Optimization for Machine Learning
with a focus on proximal gradient descent algorithm

Quanming Yao

Department of Computer Science and Engineering
Outline

1. History & Trends
2. Proximal Gradient Descent
3. Three Applications
A Brief History

A. Convex optimization: before 1980
   - Develop more efficient algorithms for convex problems

   - Apply mature convex optimization algorithms on machine learning models

C. Large scale machine learning: 2000-2010
   - Convexity may not longer be required
   - Algorithms become more specific to machine learning models

D. Huge scale machine learning: 2010-now
   - Convexity is longer a problem for optimization
   - Parallel & distributed optimization
Convex Optimization: Before 1980

Linear least squares regression

$$
\min_x \frac{1}{2N} \sum_{n=1}^{N} (y_i - a_i^T x)^2 + \frac{\lambda}{2} \|x\|_2^2
$$

- Gradient descent
- Conjugate gradient descent
- Quasi-Newton method: L-BFGS
Support vector machine

\[
\min \frac{1}{N} \sum_{n=1}^{N} \epsilon_i + \frac{\lambda}{2} \|x\|_2^2,
\]

s.t. \[y_i \left( a_i^T x + b \right) \geq 1 - \epsilon_i\]

- Sequential minimal optimization
Large Scale Machine Learning

Support vector machine
- Core-set selection: *CVM*
- Coordinate descent: *LibLinear*
- Stochastic gradient descent (SGD): *Pegasos*

Sparse coding and matrix completion
- Proximal gradient descent (PG)
- Frank-Wolfe algorithm (FW)
- Alternating Direction Method of Multipliers (ADMM)

Neural networks
- Forward and Backward-propagation
Huge Scale Machine Learning: 2010-now

Deep Neural networks

- Tailored-made SGD (Resprop, Adam)
- Making algorithms (PG, FW, ADMM) more efficient & applicable
  - Parallel and distributed version
  - Stochastic version
  - Convergence analysis without convexity
Three Interesting Trends

Trends

- General propose to tailor-made for model, even for data
- Convex to nonconvex
- Easy implementation to systematic approach

Requirements

- Deep understanding of optimization algorithms
- Deep understanding of machine learning models
- Experienced with coding & computer systems
- Domain knowledge from specific areas
Why?

Limits of Our Model

Start Point

Error by Our Model

God Model

Error by Optimization

- Big model (nonconvex) and big data (distributed)
  - Smaller model error, but increase the difficulty on optimization which leads to large optimization error
- Good algorithms
  - Keep small optimization error
Gradient Descent

Minimization problem: \( \min F(x) \)
- \( F \) is a \textit{smooth} function
- we can take gradient \( \nabla F \)

Gradient descent

\[
x_{t+1} = x_t - \frac{1}{L} \nabla F(x_t)
\]

Example: logistic regression

\[
\min_x \frac{1}{N} \sum_{n=1}^{N} \log \left(1 + \exp \left(-y_i(a_i^\top x)\right)\right) + \frac{\lambda}{2} \|x\|_2^2
\]
Proximal Gradient Descent

$F$ is always not smooth in machine learnings.

$$\min F(x) = \underbrace{f(x)}_{\text{loss function}} + \underbrace{g(x)}_{\text{regularizer}}$$

- $f$: loss function
  - usually smooth: e.g. square loss, logistic loss
- $g$: regularizer
  - usually nonsmooth: e.g. sparse/low-rank regularizers

Examples: logistic regression + feature selection

$$\min_x \frac{1}{N} \sum_{n=1}^{N} \log \left(1 + \exp \left(-y_i(a_i^T x)\right)\right) + \lambda \|x\|_1$$
Matrix Completion: Recommender system

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>users</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \{1, 2, 3, 4\} = \text{YUCK! MEH GOOD AWESOME!} \]
Matrix Completion: Recommendation system

Birds of a feather flock together $\iff$ Rating matrix is low-rank

- $U$: users $\times$ latent factors
- $V$: items $\times$ latent factors
Matrix Completion: Recommender system

Optimization problem:

$$\min_X \frac{1}{2} \| P_\Omega (X - O) \|^2_F + \lambda \| X \|_*$$

where observed positions are indicated by $\Omega$, and ratings are in $O$.

