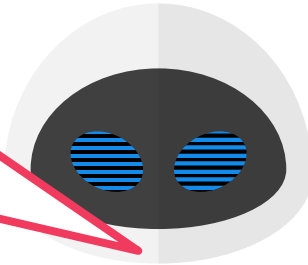
Optimization Refresher

CS771: Introduction to Machine Learning

Purushottam Kar

Topics to b

We are using $\mathbf{w}$ as the variable of optimization instead of the usual $\mathbf{x}$ since we will often use optimization in ML algos to find the "best" model and $\mathbf{w}$ is often used to denote the model



- Gradient Descent and its variants
- Practical issues with GD variants
- We will consider the following generic optimization problem

$$\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}) \quad \text{s.t. } \mathbf{w} \in \mathcal{C}$$

- Recall that $f: \mathbb{R}^d \rightarrow \mathbb{R}$ is the *objective function* and $\mathcal{C} \subseteq \mathbb{R}^d$ is the *constraint set* (for *unconstrained* problems, we have $\mathcal{C} = \mathbb{R}^d$)

From Calculus to Optimization

Method 1: First order optimality Condition

Exploits the fact that gradient must vanish at a local optimum

Also exploits the fact that for convex functions, local minima are global

***Warning**: works only for simple convex functions when there are no constraints*

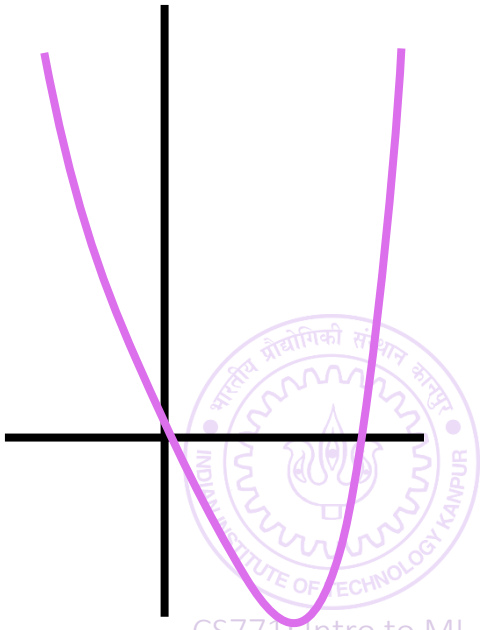
To Do: given a convex function that we wish to minimize, try finding all the stationary points of the function (set gradient to zero)

If you find only one, that has to be the global minimum ☺

Example: $f(w) = w^4 - 2w$

$f'(w) = 4w^3 - 2 = 0$ only at $w^* = \sqrt[3]{0.5}$

$f''(w) = 12x^2 \geq 0$ i.e. $f(w)$ is cvx i.e. w^* is global min



From Calculus to Optimization

Method 2: Perform (sub)gradient descent

Recall that direction opposite to gradient offers *steepest descent*

(SUB)GRADIENT DESCENT

1. **Given**: obj. func. $f: \mathbb{R}^d \rightarrow \mathbb{R}$ to minimize
2. Initialize $\mathbf{w}^0 \in \mathbb{R}^d$
3. For $t = 0, 1, \ldots$
   1. Obtain a (sub)gradient $\mathbf{g}^t \in \partial f(\mathbf{w}^t)$
   2. Choose a step length η_t
   3. Update $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta_t \cdot \mathbf{g}^t$
   4. Repeat until convergence

How to initialize $\mathbf{w}^0$?

