Optimization Techniques for ML (1)

Piyush Rai

Introduction to Machine Learning (CS771A)

August 23, 2018

Recap: Generative Classification

Class-Marginal or Class Prior

Class-Conditional

$$p(y = k|\boldsymbol{x}) = \frac{p(y = k)p(\boldsymbol{x}|y = k)}{p(\boldsymbol{x})}$$

Class-Marginal and Class-Conditional estimated from training data

$$p(y = k) = \pi_k \quad \text{and} \quad p(y) = \text{multinoulli}(\pi_1, \ldots, \pi_K)$$

$$p(\boldsymbol{x}|y = k) \text{ depends on type of } \boldsymbol{x}$$

naïve Bayes assumption: $p(\boldsymbol{x}|y = k) = \prod_{d=1}^{D} p(x_d|y = k)$

(reduces the number of parameters to be estimated for $p(\boldsymbol{x}|y = k)$)

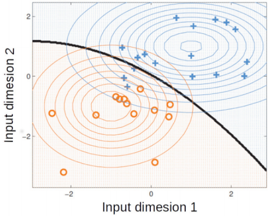
E.g.: Gaussian with Diagonal or Spherical Covariance Matrix

Recap: Generative Classification Decision Boundaries

- We can look at the case when we have Gaussians as class-conditionals

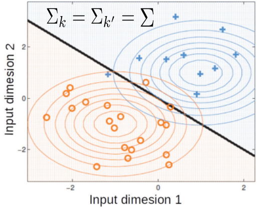
$$p(y=k|\mathbf{x}) = \frac{p(y=k)p(\mathbf{x}|y=k)}{p(\mathbf{x})} = \frac{\pi_k |\mathbf{\Sigma}_k|^{-1/2}\exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^\top \mathbf{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right]}{\sum_{k=1}^K \pi_k |\mathbf{\Sigma}_k|^{-1/2}\exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^\top \mathbf{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right]}$$

- All points $\boldsymbol{x}$ at the boundary between classes k and k' must satisfy $p(y=k|\boldsymbol{x}) = p(y=k'|\boldsymbol{x})$
- Quadratic decision boundary if covariances unequal, linear if covariances equal



$$(\mathrm{x}-\mu_k)^\top \Sigma_k^{-1}(\mathrm{x}-\mu_k) - (\mathrm{x}-\mu_{k'})^\top \Sigma_{k'}^{-1}(\mathrm{x}-\mu_{k'}) = 0$$

(a quadratic function of $\boldsymbol{x}$)



$$(\mathrm{x}-\mu_k)^\top \Sigma^{-1}(\mathrm{x}-\mu_k) - (\mathrm{x}-\mu_{k'})^\top \Sigma^{-1}(\mathrm{x}-\mu_{k'}) = 0$$

(reduces to the form $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$)

Recap: Equivalence to Discriminative Model in Linear Case

- For the Gaussian class-conditionals with equal covariances (linear case)

$$p(y = k|\boldsymbol{x}, \theta) \propto \pi_k \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right]$$

- Expanding further, we can write the above as

$$p(y = k|\boldsymbol{x}, \theta) \propto \exp\left[\boldsymbol{\mu}_k^\top \Sigma^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}_k^\top \Sigma^{-1}\boldsymbol{\mu}_k + \log \pi_k\right] \exp\left[\boldsymbol{x}^\top \Sigma^{-1}\boldsymbol{x}\right]$$

- After normalizing, the above posterior class probability can be written as

$$p(y = k|\boldsymbol{x}, \theta) = \frac{\exp\left[\boldsymbol{w}_k^\top \boldsymbol{x} + b_k\right]}{\sum_{k=1}^K \exp\left[\boldsymbol{w}_k^\top \boldsymbol{x} + b_k\right]}$$

  where $\boldsymbol{w}_k = \Sigma^{-1}\boldsymbol{\mu}_k$ and $b_k = -\frac{1}{2}\boldsymbol{\mu}_k^\top \Sigma^{-1}\boldsymbol{\mu}_k + \log \pi_k$

- Interestingly, this has exactly the same form as the softmax classification model

