Optimization for ML (3)

CS771: Introduction to Machine Learning

Piyush Rai

Stochastic Gradient Descent (SGD)

Writing as an average instead of sum. Won't affect minimization of $L(\boldsymbol{w})$

- Consider a loss function of the form $L(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} \ell_n(\boldsymbol{w})$

Expensive to compute – requires doing it for all the training examples in each iteration ☹

- The (sub)gradient in this case can be written as

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}} L(w) = \nabla_{\boldsymbol{w}}[\frac{1}{N}\sum_{n=1}^{N} \ell_n(\boldsymbol{w})] = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{g}_n$$

(Sub)gradient of the loss on $\boldsymbol{n}^{th}$ training example

- Stochastic Gradient Descent (SGD) approximates $\boldsymbol{g}$ using a <u>single</u> training example
- At iter. $\boldsymbol{t}$, pick an index $\boldsymbol{i} \in \{1,2,\ldots,N\}$ uniformly randomly and approximate $\boldsymbol{g}$ as

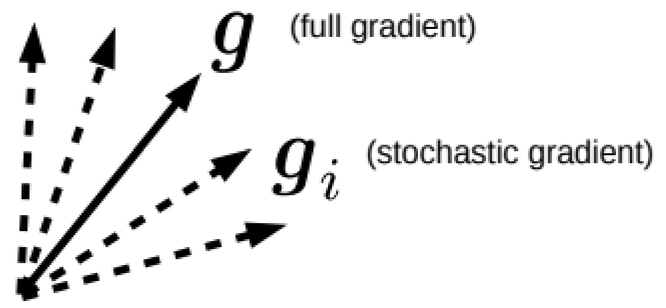
$$\boldsymbol{g} \approx \boldsymbol{g}_i = \nabla_{\boldsymbol{w}} \ell_i(\boldsymbol{w})$$

Can show that $\boldsymbol{g}_i$ is an unbiased estimate of $\boldsymbol{g}$, i.e.,$\mathbb{E}[\boldsymbol{g}_i] = \boldsymbol{g}$

- May take more iterations than GD to converge but each iteration is much faster ☺
  - SGD per iter cost is $O(D)$ whereas GD per iter cost is $O(ND)$

Minibatch SGD

- Gradient approximation using a single training example may be noisy



The approximation may have a high variance – may slow down convergence, updates may be unstable, and may even give sub-optimal solutions (e.g., local minima where GD might have given global minima)

- We can use $B > 1$ unif. rand. chosen train. ex. with indices $\{i_1, i_2, \dots, i_B\} \in \{1,2,\dots,N\}$
- Using this "minibatch" of examples, we can compute a minibatch gradient

$$\boldsymbol{g} \approx \frac{1}{B}\sum_{b=1}^{B} \boldsymbol{g}_{i_b}$$

- Averaging helps in reducing the variance in the stochastic gradient
- Time complexity is $O(BD)$ per iteration in this case

Constrained Optimization

Projected Gradient Descent

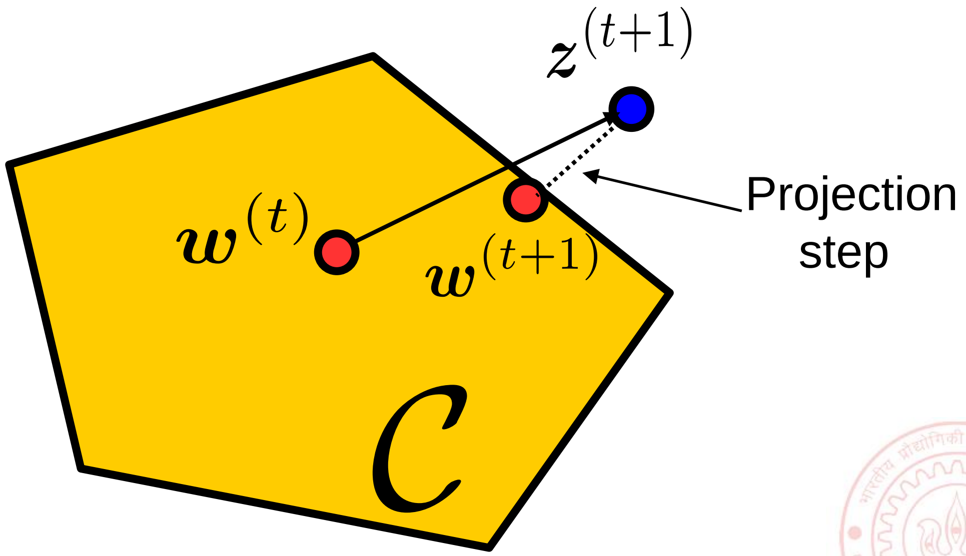
- Consider an optimization problem of the form

