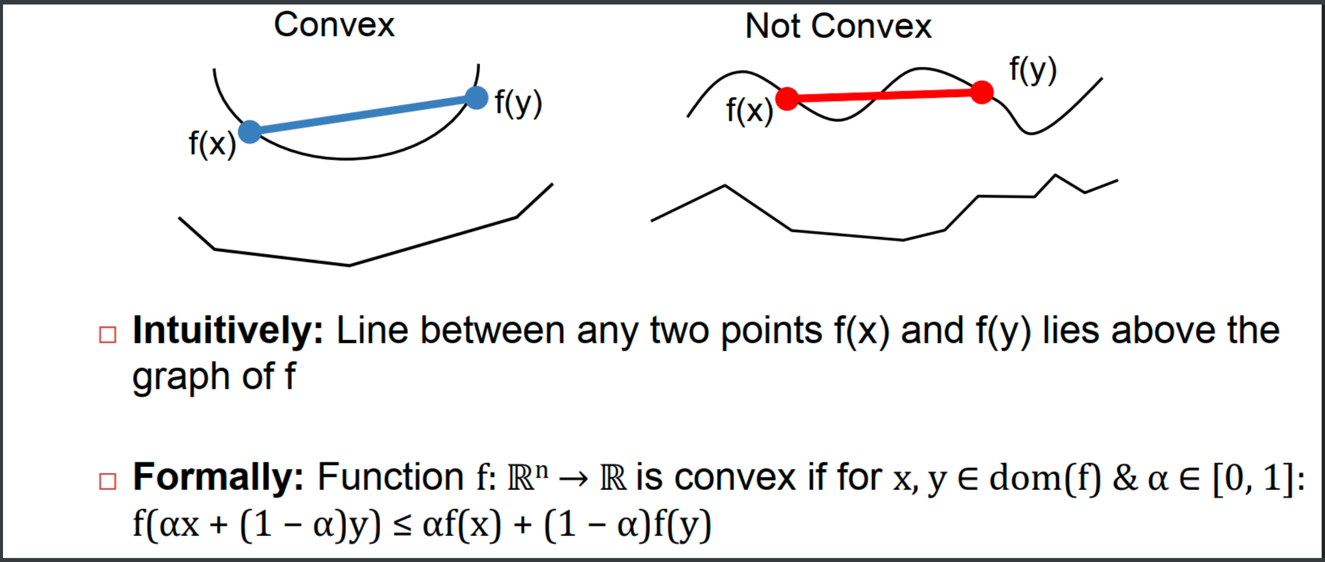
optimization

convex optimization & common techniques

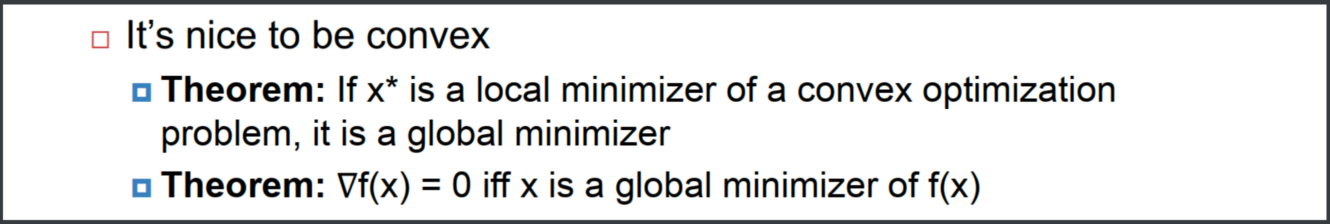
- what is (not) convex



- **Intuitively:** Line between any two points f(x) and f(y) lies above the graph of f
- **Formally:** Function $f: \mathbb{R}^n \to \mathbb{R}$ is convex if for $x, y \in \mathrm{dom}(f)$ & $\alpha \in [0, 1]$: $f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$

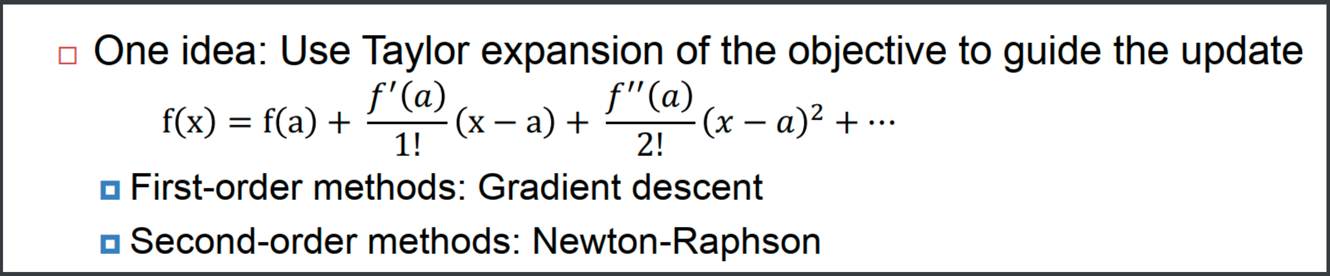
interpolation never underestimates the function value

- why convexity



- It's nice to be convex
  - **Theorem:** If x* is a local minimizer of a convex optimization problem, it is a global minimizer
  - **Theorem:** $\nabla f(x) = 0$ iff x is a global minimizer of f(x)

- first-order methods & second-order methods:



