模式识别 U3 线性回归 Linear Regression

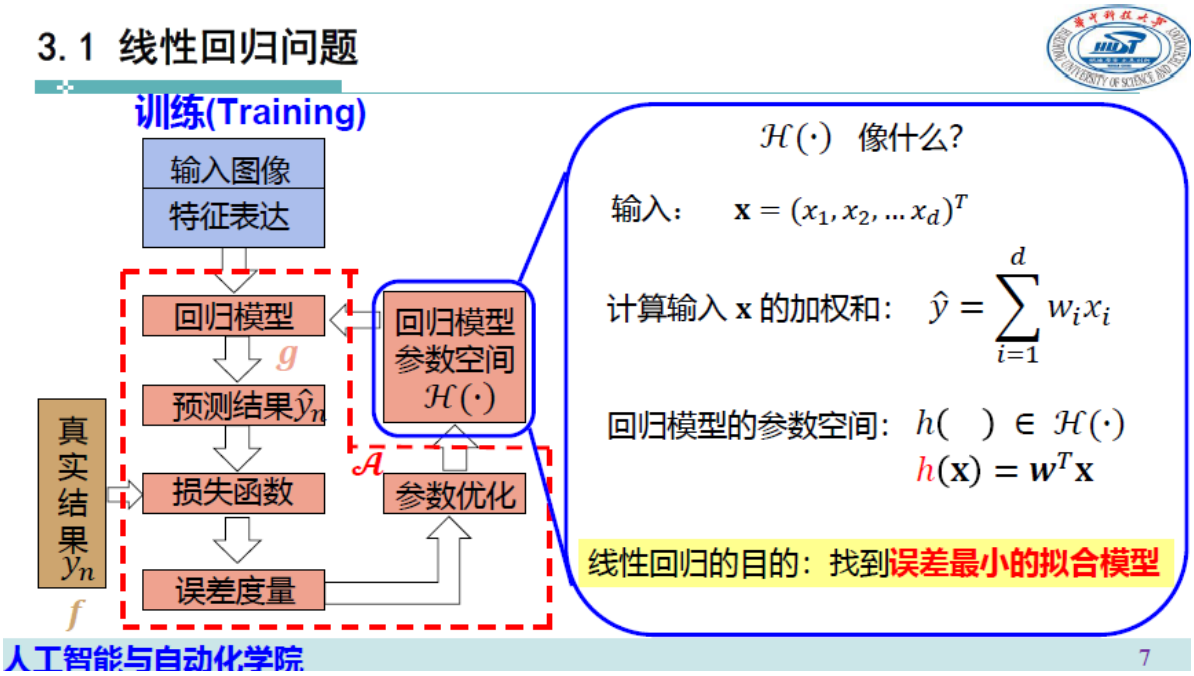
课程内容

- 3.1 线性回归问题
- 3.2 线性回归算法
- 3.3 梯度下降法

线性回归问题

机器学习的过程其实是一个找**最拟合函数**的过程，通过不断的训练，我们最终得到一个函数映射，给定函数（网络）一个输入，函数（网络）会给出相应的输出

若输出的是一个数值（scatter），我们就将这类机器学习问题称为回归Regression



线性回归算法

模型构建

为达到回归目的，我们度量模型输出结果时不再仅仅关注输出的符号

而是关注模型输出的数值

由此，我们择取“平方误差函数”作为我们的损失函数，来度量我们的模型学习效果

$$\mathcal{L} = (\hat{y_n} - y_n)^2$$

$$\mathcal{L}_{in} = \frac{1}{N}\sum_{n=1}^{N}(\hat{y_n} - y_n)^2$$

$\mathcal{L}_{in}$计算训练样本集所有样本产生的平均损失

$$\mathcal{L}_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n) - y_n)^2$$

$h(\mathbf{x}_n)$是回归模型的结果

在线性回归模型中， $\hat{y_n} = h(\mathbf{x}_n) = \mathbf{w}^T\mathbf{x}_n$

$$\mathcal{L}_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2$$

则

$$g = \arg\min_{\mathbf{w}} \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2$$

向量/矩阵形式

用矩阵/向量形式表示$L_{in}(\mathbf{w})$

$$L_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n{}^T\mathbf{w} - y_n)^2$$

$$= \frac{1}{N}\left\|\begin{matrix}\mathbf{x}_1{}^T\mathbf{w} - y_1\\ \mathbf{x}_2{}^T\mathbf{w} - y_2\\ \cdots\\ \mathbf{x}_N{}^T\mathbf{w} - y_N\end{matrix}\right\|^2 = \frac{1}{N}\left\|\begin{bmatrix}--\mathbf{x}_1{}^T--\\ --\mathbf{x}_2{}^T--\\ \cdots\\ --\mathbf{x}_N{}^T--\end{bmatrix}\mathbf{w} - \begin{bmatrix}y_1\\ y_2\\ \cdots\\ y_N\end{bmatrix}\right\|^2$$

$$= \frac{1}{N}\|\mathbf{X}\mathbf{w} - Y\|^2$$

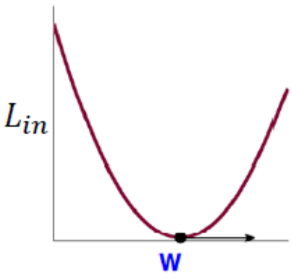
$\mathbf{X}$: $N \times (d+1)$

$\mathbf{w}$: $(d+1) \times 1$

Y: $N \times 1$

求最佳解: $\min_{\mathbf{w}} L_{in}(\mathbf{w}) = \frac{1}{N}\|\mathbf{X}\mathbf{w} - Y\|^2$

L_{in} 曲线具有连续、可微、凸函数的特点



$$\nabla L_{in}(\mathbf{w}) = \begin{bmatrix}\frac{\partial L_{in}(\mathbf{w})}{\partial w_0}\\ \frac{\partial L_{in}(\mathbf{w})}{\partial w_1}\\ \vdots\\ \frac{\partial L_{in}(\mathbf{w})}{\partial w_d}\end{bmatrix} = \begin{bmatrix}0\\ 0\\ \vdots\\ 0\end{bmatrix}$$