$$w_{opt} = \arg\min_{w \in \mathcal{C}} L(\boldsymbol{w})$$

- Projected GD is very similar to GD with an extra projection step

- Each iteration $\boldsymbol{t}$ will be of the form
  - Perform update: $\boldsymbol{z}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$
  - Check if $\boldsymbol{z}^{(t+1)}$ satisfies constraints
    - If $\boldsymbol{z}^{(t+1)} \in \mathcal{C}$, set $\boldsymbol{w}^{(t+1)} = \boldsymbol{z}^{(t+1)}$
    - If $\boldsymbol{z}^{(t+1)} \notin \mathcal{C}$, project as $\boldsymbol{w}^{(t+1)} = \Pi_{\mathcal{C}}[\boldsymbol{z}^{(t+1)}]$

Projection operator



Projected GD: How to Project?

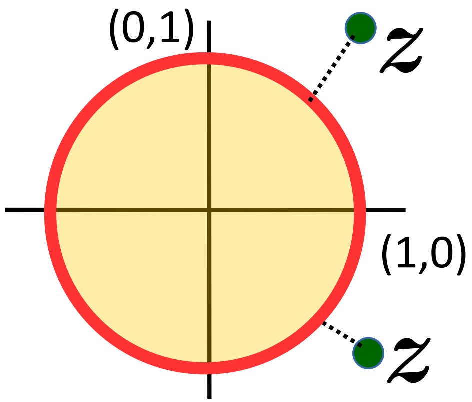
- Here projecting a point means finding the "closest" point from the constraint set

$$\Pi_{\mathcal{C}}[\boldsymbol{z}] = \arg\min_{\boldsymbol{w}\in\mathcal{C}} ||\boldsymbol{z} - \boldsymbol{w}||^2$$

Projected GD commonly used only when the projection step is simple and efficient to compute

- For some sets $\mathcal{C}$, the projection step is easy

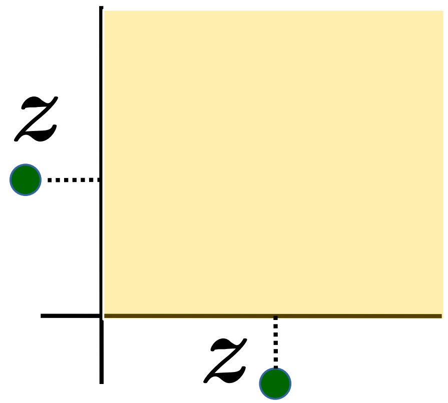
$\mathcal{C}$: Unit radius ℓ_2 ball



Projection = Normalize to unit Euclidean length vector

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

$\mathcal{C}$: Set of non-negative reals



Projection = Set each negative entry in $\boldsymbol{z}$ to be zero

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

Proximal Gradient Descent

- Consider minimizing a regularized loss function of the form

$$\arg\min_w L(\boldsymbol{w}) + R(\boldsymbol{w})$$

Note: The reg. hyperparam. λ assumed part of $R(\boldsymbol{w})$ itself

- Proximal GD popular when regularizer $R(\boldsymbol{w})$ is non-differentiable
- Basic idea: Do GD on $L(\boldsymbol{w})$ and use a prox. operator to regularize via $R(\boldsymbol{w})$
- For a func. R, its prox. operator is $\mathrm{prox}_R(\boldsymbol{z}) = \arg\min_{\boldsymbol{w}} \left[R(\boldsymbol{w}) + \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{w}\|_2^2\right]$

