Large-scale learning for image classification

Zaid Harchaoui

CVML’13, July 2013
Large-scale supervised learning

Large-scale image classification
Let \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{1, \ldots, k\}\) be labelled training images

\[
\text{Minimize}_W \quad \lambda \Omega(W) + \frac{1}{n} \sum_{i=1}^{n} L(y_i, W^T x_i)
\]

Problem: minimizing such objectives in the large-scale setting

\[n \gg 1, \quad d \gg 1, \quad k \gg 1\]
Large-scale supervised learning

Large-scale image classification
Let \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{1, \ldots, k\}\) be labelled training images

\[
\text{Minimize} \quad \lambda \Omega(W) + \frac{1}{n} \sum_{i=1}^{n} L(y_i, W^T x_i)
\]

Problem : minimizing such objectives in the large-scale setting

\[
n \gg 1, \quad d \gg 1, \quad k \gg 1
\]
Machine learning cuboid
ImageNet dataset

- Large number of examples: \( n = 17 \text{ millions} \)
- Large feature size: \( d = 4 \times 10^3, \ldots, 2 \times 10^5 \)
- Large number of categories: \( k = 10,000 \)
Strategy

Most approaches boil down to a general "divide-and-conquer" strategy

Break the large learning problem into small and easy pieces
Machine learning cuboid
Decomposition principle

- Decomposition over examples: stochastic/incremental gradient descent
- Decomposition over features: (primal) regular coordinate descent
- Decomposition over categories: one-versus-rest strategy
- Decomposition over latent structure: atomic decomposition
Decomposition principle

- Decomposition over examples: stochastic/incremental gradient descent
- Decomposition over features: (primal) coordinate descent
- Decomposition over categories: one-versus-rest strategy
- Decomposition over latent structure: atomic decomposition
Decomposition over examples

Stochastic/incremental gradient descent

- Bru, 1890: algorithm to adjust a slant $\theta$ of cannon in order to obtain a specified range $r$ by trial and error, firing one shell after another

$$\theta_t = \theta_{t-1} - \frac{\gamma_0}{t} (r - r_t)$$

- Perceptron, Rosenblatt, 1957

$$w_t = w_{t-1} - \gamma_t(y_t \phi(x_t)) \quad \text{if } y_t \phi(x_t) \leq 0$$

$$= w_{t-1} \quad \text{otherwise}$$
Stochastic/incremental gradient descent

- Bru, 1890: algorithm to adjust a slant $\theta$ of cannon in order to obtain a specified range $r$ by trial and error
- Perceptron, Rosenblatt, 1957
- 60s-70s: extensions in learning, optimal control, and adaptive signal processing
- 80s-90s: extensions to non-convex learning problems
- see "Efficient backprop" in *Neural networks: Tricks of the trade*, LeCun et al., 1998, for wise advice and overview on sgd algorithms
Stochastic/incremental gradient descent

- **Initialize**: $W = 0$
- **Iterate**: pick an example $(x_t, y_t)$

$$W_{t+1} = W_t - \gamma_t \nabla W Q(W; x_t, y_t)$$

one example at a time

Why?

Where does these update rules come from?
Plain gradient descent

Plain gradient descent versus stochastic/incremental gradient descent

Grouping the regularization penalty and the empirical risk

$$\nabla_W J(W) = \frac{1}{n} \sum_{i=1}^{n} \{ n \lambda \Omega(W) + L(y_i, W^T x_i) \}$$
Plain gradient descent versus stochastic/incremental gradient descent

Grouping the regularization penalty and the empirical risk, and expanding the sum onto the examples

\[
\nabla_{\mathbf{w}} J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^{n} \{ n\lambda \Omega(\mathbf{W}) + L(y_i, \mathbf{W}^T \mathbf{x}_i) \}
\]

\[
= \nabla_{\mathbf{w}} \left\{ \frac{1}{n} \sum_{i=1}^{n} Q(\mathbf{W}; \mathbf{x}_i, y_i) \right\}
\]
Plain gradient descent

