Large-scale visual recognition
Large-scale learning

Florent Perronnin, XRCE
Hervé Jégou, INRIA

CVPR tutorial on Large-Scale Visual Recognition
June 16, 2012
Motivation

Most popular image classification pipeline: **BOV + SVMs**

→ See results of PASCAL VOC challenge
http://pascallin.ecs.soton.ac.uk/challenges/VOC/

But can we scale SVMs to a dataset such as ImageNet…

... at a reasonable computational cost?
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding

An embedding view of the BOV and FV

Putting everything together
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding

An embedding view of the BOV and FV

Putting everything together
Learning with SGD
Acknowledgement

The following slides are largely based on material available on Léon Bottou’s webpage:
http://leon.bottou.org/projects/sgd

See also the excellent code available on the same webpage
Learning with SGD

Historical note

Stochastic Gradient Descent (SGD) was the standard for back-propagation training of Multi-Layer Perceptrons (MLP)

With the rise of large-margin classification algorithms (e.g. SVMs), MLPs fell somewhat into disuse… and so did SGD

Then, it was showed that SGD can be applied to large-margin classifiers with excellent properties
Shalev-Swartz, Singer, Srebro, “Pegasos: Primal Estimated sub-GrAdient Solver for SVM”, ICML’07.
Shalev-Swartz, Srebro, “SVM optimization: inverse dependence on training set size”, ICML’08.

→ Today very popular for large-scale learning
Learning with SGD

Objective functions

We consider objective functions of the form:

$$ C(w) = \frac{1}{N} \sum_{i=1}^{N} Q(z_i; w) $$

- the $z_i$'s correspond to the training samples
- $w$ is the set of parameters to be learned

Linear two-class SVM example (primal formulation):

$$ \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i; w) + \frac{\lambda}{2} ||w||^2 $$

with $\ell(x_i, y_i; w) = \max\{0, 1 - y_i w' x_i\}$

$$ Q(z_i; w) = \ell(x_i, y_i; w) + \frac{\lambda}{2} ||w||^2 $$
Learning with SGD
Batch Gradient Descent (BGD)

Iterative optimization algorithm:

\[ w_{t+1} = w_t - \eta_t \nabla_{w=w_t} C(w) \]

\[ = w_t - \eta_t \frac{1}{N} \sum_{i=1}^{N} \nabla_{w=w_t} Q(z_i; w) \]

where \( \eta_t \) is the step size (e.g. line search)

- assumes all samples are available in a large batch
- disk / memory requirements in O(N)
- slow convergence (more later)
Learning with SGD
Stochastic Gradient Descent (SGD)

Iterative optimization algorithm:

- pick a random sample $z_t$

- (noisy) update: $w_{t+1} = w_t - \eta_t \nabla_w Q(z_t; w)$

where $\eta_t$ is the step size (e.g. fixed schedule)

😊 disk/memory requirements in $O(1)$
😊 can handle new incoming samples
😊 faster convergence
Learning with SGD
Convergence: BGD vs SGD

Imagine you have 100 training samples and that you copy them 10 times
→ $10 \times 100 = 1,000$ training samples

If we do one pass over the data:

- BGD: gradient with 100 samples = gradient with 1,000 samples
  → computational cost x10 for the exact same result
- SGD: better results with the 1,000 samples than with 100 samples

→ SGD exploits the redundancy in the training data
Learning with SGD
Choosing the step size

To guarantee convergence to the optimum:

- $\eta_t$ should decrease fast enough: $\sum_{t=1}^{\infty} \eta_t^2 < \infty$
- $\ldots$ but not too fast: $\sum_{t=1}^{\infty} \eta_t = \infty$

$\Rightarrow$ important to tune $(a, t_0)$ correctly

However, fixed schedule $\eta_t = \eta$ can also work well in practice

Learning with SGD
Linear two-class SVM example

We have \( Q(z_t; w) = \ell(x_i, y_i; w) + \frac{\lambda}{2} \|w\|^2 \)

with \( \ell(x_i, y_i; w) = \max\{0, 1 - y_i w' x_i\} \)

Iteratively:
- pick a random sample \( z_t = (x_t, y_t) \)
- update: \( w_{t+1} = (1 - \eta_t \lambda)w_t + \delta_t \eta_t y_t x_t \)
  with \( \delta_t = 1 \) if \( \ell(x_t, y_t; w) > 0 \), 0 otherwise