- One idea: Use Taylor expansion of the objective to guide the update

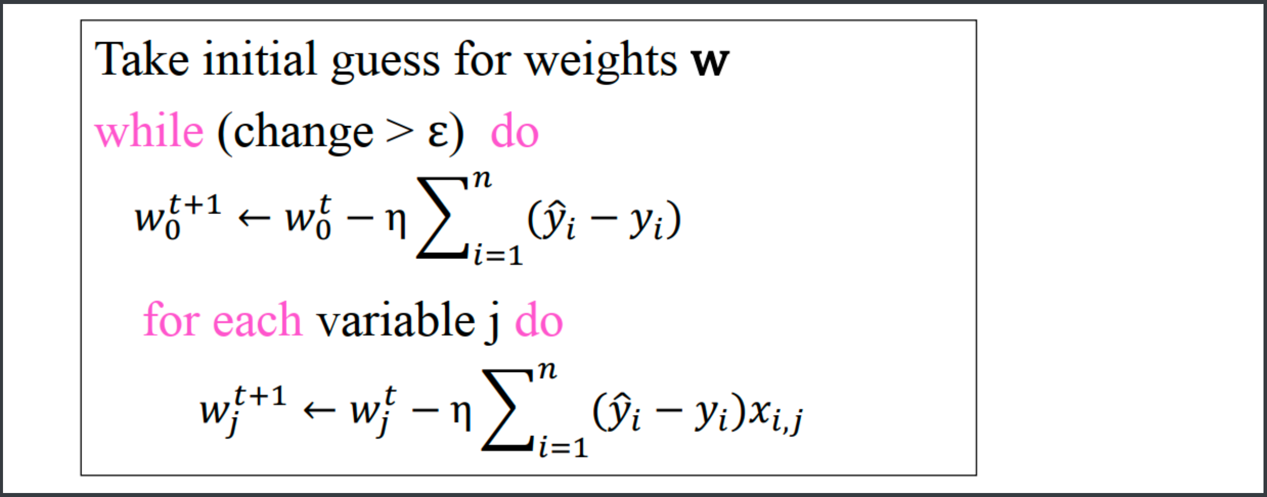
$$f(x) = f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \cdots$$

  - First-order methods: Gradient descent
  - Second-order methods: Newton-Raphson

- first-order method: gradient descent

  1. naive version

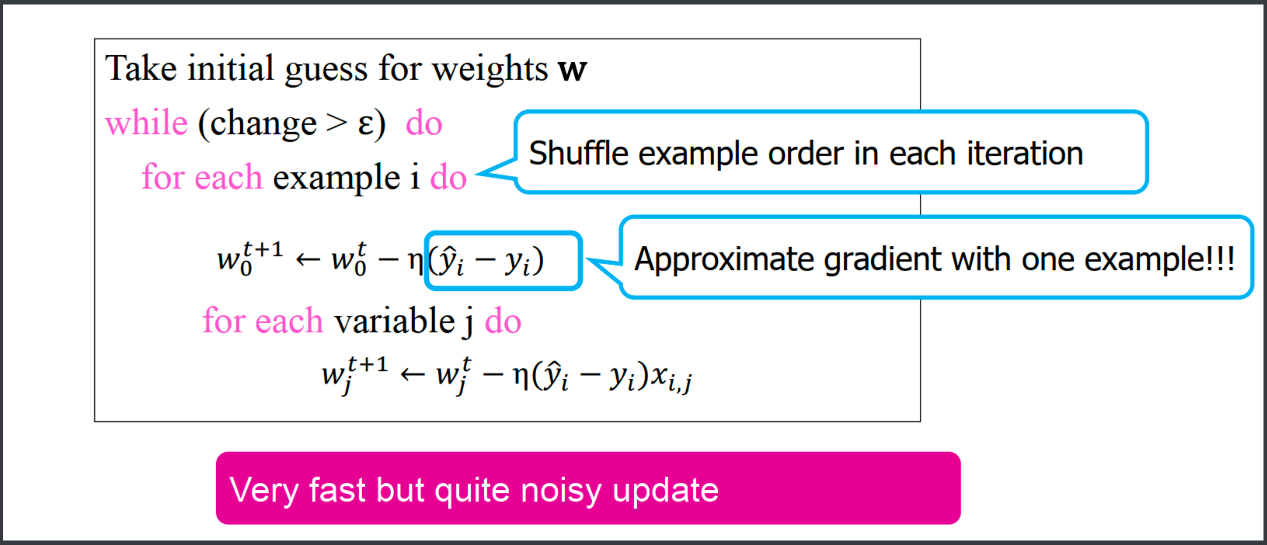
     e.g. for linear regression



```
Take initial guess for weights **w**
while (change > ε) do
    $w_0^{t+1} \leftarrow w_0^t - \eta \sum_{i=1}^{n} (\hat{y}_i - y_i)$
    for each variable j do
        $w_j^{t+1} \leftarrow w_j^t - \eta \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{i,j}$
```

this has to iterate over all examples, which is slow; -> SGD

2. SGD



```
Take initial guess for weights **w**
while (change > ε) do
    for each example i do
        $w_0^{t+1} \leftarrow w_0^t - \eta \boxed{(\hat{y}_i - y_i)}$
        for each variable j do
            $w_j^{t+1} \leftarrow w_j^t - \eta (\hat{y}_i - y_i) x_{i,j}$
```

