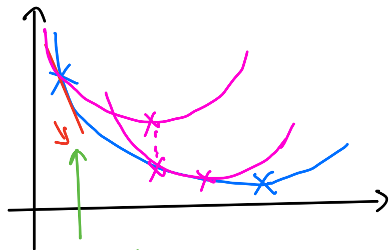
Review: Gradient Descent — first order optimization

↑ only use first derivative



first-order Taylor expansion

$y = f(a) + (x-a) f'(a)$

Q: Why must we take small steps?

A: First-order approx is not reliable when taking large steps.

Q: Can we make a better approx?

second-order Taylor expansion

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

↑

has a global minimum

take derivative, set to 0

$f'(a) + (x-a) f''(a) = 0$

$\boxed{x = V - \frac{f'(a)}{f''(a)}}$

Algorithm (Newton-Raphson method in 1D)

Guess a point $x^{(0)}$

For $t = 1, \cdots, T$

  compute second-order approx to f at $x^{(t-1)}$

  $x^{(t)} \leftarrow$ minimizer of

return $x^{(T)}$

Newton Raphson in $\mathbb{R}^d$

- ☐ second derivative?
- ☐ second-order Taylor expansion?
- ☐ minimize

Hessian Matrix

For a function $f: \mathbb{R}^d \to \mathbb{R}$, the Hessian $H(f)$ is a $d \times d$ matrix

where $H(f)_{ij} = \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_j} f(x)$

$d=2$: $\begin{bmatrix} \frac{d^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$

Example: $f(x) = x_1^3 + 5x_1^2 x_2$

$\frac{\partial f}{\partial x_1} = 3x_1^2 + 10x_1x_2$ $\Rightarrow$ $\frac{\partial f}{\partial x_1^2} = 6x_1 + 10x_2$ $\qquad \frac{\partial f}{\partial x_1 \partial x_2} = 10x_1$

$\frac{\partial f}{\partial x_2} = 5x_1^2$ $\qquad \frac{\partial f}{\partial x_2^2} = 0$

$$H(f) = \begin{bmatrix} 6x_1 + 10x_2 & 10x_1 \\ 10x_1 & 0 \end{bmatrix}$$

Second-order Taylor Expansion in $\mathbb{R}^d$

Let $g = \nabla_x f(v)$, $H = H(f)(v)$

$$f(x) \approx \underbrace{f(v) + g^T(x-v)}_{\text{first-order approx}} + \frac{1}{2}(x-v)^T H (x-v)$$

In general, the expansion $u^T A u$ is called a quadratic form.

$u^T A u = \sum_{i=1}^{d} \sum_{j=1}^{d} A_{ij} u_i u_j$ $\leftarrow$ $\nabla_u u^T A u = 2Au$

minimizing the taylor expansion

$$g + H(x-v) = 0$$

$$\boxed{x = v - H^{-1} g}$$

update rule for Newton-Raphson in $\mathbb{R}^d$

Newton-Raphson vs Gradient

Newton-Raphson: $O(d^2)$ memory to store H

Gradient: $O(d)$ memory

Second-order methods are expensive when d is large.

In practice, use a low-rank approximation to H that takes $O(d)$ memory.

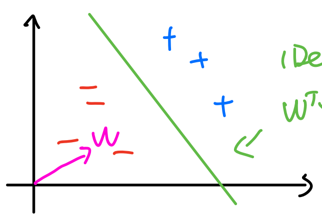
L-BFGS — approximate H, conservative update

L: limited memory

BFGS: name of algorithm

Softmax Regression ("Multinomial Logistic Regression")

multi-class classification



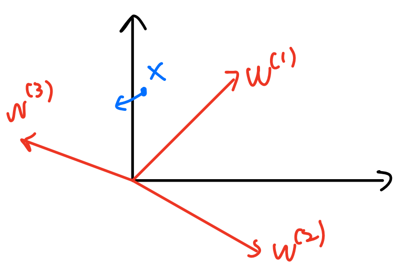
$$\begin{cases} 1 & \text{if } w^Tx+b>0 \\ -1 & \text{if } w^Tx+b\le 0 \end{cases}$$

dot product with w measures how much you look like class 1

softmax regression: parameter vectors

total $c\times d$ parameters $\underbrace{w^{(1)},\dots,w^{(c)}}_{W} \in \mathbb{R}^d$

$w^{(j)T}x$ measures how much x looks like class j



Decision rule: given x, compute $w^{(1)T}x, \dots, w^{(c)T}x$, return j with largest $w^{(j)T}x$

$$P(y=j|x;w) = \frac{\exp(w^{(j)T}x)}{\sum_{k=1}^{c}\exp(w^{(k)T}x)}$$

softmax function

$$w^{(1)T}x = 1 \rightarrow \exp \approx 2.7 \rightarrow P(y=1|x;w) = 0.27$$

$$w^{(2)T}x = -3 \rightarrow \exp \approx 0.1 \rightarrow P(y=2|x;w) = 0.01$$

$$w^{(3)T}x = 2 \rightarrow \exp \approx 7.4 \rightarrow P(y=3|x;w) = 0.72$$

$$\text{sum} \approx 10.2$$

Maximum Likelihood Estimation

NLL – "negative log likelihood" to minimize

$$NLL(w) = -\sum_{i=1}^{n} \log P(y=y^{(i)}|x^{(i)};w)$$

$$= -\sum_{i=1}^{n}\left[w^{y(i)T}x^{(i)} - \log\left(\sum_{k=1}^{C}\exp(w^{(k)T}x^{(i)})\right)\right]$$

↑ loss function

Gradient

$$\nabla_{w^{(j)}} NLL(w) = -\sum_{i=1}^{n} I\{y^{(i)}=j\}\cdot x^{(i)} - \underbrace{\frac{1}{\sum_{k=1}^{C}\exp(w^{(k)T}x^{(i)})}\cdot \exp(w^{(k)T}x^{(i)})}_{\text{scalar} \quad P(y=j|x^{(i)};w)}\cdot x^{(i)}$$

$$= \sum_{i=1}^{n}\left(P(y=j|x^{(i)};w) - I\{y^{(i)}=j\}\right)x^{(i)}$$

If $y^{(i)} \neq j$, then positive $x^{(i)} \Rightarrow$ GD subtracts multiple of $x^{(i)}$

If $y^{(i)} = j$, then negative $x^{(i)} \Rightarrow$ GD adds multiple of $x^{(i)}$