Large-scale visual recognition
Large-scale learning

Florent Perronnin, XRCE

Hervé Jégou, INRIA

CVPR tutorial on Large-Scale Visual Recognition

June 16, 2012

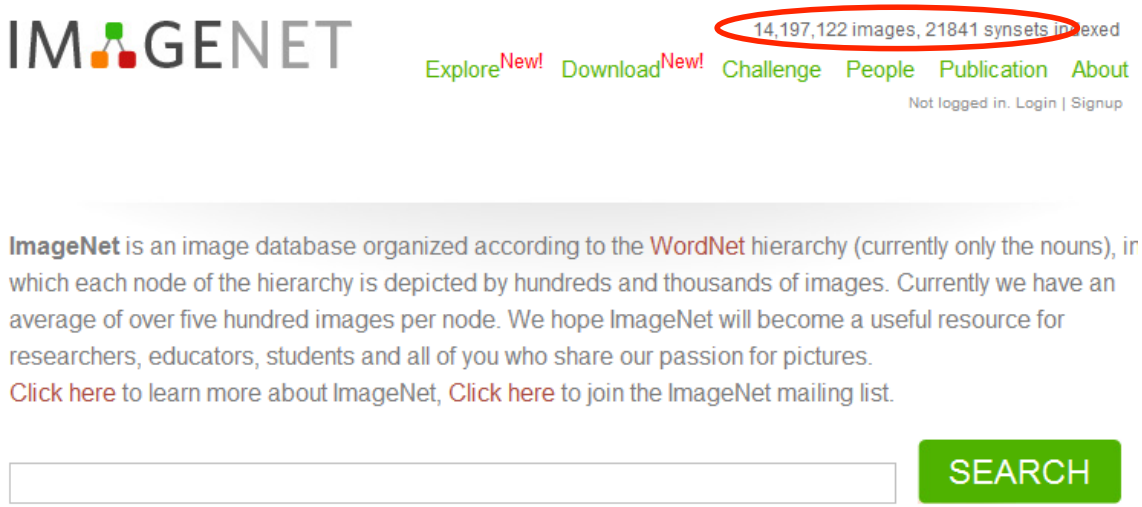
Motivation

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)

LeCun, Bottou, Orr, Müller, “Efficient BackProp”, Neural Networks: Tricks of the trade”, Springer, 1998.

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.

Bottou, Bousquet, “The Tradeoffs of large scale learning”, NIPS’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:

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

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

$$\rightarrow \quad Q(z_i; w) = \ell(x_i, y_i; w) + \frac{\lambda}{2}||w||^2$$

Learning with SGD
Batch Gradient Descent (BGD)

Iterative optimization algorithm:

$$\begin{aligned} w_{t+1} &= w_t - \eta_t \nabla_{w=w_t} C(w) \\ &= w_t - \eta_t \frac{1}{N} \sum_{i=1}^{N} \nabla_{w=w_t} Q(z_i; w) \end{aligned}$$

where η_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: $w_{t+1} = w_t - \eta_t \nabla_{w=w_t} Q(z_t; w)$

where η_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
→ 10x100 = 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:

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

$$\left.\right\} \quad \eta_t = \frac{a}{t + t_0}$$

→ important to tune (a,t0) correctly

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

Bai, Weston, Grangier, Collobert, Chapelle, Weinberger, "Supervised semantic indexing", CIKM'09.

Weston, Bengio, Usunier, "Large-scale image annotation: Learning to rank with joint word-image embeddings", ECML'10.

Perronnin, Akata, Harchaoui, Schmid, "Towards good practice in large-scale learning for image classification", CVPR'12.

Learning with SGD

Linear two-class SVM example

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

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: $\bar{w}_t = \frac{1}{t} \sum_{i=1}^{t} w_i$

Polyak, Juditsky, "Acceleration of stochastic approximation by averaging", SIAM J. Control and Optimization'92.

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, Shaledv-Swartz, Singer, "Online passive-aggressive algorithms", JMLR'06.

Best practices for large-scale learning of image classifiers :

Perronnin, Akata, Harchaoui, Schmid, "Towards good practice in large-scale learning for image classification", CVPR'12.

→ 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

PASCAL VOC 2007

Linear SVM [1]					
44.1					
Non-Linear SVM [2]					
Additive kernels			Exponential kernels		
K_{bha}	K_{chi2}	K_{int}	K_{bha}^{exp}	K_{chi2}^{exp}	K_{int}^{exp}
48.9	50.1	50.5	52.7	52.8	52.0

From: Perronnin, Sánchez and Liu, "Large-scale image categorization with explicit data embedding", CVPR'10.

Can we get the **accuracy of non-linear at the cost of linear**?