- **Initialize**: $W = 0$
- **Iterate**:

$$W_{t+1} = W_t - \gamma_t \nabla J(W)$$

$$= W_t - \gamma_t \nabla w \left\{ \frac{1}{n} \sum_{i=1}^{n} Q(W; x_i, y_i) \right\}$$
Decomposition over examples

Plain gradient descent

Plain gradient descent

- Initialize: \( W = 0 \)
- Iterate:

\[
W_{t+1} = W_t - \gamma_t \nabla J(W)
\]

\[
= W_t - \gamma_t \nabla \left\{ \frac{1}{n} \sum_{i=1}^{n} Q(W; x_i, y_i) \right\}
\]

Strengths and weaknesses

- Strength: robust to setting of step-size sequence (line-search)
- Weakness: demanding disk/memory requirements
Stochastic/incremental gradient descent

Leveraging the decomposable structure over examples

\[ \nabla_W J(W) = \frac{1}{n} \sum_{i=1}^{n} \nabla_W Q(W; x_i, y_i) \]

\[ = \frac{1}{n} \left\{ \nabla_W Q(W; x_1, y_1) + \cdots + \frac{1}{n} (\nabla_W Q(W; x_n, y_n) \right\} \]
Stochastic/incremental gradient descent

- Leveraging the decomposable structure over examples

\[ \nabla_W J(W) = \frac{1}{n} \left\{ \nabla_W Q(W; x_1, y_1) + \cdots + \nabla_W Q(W; x_n, y_n) \right\} \]

  \[ \text{cheap to compute} \]

- Make incremental gradient steps along \( Q(W; x_t, y_t) \) at each iteration \( t \), instead of full gradient steps along \( \nabla J(W) \) at each iteration
Stochastic/incremental gradient descent

- **Initialize**: $W = 0$
- **Iterate**: pick an example $(x_t, y_t)$

$$W_{t+1} = W_t - \gamma_t \nabla W Q(W; x_t, y_t)$$
Stochastic/incremental gradient descent

- **Initialize**: $W = 0$
- **Iterate**: pick an example $(x_t, y_t)$

$$W_{t+1} = W_t - \gamma_t \nabla W Q(W; x_t, y_t)$$

one example at a time

Strengths and weaknesses

- **Strength**: little disk requirements
- **Weakness**: may be sensitive to setting of step-size sequence
What’s "stochastic" in this algorithm?
Looking at the objective as a stochastic approximation of the expected training error

\[
\nabla_w J(W) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(W; x_i, y_i)
\]

\[
= \frac{1}{n} \left\{ \nabla_w Q(W; x_1, y_1) + \cdots + \frac{1}{n} \nabla_w Q(W; x_n, y_n) \right\}
\]
Decomposition over examples

Stochastic/incremental gradient descent

What’s "stochastic" in this algorithm?

\[
\nabla_{W} J(W) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{W} Q(W; x_i, y_i)
\approx E_{x,y}[\nabla_{W} Q(W; x, y)]
\]

Practical consequences

- Shuffle the examples before launching the algorithm, in case they form a correlated sequence
- Perform several passes/epochs over the training data, shuffling the examples before each pass/epoch
Mini-batch extensions

- Regular stochastic gradient descent: extreme decomposition strategy picking one example at a time
- Mini-batch extensions: decomposition onto mini-batches of size $B_t$ at iteration $t$

When to choose one or the other?

- Regular stochastic gradient descent converges for simple objectives with "moderate non-smoothness"
- For more sophisticated objectives, SGD does not converge, and mini-batch SGD is a must
Decomposition over examples

Theory digest

Fixed stepsize $\gamma_t \equiv \gamma \rightarrow$ stable convergence

Decreasing stepsize $\gamma_t = \frac{\gamma_0}{t+t_0} \rightarrow$ faster local convergence, with $\gamma_0$ and $t_0$ properly set

Note: stochastic gradient descent is an extreme decomposition strategy picking one example at a time

In practice

Pick a random batch of reasonable size, and find best pair $(\gamma_0, t_0)$ through cross-validation

Run stochastic gradient descent with sequence of decreasing stepsize $\gamma_t = \frac{\gamma_0}{t+t_0}$
Life is simpler in large-scale settings

