# Optimization Techniques for ML (contd)

CS771: Introduction to Machine Learning

Piyush Rai

## Optimization Problems in ML

- The general form of an optimization problem in ML will usually be

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

$L(\boldsymbol{w})$ may denote the training loss, or training loss + regularizer term

or

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

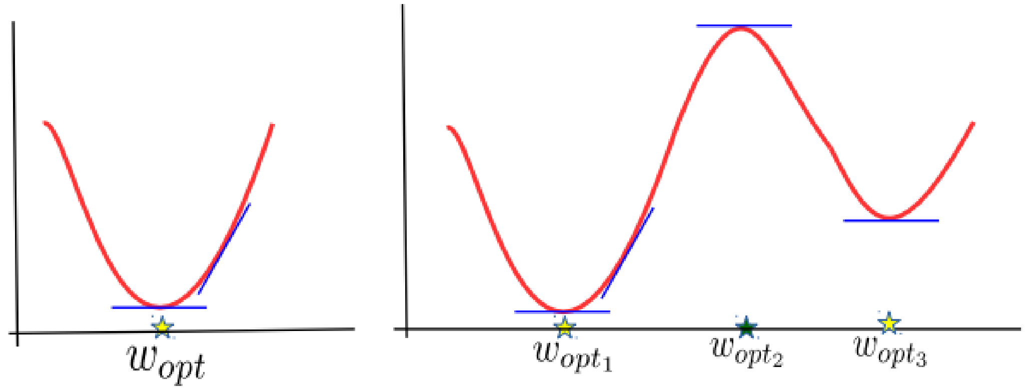
Training loss with a constraint on $\boldsymbol{w}$

- $\boldsymbol{C}$ is the constraint set that the solution must belong to, e.g.,
  - Non-negativity constraint: All entries in $\boldsymbol{w}_{opt}$ must be non-negative
  - Sparsity constraint: $\boldsymbol{w}_{opt}$ is a sparse vector with at most $\boldsymbol{K}$ non-zeros
- Constrained opt. probs can be converted into unconstrained opt. (will see later)
- For now, assume we have an unconstrained optimization problem

## Methods for Solving Optimization Problems

# Method 1: Using First-Order Optimality

- Very simple. Already used this approach for linear and ridge regression



Called "first order" since only gradient is used and gradient provides the first order info about the function being optimized

The approach works only for very simple problems where the objective is convex and there are no constraints on the values $\boldsymbol{w}$ can take

- First order optimality: The gradient $\boldsymbol{g}$ must be equal to zero at the optima

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}}[L(\boldsymbol{w})] = \mathbf{0}$$

E.g., linear/ridge regression, but not for logistic/softmax regression

- Sometimes, setting $\boldsymbol{g} = \mathbf{0}$ and solving for $\boldsymbol{w}$ gives a closed form solution
- If closed form solution is not available, the gradient vector $\boldsymbol{g}$ can still be used in iterative optimization algos, like gradient descent

## Method 2: Iterative Optimiz. via Gradient Descent

Can I used this approach to solve maximization problems?

For max. problems we can use gradient ascent
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \boldsymbol{g}^{(t)}$$

Iterative since it requires several steps/iterations to find the optimal solution