Explicit embedding
SVM formulations

Dual formulation:

☺ avoids explicit mapping of data in high-dimensional space

☹ training cost (e.g. SMO): between quadratic and cubic in # training samples

☹ runtime cost: linear in # support vectors

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

Primal formulation:

☹ requires explicit embedding of data in higher dimensional space

☺ training cost (e.g. SGD): linear in # training samples

☺ 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:

$$\boxed{K(x,z) = \varphi(x)'\varphi(z)}$$

The score becomes:

$$\begin{aligned} f(z) &= \sum_{i=1}^{N} a_i K(z, x_i) + b \\ &= \left(\sum_{i=1}^{N} \alpha_i \varphi(x_i) \right)' \varphi(z) + b \end{aligned}$$

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 φ
- learn linear classifiers in the new space (e.g. with efficient SGD)

Our focus: given a kernel, find φ

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: $\mathbf{K}\psi_e = \sigma_e^2 \psi_e$

Embed a vector z with the Nyström approximation:

- project z: $K(z, .) = [K(z, x_1), \ldots, K(z, x_M)]$
- embed: $\varphi_e(z) = \dfrac{K(z, .)' \psi_e}{\sigma_e}$

Schölkopf, Smola and Müller, "Non-linear component analysis as a kernel eigenvalue problem", Neural Computation'98.
Williams and Seeger, "Using the Nystrom 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)

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

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

→ Generalized Additive Model (GAM)

Hastie and Tibshirani, "Generalized additive models", Statistical Science'86.

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 $k(x(d), z(d)) = \varphi_d\left(x(d)\right)' \varphi_d\left(z(d)\right)$

The final embedding function is given by:

$$\boxed{\varphi(x) = \left[\varphi_1\left(x(1)\right) \ldots \varphi_D\left(x(D)\right)\right]}$$

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: $\boxed{\varphi(z) = \sqrt{z}}$

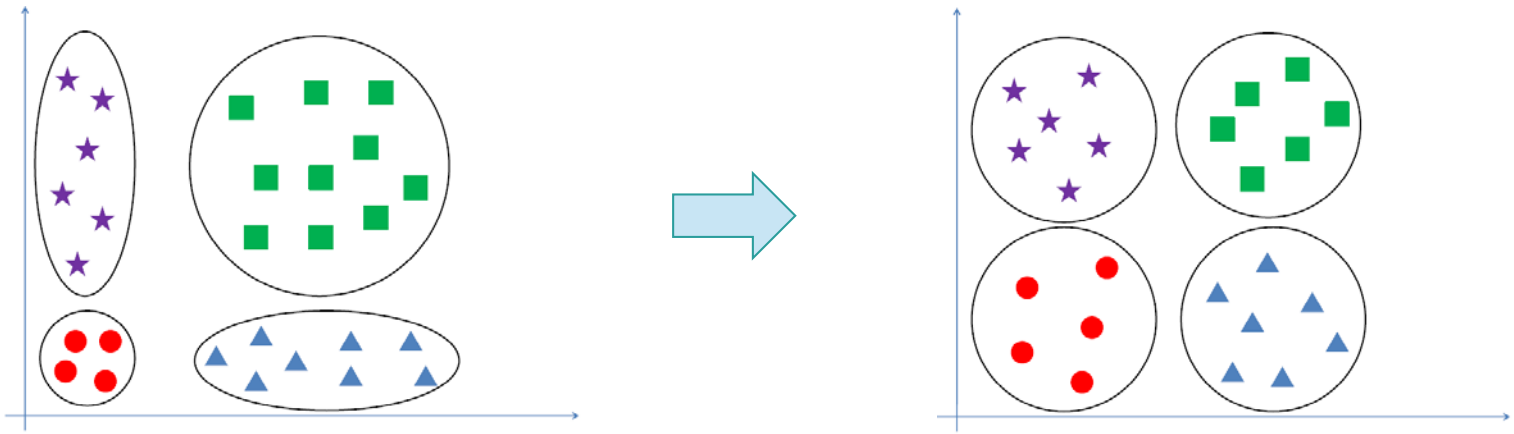
Perronnin, Sánchez and Liu, "Large-scale image categorization with explicit data embedding", CVPR'10.
Vedaldi and Zisserman, "Efficient additive kernels via explicit feature maps", CVPR'10.

Intuitive interpretation: reduce effect of bursty visual words

Jégou, Douze and Schmid, "On the burstiness of visual elements", ICCV'09.

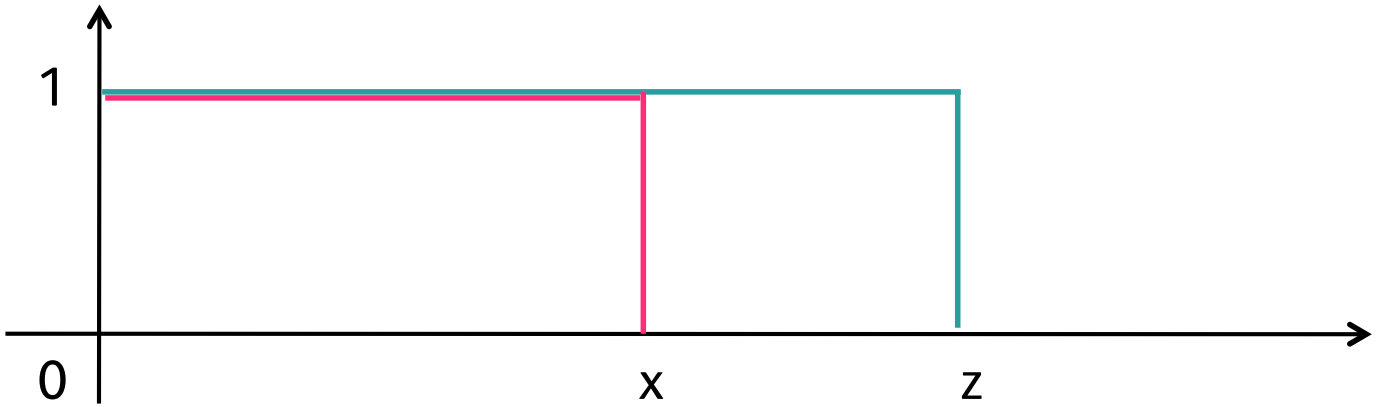
Theoretical interpretation: variance stabilizing transform