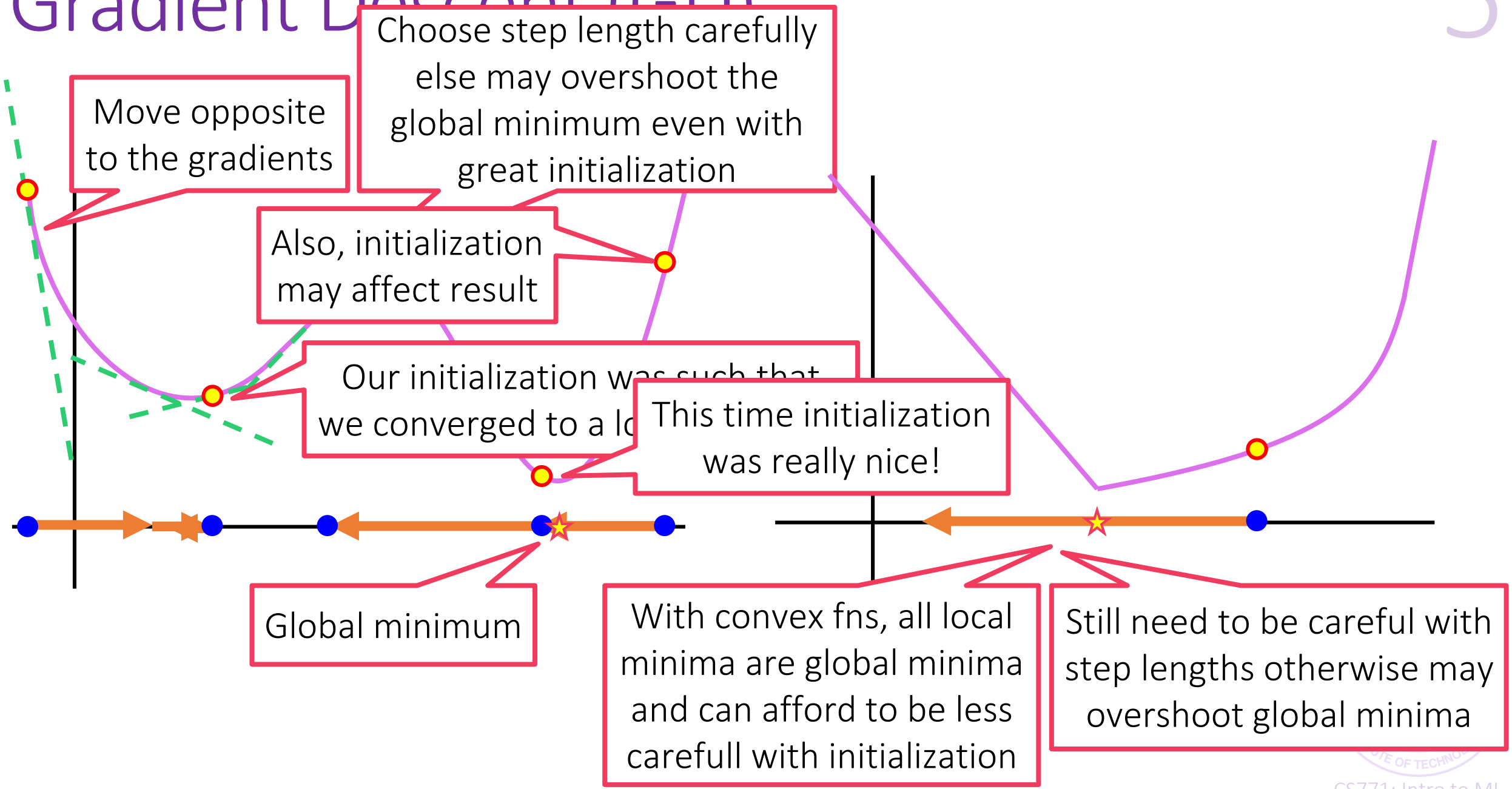
How to choose η_t

Often called "step length" or "learning rate"

What is convergence?

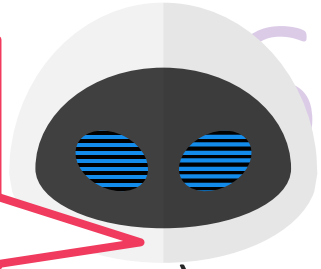
How to decide if we have converged?