Fact: Gradient gives the direction of steepest change in function's value

Will move in the direction of the gradient

For convex functions, GD will converge to the global minima

Good initialization needed for non-convex functions

### Gradient Descent

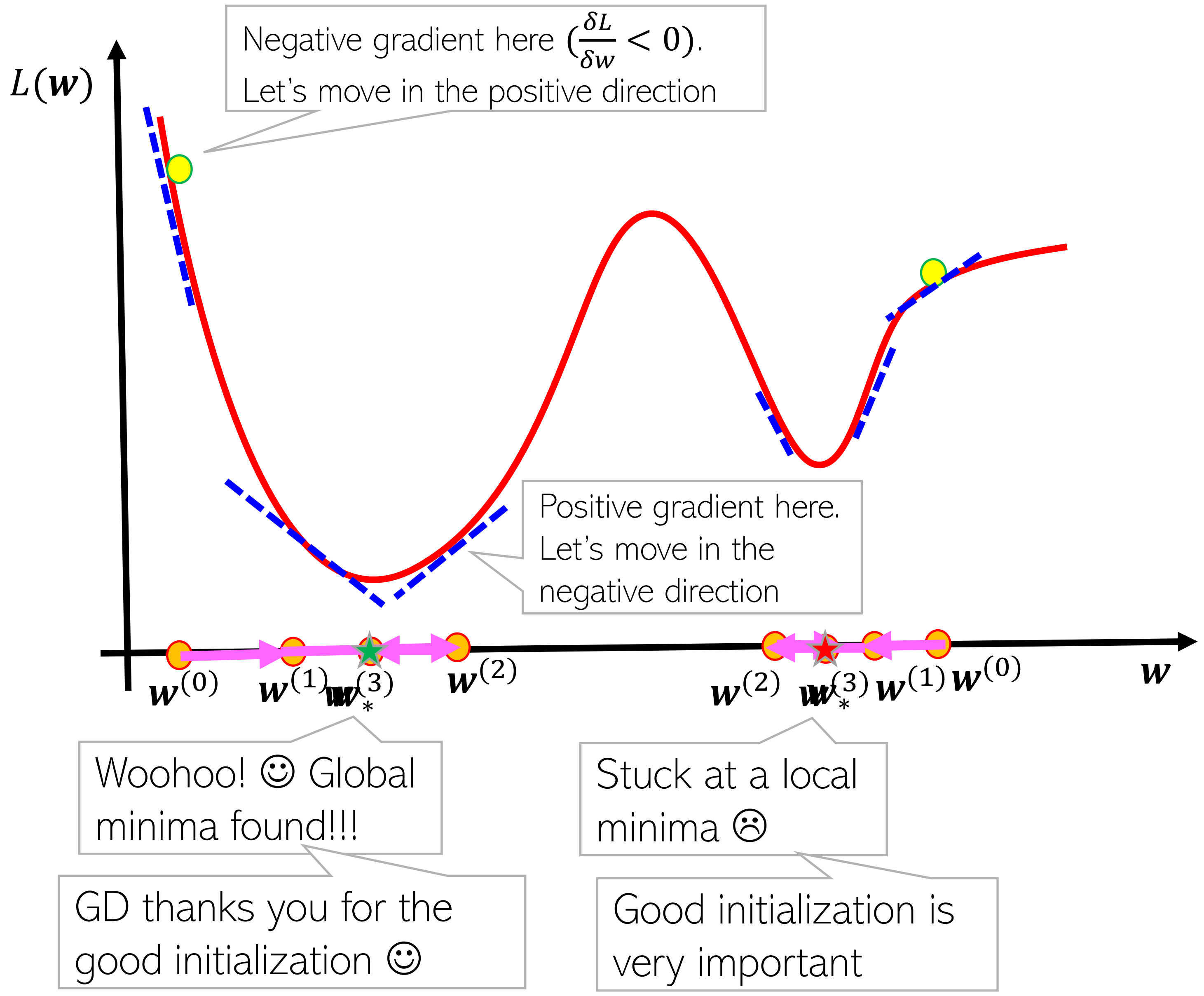
- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2,\ldots$ (or until convergence)
  - Calculate the gradient $\boldsymbol{g}^{(t)}$ using the current iterates $\boldsymbol{w}^{(t)}$
  - Set the learning rate $\eta_t$
  - Move in the opposite direction of gradient