- Shuffle the examples before launching the algorithm, and process the examples in a balanced manner w.r.t. the categories.
- Regularization through early stopping: perform only a few several passes/epochs over the training data, and stop when the accuracy on a held-out validation set does not increase anymore.
- Fixed step-size works fine: find best $\gamma$ through cross-validation on a small batch.
Stochastic/incremental gradient descent

Put the shoulder to the wheel

Let’s try it out!
Ridge regression

Training data: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}\)

Minimize \(\mathbf{w} \in \mathbb{R}^d\)
\[
\frac{\lambda}{2} \|\mathbf{w}\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} L(y_i, \mathbf{w}^T \mathbf{x}_i)
\]

Key calculations

\[
Q(\mathbf{w}; \mathbf{x}_i, y_i) = \frac{n\lambda}{2} \|\mathbf{w}\|_2^2 + (y_i - \mathbf{w}^T \mathbf{x}_i)^2
\]

\[
\nabla Q(\mathbf{w}; \mathbf{x}_i, y_i) = n\lambda \mathbf{w} + (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}
\]
Logistic regression

Training data: $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$

Minimize \[ \min_{w \in \mathbb{R}^d} \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} L(y_i, w^T x_i) \]

Key calculations

\[ Q(w; x_i, y_i) = \frac{n \lambda}{2} \|w\|^2 + \log (1 + \exp(-y_i w^T x_i)) \]

\[ \nabla Q(w; x_i, y_i) = n \lambda w + -\frac{1}{1 + \exp(y_i w^T x_i)} y_i x_i \]
Linear SVM with linear hinge loss

Training data: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}\)

Minimize \[
\min_{w \in \mathbb{R}^d} \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} L(y_i, w^T x_i)
\]

Key calculations

\[
Q(w; x_i, y_i) = \frac{n \lambda}{2} \|w\|^2 + \max(0, 1 - y_i w^T x_i)
\]

\[
\nabla Q(w; x_i, y_i) = \begin{cases} 
  n \lambda w - y_i x_i & \text{if } 1 - y_i x_i > 0 \\
  0 & \text{otherwise}
\end{cases}
\]
Linear SVM with linear hinge loss

Training data: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}\)

\[
\text{Minimize } \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} L(y_i, w^T x_i)
\]

Non-differentiable loss

- Rule: if \(Q(w; x, y)\) has a finite number of a non-differentiable points, then just make no update, and pick another example.
- Theoretical justification: the set of a non-differentiable points will have measure zero, and convergence guarantee is still valid

\[
\nabla E_{x,y}[Q(W; x, y)] = E_{x,y}[\nabla Q(W; x, y)].
\]
Decomposition over examples

A quick overview

Convergence guarantees

- Least-square loss: smooth $\rightarrow$ fast and stable convergence
- Logistic loss: smooth $\rightarrow$ fast and stable convergence
- Linear hinge loss: non-smooth $\rightarrow$ slower convergence

Take-home message: smooth loss is nicer
Machine learning cuboid
Decomposition principle

- Decomposition over examples: stochastic/incremental gradient descent
- Decomposition over categories: one-versus-rest strategy
- Decomposition over latent structure: atomic decomposition
Decomposition over categories: one-versus-rest strategy

Multi-class linear SVM with regular linear hinge loss

Training data: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{0, 1\}^k\)

\[
\min_{w \in \mathbb{R}^{d \times k}} \lambda \|w\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_i
\]

One-versus-rest reduction

- Turn original label \(y_i \in \{0, 1\}^k\) into binary label \(\tilde{y}_i \in \{-1, +1\}\)

\[
\text{BinaryHingeLoss}_i = \max(0, 1 - \tilde{y}_i w^T x_i)
\]

- Note: any loss could do, i.e. also the logistic loss
Multi-class linear SVM with regular linear hinge loss

Training data: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{0, 1\}^k\)

\[
\min_{w \in \mathbb{R}^{d \times k}} \sum_{\ell=1}^{k} \lambda_\ell \|w_\ell\|_2^2 + \frac{1}{n} \sum_{\ell=1}^{k} \sum_{i=1}^{n} \text{BinaryHingeLoss}_{i}^{y_i=\text{class } \ell}
\]