Recap: Equivalence to Prototype based Classification

- Again consider, generative clasification with Gaussian class-conditionals
- Consider the prediction rule

$$\hat{y} = \arg\max_k p(y = k|\mathbf{x}) = \arg\max_k \quad \pi_k \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right]$$

$$= \arg\max_k \quad \log \pi_k - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)$$

- This is a generalization of prototype based classification
- Generalization because we are not simply computing Euclidean distances to make predictions
- If we assume the classes to be of equal size, i.e., $\pi_k = 1/K$ and $\Sigma_k = \Sigma$. Then we will have

$$\hat{y} = \arg\min_k \quad (\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)$$

- This is equivalent to assigning $\boldsymbol{x}$ to the "closest" class in terms of a Mahalanobis distance
  - The covariance matrix "modulates" how the distances are computed
- If we further assume $\Sigma = \mathbf{I}$, we get the exact same model as prototype based clasification

Discriminative vs Generative: A Few Points

- Generative models are always probabilistic with models for $p(y)$ and $p(\boldsymbol{x}|y)$
- Some discriminative models are also non-probabilistic
  - Any model of the form $y = f(\boldsymbol{x})$ with no model for $\boldsymbol{x}$ is a discriminative model
  - Example: Support Vector Machines (SVM), DT, KNN, etc.
- Discriminative models are preferred when
  - There is plenty of training data. Modeling $\boldsymbol{x}$ doesn't usually matter much in that case
- Some situations when generative models are preferred
  - We can (afford to) learn the structure of the inputs
  - We want to do semi-supervised learning (or if we don't have much labeled data)
  - We would like to "generate" data (note that we are learning $p(\boldsymbol{x}|y)$)
- Generative and discriminative models can be combined as well

Optimization Techniques for ML

Optimization Problems in ML

- The generic form of most optimization problems in ML

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} \ell_n(\boldsymbol{w}) + R(\boldsymbol{w})$$

- $\ell_n(\boldsymbol{w})$: loss function for the n^{th} training example, $R(\boldsymbol{w})$: (optional) regularizer on the parameters
- Some common examples

$$\hat{w} = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda ||w||_2^2$$

$$\hat{w} = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda ||\boldsymbol{w}||_1$$

ℓ_1 regularizer

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} - \left[\sum_{n=1}^{N} (y_n \boldsymbol{w}^\top \boldsymbol{x}_n - \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_n))) \right] + \lambda ||\boldsymbol{w}||^2$$

ℓ_2 regularized logistic regression assuming $y_n \in \{0, 1\}$

$$\hat{w} = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} \log(1 + \exp(-y_n \boldsymbol{w}^\top \boldsymbol{x}_n) + \lambda ||\boldsymbol{w}||^2$$

ℓ_2 regularized logistic regression assuming $y_n \in \{-1, 1\}$

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} - \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{I}[y_n = k] \log \left[\frac{\exp(\boldsymbol{w}_k^\top \boldsymbol{x}_n)}{\sum_{k=1}^{K} \exp(\boldsymbol{w}_k^\top \boldsymbol{x}_n)} \right]$$

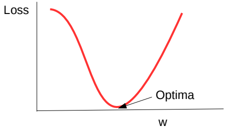
Softmax Regression with K classes (assuming no regularization)

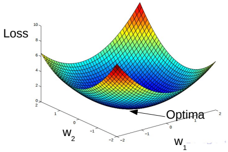
$$\hat{\theta}_{MAP} = \arg\min_{\theta} - \left[\sum_{n=1}^{N} \log p(y_n|\theta) + \log p(\theta) \right]$$

A general MAP estimation problem

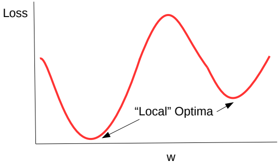
Optimization Problems in ML

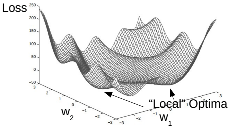
- Wish to find the optima (minima) of an objective function, that can be seen as as a curve/surface
- For simple cases, the functions may look like this





- In many cases, the functions may even look like this





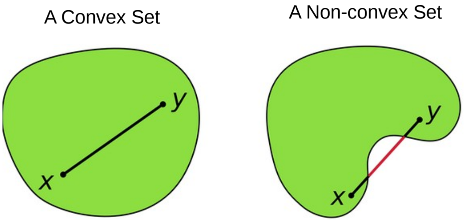
- Functions with unique minima: Convex; Functions with many local minima: Non-convex