Gradient Descent (GD)



Behind the sce

So gradient descent, although a mathematical tool from calculus, actually tries very actively to make the model perform better on all data points



$f(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}[1 - y^i \cdot \mathbf{w}^\top \mathbf{x}^i]_+$ (ignore bias b for now)

$\nabla f(\mathbf{w}) = \mathbf{w} + C \cdot \sum_{i=1}^{n} g^i y^i \cdot \mathbf{x}^i$, where $g^i \in \nabla \ell_{\text{hinge}}\left(y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right)$

$\mathbf{w}^{\text{new}} = \mathbf{w} - \eta \cdot \nabla f(\mathbf{w}) = (1-\eta) \cdot \mathbf{w} - \eta C \cdot \sum_{i=1}^{n} g^i y^i \cdot \mathbf{x}^i$

Assume $n = 1$ for a moment for sake of understanding

$\mathbf{w}^{\text{new}} = (1-\eta) \cdot \mathbf{w} - \eta C \cdot g^1 y^1 \cdot \mathbf{x}^1$

Small η: $(1-\eta)$ is large $\Rightarrow$ do not change $\mathbf{w}$ too much

Large η: Feel free to change $\mathbf{w}$ as much as the gradient di[illegible]ates

If $\boldsymbol{w}$ does well on $(\boldsymbol{x}^1, y^1)$, say $y^1 \cdot \boldsymbol{w}^\top \boldsymbol{x}^1 > 1$, then g^1 [illegible]

If $\boldsymbol{w}$ does badly on $(\boldsymbol{x}^1, y^1)$, say $y^1 \cdot \boldsymbol{w}^\top \boldsymbol{x}^1 < 0$, then [illegible]

$\mathbf{w}^{\text{new}} = (1-\eta) \cdot \mathbf{w} + \eta C \cdot y^1 \cdot \mathbf{x}^1$

$y^1 \cdot (\mathbf{w}^{\text{new}})^\top \cdot \mathbf{x}^1 = (1-\eta) y^1 \cdot \boldsymbol{w}^\top \boldsymbol{x}^1 + \eta C \cdot \|\mathbf{x}^1\|_2^2$

No change to $\mathbf{w}$ due to the data point $(\boldsymbol{x}^1, y^1)$

$\mathbf{w}^{\text{new}}$ may get much better margin on $(\boldsymbol{x}^1, y^1)$ than $\mathbf{w}$

Stochastic Gradient Method

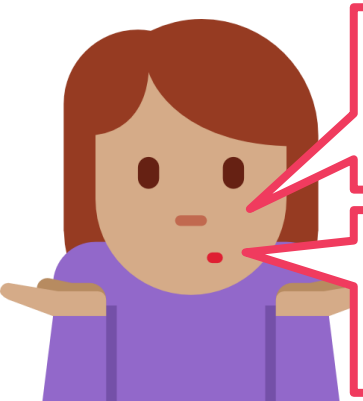
$\nabla f(\mathbf{w}) = \mathbf{w} + C \cdot \sum_{i=1}^{n} g^i y^i \cdot \mathbf{x}^i$, where $g^i \in \nabla \ell_{\text{hinge}}\left(y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right)$

Calculating each g^i takes $\mathcal{O}(d)$ time since $\mathbf{w}, \mathbf{x}^i \in \mathbb{R}^d$ - total $\mathcal{O}(nd)$

At each time, choose a random data point $\left(\mathbf{x}^{i_t}, y^{i_t}\right)$
$\nabla f(\mathbf{w}) \approx \mathbf{w} + C \cdot g^{i_t} y^{i_t} \cdot \mathbf{x}^{i_t}$ - only $\mathcal{O}(d)$ time!!

Warning: may have to perform several SGD steps than we had to do with GD but each SGD step is much cheaper than a GD step

We take a random data point to avoid being unlucky (also it is cheap)

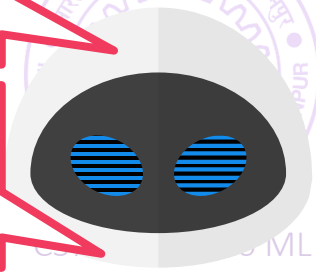


Do we really need to spend so much time on just one update?