Shuffle example order in each iteration

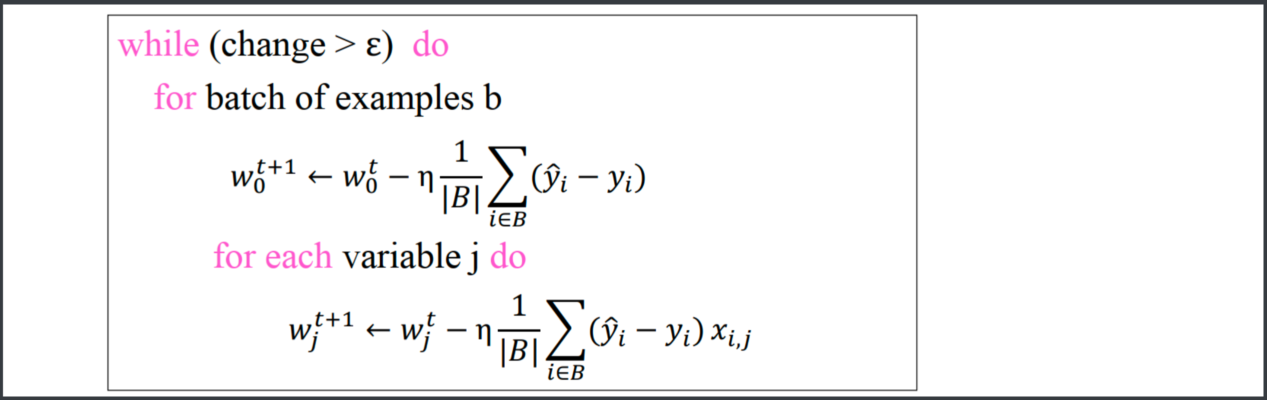
Approximate gradient with one example!!!

Very fast but quite noisy update

instead of using all examples, use only one example each time to compute the gradients

in this circumstance, gradient updates might not be accurate but still in the direction towards global optimal ("quite noisy update")

3. mini-batch gradient descent



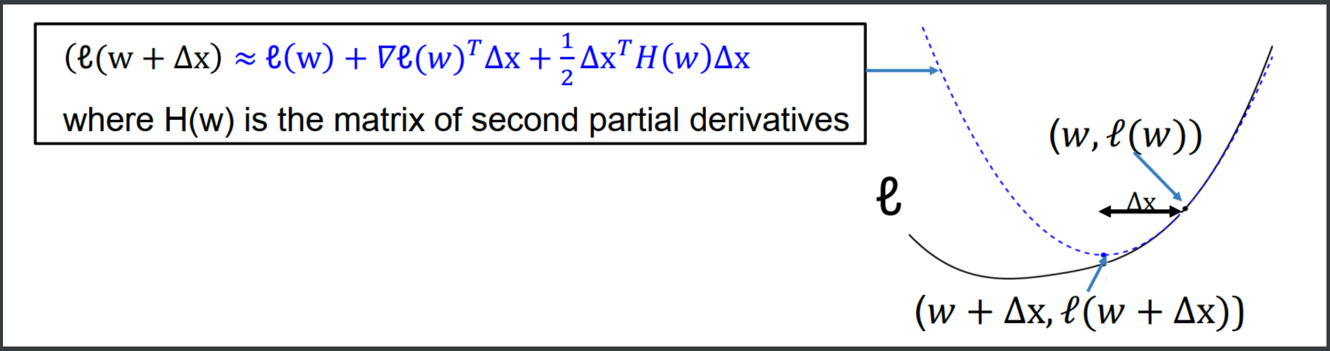
```
while (change > ε) do
    for batch of examples b
        $w_0^{t+1} \leftarrow w_0^t - \eta \frac{1}{|B|} \sum_{i \in B} (\hat{y}_i - y_i)$
        for each variable j do
            $w_j^{t+1} \leftarrow w_j^t - \eta \frac{1}{|B|} \sum_{i \in B} (\hat{y}_i - y_i) x_{i,j}$
```

averaging reduces noise, compared to SGD small batches can also exploit (data) parallelism

- second-order method

  how to update:

  according to second-order expansion,



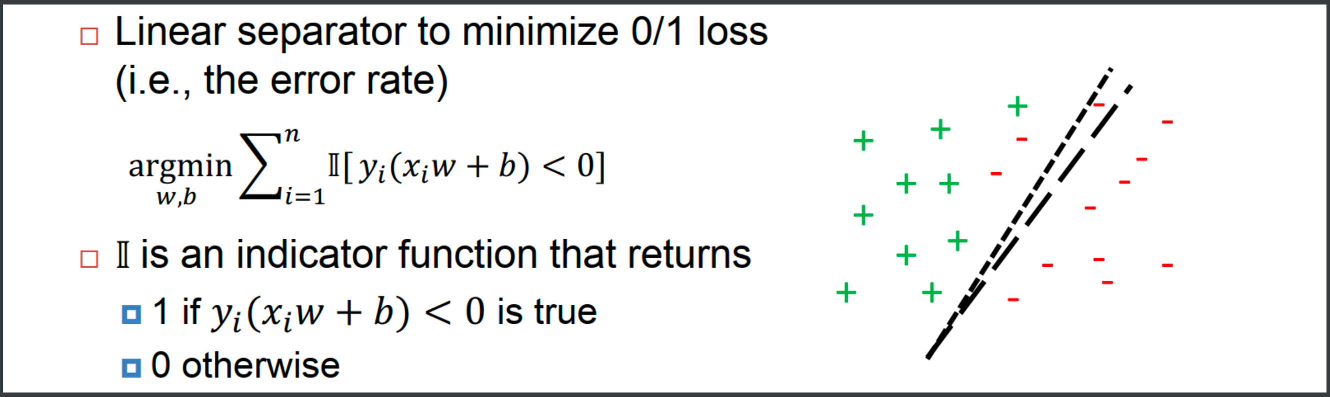
$(\ell(w+\Delta x) \approx \ell(w) + \nabla\ell(w)^T\Delta x + \frac{1}{2}\Delta x^T H(w)\Delta x$

where H(w) is the matrix of second partial derivatives

  updating rule would be, $\Delta x = -[H(w)^{-1}]\nabla\ell(w)$

  compared to GD: larger step, but longer to compute (Hessian矩阵的逆矩阵计算overhead很大，可以用近似的方法如L-BFGS)