Decomposition over categories

Leverage decomposable structure over categories
Multi-class linear SVM with regular linear hinge loss

Training data: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{0, 1\}^k\)

\[
\begin{align*}
\min_{w_1 \in \mathbb{R}^d} & \quad \lambda_1 \|w_1\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_{i, y_i = \text{class } 1} \\
\ldots & \\
\min_{w_\ell \in \mathbb{R}^d} & \quad \lambda_\ell \|w_\ell\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_{i, y_i = \text{class } \ell} \\
\ldots & \\
\min_{w_k \in \mathbb{R}^d} & \quad \lambda_k \|w_k\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_{i, y_i = \text{class } k}
\end{align*}
\]
Multi-class through one-vs-rest

- Overall: simplest multi-class classification algorithm
- Computational strength: easy to optimize by decomposition over classes
- Statistical weakness: no universally consistent loss can be decomposable over classes (do we really care? we’ll see)
Multi-class through one-vs-rest

In practice
State-of-the-art performance using a balanced version of the binary loss, and learning the optimal imbalance $\beta$ through cross-validation

$$\text{Empirical risk} = \frac{\beta}{n^+} \sum_{i \in \text{positive examples}} \text{BinaryHingeLoss}_i + \frac{1 - \beta}{n^-} \sum_{i \in \text{negative examples}} \text{BinaryHingeLoss}_i$$
Other multi-class loss functions

- Multinomial logistic loss
- Crammer & Singer multi-class loss

\[ R_{\text{MUL}} = \frac{1}{n} \sum_{i=1}^{n} \left\{ \max_y (\Delta(y_i, y) + w_y^T x_i) - w_{y_i}^T x_i \right\} \]

The multi-class binary hinge loss is the only decomposable loss
More sophisticated losses

Loss functions tailored to optimize a convex surrogate of top-\(k\) accuracy

\[
\text{Accuracy}_{\text{top-}k} = \frac{\# \text{ images whose correct label lies in top-}k\text{ scores}}{\text{Total number of images}}
\]

- Ranking losses

\[
R_{\text{RNK}} = \frac{1}{n} \sum_{i=1}^{n} \sum_{y=1}^{k} \max (0, \Delta(y_i, y) - (w_{y_i}^T w_y)^T x_i)
\]

- Weighted ranking losses, and other variations

Yet to prove themselves compared to one-vs-rest with binary loss on real-world datasets
Decomposition over categories: one-versus-rest strategy

Multi-class with non-decomposable loss functions

<table>
<thead>
<tr>
<th></th>
<th>Sampling</th>
<th>Update</th>
</tr>
</thead>
</table>
| $R_{OVR}$ | Draw $(x_i, y_i)$ from $S$             | $\delta_i = 1$ if $L_{OVR}(x_i, y_i; w) > 0$, 0 otherwise.  
$w(t) = (1 - \eta_t)w^{(t-1)} + \eta_t \delta_i x_i y_i$ |
| $R_{MUL}$ | Draw $(x_i, y_i)$ from $S$             | $\bar{y} = \arg \max_y D(y_i, y) + w'_y x_i$ and $\delta_i = \begin{cases} 
1 & \text{if } \bar{y} \neq y_i \\
0 & \text{otherwise.}
\end{cases}$  
$w_y^{(t)} = \begin{cases} 
 w_y^{(t-1)}(1 - \eta_t) + \delta_i \eta_t x_i & \text{if } y = y_i \\
 w_y^{(t-1)}(1 - \eta_t) - \delta_i \eta_t x_i & \text{if } y = \bar{y} \\
 w_y^{(t-1)}(1 - \eta_t) & \text{otherwise.}
\end{cases}$ |
| $R_{RNK}$ | Draw $(x_i, y_i)$ from $S$  
Draw $\bar{y} \neq y_i$ from $Y$ | $\delta_i = 1$ if $L_{tri}(x_i, y_i, \bar{y}; w) > 0$, 0 otherwise.  
$w_y^{(t)} = \begin{cases} 
 w_y^{(t-1)}(1 - \eta_t) + \delta_i \eta_t x_i & \text{if } y = y_i \\
 w_y^{(t-1)}(1 - \eta_t) - \delta_i \eta_t x_i & \text{if } y = \bar{y} \\
 w_y^{(t-1)}(1 - \eta_t) & \text{otherwise.}
\end{cases}$ |
Experimental results