→ extremely simple to implement: few lines of code
Learning with SGD
Beyond one-vs-rest

SGD is applicable to many optimization problems including:
  • K-means clustering
  • metric learning
  • learning to rank
  • multiclass learning
  • structured learning
  • etc.
Learning with SGD

Extensions

Mini batch SGD: sample K training samples
- if K=1 → traditional SGD setting
- If K=N → BGD

Second-order SGD: $w_{t+1} = w_t - \eta_t H_t \nabla_{w=w_t} Q(z_t; w)$
where $H_t$ is an approximation of the Hessian
Bordes, Bottou, Galinari, “SGD-QN: careful quasi-Newton stochastic gradient descent”, JMLR’09

Averaging SGD: $\bar{w}_t = \frac{1}{t} \sum_{i=1}^{t} w_i$
Xu, “Towards optimal one pass large-scale learning with averaged stochastic gradient descent”, arXiv, 2011.
Learning with SGD

Summary

Very generic principle applicable to many algorithms

Very simple to implement…

… but getting the step-size correct is not always trivial

Other online learning algorithms are closely related to SGD:

e.g. Passive-Aggressive algorithms

Crammer, Dekel, Keshet, Shalev-Swartz, Singer, “Online passive-aggressive algorithms”, JMLR’06.

Best practices for large-scale learning of image classifiers:


→ a key conclusion: one-vs-rest is a competitive strategy with respect to more complex multiclass / ranking SVM formulations

→ see also code available at http://lear.inrialpes.fr/src/jsgd/
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding

An embedding view of the BOV and FV

Putting everything together
Explicit embedding
Kernels for the BOV

Consider two families of non-linear kernels on BOV:

\[ K^{exp}(x, z) = \exp(\gamma (K(x, z) - 1)) \]

**Additive Kernels:**

\[ K_{bha}(a, b) = \sum_{i=1}^{N} \sqrt{a_i b_i}, \]

\[ K_{chi2}(a, b) = 2 \sum_{i=1}^{N} \frac{a_i b_i}{a_i + b_i}, \]

\[ K_{int}(a, b) = \sum_{i=1}^{N} \min(a_i, b_i). \]

**Exponential Kernels:**

\[ K^{exp}_{bha}(a, b) = \exp \left( -\frac{\gamma}{2} \sum_{i=1}^{N} (\sqrt{a_i} - \sqrt{b_i})^2 \right), \]

\[ K^{exp}_{chi2}(a, b) = \exp \left( -\frac{\gamma}{2} \sum_{i=1}^{N} \frac{(a_i - b_i)^2}{a_i + b_i} \right), \]

\[ K^{exp}_{int}(a, b) = \exp \left( -\gamma \sum_{i=1}^{N} |a_i - b_i| \right). \]
Explicit embedding
Non-linear vs linear

We have just discussed how to train efficiently linear classifiers…

… Yet non-linear classifiers perform significantly better on the BOV:

- dense SIFT
- codebook with 4k visual words
- [1]: Léon Bottou’s SGD
- [2]: Chang and Lin’s LIBSVM


Can we get the **accuracy of non-linear at the cost of linear**?
Explicit embedding
SVM formulations

**Dual formulation:**
- Smiley: avoids explicit mapping of data in high-dimensional space
- Frown: training cost (e.g. SMO): between quadratic and cubic in # training samples
- Frown: runtime cost: linear in # support vectors

→ Difficult to scale to very large image sets (e.g. ImageNet)

**Primal formulation:**
- Frown: requires explicit embedding of data in higher dimensional space
- Smiley: training cost (e.g. SGD): linear in # training samples
- Smiley: runtime cost: independent of # support vectors

→ Is explicit embedding such a high price to pay in large-scale?
Explicit embedding
Primal formulation

Positive definite kernel corresponds to implicit embedding:

\[ K(x, z) = \varphi(x)'\varphi(z) \]

The score becomes:

\[ f(z) = \sum_{i=1}^{N} a_i K(z, x_i) + b \]

\[ = \left( \sum_{i=1}^{N} \alpha_i \varphi(x_i) \right)' \varphi(z) + b \]

non-linear classification in the original space
= linear classification in the embedding space
Explicit embedding