Interlude: Convex Sets

- A set $\mathcal{S}$ of points is a convex set, if for any two points $x, y \in \mathcal{S}$, and $0 \leq \alpha \leq 1$

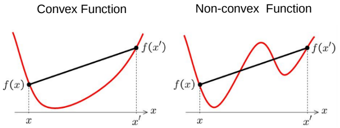
$$z = \alpha x + (1 - \alpha) y \in \mathcal{S}$$

.. i.e., all points on the line-segment between x and y lie within the set



Interlude: Convex Functions

- Note: The domain of a convex function needs to be a convex set (a required condition)
- Informally, a function $f(x)$ is convex if all of its chords lie above the function everywhere



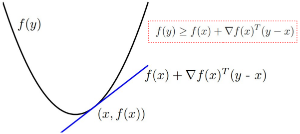
Note: "Chord lies above function" more formally means

If f is convex then given $\alpha_1, \ldots, \alpha_n$ s.t $\sum_{i=1}^{n} \alpha_i = 1$

$$f\left(\sum_{i=1}^{n} \alpha_i x_i\right) \leq \sum_{i=1}^{n} \alpha_i f(x_i)$$

Jensen's Inequality

- Formally, (assuming the function is differentiable), some conditions to test for convexity:
  - First-order convexity (graph of f must be above all the tangents)



  - Second-order convexity: Second derivative a.k.a. Hessian (if exists) must be positive semi-definite

$$\nabla^2 f(x) \succeq 0$$

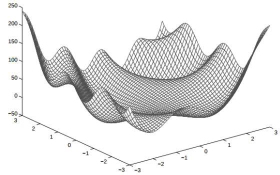
Interlude: Convex Functions

- Some basic rules to check if $f(x)$ is convex or not
  - All linear and affine functions (e.g., $ax + b$) are convex
  - $\exp(ax)$ is convex for $x \in \mathbb{R}$, for any $a \in \mathbb{R}$
  - $\log(x)$ is concave (not convex) for $x > 0$
  - x^a is convex for $x > 0$, for any $a \geq 1$ and $a < 0$, concave for $0 \leq a \leq 1$
  - $|x|^a$ is convex for $x \in \mathbb{R}$, for any $a \geq 1$
  - All norms in $\mathbb{R}^D$ are convex
  - Non-negative weighted sum of convex functions is also a convex function
  - Affine transformation preserves convexity: if $f(x)$ is convex then $f(x) = f(ax + b)$ is also convex
  - Some rules to check whether composition $f(x) = h(g(x))$ of two functions h and g is convex

f is convex if h is convex and nondecreasing, and g is convex,
f is convex if h is convex and nonincreasing, and g is concave,
f is concave if h is concave and nondecreasing, and g is concave,
f is concave if h is concave and nonincreasing, and g is convex.

  - Most of these also apply when x is a vector (and many other rules)

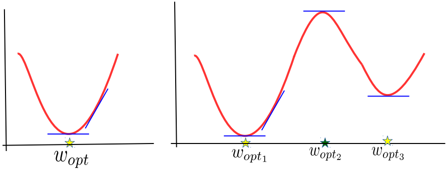
Disclaimer:
It's OK to be non-convex :-)



Many interesting ML problems are in fact non-convex and there are ways to optimize non-convex objectives (non-convex optimization is a research area in itself)

Solving Optimization Problems

- The most basic approach: Use first-order optimality condition



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

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \left[\sum_{n=1}^{N} \ell_n(\boldsymbol{w}) + R(\boldsymbol{w}) \right] = 0$$

- Sometimes, setting $\boldsymbol{g} = 0$ and solving for $\boldsymbol{w}$ gives a closed form solution (recall linear regression)
- .. and often it does NOT (recall logistic regression)
- The gradient $\boldsymbol{g}$ can still be helpful since we can use it in iterative optimization methods