Datasets

<table>
<thead>
<tr>
<th></th>
<th>Total # of images</th>
<th># of classes</th>
<th>Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fungus</td>
<td>88K</td>
<td>134</td>
<td>44K train</td>
</tr>
<tr>
<td>Ungulate</td>
<td>183K</td>
<td>183</td>
<td>91.5K val</td>
</tr>
<tr>
<td>Vehicle</td>
<td>226K</td>
<td>262</td>
<td>113K test</td>
</tr>
<tr>
<td>ILSVRC10</td>
<td>1.4M</td>
<td>1,000</td>
<td>1.2M train</td>
</tr>
<tr>
<td>ImageNet10K</td>
<td>9M</td>
<td>10,184</td>
<td>4.5M val</td>
</tr>
</tbody>
</table>

Table: Datasets considered
Stochastic gradient descent is competitive with batch solvers

Average training time (in CPU seconds) on 3 fine-grained datasets

(a) w-OVR SVM : LibSVM vs SGD

<table>
<thead>
<tr>
<th></th>
<th>LibSVM (batch) / SGD (online)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fungus</td>
</tr>
<tr>
<td>10</td>
<td>12 / 7</td>
</tr>
<tr>
<td>25</td>
<td>95 / 16</td>
</tr>
<tr>
<td>50</td>
<td>441 / 38</td>
</tr>
<tr>
<td>100</td>
<td>1,346 / 71</td>
</tr>
</tbody>
</table>

(b) MUL SVM : SVM-light vs SGD

<table>
<thead>
<tr>
<th></th>
<th>SVM-light (batch) / SGD (online)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fungus</td>
</tr>
<tr>
<td>10</td>
<td>45 / 36</td>
</tr>
<tr>
<td>25</td>
<td>99 / 72</td>
</tr>
<tr>
<td>50</td>
<td>198 / 261</td>
</tr>
<tr>
<td>100</td>
<td>972 / 522</td>
</tr>
</tbody>
</table>
Superiority of one-vs-rest with *weighted* binary loss

Superiority of one-vs-rest with *weighted* binary loss over unweighted one

**Figure:** Influence of data rebalancing in weighted one-vs-rest (w-OVR) vs unweighted one-vs-rest (u-OVR) on Fungus (134 classes).
One-vs-rest with binary hinge-loss works fine for expressive features

Figure: Comparison of Top-1 Accuracy between the w-OVR, MUL, RNK and WAR SVMs as a function of the number of Gaussians used to compute the FV (i.e. as a function of the FV dimensionality). No spatial pyramids were used to speed-up these experiments.
Beyond one-vs-rest strategies

**Large-scale learning**

Training data: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathcal{Y} = \{0, 1\}^k\)

\[
\text{Minimize} \quad \lambda \Omega(W) + \frac{1}{n} \sum_{i=1}^{n} \text{Loss}_i
\]

- Discover latent structure of the classes
- **And** keep scalability and efficiency of one-versus-rest strategies
Machine learning cuboid
Decomposition principle

- Decomposition over examples: stochastic/incremental gradient descent
- Decomposition over latent structure: atomic decomposition
Learning with atom. penalty

Learning with low-rank regularization penalty
Training data : \((x_1, y_1), \ldots, (x_n, y_n)\) \(\mathbb{R}^d \times \mathcal{Y} = \{0, 1\}^k\)

\[
\min_{W \in \mathbb{R}^{d \times k}} \lambda \text{Rank}(W) + \frac{1}{n} \sum_{i=1}^{n} \text{Loss}_i
\]