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

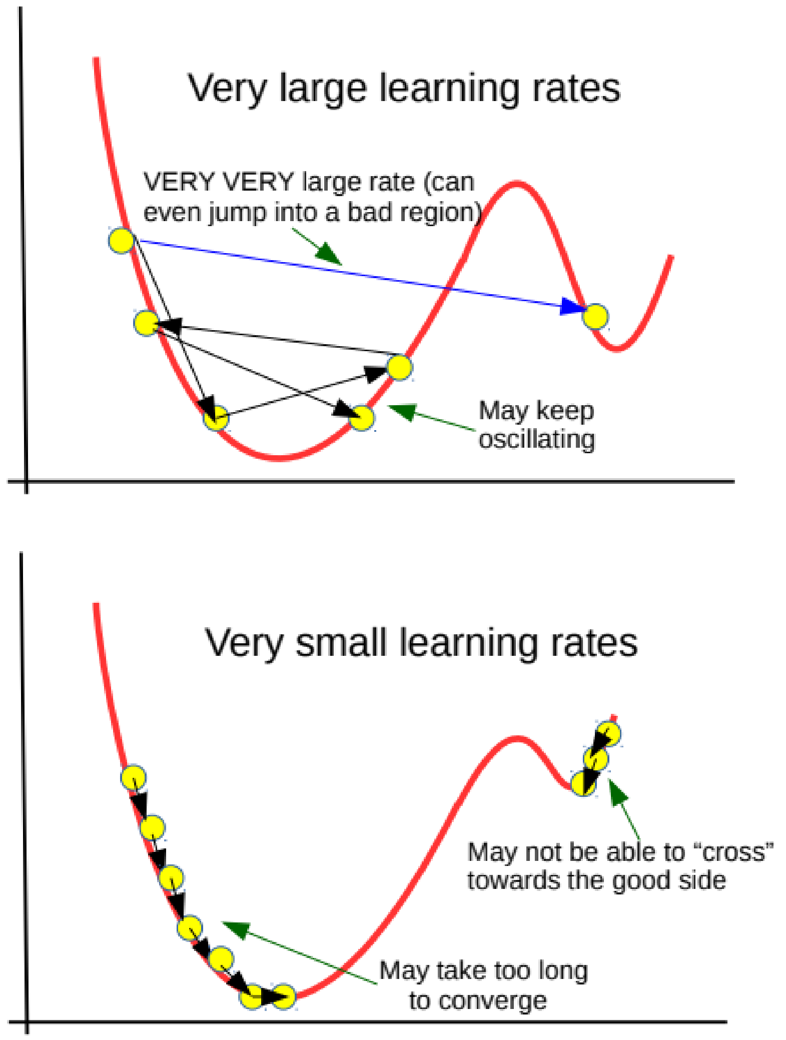
Will see the justification shortly

The learning rate very imp. Should be set carefully (fixed or chosen adaptively). Will discuss some strategies later

# Gradient Descent: An Illustration


$L(\mathbf{w})$
Negative gradient here ($\frac{\delta L}{\delta w} < 0$). Let's move in the positive direction
Positive gradient here. Let's move in the negative direction
$\mathbf{w}^{(0)}$ $\mathbf{w}^{(1)}$ $\mathbf{w}^{(3)}$ $\mathbf{w}^{*}$ $\mathbf{w}^{(2)}$
$\mathbf{w}^{(2)}$ $\mathbf{w}^{(3)}$ $\mathbf{w}^{*}$ $\mathbf{w}^{(1)}$ $\mathbf{w}^{(0)}$
$\mathbf{w}$
Woohoo! ☺ Global minima found!!!
GD thanks you for the good initialization ☺
Stuck at a local minima ☹
Good initialization is very important

## Learning rate is very important


Very large learning rates
VERY VERY large rate (can even jump into a bad region)
May keep oscillating
Very small learning rates
May not be able to "cross" towards the good side
May take too long to converge

## GD: An Example

- Let's apply GD for least squares linear regression

$$\boldsymbol{w}_{ridge} = \text{arg min}_{\boldsymbol{w}}\, L_{reg}(\boldsymbol{w}) = \text{arg min}_{\boldsymbol{w}}\, \frac{1}{N}\sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- The gradient: $\boldsymbol{g} = -\frac{2}{N}\sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n$

- Each GD update will be of the form

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \frac{2}{N}\sum_{n=1}^{N}\left(y_n - \boldsymbol{w}^{(t)^\top}\boldsymbol{x}_n\right)\boldsymbol{x}_n$$