$\nabla L_{in}(\mathbf{w}) = \mathbf{0}$时求得最佳解$\mathbf{w}^*$

广义逆解法——线性回归的解析解

$\nabla \mathcal{L}_{in}(\mathbf{w}) = 0$时，求得最佳解$\mathbf{w}^*$

$\nabla L_{in}(\mathbf{w})$求解:

$$L_{in}(\mathbf{w}) = \frac{1}{N}\|\mathbf{X}\mathbf{w} - \mathrm{Y}\|^2 = \frac{1}{N}(\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{w}^T\mathbf{X}^T\mathrm{Y} + \mathrm{Y}^T\mathrm{Y}) = \frac{1}{N}(\mathbf{w}^T A\mathbf{w} - 2\mathbf{w}^T b + c)$$

当w是单变量时

$$L_{in}(w) = \frac{1}{N}(aw^2 - 2bw + c)$$

$$\nabla L_{in}(w) = \frac{1}{N}(2aw - 2b)$$

当$\mathbf{w}$是向量时

$$L_{in}(w) = \frac{1}{N}(\mathbf{w}^T A\mathbf{w} - 2\mathbf{w}^T b + c)$$

$$\nabla L_{in}(w) = \frac{1}{N}(2A\mathbf{w} - 2b)$$

$$\nabla L_{in}(\mathbf{w}) = \frac{2}{N}(\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathrm{Y})$$

$$\nabla L_{in}(\mathbf{w}) = \frac{2}{N}(\mathrm{X}^T\mathrm{X}\mathbf{w} - \mathrm{X}^T\mathrm{Y}) = \mathbf{0}$$

$$\mathbf{w}^* = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{Y}$$

$$g = \mathbf{w}^* = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{Y} = \mathrm{X}^\dagger\ \mathrm{Y}$$

$$\mathrm{X}^\dagger = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\text{，称为广义逆}$$

线性回归算法

- 对训练样本集 $\mathcal{D}$ 构造输入特征向量矩阵 $\mathbf{X}$ 和输出向量 Y

$$\mathbf{X} = \begin{bmatrix} --\mathrm{x}_1{}^T -- \\ --\mathrm{x}_2{}^T -- \\ \vdots \\ --\mathrm{x}_N{}^T -- \end{bmatrix}, \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

- 计算广义逆： $\mathbf{X}^\dagger = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$
- 计算回归值： $g = \mathbf{w}^* = \mathbf{X}^\dagger Y$

$\mathbf{X}$: $N \times (d+1)$

Y: $N \times 1$

$\mathbf{X}^\dagger$: $(d+1) \times N$

$\mathbf{w}$: $(d+1) \times 1$

梯度下降法 Gradient Descent

各种 GD算法

$$\nabla L_{in}(\mathbf{w}) = \frac{2}{N}(\mathrm{X}^T\mathrm{X}\mathbf{w} - \mathrm{X}^T\mathrm{Y}) = \mathbf{0}$$

$$\mathbf{w}^* = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{Y}$$

$$g = \mathbf{w}^* = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{Y} = \mathrm{X}^\dagger\ \mathrm{Y}$$

$$\mathrm{X}^\dagger = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\text{，称为广义逆}$$

当$N = 10000$，$d = 9999$，$\mathbf{X}$是一个巨大的矩阵，N，d可以更大，如何计算$\mathbf{X}^\dagger$？

回顾感知器算法:

- 对样本的特征向量x和权向量w 增广化
- 初始化权向量w_0 (例如: $w_0 = \mathbf{0}$)
- $for\ t = 0,1,2,\ldots$ (t 代表迭代次数)

  ① 对某些样本n，通过下式对权向量w_t进行更新:

$$w_{t+1} = w_t + 1 \cdot \left(\left[\!\left[\mathrm{sign}\left(w_t^T \mathbf{x}_{n(t)}\right) \neq y_n\right]\!\right] \mathbf{x}_{n(t)}\right)$$

…直到满足停止条件，此时的w_{t+1}作为学到的g

回顾感知器算法:

- 对样本的特征向量x和权向量w 增广化
- 初始化权向量w_0 (例如: $w_0 = \mathbf{0}$)
- $for\ t = 0,1,2,\ldots$ (t 代表迭代次数)

  ① 对某些样本n，通过下式对权向量w_t进行更新:

$$w_{t+1} = w_t + \underbrace{1}_{\eta} \cdot \underbrace{\left(\left[\!\left[\mathrm{sign}\left(w_t^T \mathbf{x}_{n(t)}\right) \neq y_n\right]\!\right] \mathbf{x}_{n(t)}\right)}_{\mathbf{v}}$$

…直到满足停止条件，此时的w_{t+1}作为学到的g

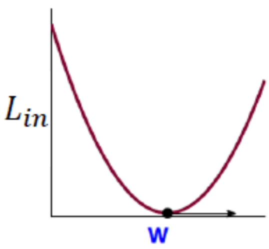
算法可理解成通过选择(η，$\mathbf{v}$)，以及确定"停止条件"的找到最佳解的迭代优化过程

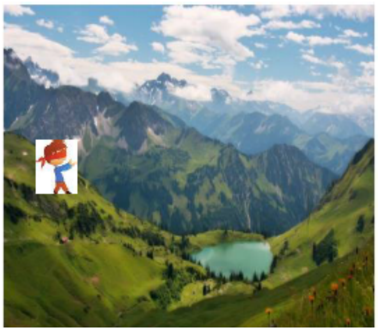
迭代优化:

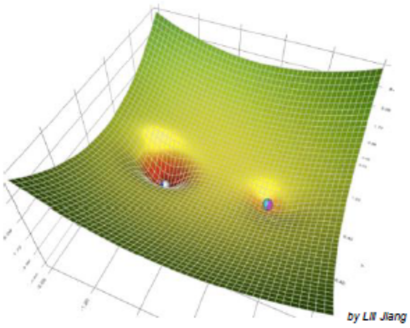
- $for\ t = 0,1,2,\ldots$ (t 代表迭代次数)