- Embedding motivation : classes may embedded in a low-dimensional subspace of the feature space
- Computational motivation : algorithm scales with the number of latent classes \(r\), assuming that \(r \ll k\)
- Extension of Reduced-Rank Regression (see e.g. Velu, Reinsel, 1998) → non-smooth, non-convex optimization problem
Learning with atom. penalty

Learning with low-rank regularization penalty

Training data: \((x_1, y_1), \ldots, (x_n, y_n)\in \mathbb{R}^d \times \mathcal{Y} = \{0, 1\}^k\)

\[
\min_{W \in \mathbb{R}^{d \times k}} \lambda \|\sigma(W)\|_1 + \frac{1}{n} \sum_{i=1}^{n} \text{Loss}_i
\]

Embedding motivation: classes may be embedded in a low-dimensional subspace of the feature space.

Computational motivation: algorithm scales with the number of latent classes \(r\), assuming that \(r \ll k\).

Extension of Reduced-Rank Regression (see e.g. Velu, Reinsel, 1998) → non-smooth, non-convex optimization problem.
Learning with low-rank regularization penalty

Let \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{1, \ldots, k\}\) be labelled training images

\[
\text{Minimize} \quad \lambda \| \sigma(W) \|_1 + \frac{1}{n} \sum_{i=1}^{n} \text{Loss}_i
\]

- Tight convex relaxation (Amit et al., 2007; Argyriou et al., 2007)
- Enforces a low-rank structure of \(W\) (sparsity of spectrum \(\sigma(W)\))
- Convex, but non-differentiable
Learning with low-rank regularization penalty

Let \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{1, \ldots, k\}\) be labelled training images

\[
\text{Minimize} \quad \frac{\lambda}{n} \|\sigma(W)\|_1 + R_n(W)
\]

where \(R_n(W)\) is the empirical risk with the multinomial logistic loss

\[
R_n(W) = \frac{1}{n} \sum_{i=1}^{n} \log \left( 1 + \sum_{\ell \in \mathcal{Y}\setminus\{y_i\}} \exp \left\{ w_\ell^T x_i - w_{y_i}^T x_i \right\} \right)
\]
Learning with atom. penalty

Learning with low-rank regularization penalty
Let \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{1, \ldots, k\}\) be labelled training images

\[
\text{Minimize } \min_{W \in \mathbb{R}^{d \times k}} \lambda \|\sigma(W)\|_1 + R_n(W)
\]

where \(R_n(W)\) is the empirical risk with the multinomial logistic loss

\[
R_n(W) = \frac{1}{n} \sum_{i=1}^{n} \log \left( 1 + \sum_{\ell \in \mathcal{Y} \setminus \{y_i\}} \exp \{ w_\ell^T x_i - w_{y_i}^T x_i \} \right)
\]
Stochastic atom descent

We want an efficient and scalable algorithm
Let’s get inspiration from $\ell_1$ case...

Atom-descent algorithms

- Leverage a decomposable structure of regularization: atomic decomposition
- Perform a stochastic version of coordinate descent on this representation
- Efficient and scalable algorithms

Atom-descent for trace-norm regularization

- Leverage a non-flat decomposable structure, in contrast to one-vs-rest
- Learn a latent embedding of the classes
Lifting to an infinite-dimensional space

The trace-norm is the smallest $\ell_1$-norm of the weight vector associated with an atomic decomposition onto rank-one subspaces

\[
\|\sigma(W)\|_1 = \min_{\theta} \left\{ \|\theta\|_1, \exists N, M_i \in \mathcal{M}, \text{ with } W = \sum_{i=1}^{N} \theta_i M_i \right\}
\]

\[
\mathcal{M} = \{ uv^T \mid u \in \mathbb{R}^d, v \in \mathbb{R}^k, \|u\|_2 = \|v\|_2 = 1 \}
\]
Lifted objective

Lifting

- **Original objective**:
  \[
  J(W) := \lambda \|\sigma(W)\|_1 + R_n(W)
  \]

- **Lifted objective**:
  \[
  I(\theta) := \lambda \sum_{j \in \text{supp}(\theta)} \theta_j + R_n(W\theta)
  \]
Equivalence