Prediction error of current model $\boldsymbol{w}^{(t)}$ on the $n^{th}$ training example

Training examples on which the current model's error is large contribute more to the update

- Exercise: Assume $N = 1$, and show that GD update improves prediction on the training input $(\boldsymbol{x}_n, y_n)$, i.e, $y_n$ is closer to $\boldsymbol{w}^{(t+1)^\top}\boldsymbol{x}_n$ than to $\boldsymbol{w}^{(t)^\top}\boldsymbol{x}_n$
  - This is sort of a proof that GD updates are "corrective" in nature (and it actually is true not just for linear regression but can also be shown for various other ML models)

# Faster GD: Stochastic Gradient Descent (SGD)

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

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

- The gradient in this case can be written as

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

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

Gradient of the loss on $n^{th}$ training example

- Stochastic Gradient Descent (SGD) approximates $\boldsymbol{g}$ using a single training example

- At iter. $\boldsymbol{t}$, pick an index $\boldsymbol{i} \in \{1,2,\dots,N\}$ uniformly randomly and approximate $\boldsymbol{g}$ as

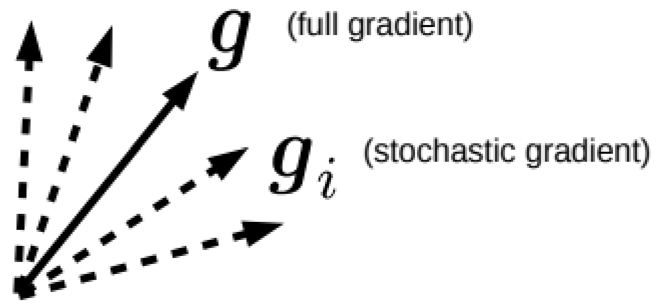
$$\boldsymbol{g} \approx \boldsymbol{g}_i = \nabla_{\boldsymbol{w}} \ell_i(\boldsymbol{w})$$

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 $\boldsymbol{O(D)}$ whereas GD per iter cost is $\boldsymbol{O(ND)}$

## 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

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

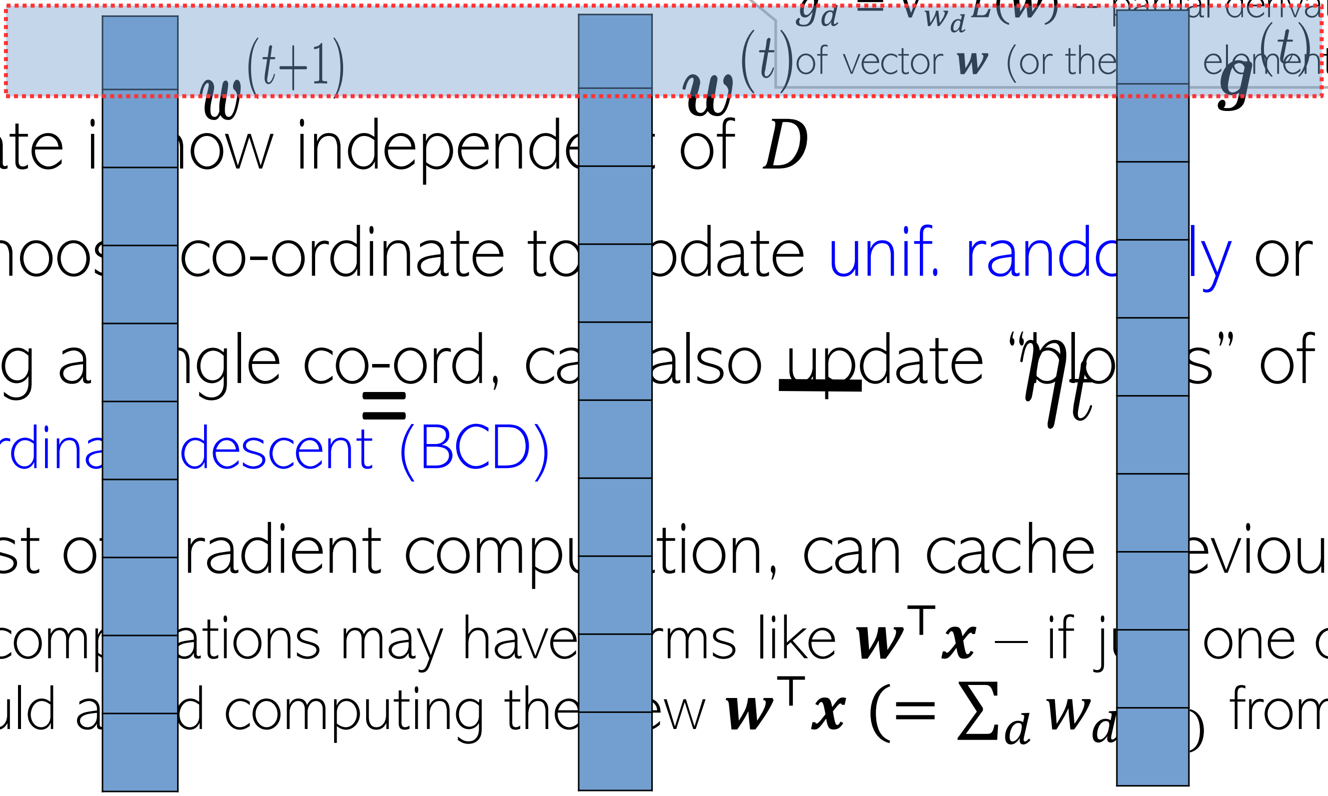
- Averaging helps in reducing the variance in the stochastic gradient
- Time complexity is $O(BD)$ per iteration in this case