- for classification problems



- Linear separator to minimize 0/1 loss (i.e., the error rate)

$$\underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{n} \mathbb{I}[\,y_i(x_i w + b) < 0]$$

- $\mathbb{I}$ is an indicator function that returns
  - 1 if $y_i(x_i w + b) < 0$ is true
  - 0 otherwise

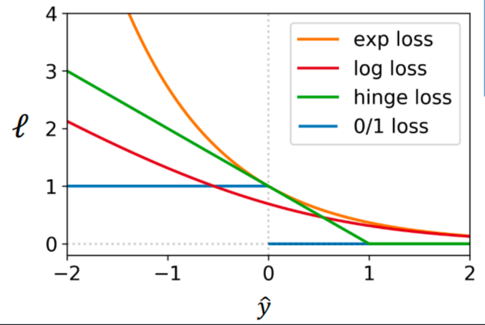
  problem: 0/1 loss is NP-hard to minimize (small changes in w / b can have a big impact on loss)

  solution: convex surrogate loss functions （凸替代loss functions）常见的有：

- Assume that
  - y = true label
  - $\hat{y}$ = predicted label
- Common Losses

$$\ell^{(0/1)} = \mathbb{I}(y\hat{y} < 0)$$

$$\left\{\begin{aligned} \ell^{(hinge)} &= max(0, 1 - y\hat{y}) \\ \ell^{(logistic)} &= \frac{1}{\log(2)}\log(1 + e^{[-y\hat{y}]}) \\ \ell^{(exp)} &= e^{[-y\hat{y}]} \\ \ell^{(sqr)} &= (y - \hat{y})^2 \end{aligned}\right.$$



其中hinge loss用于SVMs，exponential loss 用于adaboost

advanced techniques

how to set the learning rate of GD

too big / too small / function's surface steepness varies by dimension

idea 1: momentum, accelerate gradients

对于方向一致的梯度，累积可以使更新步长更大

对于方向不一致的梯度，累积可以抵消震荡

- Take average of gradients: $v_j^{(t+1)} = \beta v_j^{(t)} + g_j$

still, momentum might be sensitive to β

- Intuition: Ball rolling down loss function's surface

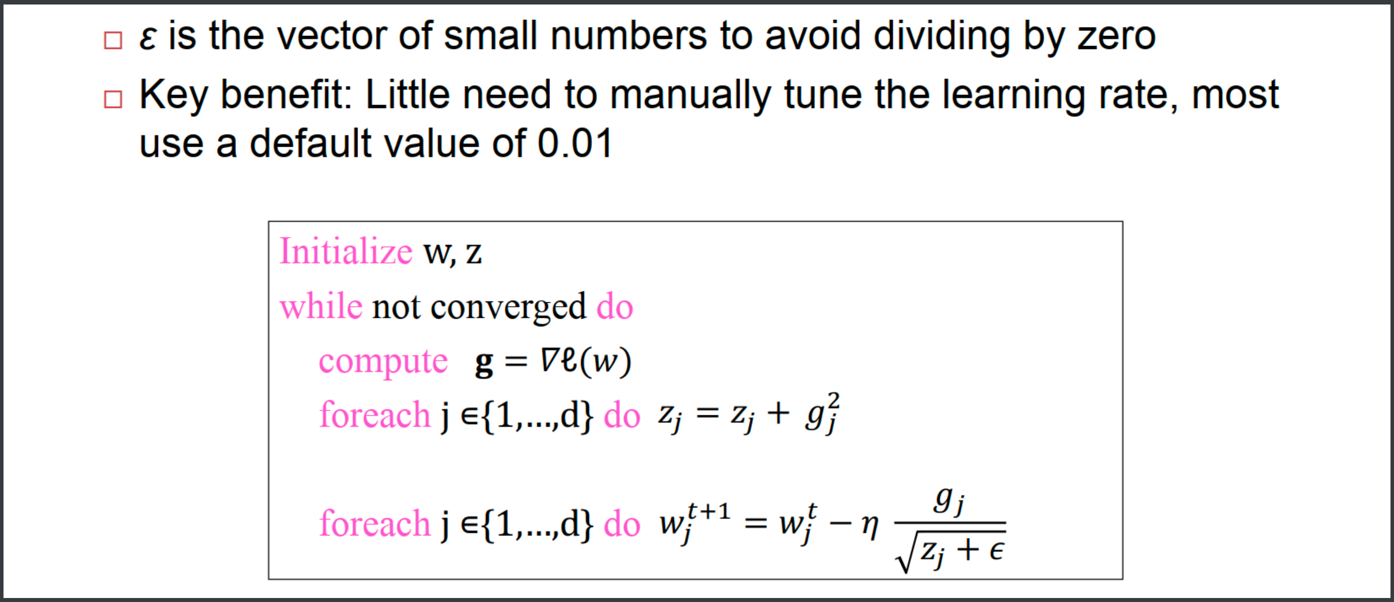
```
Initialize w, v        V initialized to 0s
while not converged do
    compute  $\mathbf{g} = \nabla\ell(w)$
    foreach j ∈{1,...,d} do
        $v_j^{(t+1)} = \beta v_j^{(t)} + g_j$
        $w_j^{(t+1)} = w_j^{(t)} - \eta v_j^{(t+1)}$
```

common settings for β : 0.5 ~ 0.99

idea 2: AdaGrad

set individual learning rate for each dimension,

works best for sparse data (e.g. word counts)