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: $\mathbb{1}_{(a,b)}(t) = \begin{cases} 1 \text{ if } a \leq t \leq b \\ 0 \text{ otherwise} \end{cases}$

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

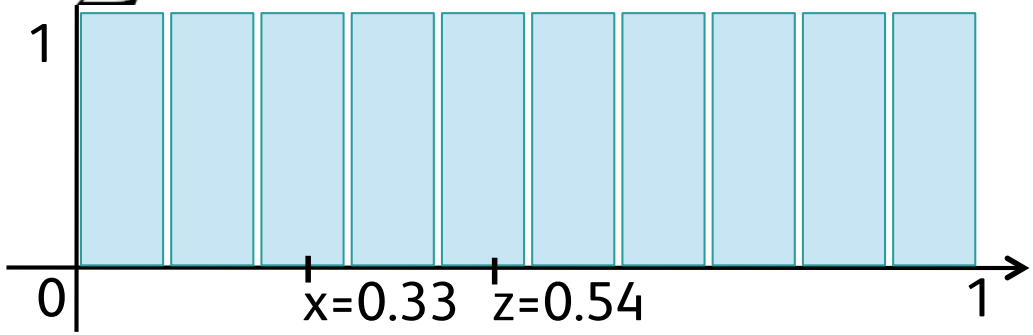
How to approximate the ∞ dimensional feature map with a finite one?

→ **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: $U_1(t) = [\overbrace{111\ldots1}^{\times t}\overbrace{000\ldots0}^{\times(N-t)}]$ and $R(t) = \lfloor t + 0.5 \rfloor$

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

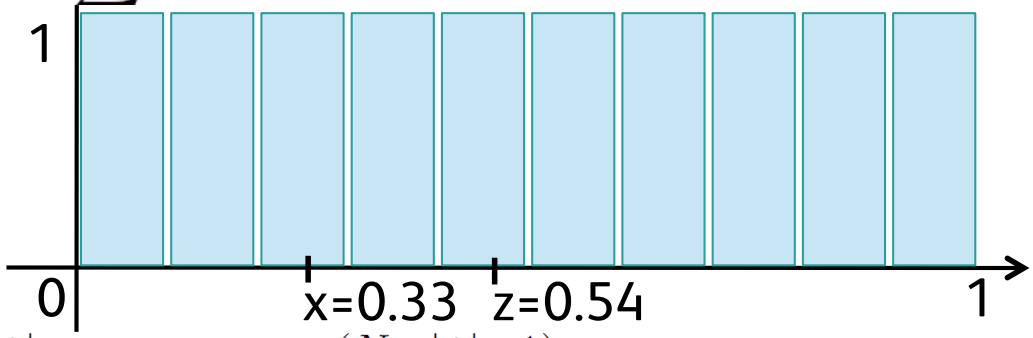
$$\begin{aligned} & min(0.33, 0.54) \\ \approx\ & \varphi_1(0.33)'\varphi_1(0.54) \\ =\ & \frac{U_1(R(3.3))}{\sqrt{10}}\frac{U_1(R(5.4))}{\sqrt{10}} \\ =\ & \frac{1}{10}[11100\ldots0]'[111110\ldots0] \\ =\ & 0.3 \end{aligned}$$

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

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 notation: $U_2(t) = [\overbrace{111\ldots1}^{\times\lfloor t\rfloor}(t-\lfloor t\rfloor) \quad \overbrace{00\ldots0}^{\times(N-\lfloor t\rfloor-1)}]$

We have: $\boxed{\varphi_2(t) = \frac{U_2(Nt)}{\sqrt{N}}}$

$$\begin{aligned} & min(0.33, 0.54) \\ \approx\ & \varphi_2(0.33)'\varphi_2(0.54) \\ =\ & \frac{U_2(3.3)}{\sqrt{10}}\frac{U_2(5.4)}{\sqrt{10}} \\ =\ & \frac{1}{10}[111\ 0.3\ 0\ldots0]'[11111\ 0.4\ 0\ldots0] \\ =\ & 0.33 \end{aligned}$$

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

Explicit embedding

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

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

→ closed form formulas exist for standard computer vision kernels

Vedaldi and Zisserman, "Efficient additive kernels via explicit feature maps", CVPR'10.