$$w_{t+1} = w_t + \eta \cdot \mathbf{v}$$

…直到满足停止条件，此时的w_{t+1}作为学到的g





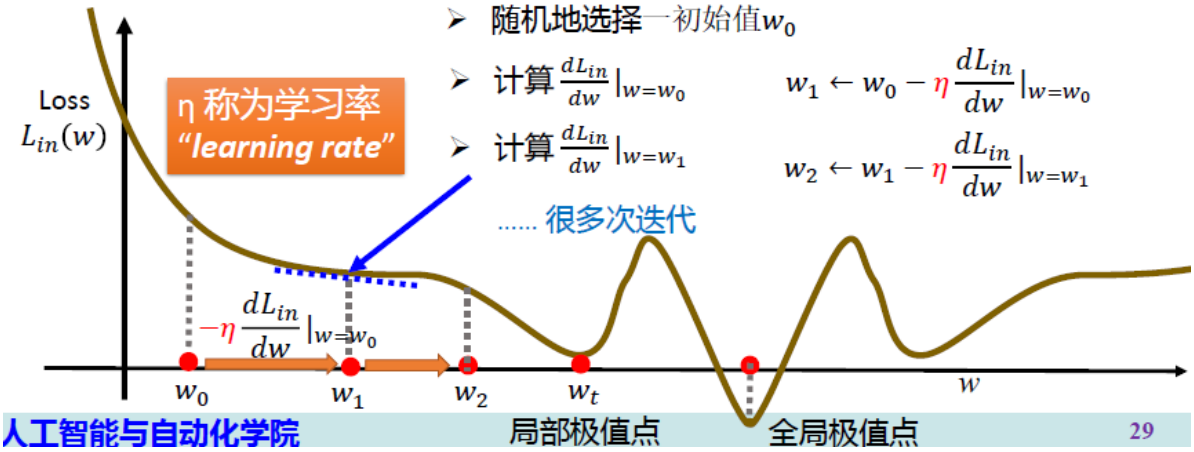


迭代优化:

假设$L_{in}(w)$是单变量w的函数 $w^* = arg\min_{w} L_{in}(w)$

- 随机地选择一初始值w_0
- 计算 $\frac{dL_{in}}{dw}|_{w=w_0}$ $w_1 \leftarrow w_0 - \eta \frac{dL_{in}}{dw}|_{w=w_0}$
- 计算 $\frac{dL_{in}}{dw}|_{w=w_1}$ $w_2 \leftarrow w_1 - \eta \frac{dL_{in}}{dw}|_{w=w_1}$

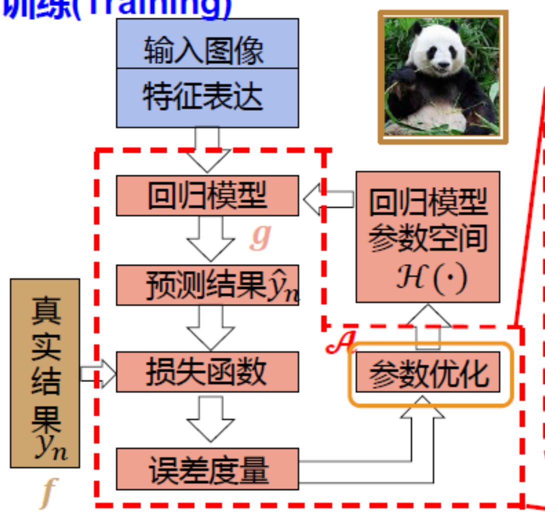
...... 很多次迭代



梯度下降法

3.3 梯度下降法

训练(Training)



梯度下降法:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla L_{in}(\mathbf{w}_t)$$

$$L_{in}(\mathbf{w}) = \frac{1}{N}\|\mathbf{X}\mathbf{w} - Y\|^2$$

$$\nabla L_{in}(\mathbf{w}) = \frac{2}{N}(\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T Y)$$

$$L_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2$$

$$\nabla L_{in}(\mathbf{w}) = \frac{2}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)\mathbf{x}_n$$

梯度下降法实现线性回归

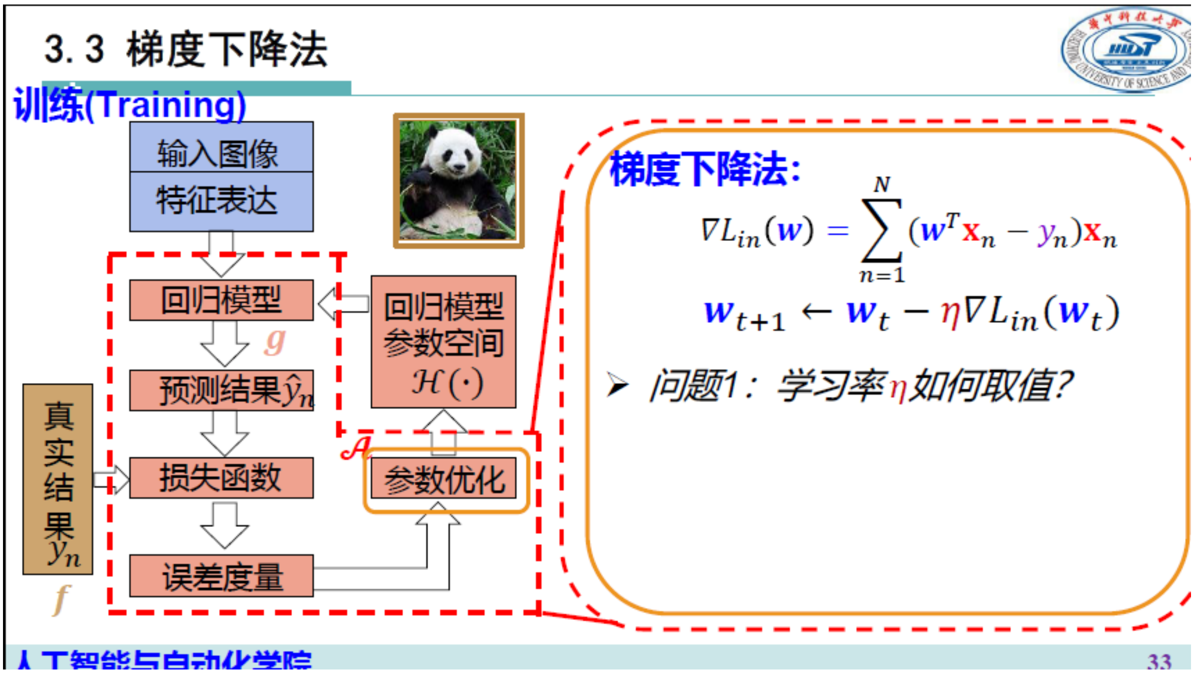
- 初始化权向量w_0
- $for\ t=0,1,2,\ldots$（t 代表迭代次数）