Principle

Non-linear classification with explicit embedding:
- perform an explicit (approximate) embedding of the data with \( \varphi \)
- learn linear classifiers in the new space (e.g. with efficient SGD)

Our focus: given a kernel, find \( \varphi \)

Generally a two-step procedure:
- find exact embedding function
- find good (i.e. accurate and efficient) approximation
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding:
  • general case
  • additive kernels
  • exponential kernels

An embedding view of the BOV and FV

Putting everything together
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding:
  • general case
  • additive kernels
  • exponential kernels

An embedding view of the BOV and FV

Putting everything together
Explicit embedding

A generic solution: kPCA

Problem: given an (unlabeled) training set \( \{ x_i, i = 1 \ldots M \} \) with \((M>E)\)
find: \( \varphi : \mathbb{R}^D \rightarrow \mathbb{R}^E \)

Kernel principal Component Analysis (kPCA):
- compute the MxM kernel matrix
- compute eigen-values / -vectors: \( K \psi_e = \sigma_e^2 \psi_e \)

Embed a vector \( z \) with the Nyström approximation:
- project \( z \): \( K(z, :) = [K(z, x_1), \ldots, K(z, x_M)] \)
- embed: \( \varphi_e(z) = \frac{K(z, :) \psi_e}{\sigma_e} \)

Schölkopf, Smola and Müller, “Non-linear component analysis as a kernel eigenvalue problem”, Neural Computation’98.
Williams and Seeger, “Using the Nyström method to speed-up kernel machines”, NIPS’02.
Explicit embedding
A generic solution: kPCA

Embedding cost:
- kernel matrix computation: $O(M^2D)$
- eigen-decomposition (incomplete Cholesky): $O(M^2E)$
- embedding one sample: $O(M(D+E))$

If we have $N$ labeled training samples to learn our SVMs:
- kPCA embedding cost at training time: $O(M(M+N)(D+E))$
- to be compared to direct kernel computation: $O(N^2D)$

😊 works well if data lies in low-dimensional manifold (e.g. faces, digits)
보호 does not work so well on complex high-dimensional visual data

→ Find specific solution for additive and exponential kernels
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding:
  • general case
  • additive kernels
  • exponential kernels

An embedding view of the BOV and FV
Explicit embedding

Additive kernels

Kernels of the form: $K(x, z) = \sum_{d=1}^{D} k(x(d), z(d))$

Score becomes: $f(z) = \sum_{i=1}^{N} a_i K(z, x_i) + b = \sum_{d=1}^{D} f_d(z(d)) + b$

with $f_d(z(d)) = \sum_{i=1}^{N} a_i k(z(d), x_i(d))$

→ Generalized Additive Model (GAM)


$f_d$ can be approximated by piecewise constant/linear functions.

→ 1D functions enable efficient classification

Maji, Berg and Malik, “Classification using intersection kernel support vector machines is efficient”, CVPR’08.
Explicit embedding
Additive kernels

Embedding 1D functions: find $\varphi_d : \mathbb{R} \rightarrow \mathbb{R}^E$

such that $k(x(d), z(d)) = \varphi_d(x(d))' \varphi_d(z(d))$

The final embedding function is given by:

$$\varphi(x) = [\varphi_1(x(1)) \ldots \varphi_D(x(D))]$$

Maji and Berg, “Max-margin additive classifiers for detection”, ICCV’09.
Explicit embedding
Bhattacharya / Hellinger kernel

\[ K_{bha}(a, b) = \sum \sqrt{a_i b_i} \]

Explicit mapping is trivial and exact:
\[ \varphi(z) = \sqrt{z} \]


Intuitive interpretation: reduce effect of bursty visual words

Theoretical interpretation: variance stabilizing transform
Winn, Criminisi and Minka, “Object categorization by learned universal visual vocabulary”, ICCV’05.
Explicit embedding

Intersection kernel \( K_{int}(a, b) = \sum \min(a_i, b_i) \)

Introducing the indicator function: 
\[
1_{(a,b)}(t) = \begin{cases} 
1 & \text{if } a \leq t \leq b \\
0 & \text{otherwise}
\end{cases}
\]

we have: 
\[
\min(x, z) = \int_{t=0}^{\infty} 1_{(0,x)}(t)1_{(0,z)}(t)dt
\]

