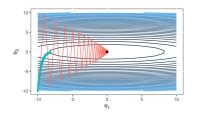
## **Optimization in Machine Learning**

# First order methods Step size and optimality





#### Learning goals

- Impact of step size
- Fixed vs. adaptive step size
- Exact line search
- Armijo rule & Backtracking
- Bracketing & Pinpointing

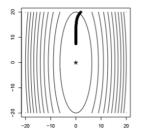
#### CONTROLLING STEP SIZE: FIXED & ADAPTIVE

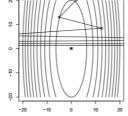
Iteration t: Choose not only descent direction  $\mathbf{d}^{[t]}$ , but also step size  $\alpha^{[t]}$ 

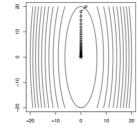
First approach: **Fixed** step size  $\alpha^{[t]} = \alpha > 0$ 

- $\bullet \;\;$  If  $\alpha$  too small, procedure may converge very slowly (left)
- ullet If lpha too large, procedure may not converge ightarrow "jumps" around optimum (middle)

**Adaptive** step size  $\alpha^{[t]}$  can provide better convergence (right)







Steps of line searches for  $f(\mathbf{x}) = 10x_1^2 + x_2^2/2$ 



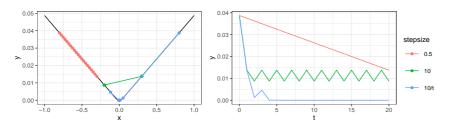
#### STEP SIZE CONTROL: DIMINISHING STEP SIZE

How can we adaptively control step size?

A natural way of selecting  $\alpha^{[t]}$  is to decrease its value over time

Example: GD on

$$f(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \le \delta, \\ \delta \cdot (|x| - 1/2 \cdot \delta) & \text{otherwise.} \end{cases}$$



GD with small constant (**red**), large constant (**green**), and diminishing (**blue**) step size



#### STEP SIZE CONTROL: EXACT LINE SEARCH

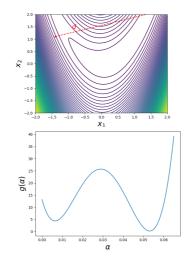
Use optimal step size in each iteration:

$$\alpha^{[t]} = \mathop{\arg\min}_{\alpha \in \mathbb{R}_{\geq 0}} g(\alpha) = \mathop{\arg\min}_{\alpha \in \mathbb{R}_{\geq 0}} f(\mathbf{x}^{[t]} + \alpha \mathbf{d}^{[t]})$$

Need to solve a **univariate** optimization problem in each iteration ⇒ univariate optimization methods

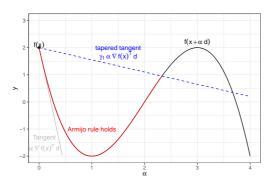
**Problem:** Expensive, prone to poorly conditioned problems

**But:** No need for *optimal* step size. Only need a step size that is "good enough". **Reason:** Effort may not pay off, but in some cases slows down performance.





#### **ARMIJO RULE**

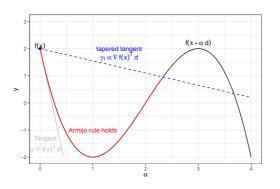




**Inexact line search:** Minimize objective "sufficiently" without computing optimal step size exactly

Common condition to guarantee "sufficient" decrease: Armijo rule

#### **ARMIJO RULE**



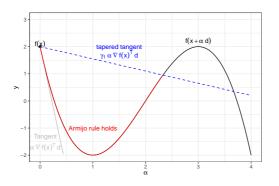


Fix  $\gamma_1 \in (0,1)$ .  $\alpha$  satisfies **Armijo rule** in **x** for descent direction **d** if

$$f(\mathbf{x} + \alpha \mathbf{d}) \leq f(\mathbf{x}) + \gamma_1 \alpha \nabla f(\mathbf{x})^{\top} \mathbf{d}.$$

Note:  $\nabla f(\mathbf{x})^{\top} \mathbf{d} < 0$  (**d** descent dir.)  $\implies f(\mathbf{x} + \alpha \mathbf{d}) < f(\mathbf{x})$ .

#### **ARMIJO RULE**





**Feasibility:** For descent direction **d** and  $\gamma_1 \in (0,1)$ , there exists  $\alpha > 0$  fulfilling Armijo rule. In many cases, Armijo rule guarantees local convergence of GD and is therefore frequently used.

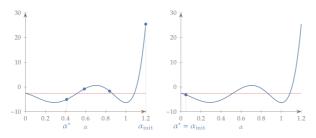
#### **BACKTRACKING LINE SEARCH**

Procedure to meet the Armijo rule: Backtracking line search

**Idea:** Decrease  $\alpha$  until Armijo rule is met

#### Algorithm Backtracking line search

- 1: Choose initial step size  $\alpha=\alpha_{\rm init},\, {\rm 0}<\gamma_{\rm 1}<{\rm 1}$  and  ${\rm 0}<\tau<{\rm 1}$
- 2: while  $f(\mathbf{x} + \alpha \mathbf{d}) > f(\mathbf{x}) + \gamma_1 \alpha \nabla f(\mathbf{x})^{\top} \mathbf{d}$  do
- 3: Decrease  $\alpha$ :  $\alpha \leftarrow \tau \cdot \alpha$
- 4: end while



(Source: Martins and Ning. Engineering Design Optimization, 2021.)



### **BACKTRACKING LINE SEARCH / 2**



#### **WOLFE CONDITIONS**

Backtracking is simple and shows good performance in practice

But: Two undesirable scenarios

- Initial step size  $\alpha_{\text{init}}$  is too large  $\Rightarrow$  need multiple evaluations of f
- Step size is too small with highly negative slopes



- Fix  $\gamma_2$  with  $0 < \gamma_1 < \gamma_2 < 1$ .
- ullet  $\alpha$  satisfies sufficient curvature condition in x for d if

$$|\nabla f(\mathbf{x} + \alpha \mathbf{d})^{\top} \mathbf{d}| \leq \gamma_2 |\nabla f(\mathbf{x})^{\top} \mathbf{d}|.$$

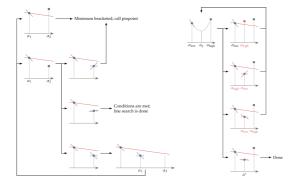
Armijo rule + sufficient curvature condition = **Wolfe conditions** 



#### **WOLFE CONDITIONS / 2**

**Algorithm** for finding a Wolfe point (point satisfying Wolfe conditions):

- Bracketing: Find interval containing Wolfe point
- Pinpointing: Find Wolfe point in interval from bracketing



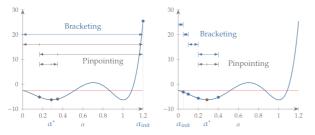
**Left:** Bracketing. **Right:** Pinpointing. (Source: Martins and Ning. *EDO*, 2021.)



#### **BRACKETING & PINPOINTING**

#### Example:

- Large initial step size results in quick bracketing but multiple pinpointing steps (left).
- Small initial step size results in multiple bracketing steps but quick pinpointing (right).



Source: Martins and Ning. EDO, 2021.