- ε is the vector of small numbers to avoid dividing by zero
- Key benefit: Little need to manually tune the learning rate, most use a default value of 0.01

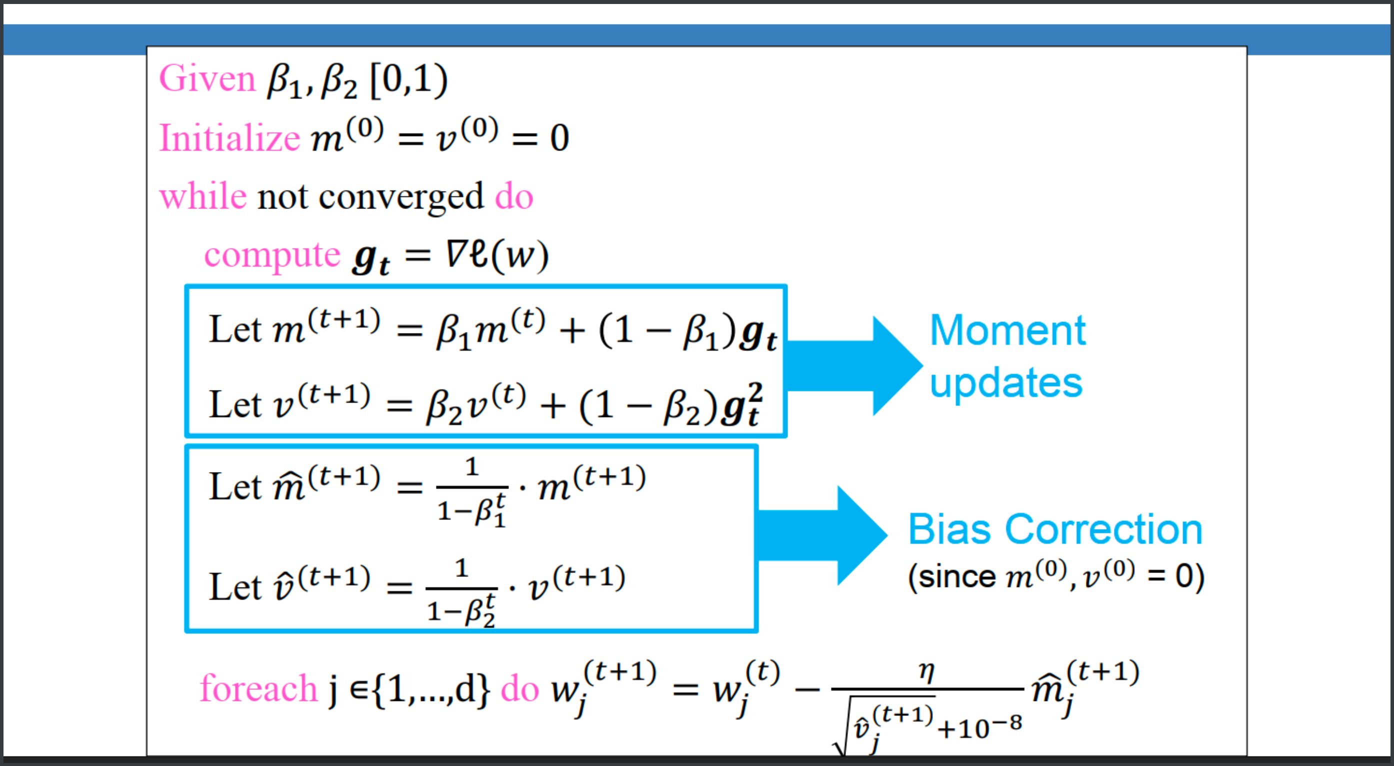
```
Initialize w, z
while not converged do
    compute  $\mathbf{g} = \nabla \ell(w)$
    foreach j ∈{1,...,d} do  $z_j = z_j + g_j^2$

    foreach j ∈{1,...,d} do  $w_j^{t+1} = w_j^t - \eta \frac{g_j}{\sqrt{z_j + \epsilon}}$
```

note that g_j^2 and therefore z_j grows quickly for large gradients

-> AdaGrad decreases step size very aggressively

idea 3: Adam (combination of idea1 and idea2)



```
Given $\beta_1, \beta_2$ [0,1)
Initialize $m^{(0)} = v^{(0)} = 0$
while not converged do
    compute $\boldsymbol{g_t} = \nabla\ell(w)$
    Let $m^{(t+1)} = \beta_1 m^{(t)} + (1-\beta_1)\boldsymbol{g_t}$
    Let $v^{(t+1)} = \beta_2 v^{(t)} + (1-\beta_2)\boldsymbol{g_t^2}$
    Let $\widehat{m}^{(t+1)} = \frac{1}{1-\beta_1^t} \cdot m^{(t+1)}$
    Let $\hat{v}^{(t+1)} = \frac{1}{1-\beta_2^t} \cdot v^{(t+1)}$
    foreach j ∈{1,...,d} do $w_j^{(t+1)} = w_j^{(t)} - \frac{\eta}{\sqrt{\hat{v}_j^{(t+1)}+10^{-8}}}\widehat{m}_j^{(t+1)}$
```

Moment updates

Bias Correction
(since $m^{(0)}, v^{(0)}$ = 0)