Especially in the beginning, when we are far away from the optimum!

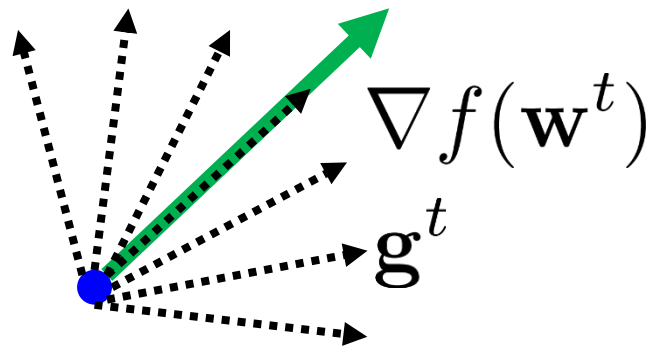
Initially, all we need is a general direction in which to move

No, SGD gives a cheaper way to perform gradient descent



Mini-batch SGD

If data is very diverse, the "stochastic" gradient may vary quite a lot depending on which random data point is chosen



This is called *variance* (more on this later) but this can slow down the SGD process – make it jittery

One solution, choose more than one random point

At each step, choose B random data points (B = *mini batch size)* without replacement, say $\left(x^{i_t^1}, y^{i_t^1}\right), \ldots, \left(x^{i_t^B}, y^{i_t^B}\right)$ and use

$$\nabla f(\mathbf{w}) \approx \mathbf{w} + C \cdot \sum_{b=1}^{B} g^{i_t^b} y^{i_t^b} \cdot \mathbf{x}^{i_t^b}$$

Takes $\mathcal{O}(Bd)$ time to execute MBSGD – more expensive than SGD

Notice that if $B = n$ then MBSGD becomes plain GD

Constrained Optimization

Recall that an optimization problem has an objective and constraints

$$\min_x f(x)$$

such that $p(x) < 0$ and $q(x) > 0$ etc.

Objective

Constraints

The set of points that satisfy *all* the constraints is called the *feasible set* $\mathcal{C}$

$$\mathcal{C} \triangleq \{x : p(x) < 0 \text{ and } q(x) > 0 \text{ and } \ldots.\}$$

Problems with constraints more challenging

Method 1: Interior Point Methods

Find a way to initialize within $\mathcal{C}$ and then take steps that never go out