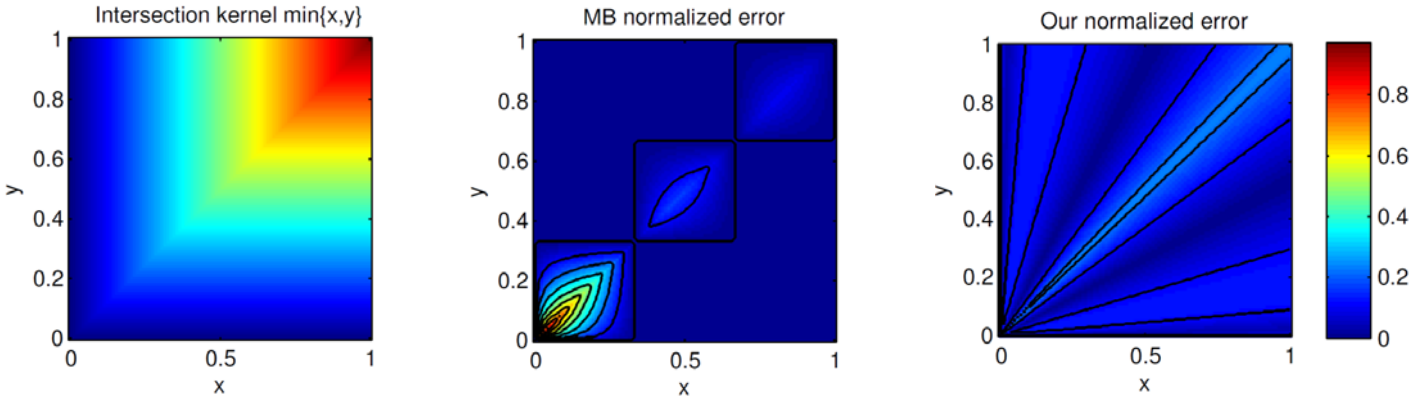
Explicit embedding
Additive homogeneous kernels

How to approximate the ∞ dimensional feature map? → **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]:



From: Vedaldi and Zisserman, "Efficient additive kernels via explicit feature maps", CVPR'10.

☺ 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:

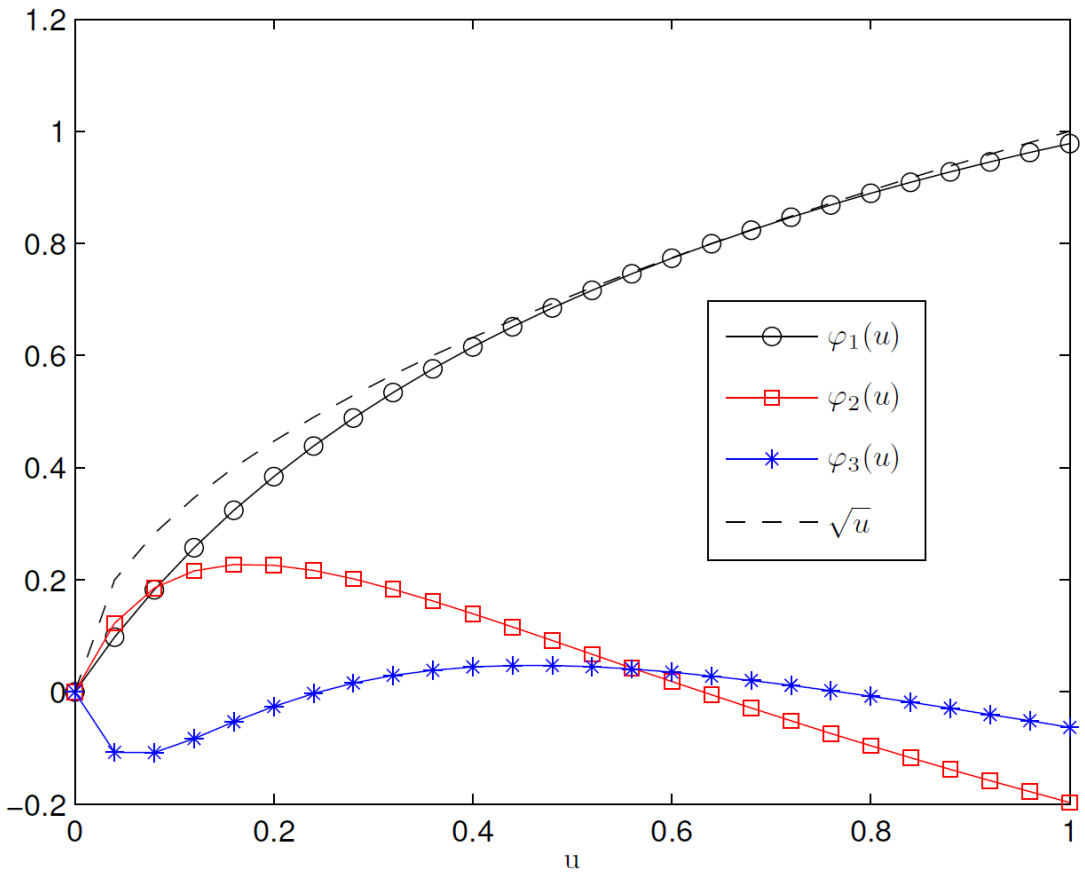
- computing D kernel matrices: O(M^2D)
- eigen-decomposition (Cholesky): O(M^2E)
- embedding of a sample: O(M(D+E))

