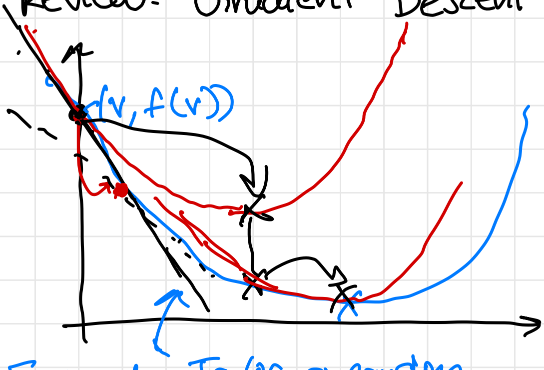
1/24/2023: Second-order optimization, Softmax Regression

Review: Gradient Descent = First-order optimization

only use first derivative



First-order Taylor expansion

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

Q: Why must we take small steps?

A: First-order approx is not that reliable when taking large step

Q: Can we make a better approximation?

Second-order Taylor expansion

constant

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

Advantage: this has a global minimum if $f''(v) > 0$

Algorithm (Newton-Raphson method in 1D)

Guess a point $x^{(0)}$ (e.g. 0)

for $t = 1, \ldots, T$

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

$x^{(t)} \leftarrow$ minimizer of this $\quad x^{(t)} \leftarrow x^{(t-1)} - \frac{f'(x^{(t-1)})}{f''(x^{(t-1)})}$

return $x^{(T)}$

take derivative, set equal to 0

$f'(v) + (x-v) \cdot f''(v) = 0$

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

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

- [x] Second derivative?
- [x] 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 $H(f)_{ij} = \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j} f(x)$

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

Example

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

$$\frac{\partial f}{\partial x_1} = 3x_1^2 + 10x_1x_2$$
$$\frac{\partial f}{\partial x_2} = 5x_1^2$$

$$\frac{\partial^2 f}{\partial x_1^2} = 6x_1 + 10x_2$$
$$\frac{\partial^2 f}{\partial x_2} = 0$$
$$\frac{\partial}{\partial x_1}\frac{\partial}{\partial x_2} = 10x_1$$

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

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

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

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

$1 \times d$ $\quad$ $d \times d$ $\quad$ $d \times 1$ $= \square$

In general, the expression $u^T A u$ is called a quadratic form (general form of a quadratic)

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

$\nabla_u u^T A u = 2Au$

Minimizing the taylor expansion

→ compute gradient, set equal to 0

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

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

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

Newton-Raphson vs. Gradient Descent

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

Gradient Descent ↓ $O(d)$ memory

Second-order methods expensive when d is large

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

$\underbrace{\text{L-BFGS}}$: Approximate H + conservative update

L ↑ Limited Memory

BFGS: Name of Algorithm

Announcements

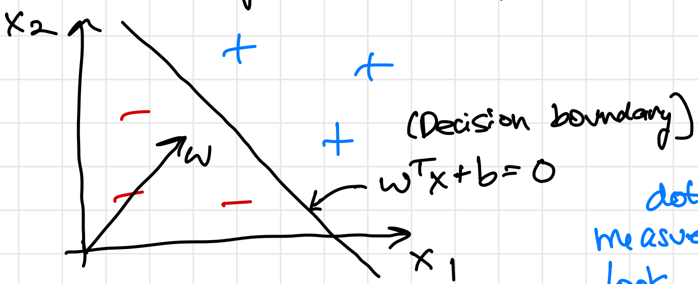
HW0 — Solutions
— Grades by thurs.
— Section Fri

HW1 — out tonight (due Feb 7)

Softmax Regression ("Multinomial Logistic Regression")

<u>Multi-class classification</u>: C different classes

- Handwritten digit $\rightarrow$ $[0, 1, 2, \ldots, 9]$ $(C=10)$
- Image $\rightarrow$ [bird, mammal, reptile, ...]



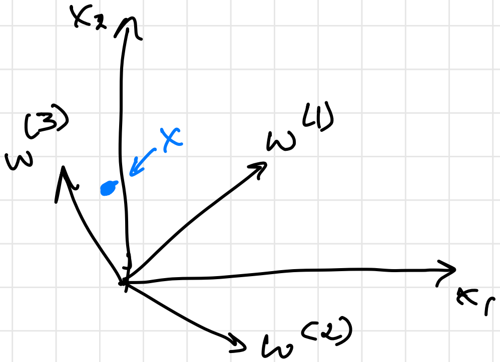
predict 1 if $w^Tx+b>0$
-1 else

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

Softmax Regression: parameter vectors $\underbrace{w^{(1)}, \dots, w^{(C)}}_{w} \in \mathbb{R}^d$

total $C \times d$ parameters

$w^{(j)T}x$ 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 $w^{(j)T}x$

Probabilistic Story:

$$p(y=j \mid x; w) = \frac{\exp(w^{(j)T}x)}{\sum_{k=1}^{C} \exp(w^{(k)T}x)}$$

softmax function

$w^{(1)T}x = 1 \longrightarrow \exp \approx 2.7 \longrightarrow p(y=1 \mid x; w) \approx .27$

$w^{(2)T}x = -3 \longrightarrow \exp \approx 0.1 \longrightarrow p(y=2 \mid x; w) \approx .01$

$\boxed{w^{(3)T}x = 2} \longrightarrow \exp \approx 7.4 \longrightarrow p(y=3 \mid x; w) \approx \boxed{.72}$

sum ≈ 10.2

Maximum Likelihood Estimation

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

"negative log likelihood" want to minimize

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

our loss function!

Gradient time:

$$\nabla_{w^{(j)}} NLL(w) = -\sum_{i=1}^{n} \mathbb{1}[y^{(i)} = j] \cdot x^{(i)} - \underbrace{\frac{1}{\sum_{k=1}^{C} \exp(w^{(k)T} x^{(i)})} \cdot \exp(w^{(j)T} x^{(i)})}_{\text{scalar } p(y=j \mid x^{(i)}; w)} \cdot x^{(i)}$$

(Do this for every j)

$$= \sum_{i=1}^{n} \Big(\underbrace{p(y=j \mid x^{(i)}; w)}_{\text{between } 0 \text{ \& } 1} - \mathbb{1}[y^{(i)}=j] \Big) x^{(i)}$$

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

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

$$w^{(t)} \leftarrow w^{(t-1)} \underbrace{- \eta \cdot \nabla f(w^{(t-1)})}$$