A very powerful family of methods – also very involved

Not very popular in machine learning as they can be expensive

Beyond the scope of CS771

Constrained Optimization

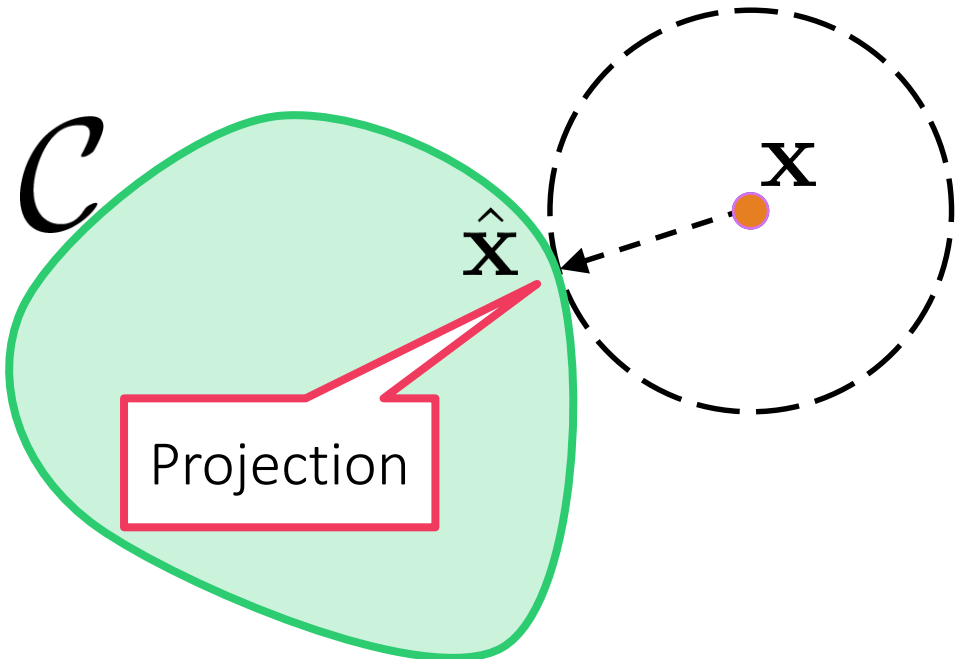
Method 2: Projected Gradient Descent

Perform (stochastic) gradient descent as usual. However, if this causes us to step outside the feasible set $\mathcal{C}$, go back to the feasible set

Process of "going back" into $\mathcal{C}$: projection step

$$\hat{\mathbf{x}} = \Pi_{\mathcal{C}}(\mathbf{x}) = \arg\min_{\mathbf{z}\in\mathcal{C}} \|\mathbf{x} - \mathbf{z}\|_2^2$$

***Warning**: works only if $\mathcal{C}$ is such that projection step is easy*



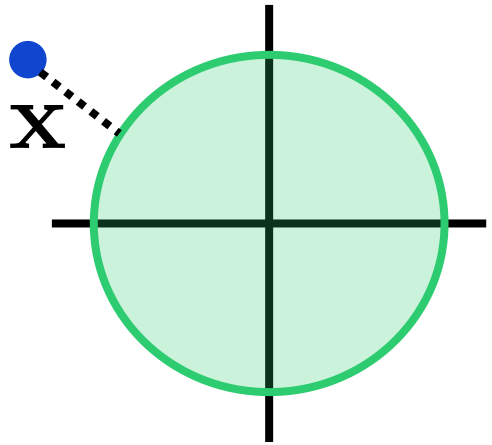
PROJECTED (SUB)GRADIENT DESCENT

1. Choose $\mathbf{g}^t \in \partial f(\mathbf{w}^t)$, step length η_t
2. Update $\mathbf{u}^{t+1} \leftarrow \mathbf{w}^t - \eta_t \cdot \mathbf{g}^t$
3. Project $\mathbf{w}^{t+1} \leftarrow \Pi_{\mathcal{C}}(\mathbf{u}^{t+1})$
4. Repeat until convergence

A few useful Projections

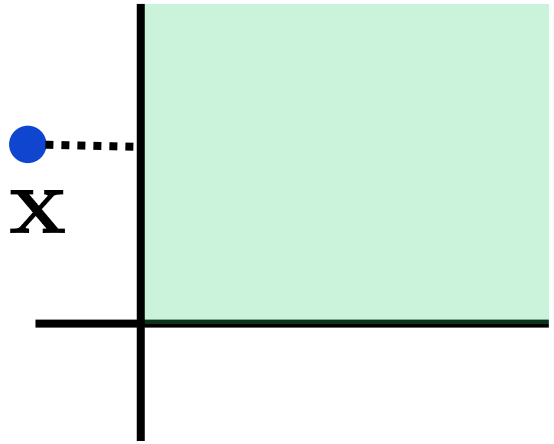
$$\hat{\mathbf{x}} = \Pi_{\mathcal{C}}(\mathbf{x})$$

$$\mathcal{C} = \{\mathbf{x} : \|\mathbf{x}\|_2 \leq 1\}$$



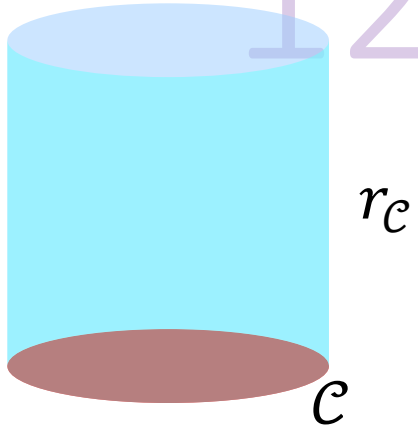
$$\hat{\mathbf{x}} = \begin{cases} \mathbf{x} & \text{if } \|\mathbf{x}\|_2 \leq 1 \\ \frac{\mathbf{x}}{\|\mathbf{x}\|_2} & \text{if } \|\mathbf{x}\|_2 > 1 \end{cases}$$