Gradient Descent

Gradient Descent

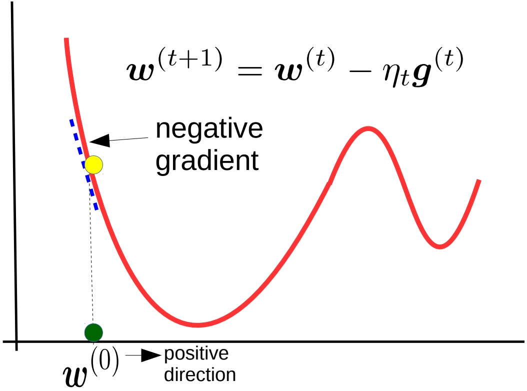
1. Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
2. Update $\boldsymbol{w}$ as follows

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

3. Repeat until convergence

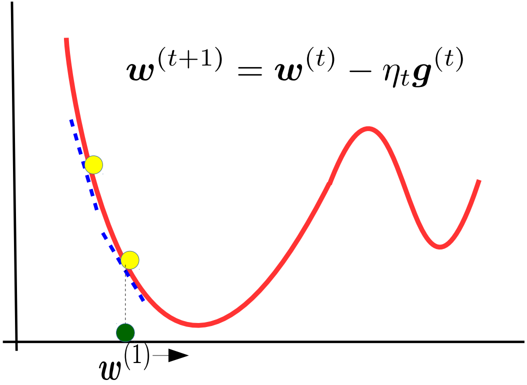
Gradient Descent

- A very simple, first-order method (uses only the gradient $\boldsymbol{g}$ of the objective)
- Basic idea: Start at some location $\boldsymbol{w}^{(0)}$ and move in the opposite direction of the gradient



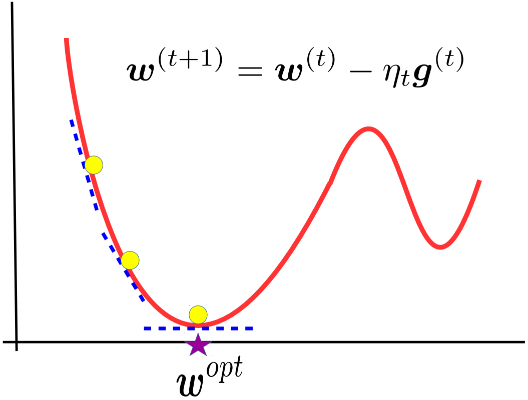
Gradient Descent

- A very simple, first-order method (uses only the gradient $\boldsymbol{g}$ of the objective)
- Basic idea: Start at some location $\boldsymbol{w}^{(0)}$ and move in the opposite direction of the gradient



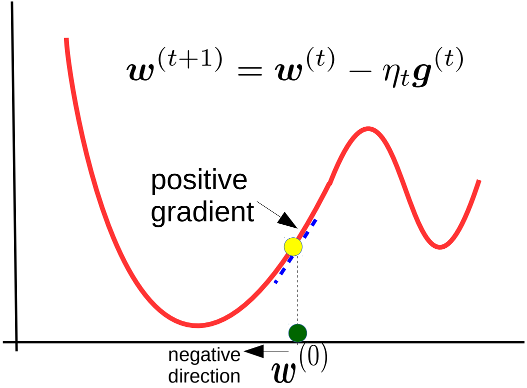
Gradient Descent

- A very simple, first-order method (uses only the gradient $\boldsymbol{g}$ of the objective)
- Basic idea: Start at some location $\boldsymbol{w}^{(0)}$ and move in the opposite direction of the gradient



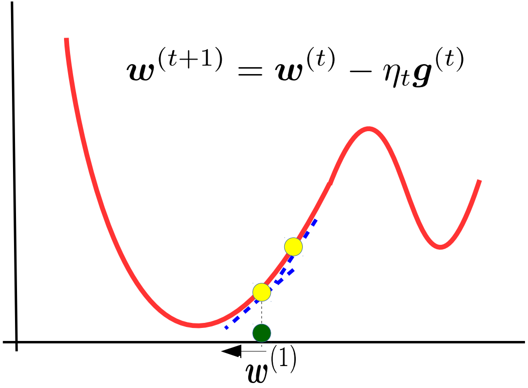
Gradient Descent

- A very simple, first-order method (uses only the gradient $\boldsymbol{g}$ of the objective)
- Basic idea: Start at some location $\boldsymbol{w}^{(0)}$ and move in the opposite direction of the gradient