  ① 计算梯度：$\nabla L_{in}(\mathbf{w})=\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n-y_n)\mathbf{x}_n$

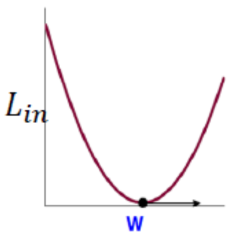
  ② 对权向量$\mathbf{w}_t$进行更新：$\mathbf{w}_{t+1}\leftarrow\mathbf{w}_t-\eta\nabla L_{in}(\mathbf{w}_t)$

…直到$\nabla L_{in}(\mathbf{w})=\mathbf{0}$，或者迭代足够多次数

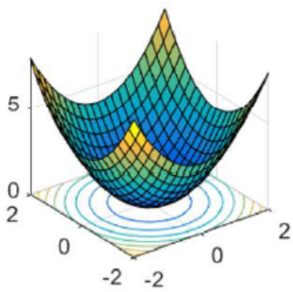
返回最终的$\mathbf{w}_{t+1}$作为学到的g



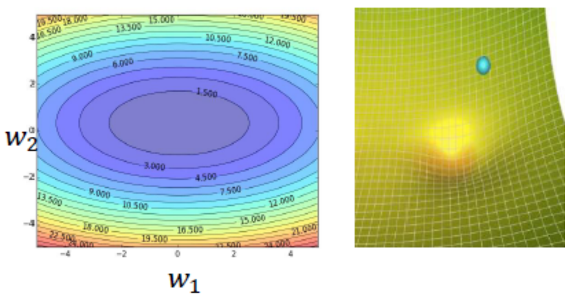
损失函数曲线的可视化：



单变量w的损失函数曲线



两个变量w的损失函数曲线

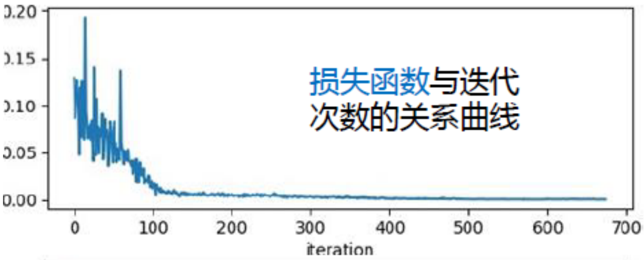


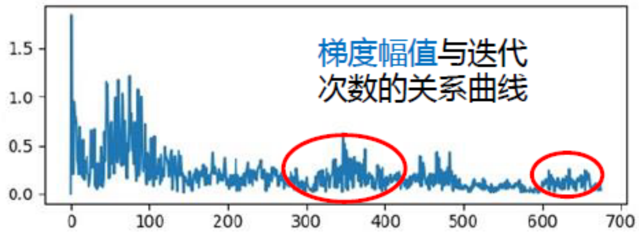
损失函数曲线的等高线表示

（*红色陡峭，蓝色平缓*）

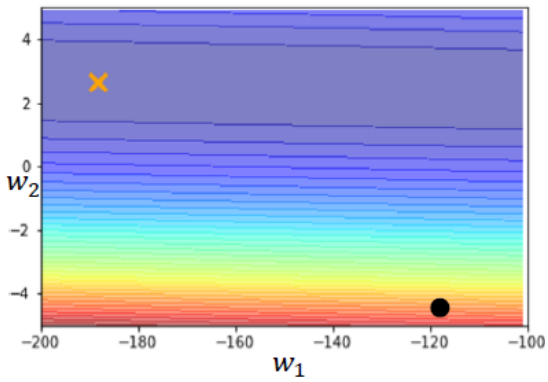
AdaGrad 自适应梯度下降法>

学习率 η 的取值讨论：



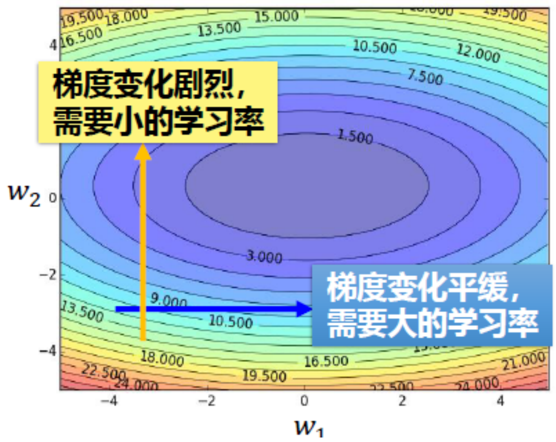


此时已经到达谷底了?



损失函数曲线的等高线表示

学习率 η 的取值讨论：



$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \nabla L_{in}(\boldsymbol{w}_t)$$

$$w_{i,t+1} \leftarrow w_{i,t} - \eta \frac{\partial L_{in}}{\partial w_{i,t}}$$

$$w_{i,t+1} \leftarrow w_{i,t} - \boxed{\frac{\eta}{\sigma_{i,t}}} \frac{\partial L_{in}}{\partial w_{i,t}}$$

与 i 有关，
与迭代时刻的梯度有关

3. 3 梯度下降法

Root Mean Square：

$$w_{i,t+1} \leftarrow w_{i,t} - \frac{\eta}{\sigma_{i,t}} \frac{\partial L_{in}}{\partial w_{i,t}}$$

$$w_{i,1} \leftarrow w_{i,0} - \frac{\eta}{\sigma_{i,0}} \frac{\partial L_{in}}{\partial w_{i,0}} \qquad \sigma_{i,0} = \sqrt{\left(\frac{\partial L_{in}}{\partial w_{i,0}}\right)^2}$$

$$w_{i,2} \leftarrow w_{i,1} - \frac{\eta}{\sigma_{i,1}} \frac{\partial L_{in}}{\partial w_{i,1}} \qquad \sigma_{i,1} = \sqrt{\frac{1}{2}\left[\left(\frac{\partial L_{in}}{\partial w_{i,0}}\right)^2 + \left(\frac{\partial L_{in}}{\partial w_{i,1}}\right)^2\right]}$$

$$w_{i,3} \leftarrow w_{i,2} - \frac{\eta}{\sigma_{i,2}} \frac{\partial L_{in}}{\partial w_{i,2}} \qquad \sigma_{i,2} = \sqrt{\frac{1}{3}\left[\left(\frac{\partial L_{in}}{\partial w_{i,0}}\right)^2 + \left(\frac{\partial L_{in}}{\partial w_{i,1}}\right)^2 + \left(\frac{\partial L_{in}}{\partial w_{i,2}}\right)^2\right]}$$

$$w_{i,t+1} \leftarrow w_{i,t} - \frac{\eta}{\sigma_{i,t}} \frac{\partial L_{in}}{\partial w_{i,t}} \qquad \sigma_{i,t} = \sqrt{\frac{1}{t+1}\sum_{t=0}^{t}\left(\frac{\partial L_{in}}{\partial w_{i,t}}\right)^2}$$