在initial time steps 如果decay rate β 比较小，可能会出现动量bias

solution是引入一个bias correlation项

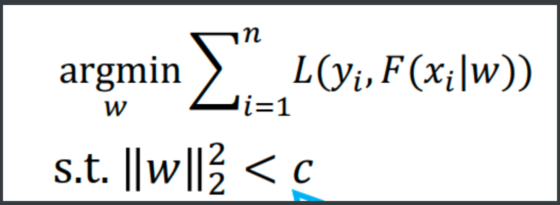
regularization

更fit outlier的拟合或许有更小的loss，但不是我们的目标

regularization introduces a penalty on the size of the weights & encodes a preference for "simpler models", prevent overfitting

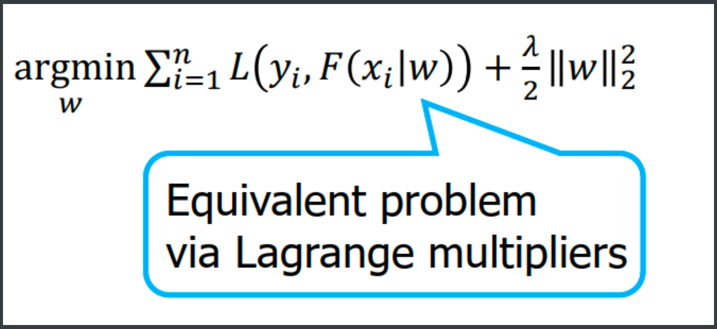
ridge regression: linear regression with L2 regularization

add constraint: c可以理解为weight的budget



$$\underset{w}{\text{argmin}} \sum_{i=1}^{n} L(y_i, F(x_i|w))$$
$$\text{s.t. } \|w\|_2^2 < c$$

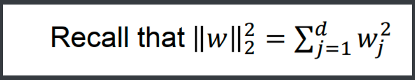
通过lagrange multipler转换等价问题：



$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F(x_i|w)) + \frac{\lambda}{2}\|w\|_2^2$$

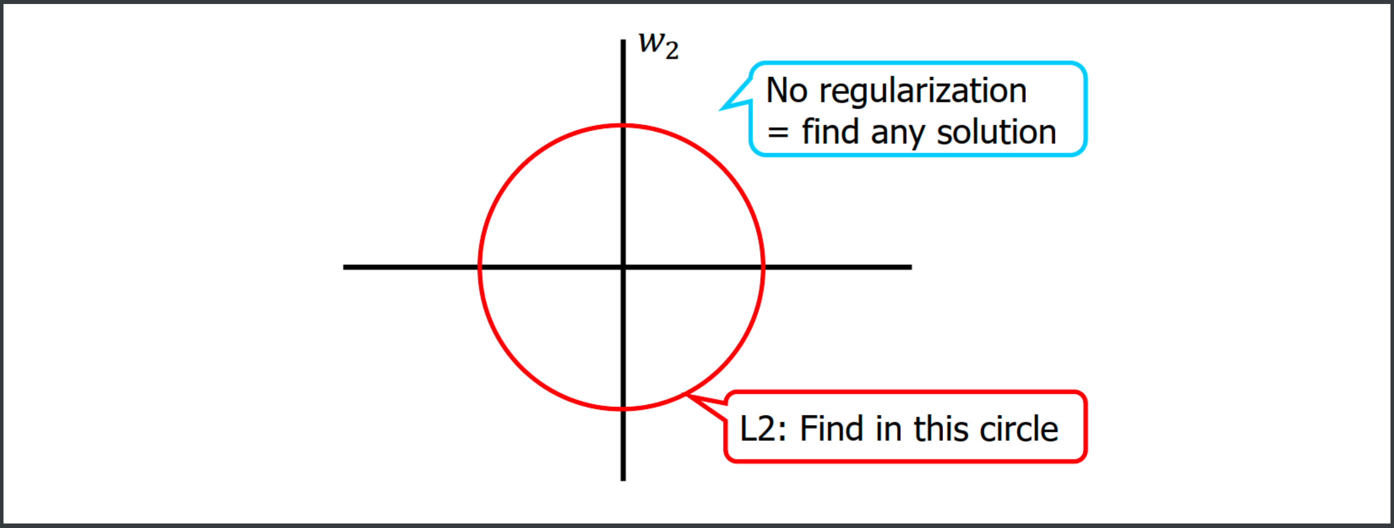
Equivalent problem
via Lagrange multipliers

注意不要正则化偏置项w$_0$，

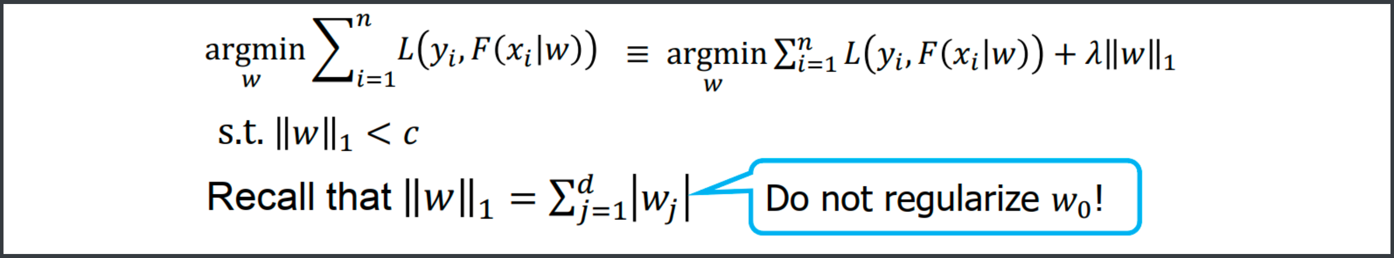


Recall that $\|w\|_2^2 = \sum_{j=1}^{d} w_j^2$

（w$_0$对模型复杂度没有影响）



LASSO / L1 regularization



$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F(x_i|w)) \equiv \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F(x_i|w)) + \lambda\|w\|_1$$

