Large-scale learning for visual recognition

Zaid Harchaoui

CVML 2012, June 11th 2012
Large-scale image datasets

From "The Promise and Perils of Benchmark Datasets and Challenges", D. Forsyth, A. Efros, F.-F. Li, A. Torralba and A. Zisserman, Talk at "Frontiers of Computer Vision"
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

\[n \gg 1, \quad d \gg 1, \quad k \gg 1\]
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

\[n \gg 1, \quad d \gg 1, \quad k \gg 1\]
Working example: ImageNet dataset

ImageNet dataset

- Large number of examples: $n = 17 \text{ millions}$
- Large feature size: $d = 4 \cdot 10^3, \ldots, 2 \cdot 10^5$
- Large number of categories: $k = 10,000$
General strategy for large-scale problems

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

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

$$= w_{t-1} \quad \text{otherwise}$$
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
- 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

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

$$\nabla W J(W) = \frac{1}{n} \sum_{i=1}^{n} \left\{ n \lambda \Omega(W) + L(y_i, W^T x_i) \right\}$$

$$= \nabla W \left\{ \frac{1}{n} \sum_{i=1}^{n} Q(W; x_i, y_i) \right\}$$
Plain gradient descent

- Initialize: $W = 0$
- Iterate:

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

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

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

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

- **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

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

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

Stochastic/incremental gradient descent

- Leveraging the decomposable structure over examples

\[ \nabla_{\mathbf{W}} J(\mathbf{W}) = \frac{1}{n} \left\{ \nabla_{\mathbf{W}} Q(\mathbf{W}; \mathbf{x}_1, \mathbf{y}_1) + \cdots + \nabla_{\mathbf{W}} Q(\mathbf{W}; \mathbf{x}_n, \mathbf{y}_n) \right\} \]

  cheap to compute  
  cheap to compute

- Make incremental gradient steps along \( Q(\mathbf{W}; \mathbf{x}_t, \mathbf{y}_t) \) at each iteration \( t \), instead of full gradient steps along \( \nabla J(\mathbf{W}) \) at each iteration
Stochastic/incremental gradient descent

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

\[
W_{t+1} = W_t - \gamma_t \nabla_w Q(W; x_t, y_t)
\]
Decomposition over examples

Stochastic/incremental gradient descent

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

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

Strengths and weaknesses

- **Strength**: little disk requirements
- **Weakness**: may be sensitive to setting of step-size sequence
Decomposition over examples

Stochastic/incremental gradient descent

What’s "stochastic" in this algorithm?

Looking at the objective as a stochastic approximation of the expected training error

\[
\nabla \mathbf{w} J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^{n} \nabla \mathbf{w} Q(\mathbf{W}; \mathbf{x}_i, y_i) \\
= \frac{1}{n} \left\{ \nabla \mathbf{w} Q(\mathbf{W}; \mathbf{x}_1, y_1) + \cdots + \frac{1}{n} (\nabla \mathbf{w} Q(\mathbf{W}; \mathbf{x}_n, y_n) \right\}
\]
Stochastic/incremental gradient descent

What’s "stochastic" in this algorithm?

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

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
**Theory digest**

- **Fixed stepsize** \( \gamma_t \equiv \gamma \rightarrow \text{stable convergence} \)

- **Decreasing stepsize** \( \gamma_t = \frac{\gamma_0}{t+t_0} \rightarrow \text{faster local convergence, with } \gamma_0 \text{ and } t_0 \text{ properly set} \)

- 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

\[
\gamma_t = \frac{\gamma_0}{t+t_0}
\]
Tricks of the trade: life is simpler in large-scale settings

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.
- Fixed step-size works fine: find best $\gamma$ through cross-validation on a small batch.
Stochastic/incremental gradient descent

Now some exercises

Try it out!
Ridge regression

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

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

Key calculations

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

\[\nabla Q(\mathbf{w}; x_i, y_i) = ?\]
Ridge regression

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

Minimize \(w \in \mathbb{R}^d\)

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

Key calculations

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

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

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

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

Key calculations

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

\[ \nabla Q(w; x_i, y_i) = ? \]
Logistic regression

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

Minimize

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

Key calculations

\[
Q(\mathbf{w}; \mathbf{x}_i, y_i) = \frac{n\lambda}{2} \| \mathbf{w} \|_2^2 + \log \left(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)\right)
\]

\[
\nabla Q(\mathbf{w}; \mathbf{x}_i, y_i) = n\lambda \mathbf{w} + \frac{1}{1 + \exp(y_i \mathbf{w}^T \mathbf{x}_i)} y_i \mathbf{x}_i
\]
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

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

Key calculations

\[
Q(\mathbf{w}; \mathbf{x}_i, y_i) = \frac{n\lambda}{2} \|\mathbf{w}\|_2^2 + \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)
\]

\[
\nabla Q(\mathbf{w}; \mathbf{x}_i, y_i) = \begin{cases} 
 n\lambda \mathbf{w} - y_i \mathbf{x}_i & \text{if } 1 - y_i \mathbf{x}_i > 0 \\
 0 & \text{otherwise}
\end{cases}
\]
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

\[
\nabla E_{x,y}[Q(W; x, y)] = E_{x,y}[\nabla Q(W; x, y)].
\]
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
Multi-class linear SVM with regular linear hinge loss

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

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

One-versus-rest reduction

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

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

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

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

\[
\min_{w \in \mathbb{R}^{d \times k}} \sum_{\ell=1}^{k} \lambda_{\ell} \|w_{\ell}\|_2^2 + \frac{1}{n} \sum_{\ell=1}^{k} \sum_{i=1}^{n} \text{BinaryHingeLoss}_{i} \quad \text{subject to} \quad \sum_{\ell=1}^{k} w_{\ell} = 0
\]

