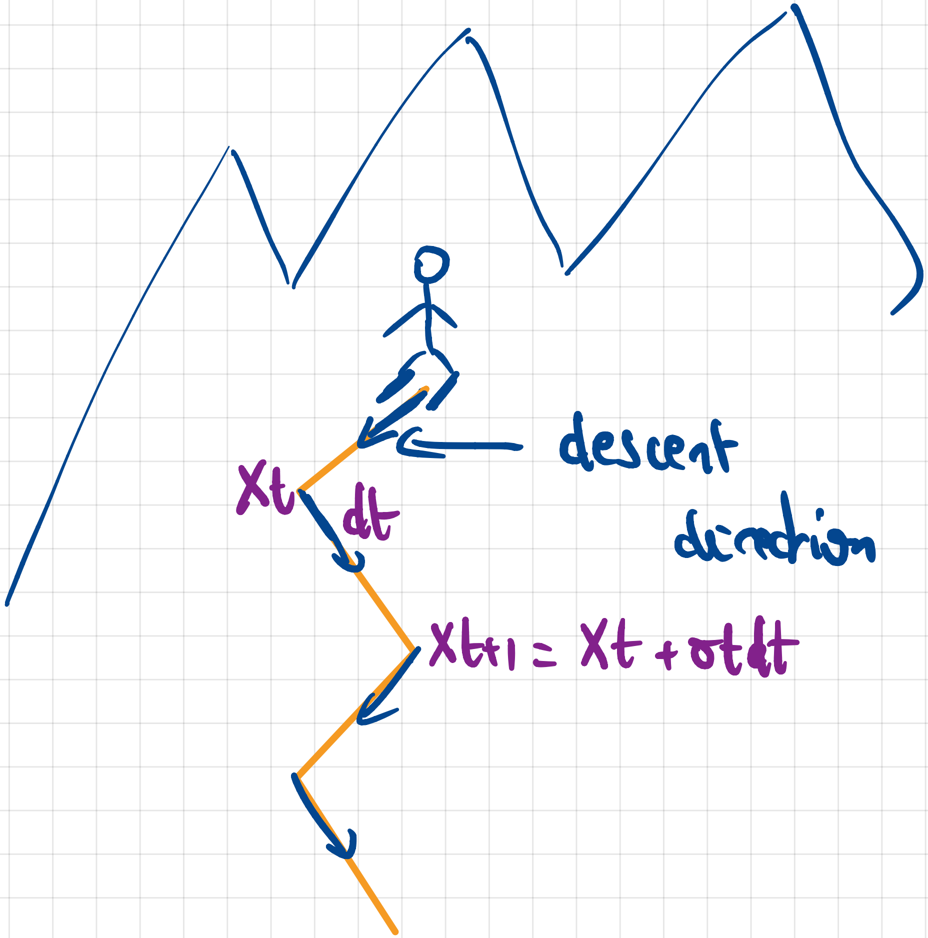
OPTIMIZATION FOR MACHINE LEARNING 2022 CLASS 4

DESCENT METHODS



obj: minimize $f: \mathbb{R}^n \to \mathbb{R}$

General principle

1/ Choose an initial point $x_0 \in \mathbb{R}^n$
$t = 0$

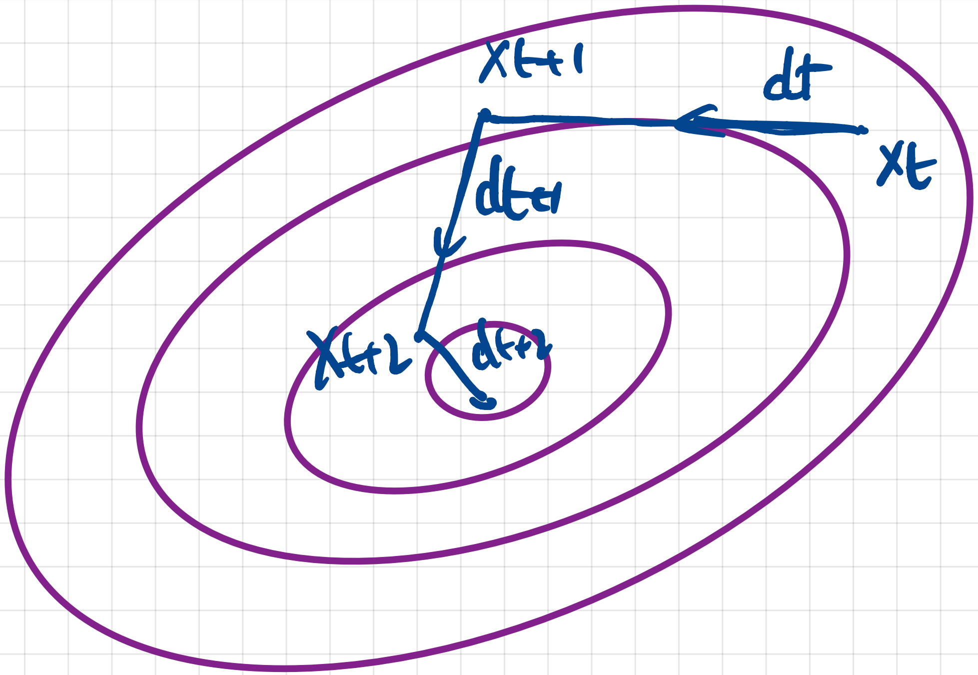
While not happy (while f not minimize enough)