- $f$: square loss is used
  - smooth
- $g$: the nuclear norm regularizer
  - $\| X \|_* = \sum_i \sigma_i(X)$, sum of singular values, nonsmooth
**Topic Model: Tree-structured lasso**

<table>
<thead>
<tr>
<th>Topics</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene</td>
<td>0.04</td>
</tr>
<tr>
<td>dna</td>
<td>0.02</td>
</tr>
<tr>
<td>genetic</td>
<td>0.01</td>
</tr>
<tr>
<td>life</td>
<td>0.02</td>
</tr>
<tr>
<td>evolve</td>
<td>0.01</td>
</tr>
<tr>
<td>organism</td>
<td>0.01</td>
</tr>
<tr>
<td>brain</td>
<td>0.04</td>
</tr>
<tr>
<td>neuron</td>
<td>0.02</td>
</tr>
<tr>
<td>nerve</td>
<td>0.01</td>
</tr>
<tr>
<td>data</td>
<td>0.02</td>
</tr>
<tr>
<td>number</td>
<td>0.02</td>
</tr>
<tr>
<td>computer</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**Documents**

**Seeking Life's Bare (Genetic) Necessities**

*COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here, two genome researchers with radically different approaches presented complimentary views of the basic genes needed for life.*

One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest lifeforms required a mere 128 genes. The other research team mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions are not all that far apart, especially in comparison to the 75,000 genes in the human genome, notes Sydney Brenner, a molecular biologist at the University of Cambridge, in Cambridge, England. But coming up with a consensus answer may be more than just a matter of numbers. Some, particularly more and more organisms are complexly mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arek Fishbein, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing all...


**Topic proportions and assignments**

SCIENCE • VOL. 272 • 24 MAY 1996

Quanming Yao

Optimization for Machine Learning
Topics naturally have a hierarchical structure
Topic Model: Tree-structured lasso

Bag of words (BoW) representation of a document

Group Structure

Feature

FW  PG  SGD  ADMM  Low-Rank
Optimization problems

\[
\min_x \frac{1}{2} \| y - Dx \|_2^2 + \lambda \sum_{g=1} \| x_{I_g} \|_2
\]

where \( y \) is the BoW representation of a document, and different groups are indicated by \( I_g \):

- \( f \): square loss is used - **smooth**
- \( g \): tree-structured lasso regularizer - **nonsmooth**
PGD Algorithm: Overview

Gradient Descent

\[ x_{t+1} = x_t - \frac{1}{L} \nabla f(x_t) \]

\[ = \arg \min_{x} \frac{1}{2} \| x - (x_t - \frac{1}{L} \nabla f(x_t)) \|_2^2 \]

\[ = \arg \min_{x} f(x_t) + (x - x_t)^\top \nabla f(x_t) + \frac{L}{2} \| x - x_t \|_2^2 \]

Proximal Gradient Descent

\[ x_{t+1} = \text{prox}_{\frac{\lambda}{L} g} \left( x_t - \frac{1}{L} \nabla f(x_t) \right) : \text{proximal step} \]

\[ = \arg \min_{x} \frac{1}{2} \| x - (x_t - \frac{1}{L} \nabla f(x_t)) \|_2^2 + \lambda g(x) \]

\[ = \arg \min_{x} f(x_t) + (x - x_t)^\top \nabla f(x_t) + \frac{L}{2} \| x - x_t \|_2^2 + \lambda g(x) \]

nonsmooth term
PGD Algorithm: Overview

Proximal step $\text{prox}_{\lambda \frac{1}{L}} g \left( \cdot \right)$

Cheap closed-form solution

- Fast $O(1/T^2)$ convergence in convex problems

\[ y_t = x_t + \frac{t - 1}{t + 2} (x_t - x_{t-1}), \]

\[ x_{t+1} = \text{prox}_{\lambda \frac{1}{L}} g \left( y_t - \frac{1}{L} \nabla f(y_t) \right). \]

- Sound convergence even when $f$ and $g$ are not convex
  - Not for ADMM, SGD and FW algorithms
PGD Algorithm: Matrix completion

Optimization problem: \( \min_X \frac{1}{2} \| P_\Omega (X - O) \|_F^2 + \lambda \| X \|_* \)

- Proximal step

\[
X_{t+1} = \text{prox}_{\frac{\lambda}{L} \| \cdot \|_*} (Z_t), \quad Z_t = X_t - P_\Omega (X_t - O)
\]

- SVD is needed, expensive for large matrices
  - \( O(m^2 n) \) for \( X \in \mathbb{R}^{m \times n} (m \leq n) \)

How to decrease iteration time complexity while guarantee convergence?
PGD Algorithm: Matrix completion

Key observation: \( Z_t \) is \( \text{sparse} + \text{low-rank} \)

\[
Z_t = P_\Omega(X_t - O) + X_t, \quad X_{t+1} = \text{prox}_{\lambda \| \cdot \|_*}(Z_t)
\]