$$\mathcal{C} = \{\mathbf{x} : \mathbf{x}_i \geq 0\}$$



$$\hat{\mathbf{x}}_i = \begin{cases} \mathbf{x}_i & \text{if } \mathbf{x}_i \geq 0 \\ 0 & \text{if } \mathbf{x}_i < 0 \end{cases}$$

Proximal Gradient Descent



Can think of it as a more powerful cousin of projected GD

Many opt problems that come up in ML look like

$$\min_{\mathbf{w}\in\mathbb{R}^d} r(\mathbf{w}) + \ell(\mathbf{w})$$

In SVM problem, $r(\mathbf{w}) = 0.5 \cdot \|\mathbf{w}\|_2^2$ *and* $\ell(\mathbf{w}) = C \cdot \sum_{i=1}^{n} \left[1 - y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right]_+$

$\ell(\cdot)$ is ofte $(\cdot)$ *the "regularizer (more later)*

Note that l *t regularizer only on model*

Such a function is often called an "indicator function" since it indicates if a point is inside or outside $\mathcal{C}$

The regularizer can even encode constraints e.g. $r_{\mathcal{C}}(\mathbf{w}) = \begin{cases} 0, \boldsymbol{w} \in \mathcal{C} \\ \infty, \boldsymbol{w} \notin \mathcal{C} \end{cases}$

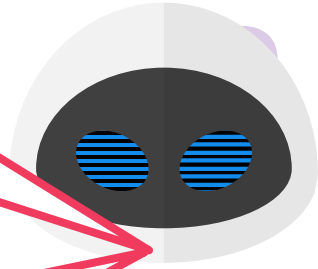
Solving $\min_{\mathbf{w}\in\mathbb{R}^d} r_{\mathcal{C}}(\mathbf{w}) + \ell(\mathbf{w})$ *is the same as solving* $\min_{\mathbf{w}\in\mathbb{R}^d} \ell(\mathbf{w})$ *s.t.* $\mathbf{w} \in \mathcal{C}$

Prox-GD offers a very attractive technique when the regularizer is not a differentiable function (will see examples soon)

Proximal (

Nice! So the prox step is trying to lower $r(\cdot)$ value without undoing progress made by gradient step

The intuition behind this definition is that we wish to stay in the **proximity** of the input $\mathbf{u}$, but also lower $r(\cdot)$ value



Indeed – prox GD is very useful for problems where regularizer is non-differentiable

PROXIMAL GRADIENT DESCENT

1. **Given**: loss fn $\ell(\cdot)$ regularizer $r(\cdot)$
2. Initialize $\mathbf{w}^0 \in \mathbb{R}^d$
3. For $t = 0, 1, \ldots$
   1. Let $\mathbf{g}^t \in \partial\ell(\mathbf{w}^t)$ and choose η_t
   2. Let $\mathbf{u}^{t+1} \leftarrow \mathbf{w}^t - \eta_t \cdot \mathbf{g}^t$
   3. Let $\mathbf{w}^{t+1} \leftarrow \mathbf{prox}_r(\mathbf{u}^{t+1})$
   4. Repeat until convergence

If $r(\cdot)$ is indicator fn of a set $\mathcal{C}$, proxGD becomes projected GD

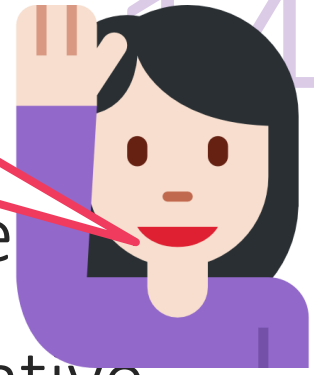
$$\mathbf{prox}_r(\mathbf{u}) = \arg\min_{\mathbf{z}\in\mathcal{C}} \|\mathbf{u} - \mathbf{z}\|_2$$

