Optimization Techniques for ML (1)

Piyush Rai

Introduction to Machine Learning (CS771A)

August 23, 2018
Recap: Generative Classification

Class-Marginal or Class Prior

\[ p(y = k | \mathbf{x}) = \frac{p(y = k)p(\mathbf{x} | y = k)}{p(\mathbf{x})} \]

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

\[ p(\mathbf{x} | y = k) \text{ depends on type of } \mathbf{x} \]

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

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

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

\[ p(y = k|x) = \frac{p(y = k)p(x|y = k)}{p(x)} = \frac{\pi_k |\Sigma_k|^{-1/2} \exp \left[ -\frac{1}{2} (x - \mu_k)^\top \Sigma_k^{-1} (x - \mu_k) \right]}{\sum_{k=1}^{K} \pi_k |\Sigma_k|^{-1/2} \exp \left[ -\frac{1}{2} (x - \mu_k)^\top \Sigma_k^{-1} (x - \mu_k) \right]} \]

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

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

(a quadratic function of \( x \))

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

(reduces to the form \( w^\top x + b = 0 \))
Recap: Equivalence to Discriminative Model in Linear Case

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

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

- Expanding further, we can write the above as

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

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

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

where \( w_k = \Sigma^{-1} \mu_k \) and \( b_k = -\frac{1}{2} \mu_k^\top \Sigma^{-1} \mu_k + \log \pi_k \)

- Interestingly, this has exactly the same form as the **softmax classification model**
Recap: Equivalence to Prototype based Classification

- Again consider, generative classification with Gaussian class-conditionals

- Consider the prediction rule

\[
\hat{y} = \arg \max_k p(y = k | x) = \arg \max_k \pi_k \exp \left[ -\frac{1}{2} (x - \mu_k)^\top \Sigma_k^{-1} (x - \mu_k) \right]
\]

\[
= \arg \max_k \log \pi_k - \frac{1}{2} (x - \mu_k)^\top \Sigma_k^{-1} (x - \mu_k)
\]

- 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

\[
\hat{y} = \arg \min_k (x - \mu_k)^\top \Sigma^{-1} (x - \mu_k)
\]

- This is equivalent to assigning \( x \) to the “closest” class in terms of a Mahalanobis distance

  - The covariance matrix “modulates” how the distances are computed

- If we further assume \( \Sigma = I \), we get the exact same model as prototype based classification
Discriminative vs Generative: A Few Points

- Generative models are always probabilistic with models for $p(y)$ and $p(x|y)$
- Some discriminative models are also non-probabilistic
  - Any model of the form $y = f(x)$ with no model for $x$ is a discriminative model
  - Example: Support Vector Machines (SVM), DT, KNN, etc.

- Discriminative models are preferred when
  - There is plenty of training data. Modeling $x$ 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 $p(x|y)$)

- Generative and discriminative models can be combined as well
Optimization Techniques for ML
The generic form of most optimization problems in ML

\[
\hat{w} = \text{arg min}_w \mathcal{L}(w) = \text{arg min}_w \sum_{n=1}^{N} \ell_n(w) + R(w)
\]

\(\ell_n(w)\): loss function for the \(n^{th}\) training example, \(R(w)\): (optional) regularizer on the parameters

Some common examples

\[
\hat{w} = \text{arg min}_w \sum_{n=1}^{N} (y_n - w^T x_n)^2 + \lambda \|w\|_2^2
\]

\(\ell_2\) regularized logistic regression assuming \(y_n \in \{0,1\}\)

\[
\hat{w} = \text{arg min}_w - \sum_{n=1}^{N} (y_n w^T x_n - \log(1 + \exp(w^T x_n))) + \lambda \|w\|_2^2
\]

\(\ell_2\) regularized logistic regression assuming \(y_n \in \{-1,1\}\)

\[
\hat{W} = \text{arg min}_w \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{I}[y_n = k] \log \left[ \frac{\exp(w_k^T x_n)}{\sum_{k=1}^{K} \exp(w_k^T x_n)} \right]
\]

Softmax Regression with \(K\) classes (assuming no regularization)

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

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

![Simple Objective Function](image)

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

![Complex Objective Function](image)

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

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

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

.. 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**

![Convex Function vs. Non-convex Function](image)

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

![Convex Function](image)

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

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

Note: “Chord lies above function” more formally means

If $f$ is convex then given

\[
\alpha_1, \ldots, \alpha_n \text{ s.t. } \sum_{i=1}^{n} \alpha_i = 1
\]

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

**Jensen’s Inequality**
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 \( \mathbf{g} \) must be equal to zero at (each of) the optima

\[
\mathbf{g} = \nabla_w L(w) = \nabla_w \left[ \sum_{n=1}^{N} \ell_n(w) + R(w) \right] = 0
\]