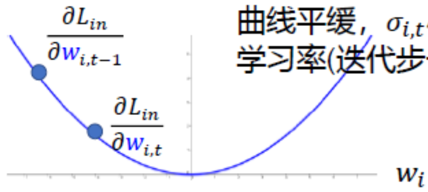
自适应动态学习率

(Learning rate adapts dynamically):

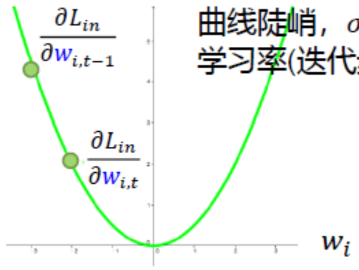
$$w_{i,t+1} \leftarrow w_{i,t} - \frac{\eta}{\sigma_{i,t}} \frac{\partial L_{in}}{\partial w_{i,t}}$$

$$\sigma_{i,t} = \sqrt{\frac{1}{t+1}\sum_{t=0}^{t}\left(\frac{\partial L_{in}}{\partial w_{i,t}}\right)^2}$$

AdaGrad

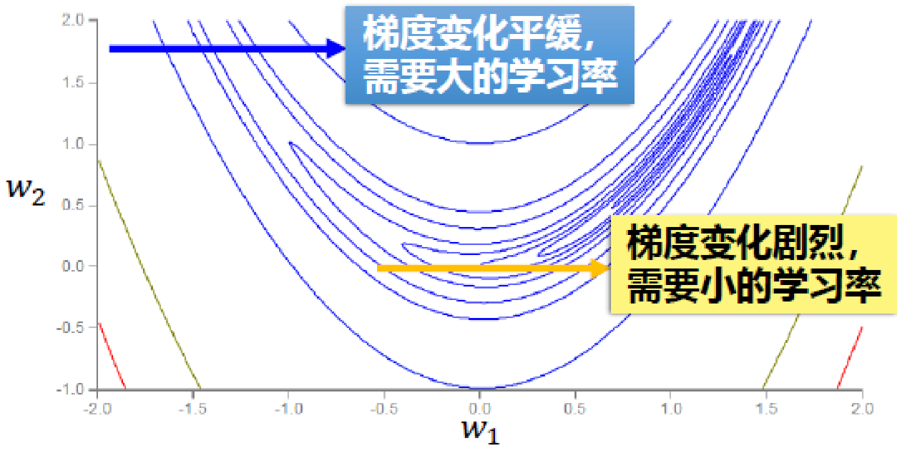


曲线平缓，$\sigma_{i,t}$小
学习率(迭代步长)大



曲线陡峭，$\sigma_{i,t}$大
学习率(迭代步长)小

自适应动态学习率(Learning rate adapts dynamically):



RMSProp

RMSProp：

$$w_{i,t+1} \leftarrow w_{i,t} - \frac{\eta}{\sigma_{i,t}} \frac{\partial L_{in}}{\partial w_{i,t}}$$

$$w_{i,1} \leftarrow w_{i,0} - \frac{\eta}{\sigma_{i,0}} \frac{\partial L_{in}}{\partial w_{i,0}} \qquad \sigma_{i,0} = \sqrt{(\frac{\partial L_{in}}{\partial w_{i,0}})^2}$$

$$0 < \alpha < 1$$

$$w_{i,2} \leftarrow w_{i,1} - \frac{\eta}{\sigma_{i,1}} \frac{\partial L_{in}}{\partial w_{i,1}} \qquad \sigma_{i,1} = \sqrt{\alpha(\sigma_{i,0})^2 + (1-\alpha)(\frac{\partial L_{in}}{\partial w_{i,1}})^2}$$

$$w_{i,3} \leftarrow w_{i,2} - \frac{\eta}{\sigma_{i,2}} \frac{\partial L_{in}}{\partial w_{i,2}} \qquad \sigma_{i,2} = \sqrt{\alpha(\sigma_{i,1})^2 + (1-\alpha)(\frac{\partial L_{in}}{\partial w_{i,2}})^2}$$

$$w_{i,t+1} \leftarrow w_{i,t} - \frac{\eta}{\sigma_{i,t}} \frac{\partial L_{in}}{\partial w_{i,t}} \qquad \sigma_{i,t} = \sqrt{\alpha(\sigma_{i,t-1})^2 + (1-\alpha)(\frac{\partial L_{in}}{\partial w_{i,t}})^2}$$

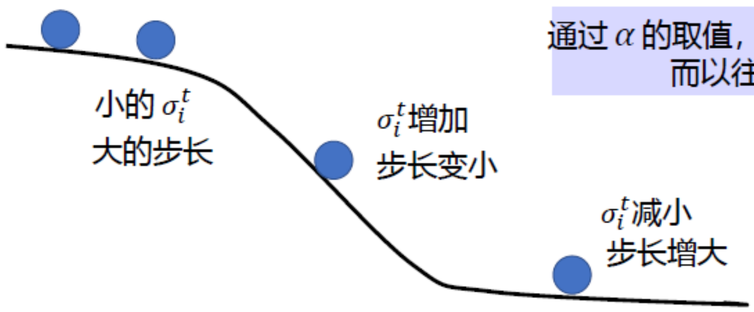
RMSProp：

$$\frac{\partial L_{in}}{\partial w_{i,0}}, \frac{\partial L_{in}}{\partial w_{i,1}}, \ldots, \frac{\partial L_{in}}{\partial w_{i,t-1}},$$

$$0 < \alpha < 1$$

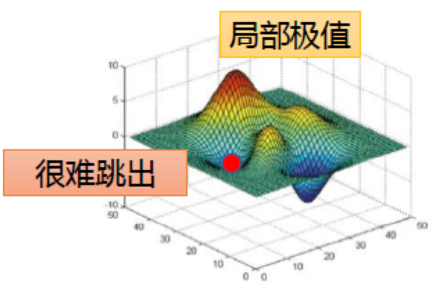
$$w_{i,t+1} \leftarrow w_{i,t} - \frac{\eta}{\sigma_{i,t}} \frac{\partial L_{in}}{\partial w_{i,t}} \qquad \sigma_{i,t} = \sqrt{\alpha(\sigma_{i,t-1})^2 + (1-\alpha)(\frac{\partial L_{in}}{\partial w_{i,t}})^2}$$