Assume that the loss function $L(y, \cdot)$ is convex and smooth. Then the two problems are equivalent
**Sketch**

- At each iteration, pick a random mini-batch $B_t$, then pick the rank-1 subspace yielding the **steepest descent**, and perform descent along that direction.
- Periodically perform **second-order minimization** on current subspace.
Stochastic atom descent

Algorithm

- **Initialize**: $\theta = 0$
- **Iterate**: pick a random mini-batch $B_t$, find coordinate $\theta_i$ of steepest descent

\[
i(t) = \text{Arg max}_i \frac{\partial I_{B_t}(\theta)}{\partial \theta_i} = \text{Arg max}_i \langle u_i v_i^T, -\nabla R_{B_t}(W\theta) \rangle = \text{Arg max}_i u_i^T (-\nabla R_{B_t}(W\theta)) v.
\]

then perform a 1D line-search along $\theta_{i(t)}$

\[
(\theta_{t+1,1}, \cdots, \theta_{t+1,i(t)}, \cdots) \leftarrow (\theta_{t,1}, \cdots, \theta_{t,i(t)}, \cdots) + \delta(0, \cdots, 1, \cdots)
\]
Stochastic atom descent

Algorithm

- **Initialize**: $\theta = 0$
- **Iterate**: pick a random mini-batch $B_t$, find coordinate $\theta_i$ of steepest descent

\[
i(t) = \text{Arg max}_i \frac{\partial I_{B_t}(\theta)}{\partial \theta_i}
\]

\[
= \text{Arg max}_i \langle u_i v_i^T, -\nabla R_{B_t}(W\theta) \rangle
\]

\[
= \text{Arg max} \quad u^T \left( -\nabla R_{B_t}(W\theta) \right) v .
\]

then perform a 1D line-search along $\theta_{i(t)}$

\[
\theta_{t+1} \leftarrow \theta_t + \delta e_{t(i)}
\]
Stochastic atom descent

Algorithm

- Initialize: \( \theta = 0 \)
- Iterate: pick a random mini-batch \( B_t \), find coordinate \( \theta_i \) of steepest descent

Finding \( i(t) \) corresponds to finding top singular vectors \( u_1 \) and \( v_1 \) of \(-\nabla R_{B_t}(W\theta)\)

Runtime: \( O(dk) \) (few Power/Lanczos iter.)

Descend along \( \theta_{i(t)} \)

\[
\theta_{t+1} \leftarrow \theta_t + \delta e_{i(t)} \\
W_{t+1} \leftarrow W_t + \delta u_t v_t^T
\]

Periodically minimize \( I(\theta) \) over \( \text{supp}(\theta) \) up to optimality.
Algorithm

- **Initialize**: $\theta = 0$
- **Iterate**: pick a random mini-batch $B_t$, find coordinate $\theta_i$ of steepest descent

Find $i^*$ corresponds to find top singular vectors $u_1$ and $v_1$ of $R_{B_t}(W_\theta)$
then descend along $\theta_{i(t)}$

**Periodically** minimize $I(\theta)$ over $\text{supp}(\theta)$ up to optimality.

(quasi-)Newton method with box constraints
Subspace acceleration

Let $s = \text{supp}(\theta)$ be the size of the support of $\theta$ at iteration $t$. Coordinates of $\theta$ are re-ordered using indexes $j = 1, \ldots, s$.

$$
\begin{aligned}
\min_{\theta_1, \ldots, \theta_s} & \quad \lambda \sum_{j=1}^{s} \theta_j + R_{\text{emp}} \left( \sum_{j=1}^{s} \theta_j u_j v_j^T \right) \\
\text{subject to} & \quad \theta_j \geq 0, \quad j = 1, \ldots, s
\end{aligned}
$$

- convex and smooth objective with simple box constraint
- coordinate-descent works fine
- Quasi-Newton algorithm with box constraints (L-BFGS-B) works fine
Generalization to gauge regularization penalty

Properties

- $\Omega(tW) = t\Omega(W)$ for all $W$ and $t \geq 0$
- $\Omega(W + W') \leq \Omega(W) + \Omega(W')$ for all $W$ and $W'$.