- choose a descent direction $d_t \neq 0$ $d_t \in \mathbb{R}^n$
- line search
  - $\to$ choose step-size $\alpha_t > 0$
  - $\to$ set $x_{t+1} = x_t + \alpha_t d_t$

set $t = t+1$

Remains to decide : - what is d_t ?

- how to choose α_t.



How to choose a descent direction ?

Proposition

$$d_t = -\nabla f(x_t)$$

For a small enough step-size σ, if I follow $-\nabla f(x_t)$

then $f(x_t + \sigma(-\nabla f(x_t))) < f(x_t)$

If σ is small enough

$$f(x_t - \sigma \nabla f(x_t)) < f(x_t) \text{ and } \nabla f(x_t) \neq 0$$

Taylor Formula

f. Differentiable

$$f(x+h) = f(x) + h^T \nabla f(x) + O(\|h\|^2)$$

(Also true : $f(x+h) = f(x) + h^T \nabla f(x) + o(\|h\|)$)

$$\frac{O(\|h\|^2)}{\|h\|^2} < C \quad \text{(Bounded for } h \text{ small)} \qquad \frac{o(\|h\|)}{\|h\|} \xrightarrow[\|h\| \to 0]{} 0$$

Apply Taylor Formula to $f(x_t - \sigma Df(x_t))$

$$f(x_t \underbrace{- \sigma Df(x_t)}_{h}) = f(x_t) + \left(-\sigma Df(x_t)\right)^T \nabla f(x_t) + o(\|\sigma Df(x_t)\|^2)$$

$$= f(x_t) - \sigma \|Df(x_t)\|^2 + \sigma^2 o(\|Df(x_t)\|^2)$$

For σ small enough

$$f(x_t - \sigma Df(x_t)) \approx f(x_t) \underbrace{- \sigma \|Df(x_t)\|^2}_{<0}$$

$$< f(x_t)$$

↳ $Df(x_t)$ is a descent direction.

$$O(\sigma^2 \|\nabla f(x)\|^2) = \underline{\sigma^2 \, O(\|\nabla f(x)\|^2)}$$

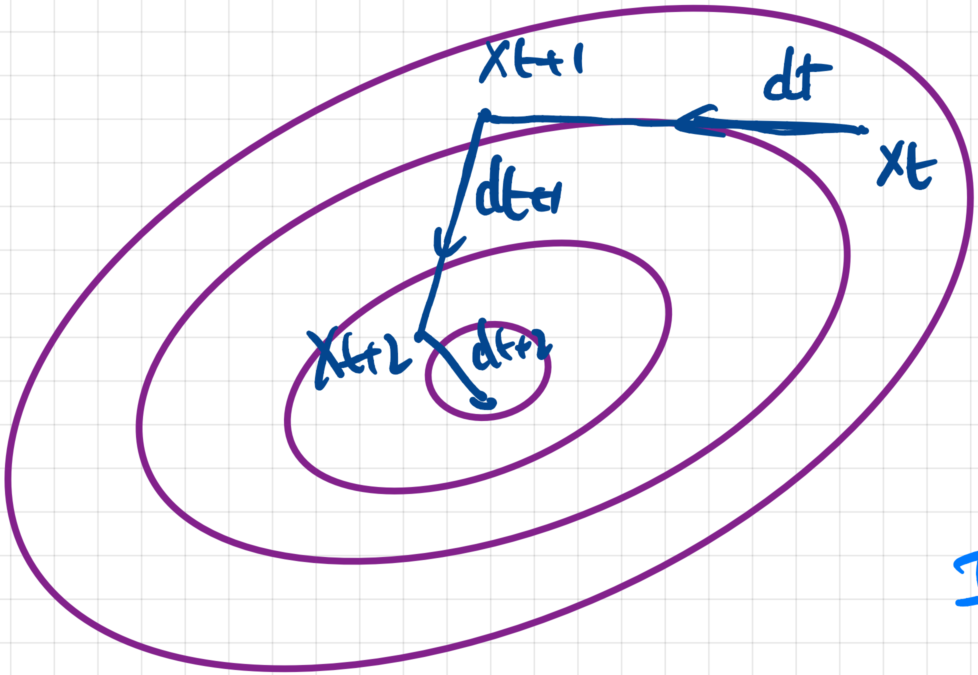
$$O(\sigma \|h\|) = \sigma \, O(\|h\|)$$