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

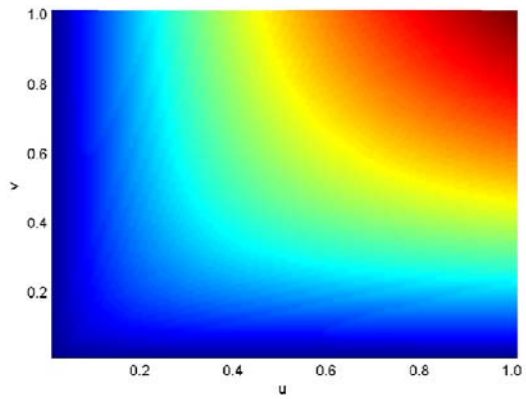
☺ Exploits data distribution: variable number of output embedding dimensions per input dimension

Perronnin, Sánchez and Liu, "Large-scale image categorization with explicit data embedding", CVPR'10.

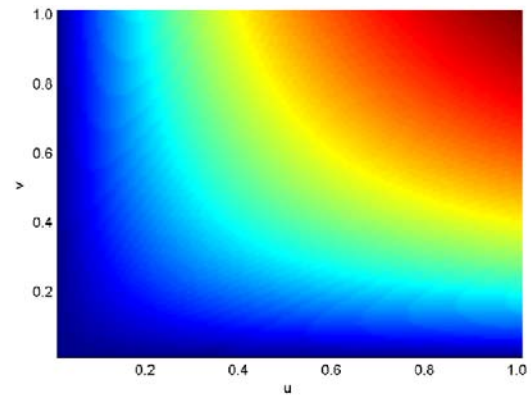
Explicit embedding
AddkPCA on Kchi2



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



$$\varphi_1(u)'\varphi_1(v)$$



Perronnin, Sánchez and Liu, “Large-scale image categorization with explicit data embedding”, CVPR’10.

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

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:
$$\begin{aligned} K(x-y) &= \int_{\omega \in \mathbb{R}^D} p(\omega) e^{i\omega'(x-y)} d\omega \\ &= E_{\omega \sim p}\left[\varphi_\omega(x)^* \varphi_\omega(y)\right] \end{aligned}$$

with: $\boxed{\varphi_\omega(x) = e^{i\omega' x}}$

Approximate embedding by sampling: draw ω's iid from p

$$\boxed{\varphi(x) = \sqrt{\frac{2}{E}}\left[\cos(\omega_1' x), \ldots, \cos(\omega'_{\frac{E}{2}} x), \sin(\omega_1' x), \ldots, \sin(\omega'_{\frac{E}{2}} x)\right]}$$

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:

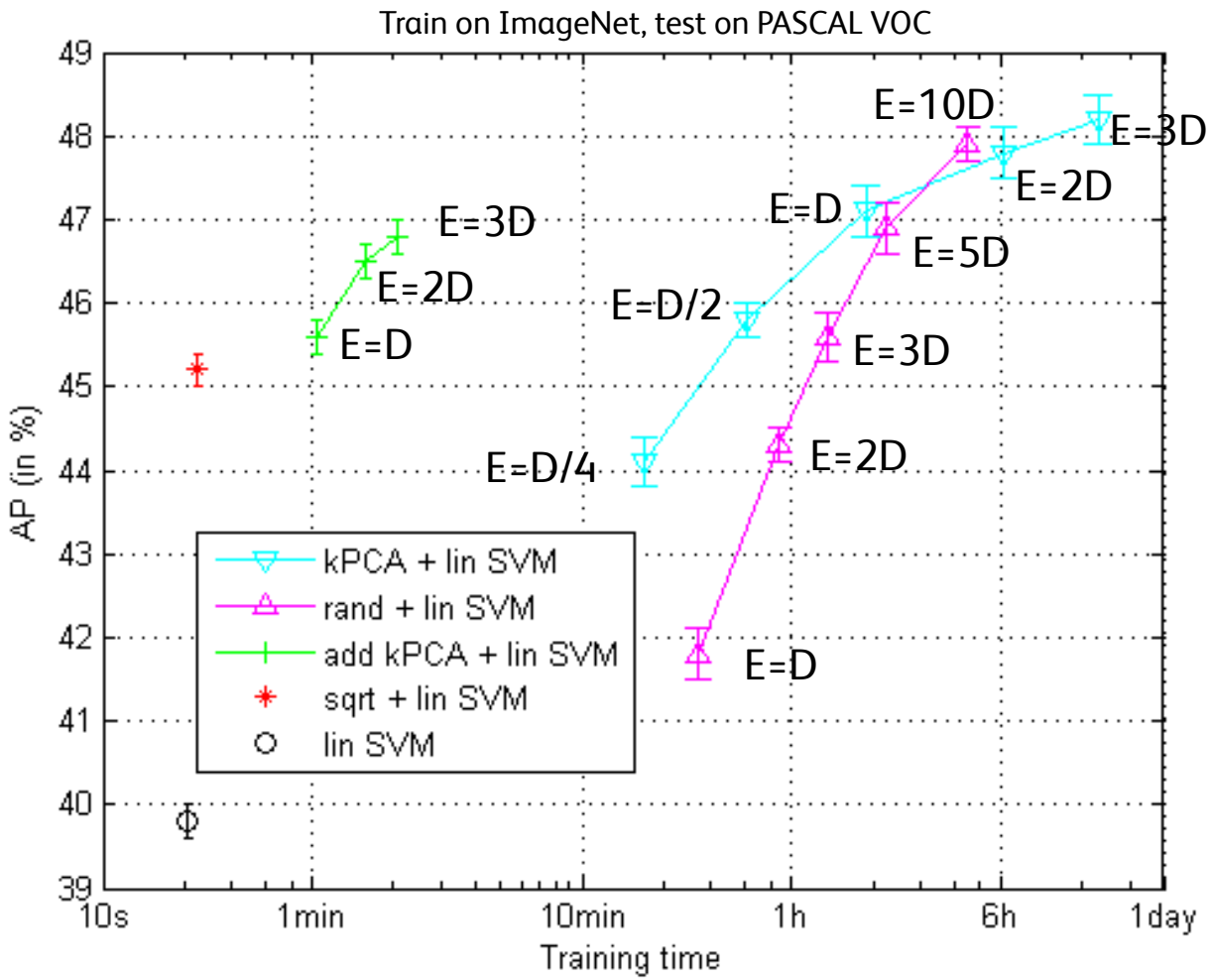
$$\boxed{\begin{aligned} K^{exp}(x,y) &= \exp(\gamma(K(x,y)-1)) \\ &= \exp\left(-\frac{\gamma}{2}||\varphi(x)-\varphi(y)||^2\right) \end{aligned}}$$