Decomposition over categories

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

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

\[
\begin{align*}
\min_{w_1 \in \mathbb{R}^d} & \quad \lambda_1 \|w_1\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_i \\
\min_{w_2 \in \mathbb{R}^d} & \quad \lambda_2 \|w_2\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_i \\
\min_{w_k \in \mathbb{R}^d} & \quad \lambda_k \|w_k\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_i
\end{align*}
\]
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

\[
\text{Empirical risk} = \frac{\beta}{n^+} \sum_{i \in \text{positive examples}} \text{BinaryHingeLoss}_i + \frac{1 - \beta}{n^-} \sum_{i \in \text{negative examples}} \text{BinaryHingeLoss}_i
\]
Multi-class through one-vs-rest

Other multi-class loss functions

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

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
- Weighted ranking losses
- and many others

Yet to prove themselves compared to one-vs-rest with binary loss on real-world datasets
Experimental results

Datasets

<table>
<thead>
<tr>
<th></th>
<th>Total # of</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>images</td>
<td>classes</td>
<td>train</td>
<td>val</td>
</tr>
<tr>
<td>Fungus</td>
<td>88K</td>
<td>134</td>
<td>44K</td>
<td>5K</td>
</tr>
<tr>
<td>Ungulate</td>
<td>183K</td>
<td>183</td>
<td>91.5K</td>
<td>5K</td>
</tr>
<tr>
<td>Vehicle</td>
<td>226K</td>
<td>262</td>
<td>113K</td>
<td>5K</td>
</tr>
<tr>
<td>ILSVRC10</td>
<td>1.4M</td>
<td>1,000</td>
<td>1.2M</td>
<td>50K</td>
</tr>
<tr>
<td>ImageNet10K</td>
<td>9M</td>
<td>10,184</td>
<td>4.5M</td>
<td>50K</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

<table>
<thead>
<tr>
<th></th>
<th><strong>LibSVM (batch) / SGD (online)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Fungus</strong></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>

<table>
<thead>
<tr>
<th></th>
<th><strong>SVM-light (batch) / SGD (online)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Fungus</strong></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).
Beyond one-vs-rest strategies

Large-scale learning

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

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

- Discover latent structure of the classes
- And keep scalability and efficiency of one-versus-rest strategies
Learning with atom. penalty

Learning with low-rank regularization penalty

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

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

- Embedding motivation: classes may 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 atom. penalty

Learning with low-rank regularization penalty

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

\[
\begin{align*}
\text{Minimize} & \quad \lambda \| \sigma(W) \|_1 + \frac{1}{n} \sum_{i=1}^{n} \text{Loss}_i \\
\text{subject to} & \quad W \in \mathbb{R}^{d \times k}
\end{align*}
\]

- 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

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

- 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 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

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

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

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

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

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

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

$$\mathcal{M} = \{uv^T | u \in \mathbb{R}^d, v \in \mathbb{R}^k, \|u\|_2 = \|v\|_2 = 1\}$$
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.

\[ I(\theta^*) = J(W^*) \]
Atom-descent: high-level idea

Sketch

- At each iteration, pick the rank-1 subspace yielding the steepest descent, and perform descent along that direction
- Periodically perform second-order minimization on current subspace

\[
W = \theta_1 + \cdots + \theta_i + \cdots
\]
Stochastic atom descent

Algorithm

- **Initialize**: $\theta = 0$
- **Iterate**: find coordinate $\theta_i$ of steepest descent

\[
i^* = \text{Arg max}_i \frac{\partial I(\theta)}{\partial \theta_i} = \text{Arg max}_i \langle u_i v_i^T, -\nabla R(W_\theta) \rangle = \text{Arg max} \frac{u^T}{\|u\|=\|v\|=1} (-\nabla R(W_\theta)) \frac{v}{v}.
\]

then descend along $\theta_{i^*}$

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

Algorithm

- **Initialize**: $\theta = 0$
- **Iterate**: find coordinate $\theta_i$ of steepest descent

Finding $i^*$ corresponds to finding top singular vectors $u_1$ and $v_1$ of $-\nabla R(W_\theta)$

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

Descend along $\theta_i^*$

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

- **Initialize**: $\theta = 0$
- **Iterate**: find coordinate $\theta_i$ of steepest descent

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

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

(quasi-)Newton method with box constraints
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**  \[
\min_{W \in \mathbb{R}^{d \times k}} \sum_{\ell=1}^{k} \lambda_{\ell} \left\| w_{\ell} \right\|_{2}^{2} + \frac{1}{n} \sum_{i=1}^{n} \text{BinaryHingeLoss}_{i}
\]

**L2-Multiclass**  \[
\min_{W \in \mathbb{R}^{d \times k}} \lambda \left\| W \right\|_{2}^{2} + \frac{1}{n} \sum_{i=1}^{n} \text{MultinomialLogisticLoss}_{i}
\]

**TR-Multiclass**  \[
\min_{W \in \mathbb{R}^{d \times k}} \lambda \left\| \sigma(W) \right\|_{1} + \frac{1}{n} \sum_{i=1}^{n} \text{MultinomialLogisticLoss}_{i}
\]
Experimental results
A posteriori low-dimensional embedding
Conclusion

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

- Joust SGD: available at lear.inrialpes.fr/software
Recent references

- Large-scale classification with trace-norm regularization, Z. Harchaoui, M. Douze, M. Paulin, J. Malick, CVPR 2012
- Improving the Fisher kernel for large-scale image classification, F. Perronnin, J. Sanchez, T. Mensink, ECCV 2010
- Efficient additive kernels via explicit feature maps, A. Vedaldi, A. Zisserman, CVPR 2010
General references