- Let \( X_t = U_t \Sigma_t V_t^\top \). For any \( u \in \mathbb{R}^n \),

\[
Z_t u = P_\Omega(O - X_t)u + U_t \Sigma_t (V_t^\top u)
\]

- \( \text{Rank-}k \text{ SVD takes } O(\| \Omega \|_1 k + (m + n)k^2) \text{ time, instead of } O(mnk) \) (similarly, for \( Z_t^\top v \))
  - \( k \) is much smaller than \( m \) and \( n \)
  - \( \| \Omega \|_1 \) much smaller than \( mn \)

Use approximate SVD (power method) instead of exact SVD
Proposed **AIS-Impute** [IJCAI-2015] is in black

(a) MovieLens-100K.  
(b) MovieLens-10M.

Small datasets
PGD Algorithm: Parallelization


(a) Yahoo: 12 threads.  
(b) Speedup v.s. threads.

Large datasets
PGD Algorithm: Tree-structured lasso

Optimization problems: \( \min_x \frac{1}{2} \| y - Dx \|_2^2 + \lambda \sum_{g=1}^{G} \| x_{I_g} \|_2 \)

- Proximal step
  \[ x_{t+1} = \operatorname{prox}_{\lambda \frac{1}{L} \sum_{g=1}^{G} \| (z_t)_{I_g} \|_2^2} (z_t), \quad z_t = x_t - \frac{1}{L} D^\top (Dx_t - y) \]

- Cheap cheap-closed form exists
  - complex, but convex
  - one pass of all groups, \( O(\log(d)d) \) time for \( x \in \mathbb{R}^d \)

Performance of convex regularizers is not good enough.
### PGD Algorithm: Tree-structured lasso

#### Convex v.s. nonconvex regularizers

<table>
<thead>
<tr>
<th>regularizer</th>
<th>convex</th>
<th>nonconvex</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimization</td>
<td>😊 ✓</td>
<td>😞 ×</td>
</tr>
<tr>
<td>performance</td>
<td>😞 ×</td>
<td>😊 ✓</td>
</tr>
</tbody>
</table>

#### Examples regularizers

- GP: Geman penalty
- Laplace penalty
- LSP: log-sum penalty
- MCP: minimax concave penalty
- SCAD: smoothly clipped absolute deviation penalty
PGD Algorithm: Tree-structured lasso

Optimization problems

\[
\min_x \frac{1}{2}\|y - Dx\|^2 + \lambda \sum_{g=1}^{\cap} \kappa (\|x_{I_g}\|_2)
\]

where \(\kappa\) is the nonconvex penalty function

- **Proximal step**
  \[
  x_{t+1} = \text{prox}_{\lambda \sum_{g=1}^{\cap} \kappa (\|z_t_{I_g}\|_2)} (z_t), \quad z_t = x_t - \frac{1}{L} D^\top (Dx_t - y)
  \]

- **No closed-form solution**
  - Needs to be iteratively solved with other algorithms
  - \(O(T_p d)\) is required, \(T_p\) is number of iterations: expensive

Better performance, much less efficient optimization
PGD Algorithm: Tree-structured lasso

Problem transformation [ICML-2016]:

\[
\frac{1}{2} \| y - Dx \|_2^2 + \lambda \sum_{g=1} \left[ \kappa \left( \| x_{I_g} \|_2 \right) - \kappa_0 \| x_{I_g} \|_2 \right] + \lambda \kappa_0 \sum_{g=1} \| x_{I_g} \|_2
\]

\( \bar{f} \): augmented loss

\( \bar{g} \): convex regularizer

Move the nonconexity from regularizer to the loss

- augmented loss: \( \bar{f} \) is still smooth
- convex regularizer: \( \bar{g} \) is standard tree-structured regularizer
PGD Algorithm: Tree-structured lasso

Optimizing transformed problem: \( \min_x \bar{f}(x) + \lambda \kappa_0 \sum_{g=1} \| x_{I_g} \|_2 \)

- Proximal step

\[ x_{t+1} = \text{prox}_{\frac{\lambda \kappa_0}{L} \sum_{g=1} \| (z_t)_{I_g} \|_2} (z_t), \quad z_t = x_t - \frac{1}{L} \nabla \bar{f}(x) \]

- Cheap closed-form solution
- Same convergence guarantee on original/transform problems

Better performance, efficient optimization
PGD Algorithm: Tree-structured lasso

Table: Results on tree-structured group lasso.