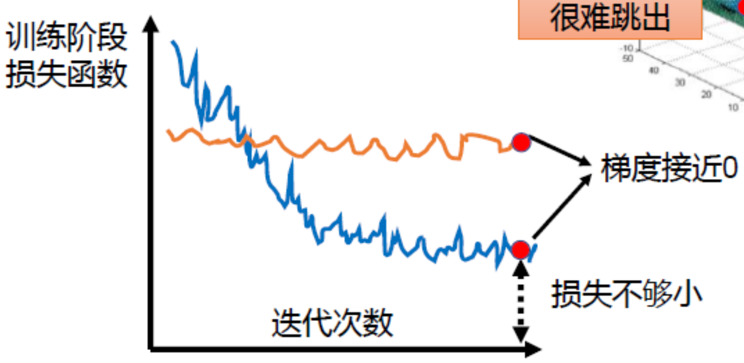
通过 α 的取值，使得当前的梯度影响更大，而以往的梯度影响较小

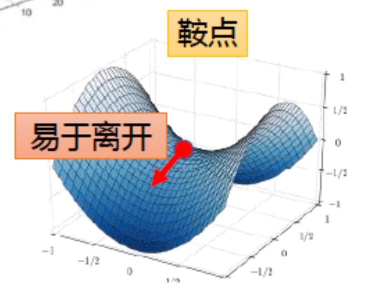


问题2：梯度为0就能得到全局最优解吗？ ×

梯度为0时没能得到最佳解





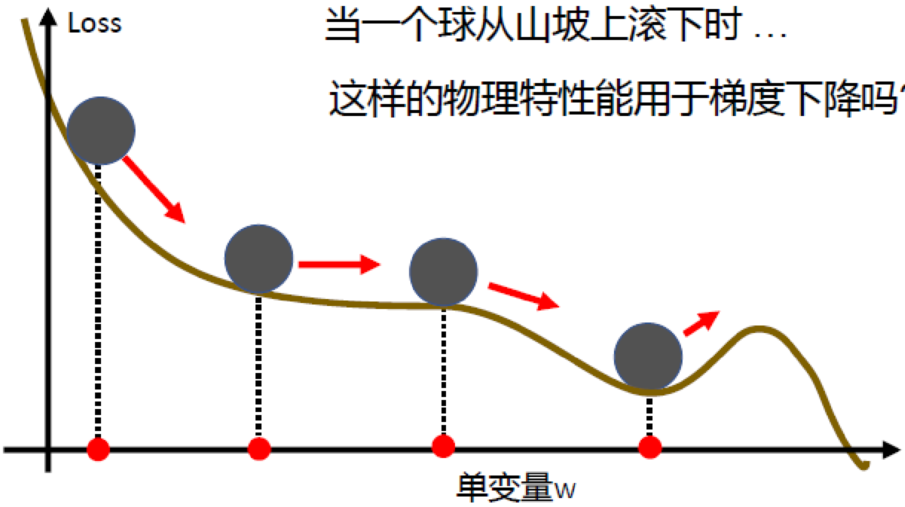


Momentum 动量法梯度下降

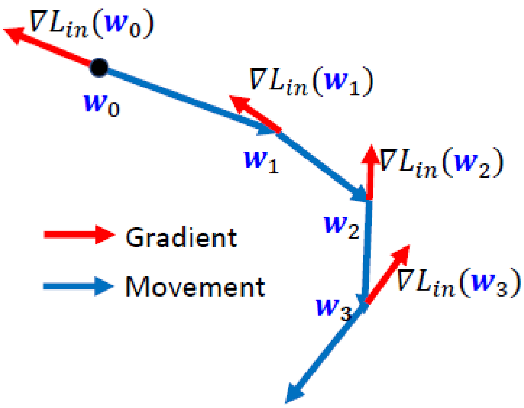
梯度较小时几乎停止更新合理吗?

当一个球从山坡上滚下时 …

这样的物理特性能用于梯度下降吗?



用向量几何示意一般的梯度下降过程



初始值为$\boldsymbol{w}_0$

计算梯度：$\nabla L_{in}(\boldsymbol{w}_0)$

权向量更新：$\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_0 - \eta \nabla L_{in}(\boldsymbol{w}_0)$

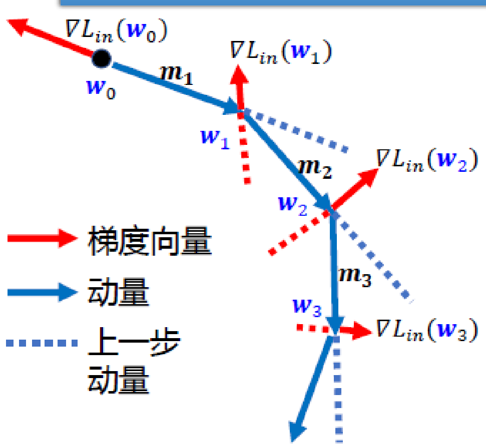
计算梯度：$\nabla L_{in}(\boldsymbol{w}_1)$

权向量更新：$\boldsymbol{w}_2 \leftarrow \boldsymbol{w}_1 - \eta \nabla L_{in}(\boldsymbol{w}_1)$

⋮

结合动量特性的梯度下降法(*Gradient Descent + Momentum*)

动量:上一步的动量减去当前梯度



初始值为$\boldsymbol{w}_0$

动量 $\boldsymbol{m_0} = \boldsymbol{0}$

计算梯度：$\nabla L_{in}(\boldsymbol{w}_0)$

动量 $\boldsymbol{m_1} = \lambda \boldsymbol{m_0} - \eta \nabla L_{in}(\boldsymbol{w}_0)$

权向量更新$\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_0 + \boldsymbol{m_1}$