Additional properties

Assuming $0 \in \text{int } B$, we also have

- $\Omega(W) \geq 0$, with equality if and only if $W = 0$
- $\{W : \Omega(W) \leq t\} = tB$ for $t \geq 0$, i.e., level sets are compact.

Polar duality

- Support function: $\Omega^\circ(G) := \sup_{M \in B} \langle M, G \rangle = \sup_{M \in M} \langle M, G \rangle$. 
Different types of atomic decomposition

\[ M_{\ell_1\text{-norm}} = \{ se_{j\ell} e^T \mid s \in \{-1, 1\} \]
\[ j \in \{1, \ldots, d\}, \ell \in \{1, \ldots, k\} \}\]

\[ M_{\ell_1/\ell_2\text{-norm}} = \{ e_j v^T \mid j \in \{1, \ldots, d\}, v \in \mathcal{R}^k, \|v\|_2 = 1 \}\]

\[ M_{\text{trace-norm}} = \{ uv^T \mid u \in \mathcal{R}^d, v \in \mathcal{R}^k, \|u\|_2 = \|v\|_2 = 1 \}\]

where \((e_1, \ldots, e_d)\) form the canonical basis of \(\mathcal{R}^d\).
Experimental results

Benchmark

- ImageNet dataset
- Subset of classes “Vehicles262”, “Fungus134”, and “Ungulate183”

Fisher vector image representation (Perronnin & Dance, 2007)

1. Extracted SIFT and local color descriptors reduced to 128 D
2. Train a Gaussian mixture model of 16 centroids → Fisher vectors of dim. 4096
3. Explicit embedding (Perronnin et al., 2010; Vedaldi & Zisserman, 2010)
Experimental results

Computations on each mini-batch

1. parallelized objective evaluation and gradient evaluation
2. efficient matrix computations for high-dimensional features

Efficient strategy: training with compression and testing without compression

1. product quantization of visual descriptors (Jegou et al., 2011)
2. during training, all matrix computations on features are performed in the compressed domain
3. for testing, all matrix computations are performed on uncompressed features
4. Note: compared to train/test without compression, average loss of performance only of 0.9\% on a subset of Vehicles with 10 categories.
Experimental results

Classification accuracy comparison

- Classification accuracy: top-$k$ accuracy, i.e.

\[
\text{Accuracy}_{\text{top-}k} = \frac{\text{# images whose correct label lies in top-k scores}}{\text{Total number of images}}
\]

- Competitors: our approach (TR-Multiclass) and $k$ independently trained one-vs-rest classifiers (OVR)
Experimental results

Cheatsheet

**OVR**
Minimize \( \min_{W \in \mathbb{R}^{d \times k}} \sum_{\ell=1}^{k} \lambda_{\ell} \| w_{\ell} \|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_{i} \)

**L2-Multiclass**
Minimize \( \min_{W \in \mathbb{R}^{d \times k}} \lambda \| W \|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{MultinomialLogisticLoss}_{i} \)

**TR-Multiclass**
Minimize \( \min_{W \in \mathbb{R}^{d \times k}} \lambda \| \sigma(W) \|_1 + \frac{1}{n} \sum_{i=1}^{n} \text{MultinomialLogisticLoss}_{i} \)
Experimental results

- **Fungus**
- **Ungulate**
- **Vehicle**

Graphs showing accuracy and top k accuracy for different datasets.
A posteriori low-dimensional embedding
Large-scale learning through decomposition

- Stochastic gradient descent is a decomposition over examples
- One-vs-rest is a decomposition over categories
- Stochastic atom descent is a decomposition over latent structure
Conclusion

Be your own cook

- Mix these decompositions and come up with your own algorithms for new problems
- Implement your own codes and master the algorithm before using an off-the-shelf implementation

Public code

- Download it and start running your own large-scale experiments
- Joust SGD: lear.inrialpes.fr/software
Recent references

- *Efficient additive kernels via explicit feature maps*, A. Vedaldi, A. Zisserman, CVPR 2010