<table>
<thead>
<tr>
<th></th>
<th>SCP</th>
<th>GIST</th>
<th>GD-PAN</th>
<th>nmAPG</th>
<th>N2C</th>
<th>FISTA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>accuracy (%)</strong></td>
<td>99.6±0.9</td>
<td>99.6±0.9</td>
<td>99.6±0.9</td>
<td>99.6±0.9</td>
<td>99.6±0.9</td>
<td>97.2±1.8</td>
</tr>
<tr>
<td><strong>sparsity (%)</strong></td>
<td>5.5±0.4</td>
<td>5.7±0.4</td>
<td>6.9±0.4</td>
<td>5.4±0.3</td>
<td>5.1±0.2</td>
<td>9.2±0.2</td>
</tr>
<tr>
<td><strong>CPU time (sec)</strong></td>
<td>7.1±1.6</td>
<td>50.0±8.1</td>
<td>14.2±2.6</td>
<td>3.8±0.4</td>
<td>1.9±0.3</td>
<td>1.0±0.4</td>
</tr>
</tbody>
</table>

(a) Objective v.s. CPU time.  
(b) Objective v.s. iterations.

**N2C** is PG algorithm on the transformed problem.
PGD Algorithm: Binary nets

Example: Small Alex-net
- convolution layers: 2M parameters
- fully connect layers: 60M parameters

Much larger (10x & 100x) networks are always prefer for big data
- Not available for mobile devices
Binary nets: $\min_x f(x) : \text{s.t. } x = \{\pm 1\}^n$.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST</th>
<th>CIFAR-10</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>No regularizer</td>
<td>1.30 ± 0.04%</td>
<td>10.64%</td>
<td>2.44%</td>
</tr>
<tr>
<td>BinaryConnect (det.)</td>
<td>1.29 ± 0.08%</td>
<td>9.90%</td>
<td>2.30%</td>
</tr>
<tr>
<td>BinaryConnect (stoch.)</td>
<td>1.18 ± 0.04%</td>
<td>8.27%</td>
<td>2.15%</td>
</tr>
<tr>
<td>50% Dropout</td>
<td>1.01 ± 0.04%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maxout Networks [29]</td>
<td>0.94%</td>
<td>11.68%</td>
<td>2.47%</td>
</tr>
<tr>
<td>Deep L2-SVM [30]</td>
<td>0.87%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network in Network [31]</td>
<td></td>
<td>10.41%</td>
<td>2.35%</td>
</tr>
<tr>
<td>DropConnect [21]</td>
<td></td>
<td></td>
<td>1.94%</td>
</tr>
<tr>
<td>Deeply-Supervised Nets [32]</td>
<td>9.78%</td>
<td>1.92%</td>
<td></td>
</tr>
</tbody>
</table>

Neural network with binary weights
- Float (32 bits) to binary (1 bit) - 32x compression
- Multiplication to addition - 4 times faster
- Binary constraint act as regularization - prevent overfitting
PGD Algorithm: Binary nets

During training

- weights are first binarized \( \hat{x}_t = \text{sign}(x_t) \);
- then \( \hat{x}_t \) propagates to next layer

Take \( x \in \{\pm 1\} \) as the regularizer \( g \), using PG

\[
x_{t+1} = \text{prox}_{x \in \{\pm 1\}} (x_t - \nabla f(x_t)) = \text{sign}(x_t - \nabla f(x_t))
\]

- loss is considered in binarization [ICLR-2016]
**PGD Algorithm: Binary nets**

**BPN** is proposed method, “loss-aware” is used with Adam

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th>CIFAR-10</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>(no binarization)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>full-precision</td>
<td>1.190</td>
<td>11.900</td>
<td>2.277</td>
</tr>
<tr>
<td>(binarize weights)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BinaryConnect</td>
<td>1.280</td>
<td>9.860</td>
<td>2.450</td>
</tr>
<tr>
<td>BWN</td>
<td>1.200</td>
<td>11.030</td>
<td>2.531</td>
</tr>
<tr>
<td>BPN</td>
<td>1.170</td>
<td>10.500</td>
<td>2.354</td>
</tr>
<tr>
<td>(binarize weights and activations)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNN</td>
<td>1.470</td>
<td>12.870</td>
<td>3.500</td>
</tr>
<tr>
<td>XNOR</td>
<td>1.450</td>
<td>12.370</td>
<td>3.580</td>
</tr>
<tr>
<td>BPN2</td>
<td>1.380</td>
<td>12.280</td>
<td>3.362</td>
</tr>
</tbody>
</table>
Thanks & QA