Two-stage embedding process:

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

Perronnin, Sánchez and Liu, "Large-scale image categorization with explicit data embedding", CVPR'10.
Vempati, Vedaldi, Zisserman and Jawahar, "Generalized RBF features maps for efficient detection", BMVC'10.

Explicit embedding
Results



Perronnin, Sánchez and Liu, “Large-scale image categorization with explicit data embedding”, CVPR’10.

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.
Wang, Yang, Yu, Lv, Huang and Gong, "Locality-constrained Linear Coding for Image Classification", CVPR'10

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

Boureau, Bach, LeCun and Ponce, "Learning mid-level features for recognition", CVPR'10.
See also: Boureau, Ponce and LeCun, "A Theoretical analysis of feature pooling in visual recognition", ICML'10.

Example on Caltech 101:

	AVG POOLING		MAX POOLING	
	LK	IK	LK	IK
Hard Quantization	51.4 (0.9)	64.2 (1.0)	64.3 (0.9)	64.3 (0.9)
Soft Quantization	57.9 (1.5)	66.1 (1.2)	69.0 (0.8)	70.6 (1.0)
Sparse Coding	61.3 (1.3)	70.3 (1.3)	71.5 (1.1)	71.8 (1.0)

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 → BOV histogram
- embed BOV histogram

→ 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 → BOV histogram
- embed BOV histogram

→ Aren't two rounds of embeddings suboptimal?

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

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

→ 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: $$\varphi_{FV}(x_t) = \left[0, \ldots, 0, \overbrace{\frac{1}{\sqrt{w_i}} \left(\frac{x_t - \mu_i}{\sigma_i} \right), \frac{1}{\sqrt{2w_i}} \left(\frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1 \right)}^{2D \text{ non-zero dim}}, 0, \ldots, 0 \right]$$

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

Csurka and Perronnin, “An efficient approach to semantic segmentation”, IJCV ‘10.

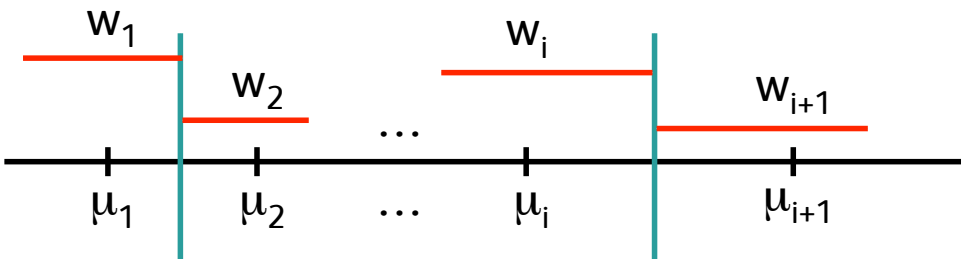
Embedding view of the BOV and FV

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

FV: $$\varphi_{FV}(x_t) = \left[0, \ldots, 0, \overbrace{\frac{1}{\sqrt{w_i}}\left(\frac{x_t - \mu_i}{\sigma_i}\right), \frac{1}{\sqrt{2w_i}}\left(\frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1\right)}^{2D \text{ non-zero dim}}, 0, \ldots, 0\right]$$

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

- in the BOV case: a piece-wise constant decision function



Csurka and Perronnin, “An efficient approach to semantic segmentation”, IJCV ‘10.