CHOICE OF THE STEP-SIZE

$$f(x_t - \sigma_t\, x_t)$$

$$\sigma_t = \frac{\sigma}{\operatorname{cond}(\nabla^2 f(x_t))}$$

→ constant ?
↳ small enough.



What is the optimal step-size?

$$f(x_t - \sigma \nabla f(x_t))$$

$$\underset{\mathbb{R}}{\sigma} \xmapsto{g} f(x_t - \sigma \nabla f(x_t))$$

I can minimize

$$\underset{\mathbb{R}}{\sigma} \mapsto f(x_t - \sigma \nabla f(x_t))$$

1D optimization

minimize f along the gradient direction starting from x_t.

<u>optimal step size</u>

$$\delta_t = \arg\min_{\delta \geq 0} f(x_t - \delta \nabla f(x_t))$$

Typically too expensive to do those 1D optimization perfectly

There exists different rules to approximate optimal step size. One widely used is called <u>Armijo rule</u>.

Stopping criterion:

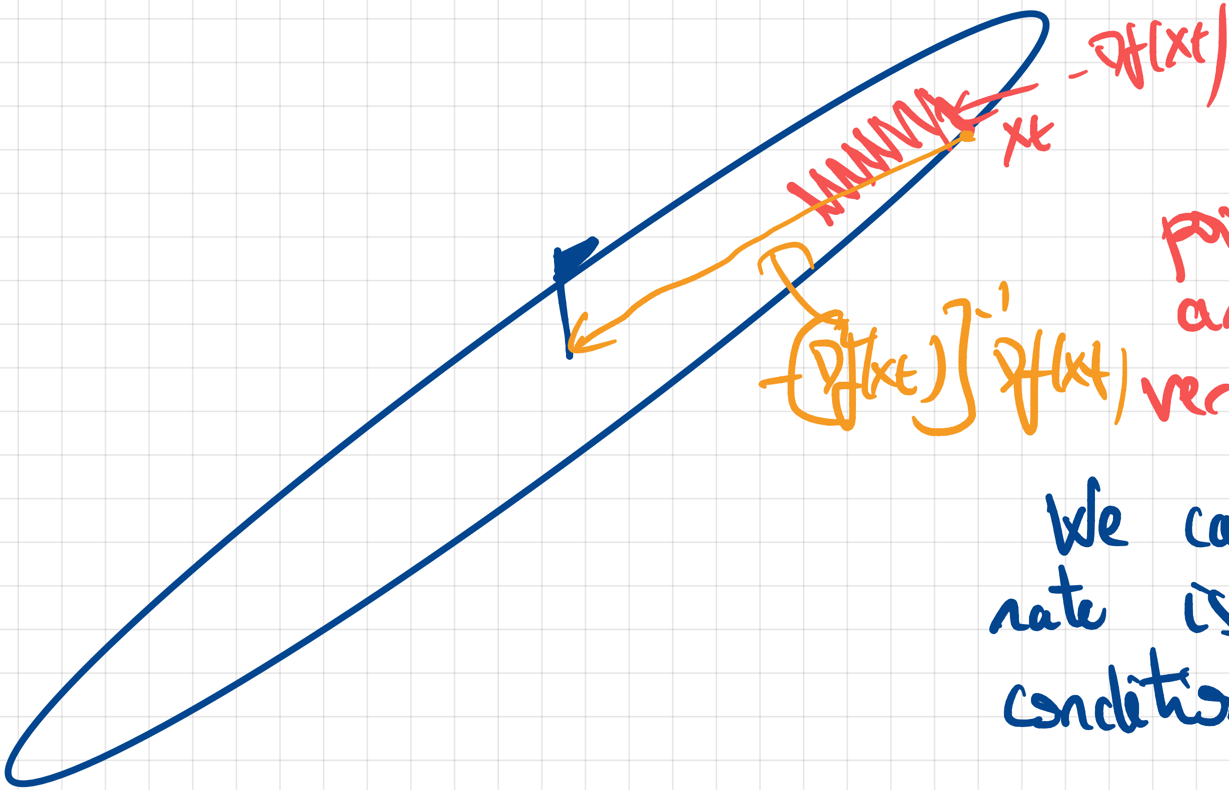
- If $|f(x_{t+1}) - f(x_t)| < \varepsilon$
- Stop after a certain number of iterations
  ↳ no guarantee on whether the algorithm has converged.
- Stop if $\|Df(x_t)\|$ small