# 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

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

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



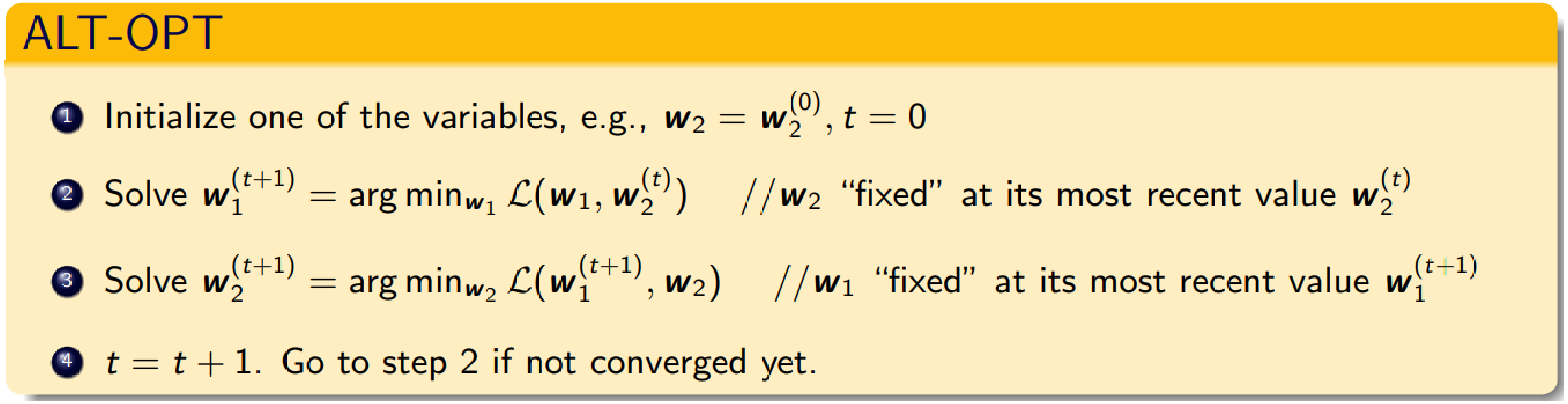
- Cost of each update i now independe of $D$
- In each iter, can choos co-ordinate to odate unif. rando ly or in cyclic order
- Instead of updating a ngle co-ord, ca also update "blo s" of co-ordinates
  - Called block co-ordina descent (BCD)
- To avoid $O(D)$ cost o radient compu tion, can cache evious computations
  - Recall that grad. comp ations may have rms like $\boldsymbol{w}^\top \boldsymbol{x}$ – if j one co-ordinate of $\boldsymbol{w}$ changes, we should a d computing the ew $\boldsymbol{w}^\top \boldsymbol{x}$ $(= \sum_d w_d$ $)$ 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