How to approximate the \( \infty \) dimensional feature map with a finite one?  
\( \rightarrow \) replace integral by sum
Explicit embedding

Intersection kernel \( K_{int}(a, b) = \sum \min(a_i, b_i) \)

Split \([0,1]\) into \(N\) bins (e.g. \(N=10\)):

Introducing notations: \(U_1(t) = [111 \ldots 1, 000 \ldots 0]^{\times t} \times (N-t)\) and \(R(t) = [t + 0.5]\)

We have: \(\varphi_1(t) = \frac{U_1(R(Nt))}{\sqrt{N}}\)

\[
\begin{align*}
\min(0.33, 0.54) &\approx \varphi_1(0.33)'\varphi_1(0.54) \\
&= \frac{U_1(R(3.3))}{\sqrt{10}} \frac{U_1(R(5.4))}{\sqrt{10}} \\
&= \frac{1}{10}[11100 \ldots 0]'[111110 \ldots 0] \\
&= 0.3
\end{align*}
\]

Maji and Berg, “Max-margin additive classifiers for detection”, ICCV’09.
Explicit embedding

**Intersection kernel**

\[ K_{\text{int}}(a, b) = \sum_{i=1}^{\min(a_i, b_i)} \]

Split [0,1] into N bins (e.g. N=10):

Introducing notation:

\[ U_2(t) = \begin{bmatrix} 111 \ldots 1(t - \lfloor t \rfloor) \\ 00 \ldots 0 \end{bmatrix} \]

We have:

\[ \varphi_2(t) = \frac{U_2(Nt)}{\sqrt{N}} \approx \min(0.33, 0.54) \]
\[ \approx \varphi_2(0.33) \varphi_2(0.54) \]
\[ = \frac{U_2(3.3)}{\sqrt{10}} \frac{U_2(5.4)}{\sqrt{10}} \]
\[ = \frac{1}{10} [111 0.3 0 \ldots 0]' [11111 0.4 0 \ldots 0] = 0.33 \]

Maji and Berg, “Max-margin additive classifiers for detection”, ICCV’09.
Explicit embedding
Intersection kernel \( K_{\text{int}}(a, b) = \sum \min(a_i, b_i) \)

😊 As accurate as original kernel classifier
😊 Embedding is (almost) costless
😊 Embedded vectors can be sparsified: encode index (and residual)
😊 Training/classification uses simple look-up table accesses

😊 Look-up table accesses can be much slower than standard arithmetic operations (e.g. addition, multiplication)
😊 Embedding space is much larger than original space (e.g. x30) and linear classifiers are not sparse

See also: Wang, Hoiem and Forsyth, “Learning image similarity from Flickr groups using SIKMA”, ICCV’09.
Explicit embedding
Additive homogeneous kernels

Homogeneous kernels: \( \forall c \geq 0 : k(cx, cy) = ck(x, y) \)

For any homogeneous kernel there exists a symmetric non-negative measure \( \kappa(\lambda)d\lambda \) such that:

\[
k(x, y) = \sqrt{xy} \int_{-\infty}^{+\infty} e^{-i\lambda \log(y/x)} \kappa(\lambda)d\lambda
\]

Hein and Bousquet, “Hilbertian metrics and positive definite kernels on probability measures”, AISTAT’05.

This can be rewritten as:

\[
k(x, y) = \int_{-\infty}^{+\infty} \varphi_\lambda(x)^* \varphi_\lambda(y)d\lambda
\]

with \( \varphi_\lambda(x) = e^{-i\lambda \log(x)} \sqrt{x} \kappa(\lambda) \)

\( \rightarrow \) closed form formulas exist for standard computer vision kernels

Explicit embedding
Additive homogeneous kernels

How to approximate the $\infty$ dimensional feature map? $\rightarrow$ sampling

$$k(x, y) \approx \sum_{k=-n}^{+n} \varphi_{kL}(x)^* \varphi_{kL}(y) \quad \text{and} \quad \varphi(x) = [\varphi_{-nL}(x) \ldots \varphi_{+nL}(x)]$$

with L the sampling step

Comparison with [Maji’09]:


😊 Higher accuracy for same dimensionality
Explicit embedding
Additive kernels: general case

Going back to kernel kPCA: we can
- perform kPCA independently in each dimension
- concatenate dimension-wise embeddings

→ Additive kPCA (addkPCA)

Embedding cost:
- computing D kernel matrices: $O(M^2D)$
- eigen-decomposition (Cholesky): $O(M^2E)$
- embedding of a sample: $O(M(D+E))$

→ Same cost as kPCA? No as $M$ much smaller for addkPCA!