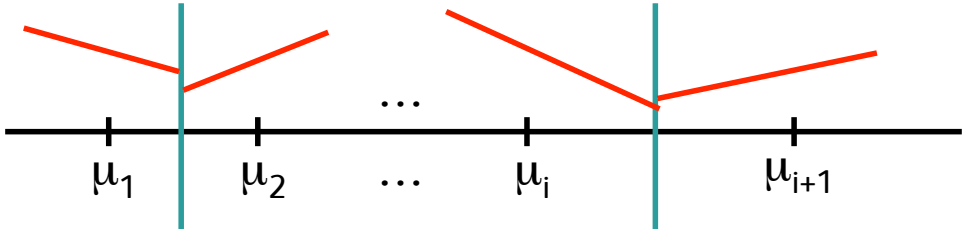
Embedding view of the BOV and FV

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

FV: $$\varphi_{FV}(x_t) = \left[0, \ldots, 0, \overbrace{\frac{1}{\sqrt{w_i}}\left(\frac{x_t - \mu_i}{\sigma_i}\right), \frac{1}{\sqrt{2w_i}}\left(\frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1\right)}^{2D \text{ non-zero dim}}, 0, \ldots, 0\right]$$

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



Csurka and Perronnin, “An efficient approach to semantic segmentation”, IJCV ‘10.

Embedding view of the BOV and FV

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

FV: $$\varphi_{FV}(x_t) = \left[0, \ldots, 0, \overbrace{\frac{1}{\sqrt{w_i}}\left(\frac{x_t - \mu_i}{\sigma_i}\right), \frac{1}{\sqrt{2w_i}}\left(\frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1\right)}^{2D \text{ non-zero dim}}, 0, \ldots, 0\right]$$

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

Csurka and Perronnin, "An efficient approach to semantic segmentation", IJCV '10.

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. ½M dim) and weakly sparse

→ Example using 4-byte floating point arithmetic:

- ILSVRC2010 ≈ 2.8TBs
- ImageNet ≈ 23TBs

… per low-level descriptor type

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

Jégou, Douze and Schmid, "Product quantization for nearest neighbor search", TPAMI'11.

Jégou, Douze, Schmid and Pérez, "Aggregating local descriptors into a compact image representation", CVPR'10.

→ 64 fold compression with zero accuracy loss in categorization

Sánchez and Perronnin, "High-dimensional signature compression for large-scale image classification", CVPR'11.

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

Sánchez and Perronnin, "High-dimensional signature compression for large-scale image classification", CVPR'11.

Also possible to learn classifiers without decompression

Vedaldi, Zisserman, "Sparse kernel approximations for efficient classification and detection", CVPR'12.

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

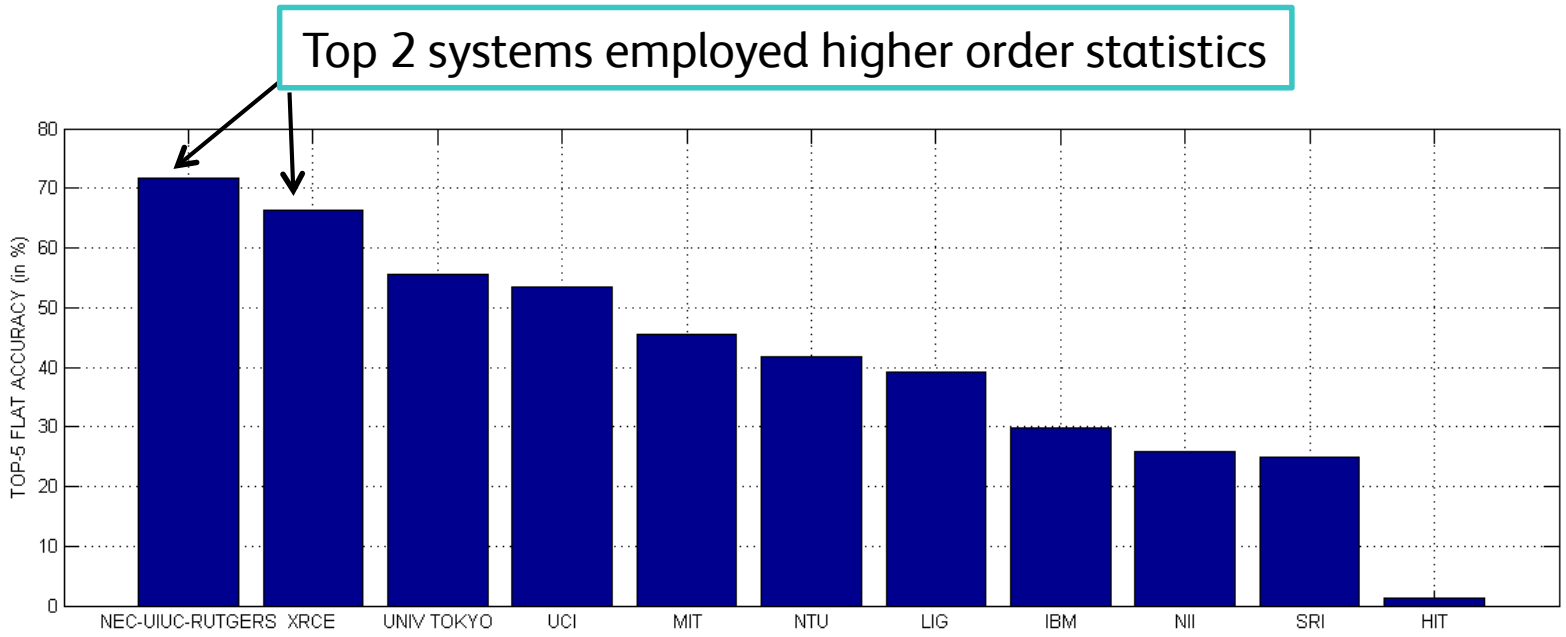
d(x,y) = height of lowest common ancestor of x and y / max. possible height