Proximal GD

- Assume reg. loss function of the form $L(\boldsymbol{w}) + R(\boldsymbol{w})$
- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2,\ldots$ (or until convergence)
  - Calculate the (sub)gradient of train. Loss (w/o reg.)

    $$\boldsymbol{g}^{(t)} \in \partial L(\boldsymbol{w}^{(t)})$$

  - Set learning rate η_t
  - Step 1: $\boldsymbol{z}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$
  - Step 2: $\boldsymbol{w}^{(t+1)} = \mathrm{prox}_R(\boldsymbol{z}^{(t+1)})$

Special Cases

For $R(\boldsymbol{w}) = 0.5 \times \|\mathbf{w}\|_2^2$

$\mathrm{prox}_R(\boldsymbol{z}) = \boldsymbol{z}/2$ i.e. scaling

That is, regularize by reducing the value of each component of the the vector $\boldsymbol{z}$ by half

If $R(\boldsymbol{w})$ defines a set based constraint

$$R(\boldsymbol{w}) := \boldsymbol{w} \in \mathcal{C}$$

$$\mathrm{prox}_R(\boldsymbol{z}) = \arg\min_{\boldsymbol{w}\in\mathcal{C}} \|\boldsymbol{z} - \boldsymbol{w}\|^2$$

Prox. GD becomes equivalent to projected GD

Constrained Opt. via Lagrangian

- Consider the following constrained minimization problem (using f instead of L)

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} f(\boldsymbol{w}), \quad \text{s.t.} \quad g(\boldsymbol{w}) \leq 0$$

- Note: If constraints of the form $g(w) \geq 0$, use $-g(w) \leq 0$
- Can handle multiple inequality and equality constraints too (will see later)
- Can transform the above into the following equivalent <u>unconstrained</u> problem

$$\hat{\boldsymbol{w}} = \arg\min f(\boldsymbol{w}) + c(\boldsymbol{w})$$

$$c(\boldsymbol{w}) = \max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) = \begin{cases} \infty, & \text{if } g(\boldsymbol{w}) > 0 \quad \text{(constraint violated)} \\ 0 & \text{if } g(\boldsymbol{w}) \leq 0 \quad \text{(constraint satisfied)} \end{cases}$$

- Our problem can now be written as

$$\boxed{\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \left\{ f(\boldsymbol{w}) + \arg\max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) \right\}}$$

Constrained Opt. via Lagrangian

- Therefore, we can write our original problem as

The Lagrangian: $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha})$

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \left\{ f(\boldsymbol{w}) + \arg\max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) \right\} = \arg\min_{\boldsymbol{w}} \left\{ \arg\max_{\alpha \geq 0} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\}$$

- The Lagrangian is now optimized w.r.t. $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ (Lagrange multiplier)
- We can defined Primal and Dual problem as

$$\hat{\boldsymbol{w}}_P = \arg\min_{\boldsymbol{w}} \left\{ \arg\max_{\alpha \geq 0} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\} \quad \text{(primal problem)}$$

$$\hat{\boldsymbol{w}}_D = \arg\max_{\alpha \geq 0} \left\{ \arg\min_{\boldsymbol{w}} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\} \quad \text{(dual problem)}$$

Both equal if $f(w)$ and the set $g(w) \leq 0$ are convex

$$\alpha_D g(\hat{\boldsymbol{w}}_D) = 0$$

complimentary slackness/Karush-Kuhn-Tucker (KKT) condition

Constrained Opt. with Multiple Constraints

- We can also have multiple inequality and equality constraints

$$\begin{aligned} \hat{\boldsymbol{w}} &= \arg\min_{\boldsymbol{w}} f(\boldsymbol{w}) \\ \text{s.t.} \quad & g_i(\boldsymbol{w}) \leq 0, \quad i = 1, \ldots, K \\ & h_j(\boldsymbol{w}) = 0, \quad j =, 1, \ldots, L \end{aligned}$$