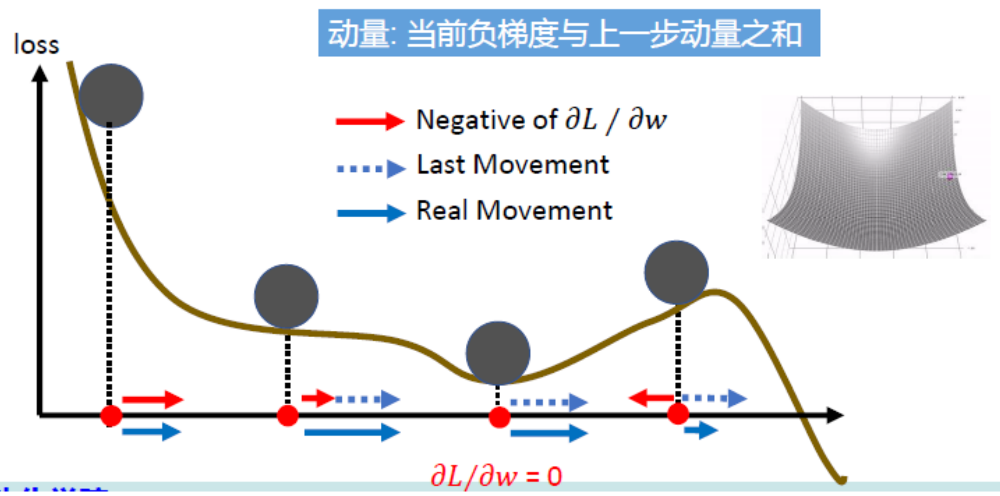
计算梯度：$\nabla L_{in}(\boldsymbol{w}_1)$

动量 $\boldsymbol{m_2} = \lambda \boldsymbol{m_1} - \eta \nabla L_{in}(\boldsymbol{w}_1)$

权向量更新$\boldsymbol{w}_2 \leftarrow \boldsymbol{w}_1 + \boldsymbol{m_2}$

权向量更新时不仅考虑负梯度方向
还同时要考虑上一步的动量

结合动量特性的梯度下降法(*Gradient Descent + Momentum*)



Adam 亚当优化器

Adam: RMSProp + Momentum

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum
$v_0 \leftarrow 0$ (Initialize 2nd moment vector) → for RMSprop
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

$$\nabla L_{in}(\mathbf{w}) = \sum_{n=1} (\mathbf{w}^T \mathbf{x}_n - y_n)\mathbf{x}_n$$

$$m_{i,t+1} = \lambda m_{i,t} - \frac{\eta}{\sigma_{i,t}} \frac{\partial L_{in}}{\partial w_{i,t}}$$

$$w_{i,t+1} \leftarrow w_{i,t} + m_{i,t+1}$$

问题3：训练样本批量大小的影响?

batch_size 的影响

梯度下降法实现线性回归

- 初始化权向量$\mathbf{w}_0$
- $for\ t = 0,1,2,\ldots$ (t 代表迭代次数)

① 计算梯度： $\nabla L_{in}(\mathbf{w}) = \boxed{\frac{2}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)\mathbf{x}_n}$

每次迭代N个样本均要计算，时间复杂度与Pocket算法相似

② 对权向量$\mathbf{w}_t$进行更新：$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta\nabla L_{in}(\mathbf{w}_t)$

…直到$\nabla L_{in}(\mathbf{w}) = \mathbf{0}$，或者迭代足够多次数

返回最终的$\mathbf{w}_{t+1}$作为学到的g

梯度下降法实现线性回归

- 初始化权向量$\mathbf{w}_0$
- $for\ t = 0,1,2,\ldots$ (t 代表迭代次数)

① 计算梯度： $\nabla L_{in}(\mathbf{w}) = (\mathbf{w}^T\mathbf{x}_n - y_n)\mathbf{x}_n$

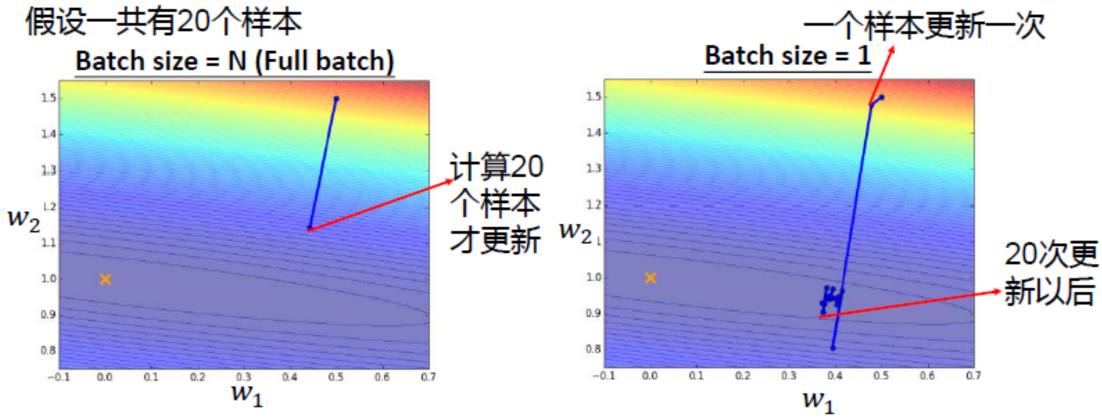
随机梯度下降法
(Stochastic Gradient Descent)
(SGD)

② 对权向量$\mathbf{w}_t$进行更新：$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta\nabla L_{in}(\mathbf{w}_t)$