If $r(\mathbf{w}) = 0.5 \cdot \|\mathbf{w}\|_2^2$ then $\mathbf{prox}_r(\mathbf{u}) = \mathbf{u}/2$ i.e. scaling

ProxGD very useful in learning *sparse models* which are offer smaller model size, test times

Coordinate Descent

Sometimes we are able to optimize completely along a given variable (even if constraints are there) – called coordinate minimization (CM)



Similar to GD except only one coordinate is changed in a single

E.g. $\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$ s.t. $\mathbf{x} \in \mathcal{C}$ with $\nabla_j f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}_j}$ as jth partial derivative

CCD: choose coordinate cyclically i.e. $j_t = 1,2,\dots,d,1,2,\dots,d,\dots$

SCD: choose j_t randomly

Block CD: choose a small set of coordinates at each t to update

Randperm: permute coordinates randomly and choose them in that order. Once the list is over, choose a new random permutation and start over (very effective)

(PROJECTED) COORDINATE DESCENT

1. For $t = 0,1,\dots$
   1. Select a coordinate $j_t \in [d]$
   2. Let $\mathbf{u}_{j_t}^{t+1} \leftarrow \mathbf{x}_{j_t}^{t} - \eta_t \cdot \nabla_{j_t} f(\mathbf{x}^t)$
   3. Let $\mathbf{u}_{j}^{t+1} \leftarrow \mathbf{x}_{j}^{t}$ for $j \neq j_t$
   4. Project $\mathbf{x}^{t+1} \leftarrow \Pi_{\mathcal{C}}(\mathbf{u}^{t+1})$
   5. Repeat until convergence

Practical Issues with GD Variants

- How to initialize?
- How to decide convergence?
- How to decide step lengths?

How to Initialize?

Initializing close to the global optimum is obviously preferable ☺

Easier said than done. In some applications however, we may have such initialization e.g. someone may have a model they trained on different data

For convex functions, bad initialization may mean slow convergence, but if step lengths are nice then GD should converge eventually

For non-convex functions (e.g. while training deepnets), bad initialization may mean getting stuck at a very bad saddle point

Random restarts most common solution to overcome this problem

For some nice non-convex problems, we do know very good ways to provably initialize close to the global optimum (e.g. collaborative filtering in recommendation systems) – details beyond scope of CS771

How to decide Convergence?

In optimization, convergence can refer to a couple of things

The algorithm has gotten within a "small" distance of a global/local optima

The algorithm is not making "much" progress e.g. $\|\mathbf{w}^{t+1} - \mathbf{w}^{t}\| \rightarrow \mathbf{0}$

GD stops making progress when it reaches a stationary point i.e. can stop making progress even without having reached a global optimum (e.g. if it has reached a saddle point)

Usually a few heuristics used to decide when to stop executing GD

If gradient vectors have become too "small", or "not much" progress is being made of if objective function value is already acceptably "small" or if assignment submission deadline is 5 minutes away

Acceptable levels e.g. "small", "not much" usually decided either by consulting domain experts or else by using performance on validation sets

How to detect convergence

Method 1: Tolerance technique

For a pre-decided tolerance value ϵ, if $f(\mathbf{w}^t) < \epsilon$, stop

Method 2: Zero-th order technique

If fn value has not changed much, stop (or else tune learning rate)!

$$|f(\mathbf{w}^{t+1}) - f(\mathbf{w}^t)| < \tau \text{ or } \|\mathbf{w}^{t+1} - \mathbf{w}^t\| < \zeta$$

Method 3: First order technique

If gradient has become too small i.e. $\|\nabla f(\mathbf{w}^t)\|_2 < \delta$, stop!

Method 4: Cross validation technique

Test the current model on validation data – if performance acceptable, stop!

Other techniques e.g. primal-dual techniques are usually infeasible for large-scale ML problems and hence not used to decide convergence