \end{aligned} \implies \begin{aligned} u_{t+1} &= \beta u_t + (1-\beta)\nabla_\theta f(\theta_t - \alpha u_t) \\ \theta_{t+1} &= \theta_t - \alpha u_t \end{aligned}$$

这对凸优化很有用，有时候在神经网络中也有效



Adam

Adam 是自适应梯度方法 (adaptive gradient method) 的一种。

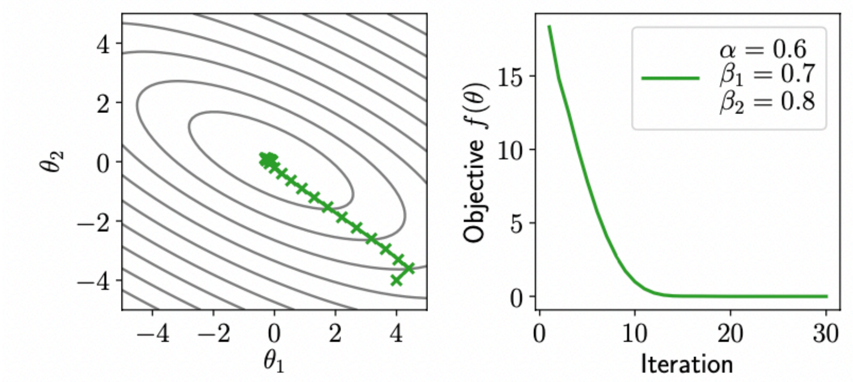
自适应梯度的核心思想：在神经网络的不同层以及不同参数之间梯度变化各有不同，对应的学习率不能一概而论。因此根据迭代中的梯度平方和来动态调整学习率。

Adam 是应用最广泛的一种，结合了 Momentum 和 Adaptive scale estimation

$$
\begin{aligned}
u_{t+1} &= \beta_1 u_t + (1-\beta_1)\nabla_\theta f(\theta_t) \\
v_{t+1} &= \beta_2 v_t + (1-\beta_2)\left(\nabla_\theta f(\theta_t)\right)^2 \\
\theta_{t+1} &= \theta_t - \alpha u_{t+1} / \left(v_{t+1}^{1/2} + \epsilon\right)
\end{aligned}
$$

(Common to use unbiased momentum estimated for both terms)

ϵ 是为了防止分母为 0, 一般不加

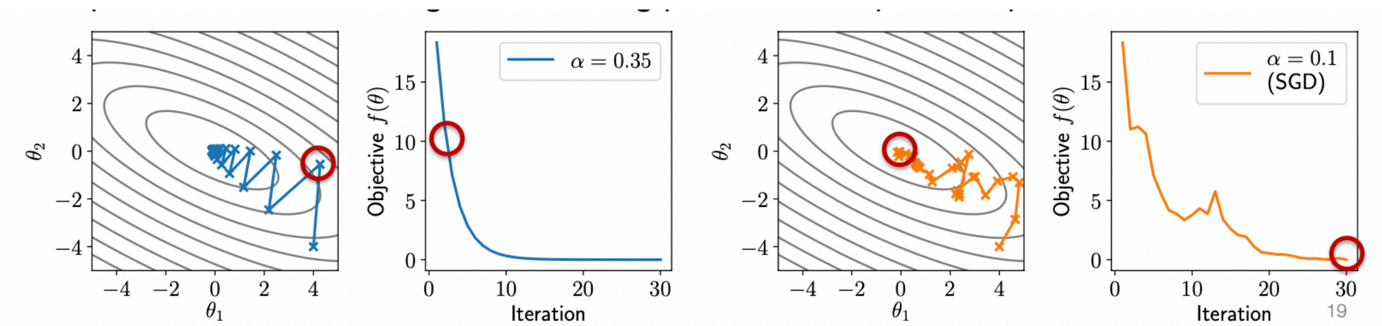


Stochastic Variants

实际使用的方法都采用 minibatch

minibatch 只考虑了样例的一个子集，可能会带来额外的噪声，但是迭代速度快，且额外的噪声有助于避免陷入局部最优

$$
\theta_{t+1} = \theta_t - \frac{\alpha}{|B|}\sum_{i\in B}\nabla_\theta l(h(x^{(i)}), y^{(i)}) \tag{9}
$$