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"



Results on ILSVRC2010

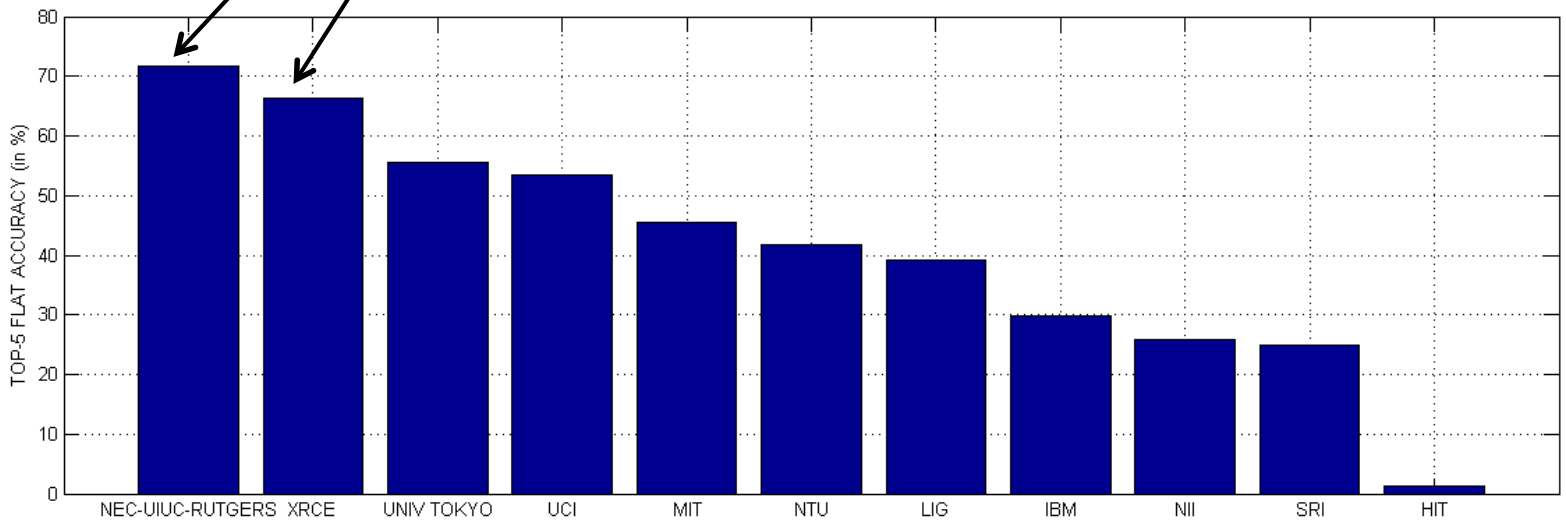
ImageNet Large-Scale Visual Recognition Challenge 2010:

- 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

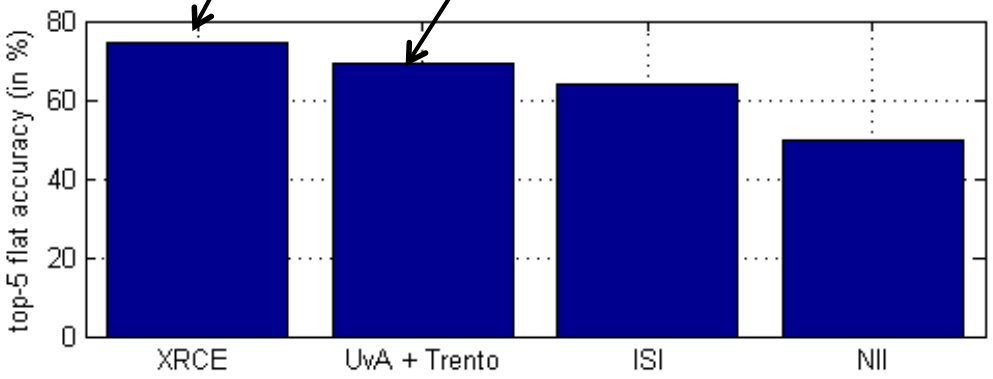
ImageNet Large-Scale Visual Recognition Challenge 2011:

- 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

Results on ImageNet10K:

- 10,184 classes (leaves and internal nodes)
- ≈ 9M images: ½ training / ½ 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)
- ≈ 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.

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

→ accuracy = **19.1%**

→ training time ≈ 1 CPU year (**trick: do not sample all negatives**)

Perronnin, Akata, Harchaoui, Schmid, "Towards good practice in large-scale learning for image classification", CVPR'12.

Questions?