😊 Exploits data distribution: variable number of output embedding dimensions per input dimension

Explicit embedding
AddkPCA on Kchi2

\[ k(u, v) = 2 \frac{uv}{u+v} \]

Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding:
  • general case
  • additive kernels
  • exponential kernels

An embedding view of the BOV and FV

Putting everything together
Explicit embedding

Shift-invariant kernels:

\[ K(x, y) = K(x - y) \]

Bochner’s theorem: a shift-invariant kernel is psd if and only if it is the Fourier transform of a non-negative measure \( p \)

Exact embedding:

\[ K(x - y) = \int_{\omega \in \mathbb{R}^D} p(\omega) e^{i\omega' (x-y)} \, d\omega \]

\[ = E_{\omega \sim p} [\varphi_\omega(x) \ast \varphi_\omega(y)] \]

with:

\[ \varphi_\omega(x) = e^{i\omega' x} \]

Approximate embedding by sampling: draw \( \omega \)’s iid from \( p \)

\[ \varphi(x) = \sqrt{\frac{2}{E}} \left[ \cos(\omega_1 x), \ldots, \cos(\frac{\omega_F}{2} x), \sin(\omega_1 x), \ldots, \sin(\frac{\omega_F}{2} x) \right] \]

Rahimi and Recht, “Random features for large-scale kernel machines”, NIPS’07.
Explicit embedding

Exponential kernels

Example: RBF kernel → \( p = \) Gaussian distribution
But RBF kernel on BOV vectors does not perform well…

However, if we have: \( K(x, x) = \|\varphi(x)\|^2 = 1 \)
then we can write exponential kernels as shift-invariant:

\[
\begin{align*}
K^{exp}(x, y) &= \exp(\gamma(K(x, y) - 1)) \\
&= \exp\left(-\frac{\gamma}{2}\|\varphi(x) - \varphi(y)\|^2\right)
\end{align*}
\]

Two-stage embedding process:

- embed additive kernel with any of the previous methods
- embed exponential kernel with random projections

Explicit embedding

Results

Train on ImageNet, test on PASCAL VOC

Explicit embedding

Summary

Explicit embedding works: from hours/days of training and testing to seconds/minutes!

In more detail:

• square-rooting the BOV already brings large improvements!
• additive kernels bring additional improvement at affordable cost
• exponential kernels bring additional gain at a significant cost

See also for a recent study: Gong and Lazebnik, “Comparing data-dependent and data-independent embeddings for classification and ranking of Internet images”, CVPR’11.

Is explicit embedding the only way to make linear classifiers work on BOV vectors? No!
Explicit embedding
Max-pooling

The pooling strategy has a significant impact: max-pooling of local statistics superior to average pooling with linear classifiers.

Yang, Yu, Gong and Huang, “Linear spatial pyramid matching using sparse coding for image classification”, CVPR’09.

Why? Reduces variance due to different proportions of foreground vs background information.


Example on Caltech 101:

<table>
<thead>
<tr>
<th></th>
<th>AVG POOLING</th>
<th>MAX POOLING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LK</td>
<td>IK</td>
</tr>
<tr>
<td>Hard Quantization</td>
<td>51.4 (0.9)</td>
<td>64.2 (1.0)</td>
</tr>
<tr>
<td>Soft Quantization</td>
<td>57.9 (1.5)</td>
<td>66.1 (1.2)</td>
</tr>
<tr>
<td>Sparse Coding</td>
<td>61.3 (1.3)</td>
<td>70.3 (1.3)</td>
</tr>
</tbody>
</table>

From [Boureau, CVPR’10]. LK = Linear kernel. IK = Intersection kernel.
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding

An embedding view of the BOV and FV

Putting everything together
Embedding view of the BOV and FV

Embedding view of the BOV: $x_t \rightarrow \varphi_{BOV}(x_t) = [0, \ldots, 0, 1, 0, \ldots, 0]$

So to obtain high-dim linearly separable BOV representations, we do:

- embed local descriptors individually
- average $\rightarrow$ BOV histogram
- embed BOV histogram

$\rightarrow$ Aren’t two rounds of embeddings suboptimal?
Embedding view of the BOV and FV

Embedding view of the BOV: $x_t \rightarrow \varphi_{BOV}(x_t) = [0, \ldots, 0, 1, 0, \ldots, 0]$

So to obtain high-dim linearly separable BOV representations, we do:
- embed local descriptors individually
- average $\rightarrow$ BOV histogram
- embed BOV histogram

$\rightarrow$ Aren’t two rounds of embeddings suboptimal?