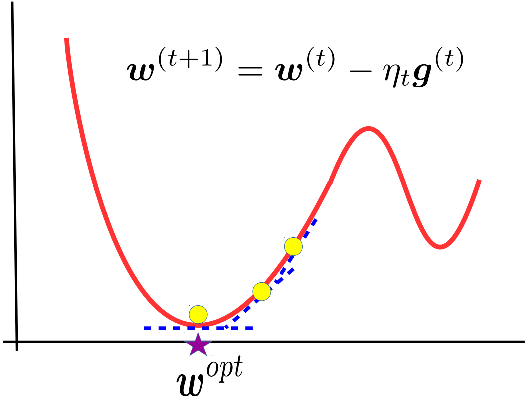
Gradient Descent

- A very simple, first-order method (uses only the gradient $\boldsymbol{g}$ of the objective)
- Basic idea: Start at some location $\boldsymbol{w}^{(0)}$ and move in the opposite direction of the gradient



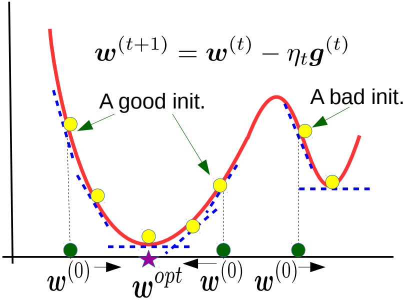
Gradient Descent

- A very simple, first-order method (uses only the gradient $\boldsymbol{g}$ of the objective)
- Basic idea: Start at some location $\boldsymbol{w}^{(0)}$ and move in the opposite direction of the gradient



Gradient Descent

- η_t is called the learning rate (can be constant or may vary at each step)

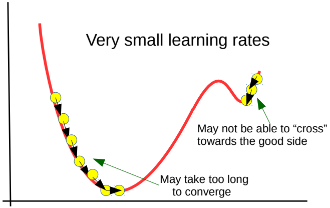


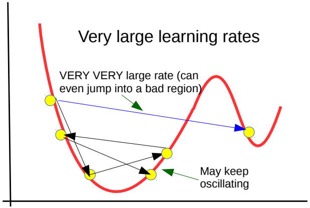
$\boldsymbol{g}^{(t)}$: Gradient at $\boldsymbol{w} = \boldsymbol{w}^{(t)}$

- Note: The effective step size (how much $\boldsymbol{w}$ moves) depends on both η_t and current gradient $\boldsymbol{g}^{(t)}$
- A good initialization $\boldsymbol{w}^{(0)}$ matters, otherwise might get trapped in a bad local optima
- If run long enough, guaranteed to converge to a local optima (=global optima for convex functions)
- **When to stop:** Many criteria, e.g., gradients become too small, or validation error starts increasing

Gradient Descent

- The learning rate η_t is important
- Very small learning rates may result in very slow convergence
- Very large learning rates may lead to oscillatory behavior or result in a bad local optima





- Many ways to set the learning rate, e.g.,
  - Constant (if properly set, can still show good convergence behavior)
  - Decreasing with t (e.g. $1/t$, $1/\sqrt{t}$, etc.)
  - Use adaptive learning rates (e.g., using methods such as Adagrad, Adam)

Gradient Descent: Gradient Computations may be Expensive

- Gradient computation in GD may be very expensive
- Reason: Need to evaluate N terms. Assuming no regularization term, something like

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

.. will be very expensive when N is very large

- A solution: Use stochastic gradient descent (SGD). Pick a random $i \in \{1, \ldots, N\}$

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

- SGD updates use this approximation of the actual gradient

Stochastic Gradient Descent

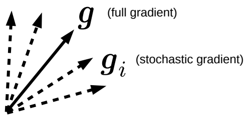
1. Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
2. Pick a random $i \in \{1, \ldots, N\}$. Update $\boldsymbol{w}$ as follows

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

3. Repeat until convergence

(Stochastic) Gradient Descent

- SGD uses a single example to compute the gradient
- Can show that $\mathbb{E}[\boldsymbol{g}_i] = \boldsymbol{g}$. Therefore $\boldsymbol{g}_i$ is an unbiased estimate of $\boldsymbol{g}$ (good)
- However, the approximate gradient will have large variance



- Many ways to control the variance in the gradient's approximation
- One simple way is to use a mini-batch containing more than one (say B) example

$$\boldsymbol{g} \approx \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{g}_i$$