> 红圈代表经过相同时间后两种方法所到达的位置

Initialization

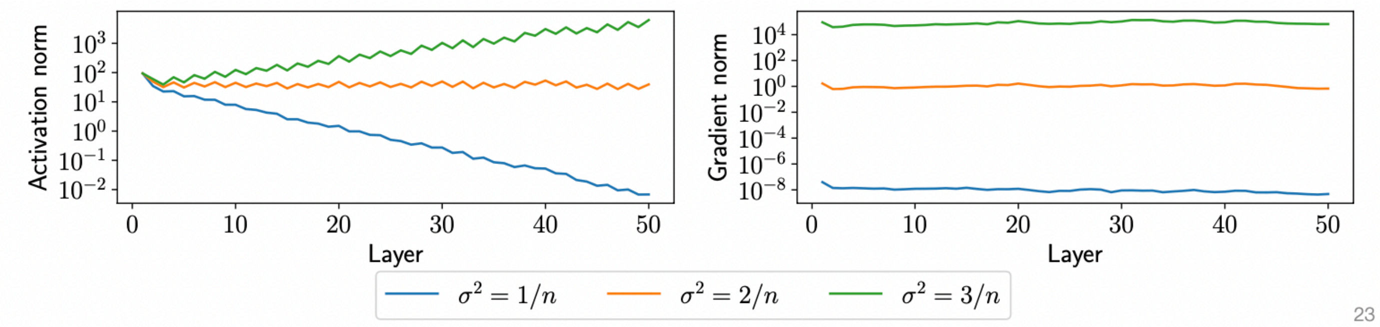
Initialization of Weights

凸优化中常会初始化为 0, 但在神经网络中 0 是鞍点，初始化为 0 之后梯度全为 0.

Choice of initialization matters

通常会初始化 W_i 为随机变量，即 $W_i \sim N(0, \sigma^2 I)$

但是对 σ^2 的选择既会影响计算结果 Z_i, 也会影响梯度 $\nabla_{W_i} l(h_\theta(X), y)$ 的范数



Weights don't move that much

误区：无论参数初始化位置在哪，总能训练到最优位置附近

事实上，比起最终期望位置，权重参数在训练后通常离初始化点更近

总之，初始化很重要

What cause these effects

考虑独立的随机变量 $x \sim N(0, 1)$, $w \sim N(0, \frac{1}{n})$

于是有

$E[x_i w_i] = E[x_i]E[w_i] = 0$, $Var[x_i w_i] = Var[x_i]Var[w_i] = 1/n$

因此

$$\begin{aligned} E[w^T x] &= 0 \\ Var[w^T x] &= 1 \end{aligned} \tag{10}$$

即由中心极限定理知 $w^T x \to N(0, 1)$

因此，如果采用线性函数 (linear activation), 且 $z_i \sim N(0, 1)$, $W_i \sim N(0, \frac{1}{n} I)$, 则有 $z_{i+1} = W_i^T z_i \sim N(0, I)$

如果采用 ReLu 作为 activation, 则 z_i 有一半设为 0, 因此需要对 W_i 的方差乘 2 以保证最终方差相同，因此 $W_i \sim N(0, \frac{2}{n} I)$

如果在 ReLu 网络中 $\sigma^2 \neq 2/n$, 则每次迭代后中间结果的方差都会乘上一个非 1 参数，对 forward pass 和 backward pass 都有影响

$$\begin{aligned} Var[Z_{i+1}] = \gamma \cdot Var[Z_i] = \gamma^i \\ Var[G_i] = \gamma \cdot Var[G_{i+1}] = \gamma^{D-i} \end{aligned} \tag{11}$$

于是

$$Var[\boldsymbol{\nabla}_{W_i} l(h_\theta(X), y)] \propto Var[Z_i^T G_i] \propto \gamma^D \tag{12}$$

因此，梯度的方差在层与层之间基本不变，但是会受到 σ^2 和深度 D 的影响。