Fisher kernels can perform directly an embedding in much higher dim:

$$
\varphi_{FV}(x_t) = \left[0, \ldots, 0, \frac{1}{\sqrt{w_i}} \left( \frac{x_t - \mu_i}{\sigma_i} \right), \frac{1}{\sqrt{2w_i}} \left( \frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1 \right), 0, \ldots, 0 \right]
$$

$\rightarrow$ works well in combination with linear classifiers
Embedding view of the BOV and FV

**BOV:** \( \varphi_{BOV}(x_t) = [0, \ldots, 0, 1, 0, \ldots, 0] \)

**FV:** \( \varphi_{FV}(x_t) = \left[ 0, \ldots, 0, \frac{1}{\sqrt{w_i}} \left( \frac{x_t - \mu_i}{\sigma_i} \right), \frac{1}{\sqrt{2w_i}} \left( \frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1 \right), 0, \ldots, 0 \right] \)

A linear classifier on these representations induces in the descriptor space:

---

Embedding view of the BOV and FV

BOV: \( \varphi_{BOV}(x_t) = [0, \ldots, 0, 1, 0, \ldots, 0] \)

FV: \( \varphi_{FV}(x_t) = 0, \ldots, 0, \frac{1}{\sqrt{w_i}} \left( \frac{x_t - \mu_i}{\sigma_i} \right), \frac{1}{\sqrt{2w_i}} \left( \frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1 \right), 0, \ldots, 0 \)

A linear classifier on these representations induces in the descriptor space:
- in the BOV case: a piece-wise constant decision function

Embedding view of the BOV and FV

BOV: \( \varphi_{BOV}(x_t) = [0, \ldots, 0, 1, 0, \ldots, 0] \)

FV: \( \varphi_{FV}(x_t) = \left[ 0, \ldots, 0, \frac{1}{\sqrt{w_i}} \left( \frac{x_t - \mu_i}{\sigma_i} \right), \frac{1}{\sqrt{2w_i}} \left( \frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1 \right), 0, \ldots, 0 \right] \)

A linear classifier on these representations induces in the descriptor space:

- in the BOV case: a piece-wise constant decision function
- in the FV case: a piecewise linear / quadratic decision function

Embedding view of the BOV and FV

BOV: \( \varphi_{BOV}(x_t) = [0, \ldots, 0, 1, 0, \ldots, 0] \)

FV: \( \varphi_{FV}(x_t) = \left[ 0, \ldots, 0, \frac{1}{\sqrt{w_i}} \left( \frac{x_t - \mu_i}{\sigma_i} \right), \frac{1}{\sqrt{2w_i}} \left( \frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1 \right), 0, \ldots, 0 \right] \)

A linear classifier on these representations induces in the descriptor space:
- in the BOV case: a piece-wise constant decision function
- in the FV case: a piecewise linear / quadratic decision function

→ FV leads to more complex decision functions for same vocabulary size

Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding

An embedding view of the BOV and FV

Putting everything together:
  - large-scale pipelines
  - large-scale results
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding

An embedding view of the BOV and FV

Putting everything together:
  • large-scale pipelines
  • large-scale results
BOV-based large-scale classification

BOV + explicit embedding + linear classifier

Memory / storage issue: embedding increases feature dim, e.g. x3 in [Vedaldi, CVPR’10] or [Perronnin, CVPR’10]

If embedding cost is low enough, efficient learning with SGD is possible:
  • pick a random sample in original (low-dimensional) space
  • embed sample on-the-fly in new (high-dimensional) space
  • feed embedded sample to SGD routine
  • discard embedded sample
FV-based large-scale classification

FV + linear classifier

Memory / storage issue: the FV is high-dim (e.g. $\frac{1}{2}$M dim) and weakly sparse