- Introduce Lagrange multipliers $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_K]$ and $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_L]$
- The Lagrangian based primal and dual problems will be

$$\hat{\boldsymbol{w}}_P = \arg\min_{\boldsymbol{w}} \{\arg\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \{f(\boldsymbol{w}) + \sum_{i=1}^{K} \alpha_i g_i(\boldsymbol{w}) + \sum_{j=1}^{L} \beta_j h_j(\boldsymbol{w})\}\}$$

$$\hat{\boldsymbol{w}}_D = \arg\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \{\arg\min_{\boldsymbol{w}} \{f(\boldsymbol{w}) + \sum_{i=1}^{K} \alpha_i g_i(\boldsymbol{w}) + \sum_{j=1}^{L} \beta_j h_j(\boldsymbol{w})\}\}$$

Some other useful optimization methods

Co-ordinate Descent (CD)

- Standard gradient descent update for : $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$
- CD: In each iter, update only one entry (co-ordinate) of $\boldsymbol{w}$. Keep all others fixed

$$w_d^{(t+1)} = w_d^{(t)} - \eta_t g_d^{(t)} \qquad d \in \{1,2,\dots,D\}$$

$g_d = \nabla_{w_d} L(\boldsymbol{w})$ – partial derivative w.r.t. the d^{th} element of vector $\boldsymbol{w}$ (or the d^{th} element of the gradient vector g)

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

- Cost of each update is now independent of D
- In each iter, can choose co-ordinate to update unif. randomly or in cyclic order
- Instead of updating a single co-ord, can also update "blocks" of co-ordinates
  - Called Block co-ordinate descent (BCD)
- To avoid $O(D)$ cost of gradient computation, can cache previous computations
  - Recall that grad. computations may have terms like $\boldsymbol{w}^\top \boldsymbol{x}$ – if just one co-ordinate of w changes, we should avoid computing the new $\boldsymbol{w}^\top \boldsymbol{x}$ $(= \sum_d w_d x_d)$ from scratch

Alternating Optimization (ALT-OPT)

- Consider opt. problems with several variables, say two variables $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$

$$\{\hat{\boldsymbol{w}}_1, \hat{\boldsymbol{w}}_2\} = \arg \min_{\boldsymbol{w}_1, \boldsymbol{w}_2} \mathcal{L}(\boldsymbol{w}_1, \boldsymbol{w}_2)$$

- Often, this "joint" optimization is hard/impossible to solve
- We can take an alternating optimization approach to solve such problems