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

1. Initialize $w$ as $w^{(0)}$
2. Update $w$ as follows
   \[ w^{(t+1)} = w^{(t)} - \eta_t g^{(t)} \]
3. Repeat until convergence
Gradient Descent

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

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

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

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

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

\[ w^{(t+1)} = w^{(t)} - \eta_t g^{(t)} \]
Gradient Descent

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

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

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

$$w^{(t+1)} = w^{(t)} - \eta_t g^{(t)}$$
A very simple, **first-order method** (uses only the gradient $g$ of the objective)

Basic idea: Start at some location $w^{(0)}$ and move in the **opposite direction** of the gradient

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

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

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

- Note: The effective step size (how much $w$ moves) depends on both $\eta_t$ and current gradient $g^{(t)}$
- A good initialization $w^{(0)}$ 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 $\eta_t$ 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
  \[ g = \nabla_w \left[ \sum_{n=1}^{N} \ell_n(w) \right] = \sum_{n=1}^{N} g_n \]
  .. will be very expensive when $N$ is very large
- A solution: Use stochastic gradient descent (SGD). Pick a random $i \in \{1, \ldots, N\}$
  \[ g \approx g_i = \nabla_w \ell_i(w) \]
- SGD updates use this approximation of the actual gradient

**Stochastic Gradient Descent**

1. Initialize $w$ as $w^{(0)}$
2. Pick a random $i \in \{1, \ldots, N\}$. Update $w$ as follows
   \[ w^{(t+1)} = w^{(t)} - \eta_i g_i^{(t)} \]
3. Repeat until convergence
(Stochastic) Gradient Descent

- SGD uses a single example to compute the gradient
- Can show that $\mathbb{E}[g_i] = g$. Therefore $g_i$ is an unbiased estimate of $g$ (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

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

This is known as mini-batch SGD
Gradient Descent: Some Simple Examples

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

  Linear Regression: \( \mathcal{L}(w) = \sum_{n=1}^{N} (y_n - w^T x_n)^2 \)

  Logistic Regression: \( \mathcal{L}(w) = -\sum_{n=1}^{N} (y_n w^T x_n - \log(1 + \exp(w^T x_n))) \) (assuming \( y_n \in \{0, 1\} \))

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

  Linear Regression: \( g = -\sum_{n=1}^{N} 2(y_n - w^T x_n)x_n \)

  Logistic Regression \( g = -\sum_{n=1}^{N} (y_n - \mu_n)x_n \) (where \( \mu_n = \sigma(w^T x_n) \))

- The GD updates in both cases will be of the form \( w^{(t+1)} = w^{(t)} - \eta_{t}g^{(t)} \)

- Note that highly mispredicted inputs \( x_n \) contribute more to \( g \) 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

\[ w = \left( \sum_{n=1}^{N} x_n x_n^\top \right)^{-1} \sum_{n=1}^{N} y_n x_n = (X^\top X)^{-1} X^\top y \]

.. 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 \( \epsilon \)-close solution, which is defined as

\[ \mathcal{L}(w^{(t)}) \leq \mathcal{L}(w^{(opt)}) + \epsilon \quad (\text{up to } \epsilon \text{ worse than optimal}) \]
Gradient Descent: Updates are “Corrective”

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

$$w^{(t+1)} = w^{(t)} + 2\eta_t \sum_{n=1}^{N} (y_n - w^{(t)^T} x_n)x_n$$

$$w^{(t+1)} = w^{(t)} + \eta_t \sum_{n=1}^{N} (y_n - \mu^{(t)}_n)x_n$$

- These updates try to correct $w$ by moving it in the right direction

- Consider the linear regression case and simplicity assume $N = 1$. Can verify (exercise)
  - If $w^{(t)^T} x_n < y_n$, the update will make $w^{(t+1)^T} x_n > w^{(t)^T} x_n$. Thus $w$ moves more towards $x_n$
  - If $w^{(t)^T} x_n > y_n$, the update will make $w^{(t+1)^T} x_n < w^{(t)^T} x_n$. Thus $w$ moves away from $x_n$

- 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 $\ell_1$ 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 $W = BS$)?
  - One option is to use alternating optimization (optimize w.r.t. one, fixing all others, and cycle through)
- What if $w$ has too many component: Can even optimize $w$ co-ordinate wise (co-ordinate descent)
- What if we have an objective with constraints on variables, e.g.,

$$\hat{w} = \arg \min_{||w|| \leq c} \sum_{n=1}^{N} (y_n - w^T x_n)^2$$

  - 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..