→ Example using 4-byte floating point arithmetic:
  - ILSVRC2010 $\approx$ 2.8TBs
  - ImageNet $\approx$ 23TBs
  … per low-level descriptor type

😊 However, we can significantly compress FVs, e.g. using PQ

→ 64 fold compression with zero accuracy loss in categorization
FV-based large-scale classification

FV + PQ (at training time) + linear classifier

Efficient SGD learning algorithm:
- pick a random compressed sample
- decompress **on-the-fly** in high-dimensional space
- feed uncompressed sample to SGD routine
- discard uncompressed sample

→ one decompressed sample alive in RAM at a time


Also possible to learn classifiers without decompression


Note: test samples do **not** need to be compressed
Outline

Efficient learning of linear classifiers with SGD

Non-linear classification with explicit embedding

An embedding view of the BOV and FV

Putting everything together:
  • large-scale pipelines
  • large-scale results
Large-scale results
Evaluation

How to measure accuracy when dealing with a large number of classes?

One cannot expect all images to be labeled with all possible classes
→ Measure a top-K loss (e.g. K=5 in ILSVRC)
Given a test image (labeled with G), predict labels $L_1, \ldots, L_K$

\[
\text{loss} = \min_k d(G, L_k) \text{ with } d(x,y) = 1 \text{ if } x=y, 0 \text{ otherwise}
\]

Not all mistakes are equally bad
→ Measure a hierarchical loss
\[d(x,y) = \text{height of lowest common ancestor of } x \text{ and } y / \text{max. possible height}\]
Results on ILSVRC2010

ImageNet Large-Scale Visual Recognition Challenge 2010:
- 1K classes (leaves only)
- 1.2M training images + 50K validation images + 150K test images
- measure top-5 accuracy (flat and hierarchical)

Berg, Deng and Fei-Fei, “Large-scale visual recognition challenge 2010”

Top 2 systems employed higher order statistics
Results on ILSVRC2010

ImageNet Large-Scale Visual Recognition Challenge 2010:
- 1K classes (leaves only)
- 1.2M training images + 50K validation images + 150K test images
- measure top-5 accuracy (flat and hierarchical)

Berg, Deng and Fei-Fei, "Large-scale visual recognition challenge 2010"

No compression: training with Hadoop map-reduce (≈ 100 mappers)
With PQ compression: training on a single server
Results on ILSVRC2011

ImageNet Large-Scale Visual Recognition Challenge 2011:
• 1K classes (leaves only)
• 1.2M training images + 50K validation images + 100K test images
• measure top-5 accuracy (flat and hierarchical)

Berg, Deng and Fei-Fei, “Large-scale visual recognition challenge 2011”

FV + PQ compression + SGD learning
BOV + non-linear SVM in dual (GPU computing)
Results on ImageNet10K:

- 10,184 classes (leaves and internal nodes)
- \( \approx 9 \)M images: \( \frac{1}{2} \) training / \( \frac{1}{2} \) test
- accuracy measured as % top-1 correct
Results on ImageNet10K

Results on ImageNet10K:
- 10,184 classes (leaves and internal nodes)
- ≈ 9M images: ½ training / ½ test
- accuracy measured as % top-1 correct

SIFT + BOV (21K-dim) + explicit embedding + linear SVM (SGD)

→ accuracy = 6.4%
→ training time ≈ 6 CPU years

Deng, Berg, Li and Fei-Fei, “What does classifying more than 10,000 image categories tell us?”, ECCV’10.
Results on ImageNet10K

Results on ImageNet10K:
- 10,184 classes (leaves and internal nodes)
- \( \approx 9\text{M} \) images: \( \frac{1}{2} \) training / \( \frac{1}{2} \) test
- accuracy measured as \% top-1 correct

SIFT + BOV (21K-dim) + explicit embedding + linear SVM (SGD)

\[ \rightarrow \text{accuracy} = 6.4\% \]
\[ \rightarrow \text{training time} \approx 6 \text{ CPU years} \]
Deng, Berg, Li and Fei-Fei, “What does classifying more than 10,000 image categories tell us?”, ECCV’10.

SIFT + FV (130K-dim) + PQ compression + linear SVM (SGD)

\[ \rightarrow \text{accuracy} = 19.1\% \]
\[ \rightarrow \text{training time} \approx 1 \text{ CPU year (trick: do not sample all negatives)} \]
Questions?