- This is known as mini-batch SGD

Gradient Descent: Some Simple Examples

- Ignoring the regularizer, consider the loss functions for linear and logistic regression

$$\text{Linear Regression:} \quad \mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

$$\text{Logistic Regression:} \quad \mathcal{L}(\boldsymbol{w}) = -\sum_{n=1}^{N}(y_n \boldsymbol{w}^\top \boldsymbol{x}_n - \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_n))) \quad (\text{assuming } y_n \in \{0, 1\})$$

- Both objectives are convex functions (can get global minima). The (full) gradients for each will be

$$\text{Linear Regression:} \quad \boldsymbol{g} = -\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n$$

$$\text{Logistic Regression} \quad \boldsymbol{g} = -\sum_{n=1}^{N}(y_n - \mu_n)\boldsymbol{x}_n \quad (\text{where } \mu_n = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n))$$

- The GD updates in both cases will be of the form $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$
- Note that highly mispredicted inputs $\boldsymbol{x}_n$ contribute more to $\boldsymbol{g}$ and thus to the weight updates!
- SGD is also straightforward (same as GD but with one or few inputs for each gradient computation)

GD and SGD: Some Comments

- Note that we could solve linear regression in closed form

$$\boldsymbol{w} = (\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top)^{-1} \sum_{n=1}^{N} y_n \boldsymbol{x}_n = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

.. this has $O(D^3 + ND^2)$ cost

- GD for linear regression avoided the matrix inversion
- In general, cost of batch GD with N examples having D features: $O(ND)$
- SGD cost will be $O(D)$ or $O(BD)$ with mini-batch of size B
- There exist theoretical results on convergence rates of GD/SGD (beyond the scope)
    - GD will take $O\left(\frac{1}{\epsilon^2}\right)$ iterations reach ϵ-close solution, which is defined as

$$\mathcal{L}(\boldsymbol{w}^{(t)}) \leq \mathcal{L}(\boldsymbol{w}^{(opt)}) + \epsilon \qquad \text{(up to } \epsilon \text{ worse than optimal)}$$

Gradient Descent: Updates are "Corrective"

- The GD updates for the linear and logistic regression case look like

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

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta_t \sum_{n=1}^{N} (y_n - \mu_n^{(t)})\mathbf{x}_n$$

- These updates try to correct $\mathbf{w}$ by moving it in the right direction
- Consider the linear regression case and simplicity assume $N = 1$. Can verify (exercise)
  - If $\mathbf{w}^{(t)\top}\mathbf{x}_n < y_n$, the update will make $\mathbf{w}^{(t+1)\top}\mathbf{x}_n > \mathbf{w}^{(t)\top}\mathbf{x}_n$. Thus $\mathbf{w}$ moves more towards $\mathbf{x}_n$
  - If $\mathbf{w}^{(t)\top}\mathbf{x}_n > y_n$, the update will make $\mathbf{w}^{(t+1)\top}\mathbf{x}_n < \mathbf{w}^{(t)\top}\mathbf{x}_n$. Thus $\mathbf{w}$ moves away from $\mathbf{x}_n$
- Try the same for the logistic regression case (reason about it in terms of probabilities)

Some Other Considerations

- What if the function is not differentiable (e.g., loss function with ℓ_1 norm reg. on weights, or absolute loss function, or many other loss functions for classification models, such as SVM)?
  - One option is to use subgradient instead of gradient (subgradient descent)
- What if there are many variables, not just one (e.g., multi-output regression with $\mathbf{W} = \mathbf{BS}$)
  - One option is to use alternating optimization (optimize w.r.t. one, fixing all others, and cycle through)
- What if $\boldsymbol{w}$ has too many component: Can even optimize $\boldsymbol{w}$ co-ordinate wise (co-ordinate descent)
- What if we have an objective with constraints on variables, e.g.,

$$\hat{\boldsymbol{w}} = \arg\min_{||\boldsymbol{w}|| \leq c} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 \qquad \text{(constraint based regularization)}$$

  - Constrained optimization problem! One option is to use Lagrangian based optimization
- Can we use more than just gradient? Yes! (e.g., Newton's method uses the Hessian)
- Will look at these in the next class..