…直到$\nabla L_{in}(\mathbf{w}) = \mathbf{0}$，或者迭代足够多次数

返回最终的$\mathbf{w}_{t+1}$作为学到的g

迭代过程中一次计算样本多少(batch)对梯度下降的影响



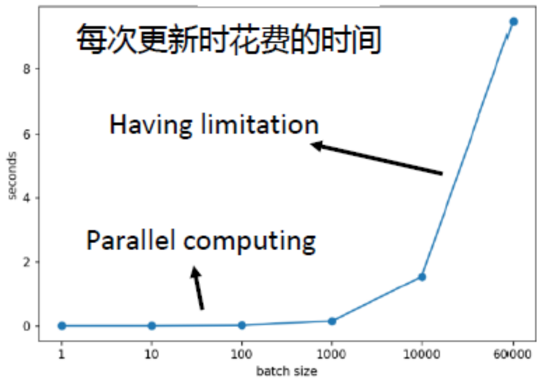
20个样本均参与计算后才更新

计算每个样本后就更新

迭代过程中一次计算样本多少(batch)对梯度下降的影响

MNIST: digit classification

Tesla V100 GPU

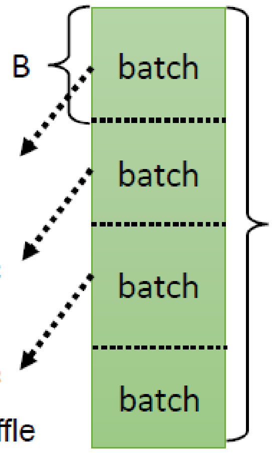


批量(batch size)大在计算梯度时并不意味着会花费更多的时间，除非batch size 太大

批量(batch)用于梯度下降

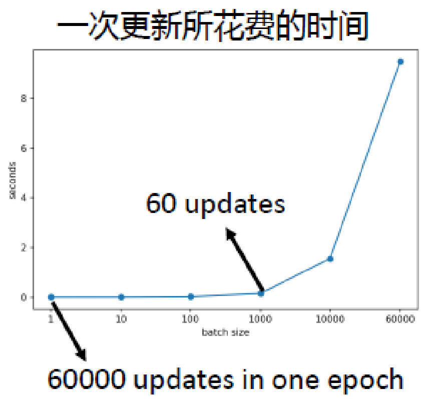
$$w^* = arg\min_{w} L$$

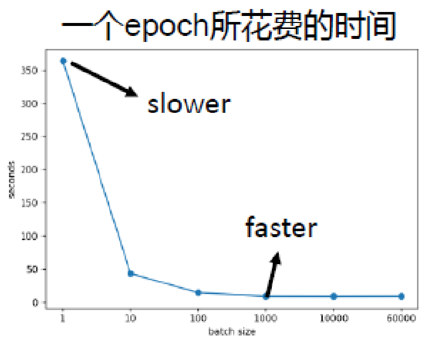
- 随机设置一个初始向量 w_0
- 计算梯度 $\nabla L_1(w_0)$，更新： $w_1 \leftarrow w_0 - \eta \nabla L_1(w_0)$
- 计算梯度 $\nabla L_2(w_1)$，更新： $w_2 \leftarrow w_1 - \eta \nabla L_2(w_1)$
- 计算梯度 $\nabla L_3(w_2)$，更新： $w_3 \leftarrow w_2 - \eta \nabla L_3(w_2)$



1 epoch = 所有的batch遍历一遍 → 每一次epoch后进行Shuffle

批量(batch)大小对训练过程速度的影响

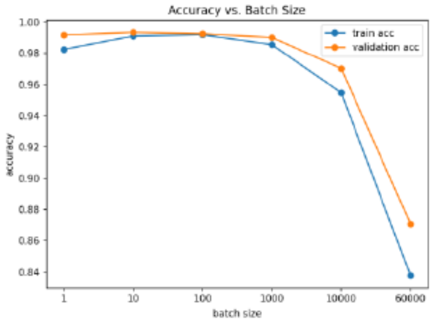




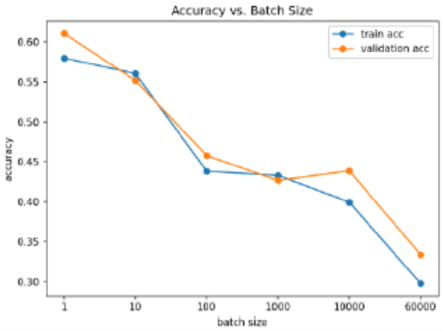
较小的批量(batch)做完一次epoch时需要花费更多的时间

批量(batch)大小对分类性能的影响

MNIST



CIFAR-10



较小的批量获得更好的性能

为什么批量大了性能会下降呢?

批量(batch)大小对分类性能的影响

SB = 256

LB = 0.1 x data set

Name	Network Type	Data set
F_1	Fully Connected	MNIST (LeCun et al., 1998a)
F_2	Fully Connected	TIMIT (Garofolo et al., 1993)
C_1	(Shallow) Convolutional	CIFAR-10 (Krizhevsky & Hinton, 2009)
C_2	(Deep) Convolutional	CIFAR-10
C_3	(Shallow) Convolutional	CIFAR-100 (Krizhevsky & Hinton, 2009)
C_4	(Deep) Convolutional	CIFAR-100

	Training Accuracy		Testing Accuracy	
Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima

https://arxiv.org/abs/1609.04836

较小的批量在测试数据集上也能获得更好的性能

批量(batch)大小对梯度下降优化时的影响总结

	批量小	批量大
一次更新需要的速度(无并行处理)	Faster	Slower
一次更新需要的速度(有并行处理)	Same	Same (not too large)
一个epoch花费的时间	Slower	Faster WIN
梯度的特点	Noisy	Stable
优化性能	Better WIN	Worse
泛化性能	Better WIN	Worse

批量大小(batch size)作为超参数(hyperparameter)由算法设计者确定

小结

3.1 线性回归问题

模型的输出为实数值，有众多应用场景

3.2 线性回归算法

损失函数为均方误差时，可通过求解广义逆得到解析解

3.3 梯度下降法

迭代优化，更一般的损失函数；固定学习率、AdaGrad、RMSProp 、动量(Momentum)、Adam、SGD、批量大小(batch size)

作业

手写作业

2，根据向量或矩阵的计算性质，证明：

$$\|\mathbf{X}\boldsymbol{w} - Y\|^2 = \boldsymbol{w}^T\mathbf{X}^T\mathbf{X}\boldsymbol{w} - 2\boldsymbol{w}^T\mathbf{X}^T\mathrm{Y} + \mathrm{Y}^T\mathrm{Y}$$

T2. 解:

证明: $\|Xw-Y\|^2 = w^TX^TXw - 2w^TX^TY + Y^TY$

$$
\begin{aligned}
\|Xw-Y\|^2 &= (Xw-Y)^T(Xw-Y) \\
&= (w^TX^T-Y^T)(Xw-Y) \\
&= w^TX^TXw - w^TX^TY - \boxed{Y^TXw} + Y^TY \\
&= w^TX^TXw - w^TX^TY - \boxed{(Xw)^TY} + Y^TY \\
&= w^TX^TXw - 2w^TX^TY + Y^TY.
\end{aligned}
$$