ALT-OPT

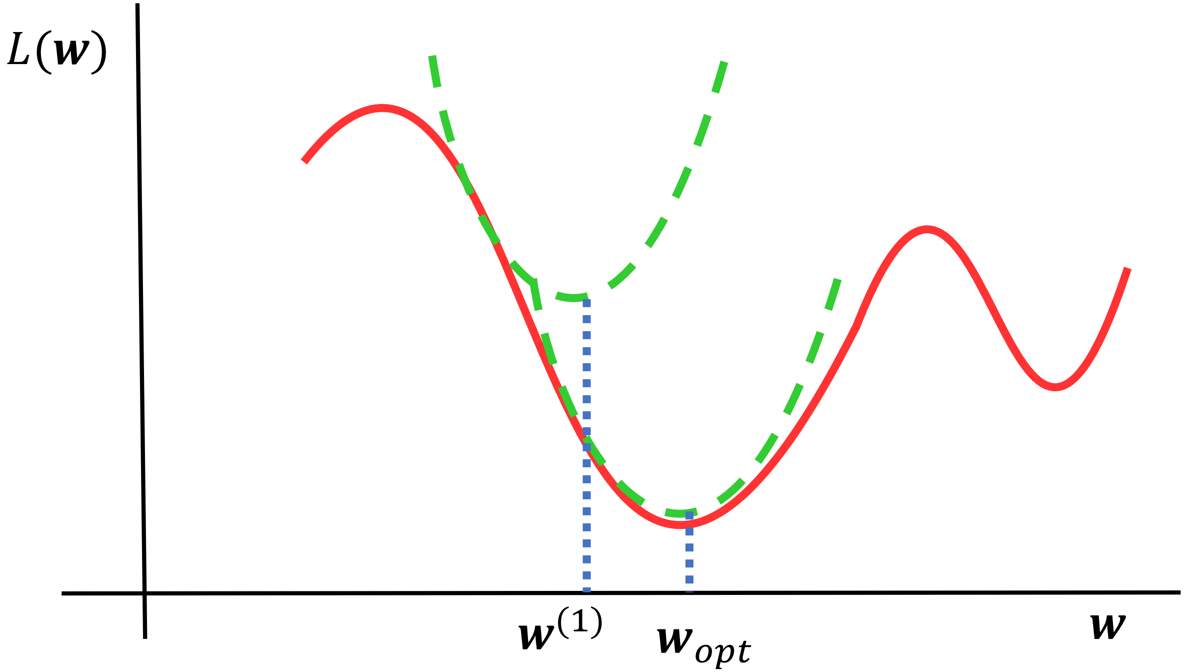
1. Initialize one of the variables, e.g., $\boldsymbol{w}_2 = \boldsymbol{w}_2^{(0)}, t = 0$
2. Solve $\boldsymbol{w}_1^{(t+1)} = \arg\min_{\boldsymbol{w}_1} \mathcal{L}(\boldsymbol{w}_1, \boldsymbol{w}_2^{(t)})$ //$\boldsymbol{w}_2$ "fixed" at its most recent value $\boldsymbol{w}_2^{(t)}$
3. Solve $\boldsymbol{w}_2^{(t+1)} = \arg\min_{\boldsymbol{w}_2} \mathcal{L}(\boldsymbol{w}_1^{(t+1)}, \boldsymbol{w}_2)$ //$\boldsymbol{w}_1$ "fixed" at its most recent value $\boldsymbol{w}_1^{(t+1)}$
4. $t = t + 1$. Go to step 2 if not converged yet.

- Usually converges to a local optima. But very very useful. Will see examples later
  - Also related to the Expectation-Maximization (EM) algorithm which we will see later

Newton's Method

- Unlike GD and its variants, Newton's method uses second-order information (second derivative, a.k.a. the Hessian)
- At each point $\boldsymbol{w}^{(t)}$, minimize the quadratic (second-order) approx. of $L(\boldsymbol{w})$

$$\boldsymbol{w}^{(t+1)} = \arg\min_{\boldsymbol{w}} \left[L\left(\boldsymbol{w}^{(t)}\right) + \nabla L\left(\boldsymbol{w}^{(t)}\right)^{\top}\left(\boldsymbol{w} - \boldsymbol{w}^{(t)}\right) + \frac{1}{2}\left(\boldsymbol{w} - \boldsymbol{w}^{(t)}\right)^{\top}\nabla^2 L\left(\boldsymbol{w}^{(t)}\right)\left(\boldsymbol{w} - \boldsymbol{w}^{(t)}\right)\right]$$



Show that $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \left(\nabla^2 L\left(\boldsymbol{w}^{(t)}\right)\right)^{-1} \nabla L\left(\boldsymbol{w}^{(t)}\right)$
$= \boldsymbol{w}^{(t)} - \left(\boldsymbol{H}^{(t)}\right)^{-1}\boldsymbol{g}^{(t)}$

Converges much faster than GD (very fast for convex functions). Also no "learning rate". But per iteration cost is slower due to Hessian computation and inversion

Faster versions of Newton's method also exist, e.g., those based on approximating Hessian using previous gradients (see L-BFGS which is a popular method)

Coming up next

- Some practical issue in optimization for ML
- Wrapping up the discussion of optimization techniques
- Probabilistic models for ML