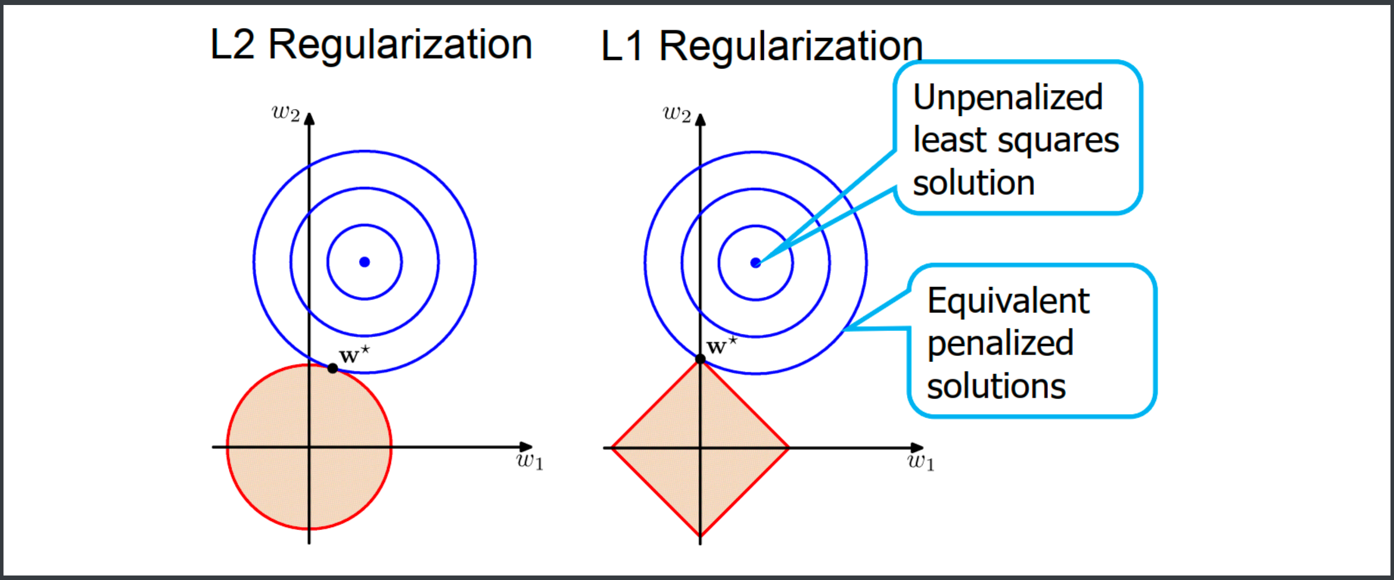
s.t. $\|w\|_1 < c$

Recall that $\|w\|_1 = \sum_{j=1}^{d} |w_j|$

Do not regularize w_0!

encodes preference for sparsity, by forcing many w$_j$ to be zero

benifits: interpretability, computational efficiency


L2 Regularization
L1 Regularization
w_2
w_1
$\mathbf{w}^\star$
Unpenalized least squares solution
Equivalent penalized solutions