Remark: If you need to maximize f and you only have a minimizer implemented, then just minimize $-f$.

If I implement gradient descent but need to maximize, I can implement gradient ascent.

$$x_{t+1} = x_t + \alpha \nabla f(x_t)$$

Gradient descent is slow on ill-conditionned problems



On a ill-conditionned function $-\nabla f$ typically points in the wron direction and the convergence will be very slow.

We can prove that the convergence rate is slower the larger the condition number.

Instead of $-Df(x_t)$, we could follow the Newton direction $-[D^2 f(x_t)]^{-1} Df(x_t)$

The Newton direction minimizes the locally quadratic approximation of f.

$$f(x+\Delta x) = \underbrace{f(x) + Df(x)^T \Delta x + \frac{1}{2} (\Delta x)^T D^2 f(x) \Delta x}_{\text{quadratic approximation of } f}$$

Newton direction minimizes this.

If we can obtain in a "cheap" way the Newton direction, we should use it.

But often too expensive to obtain $D^2 f(x_t)$ and to invert it.

In the convex-quadratic case, the function equals its second order approximation and the Newton direction is perfect as it points towards the optimum.

For non convex-quadratic case, the Newton is typically good to follow but not point directly towards the optimum.

QUASI-NEWTON METHOD: BFGS (70's)

L-BFGS

Broyden Fletcher Goldfarb Shannon

$$x_{t+1} = x_t - \eta_t H_t \nabla f(x_t)$$

H_t: approximation of $[\nabla^2 f(x_t)]^{-1}$

H_t is updated iteratively using $Df(x_t)$ (without computing the Hessian matrix) and it approximates $[D^2 f(x_t)]^{-1}$

cf wikepedia page for update.

large scale version of BFGS : L-BFGS

Limited-memory BFGS

STOCHASTIC GRADIENT DESCENT

Minimize loss function of the following form:

$$Q(w) = \frac{1}{N} \sum_{i=1}^{N} Q_i(w)$$

N : # Data
Examples

w can be weights of Neural Network.

How do we minimize Q?

Gradient descent:

$$\nabla Q(w) = \frac{1}{N} \sum_{i=1}^{N} \nabla Q_i(w)$$

$$w_{t+1} = w_t - \alpha_t \nabla Q(w_t) \quad \text{[update of weights]}$$

BACKPROPAGATION algorithm to compute $\nabla Q_i(w)$

Typically N is very large, computation of all $\nabla Q_i(w)$ $i = 1, \dots, N$ too expensive.

Instead we approximate $\nabla Q(w)$

$$\nabla Q(w) \approx \nabla Q_i(w) \quad (\text{Gradient of single example})$$

Also do mini batches:

$$\nabla Q(w) \approx \frac{1}{n_{batches}} \sum_{i=1}^{n_{batches}} \nabla Q_i(w)$$

$n_{batches} << N$

STOCHASTIC Gradient descent :

Choose an initial vectors of parameters and a step-size η

While not happy

- Randomly shuffle examples in training set
- For $i = 1, \dots, N$

$$w \leftarrow w - \eta \, \nabla Q_i(w)$$

(possibly mini-batches)

Not covered: - choice of step-size (step-size adapted using "momentum techniques" ADAM)

- increase # elements of mini-batches