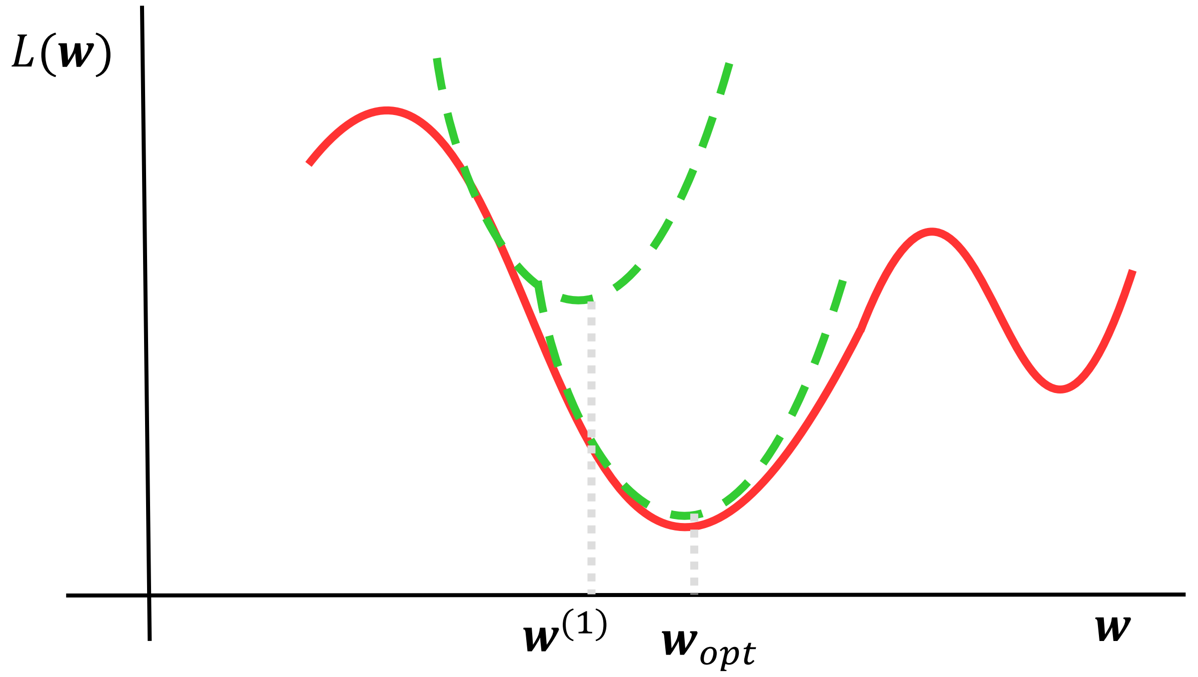
1. Initialize one of the variables, e.g., $\boldsymbol{w}_2 = \boldsymbol{w}_2^{(0)}, t = 0$
2. Solve $\boldsymbol{w}_1^{(t+1)} = \arg\min_{\boldsymbol{w}_1} \mathcal{L}(\boldsymbol{w}_1, \boldsymbol{w}_2^{(t)})$ $\quad$ //$\boldsymbol{w}_2$ "fixed" at its most recent value $\boldsymbol{w}_2^{(t)}$
3. Solve $\boldsymbol{w}_2^{(t+1)} = \arg\min_{\boldsymbol{w}_2} \mathcal{L}(\boldsymbol{w}_1^{(t+1)}, \boldsymbol{w}_2)$ $\quad$ //$\boldsymbol{w}_1$ "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

# Second Order Methods: Newton's Method

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

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



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

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

- Constrained optimization
- Optimizing non-differentiable functions
- Some practical issue in optimization for ML