problem: absolute value function is not differentiable at zero

solution: proximal methods, subgradients

- subgradient

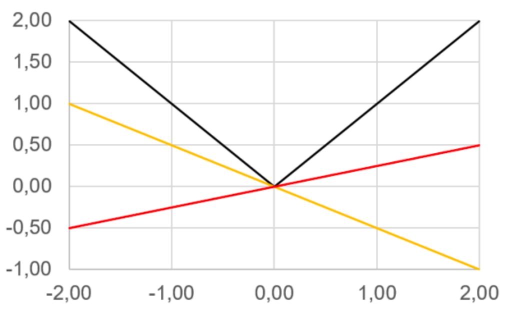
  对凸但不可导的函数，不可导点的导数：

  Subgradient set: $\partial f(x) = \{g \mid f(y) > f(x) + g^T(y-x)\ \forall y\}$

  表示: any supporting hyperplane at x that lower bounds entire function

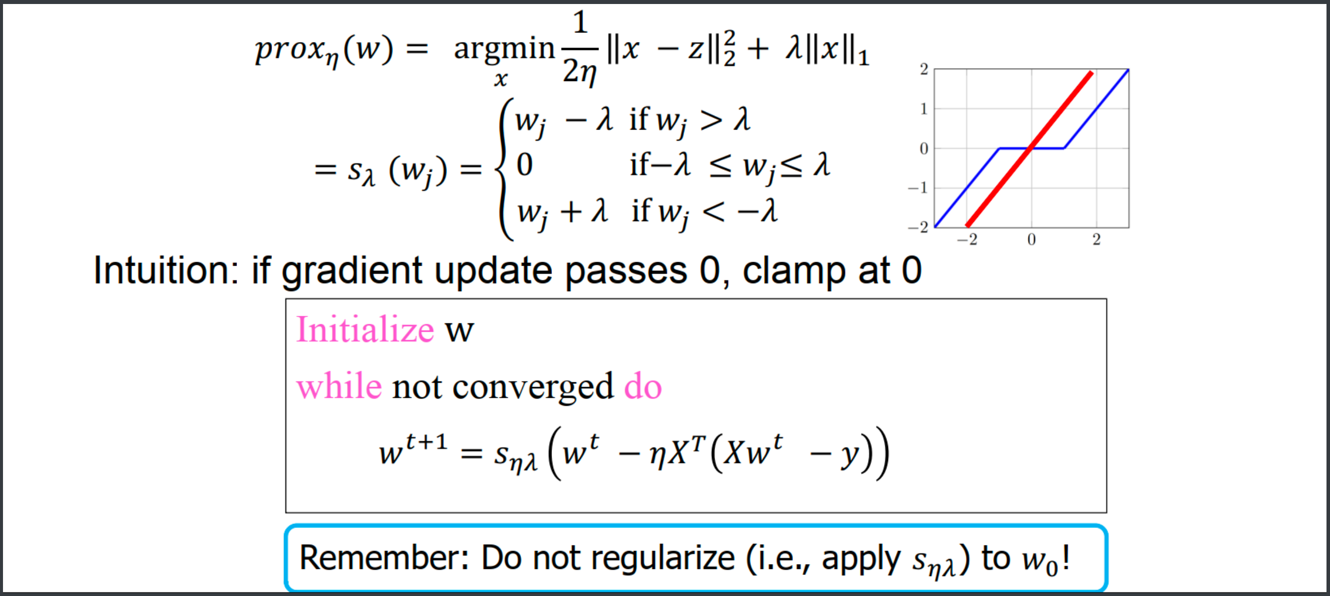
  e.g.

  □ Convex function is minimized at w if $0 \in \partial f(w)$

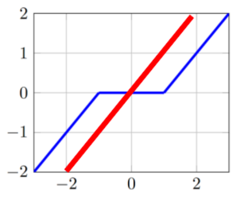
  $$\partial f(|w_j|) = \begin{cases} 1 & \text{if } w_j > 0 \\ [-1,1] & \text{if } w_j = 0 \\ -1 & \text{if } w_j < 0 \end{cases}$$

- proximal approach: ISTA



$$prox_\eta(w) = \underset{x}{\operatorname{argmin}} \frac{1}{2\eta}\|x - z\|_2^2 + \lambda\|x\|_1$$

$$= s_\lambda(w_j) = \begin{cases} w_j - \lambda & \text{if } w_j > \lambda \\ 0 & \text{if} -\lambda \le w_j \le \lambda \\ w_j + \lambda & \text{if } w_j < -\lambda \end{cases}$$



Intuition: if gradient update passes 0, clamp at 0

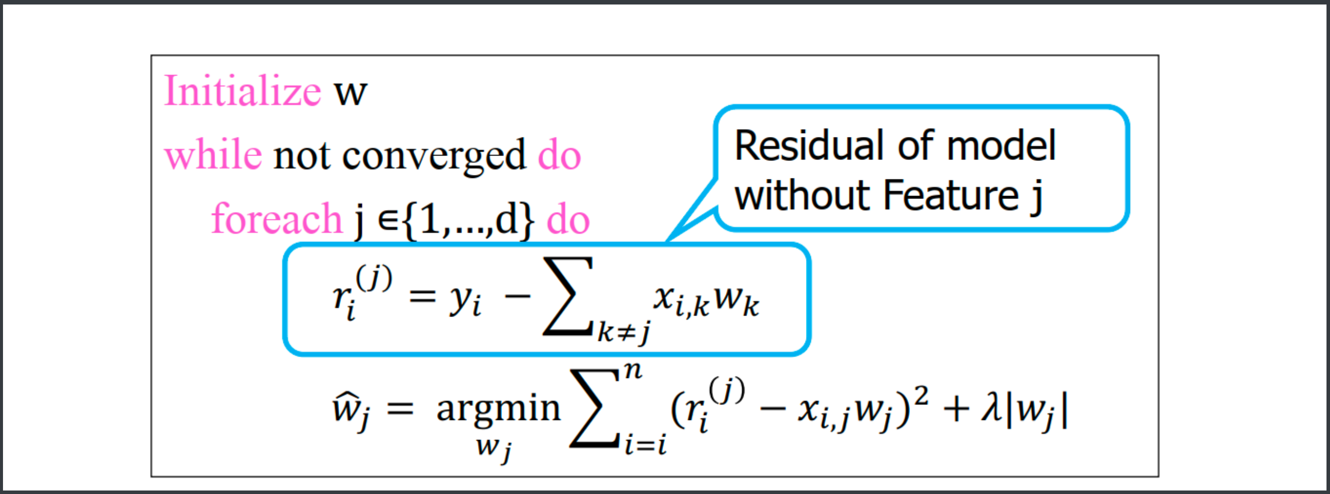
```
Initialize w
while not converged do
    $w^{t+1} = s_{\eta\lambda}\left(w^t - \eta X^T(Xw^t - y)\right)$
```

Remember: Do not regularize (i.e., apply $s_{\eta\lambda}$) to w_0!

- faster approach: coordinate descent

  adjust weight for one feature at a time (often easy to find a minimum along a single coordinate)

  converges for LASSO



```
Initialize w
while not converged do
    foreach j ∈{1,...,d} do
        $r_i^{(j)} = y_i - \sum_{k \neq j} x_{i,k} w_k$
        $\hat{w}_j = \underset{w_j}{\operatorname{argmin}} \sum_{i=i}^{n} (r_i^{(j)} - x_{i,j} w_j)^2 + \lambda |w_j|$
```

Residual of model without Feature j

- summary of LASSO:
  1. can be interpreted as performing feature selection (zero weight = unselected feature)
  2. work well with small sample sizes
  3. drawbacks: