Training Invariant Support Vector Machines using Selective Sampling

Gaëlle Loosli, Stéphane Canu, Léon Bottou

November 2005

Abstract

(author?) [3] describe the efficient online LASVM algorithm using selective sampling. On the other hand, **(author?)** [24] propose a strategy for handling invariance in SVMs, also using selective sampling. This paper combines the two approaches to build a very large SVM. We present state-of-the-art results obtained on a handwritten digit recognition problem with 8 millions points on a single processor. This work also demonstrates that online SVMs can effectively handle really large databases.

1 Introduction

Because many patterns are insensitive to certain transformations such as rotations or translations, it is widely admitted that the quality of a pattern recognition system can be improved by taking into account invariance. Very different ways to handle invariance in machine learning algorithms have been proposed [14, 20, 35, 22, 34].

In the case of kernel machines, three general approaches have been proposed. The first approach consists of learning orbits instead of points. It requires costly semi-definite programming algorithms [17]. The second approach involves specialized kernels. This turns out to be equivalent to mapping the patterns into a space of invariant features [8]. Such features are often difficult to construct. The third approach is the most general. It consists of artificially enlarging the training set by new examples obtained by deforming the available input data [33]. This approach suffers from high computational costs. Very large datasets can be generated this way. For instance, by generating 134 random deformations per digit, we have increased the MNIST training set size from 60,000 to more than 8 millions

of examples. A batch optimization algorithm working on the augmented database needs to go several times through the entire dataset until convergence. Either we store the whole dataset or we compute the deformed examples on demand, trading reduced memory requirements for increased computation time.

We propose two tools to reduce these computational costs:

- *Selective sampling* lets our system select informative points and discard useless ones. This property would reduce the amount of samples to be stored.

- Unlike batch optimization algorithms, *online learning* algorithms do not need to access each example again and again until convergence. Such repeated access are particularly costly when the deformed examples are computed on demand. This is even more wasteful because most examples are readily discarded by the selective sampling methods.

1.1 Large scale learning

Running machine learning algorithms on very large scale problems is still difficult. For instance, **(author?)** [1] discuss the problem of scaling SVM "to massive datasets" and point out that learning algorithms are typically quadratic and imply several scans of the data. Three factors limit machine learning algorithms when both the sample size n and the input dimension d grow very large. First, the training time becomes unreasonable. Second, the size of the solution affects the memory requirements during both the training and recognition phases. Third, labeling the training examples becomes very expensive. To address those limitations, there has been a lot of clever studies on solving quadratic optimization problems [7, 18], on online learning [4, 19, 11], on sparse solutions [39], and on active learning [9, 6].

The notion of computational complexity discussed in this paper is tied to the empirical performance of algorithms. Three common strategies can be distinguished to reduce this practical complexity (or observed training time). The first strategy consists in working on subsets of the training data, solving several smaller problems instead of a large one, as in the SVM decomposition method [30]. The second strategy consists in parallelizing the learning algorithm. The third strategy tries to design less complex algorithms that give an approximate solution with equivalent or superior performance. For instance, early stopping approximates the full optimization without compromising the test set accuracy. This paper focuses on this third strategy.

1.2 Learning complexity

The core of many learning algorithms essentially amounts to solving a system of linear equations of the form $Ax = b$ where A is an $n \times n$ input matrix, b an output vector and x is an unknown vector. There are many classical methods to solve such systems [16]. For instance, if we know that A is symmetric and semi-definite positive, we can factorize as $A = LL^\top$ where L is a lower triangular matrix representing the square root of A. The factorization costs $\mathcal{O}(n^3)$ operations. Similarly to this Cholesky factorization, there are many methods for every kind of matrices A such as QR, LU, etc. Their complexity is always $\mathcal{O}(n^3)$. They lead to the exact solution, even though in practice, some are more numerically stable or slightly faster. When the problem becomes large, a method with a cubic complexity is not a realistic approach.

Iterative methods are useful for larger problems. The idea is to start from some initial vector x_0 (null or random) and to iteratively update it by performing steps of size ρ_k along direction d_k, that is $x_{k+1} = x_k + \rho_k d_k$. Choosing the direction is the difficult part. The optimization literature suggests that the first order gradients yield very poor directions. Finding a second order direction costs $\mathcal{O}(n^2)$ operations (conjugate gradients, etc.) and provides the exact solution in n iterations, for a total of $\mathcal{O}(n^3)$ operations. However we obtain an approximate solution in fewer steps. Therefore these algorithms have a practical complexity of $\mathcal{O}(kn^2)$ where k is the number of steps required to have a good enough approximation. Hence we have algorithms for larger problems, but they remain too costly for really very large datasets.

Additional progress can be achieved by exploiting sparsity. The idea is to constrain the number of non null coefficients in vector x. Approximate computations can indicate which coefficients will be zero with high confidence. Then the remaining subsystem can be solved with any of the previously described methods. This approach corresponds to the fact that many examples bring very little additional information. The hope is then to have a solution with an empirical complexity $\mathcal{O}(n^d)$ with d close to 2. This idea is exploited by modern SVM algorithms.

During his optimization lecture, **(author?)** [28] said that *the extremely large-scale case ($n \gg 10^5$) rules out all advanced convex optimization technics.* He argues that there is only one option left, *first order gradient methods.* We are convinced that other approximate techniques can exploit the stochastic regularities of learning problems. For instance the LASVM method [2, 3] seems considerably more efficient than the first order online SVMs discussed in [19].

1.3 Online learning

Online learning algorithms are usually associated with problems where the complete training set is not available beforehand. However their computational properties are very useful for large scale learning. A well designed online algorithm needs less computation to reach the same test set accuracy as the corresponding batch algorithm [26, 5].

Two overlapping frameworks have been used to study online learning algorithms, by leveraging the mathematics of stochastic approximations [4], or by refining the mathematics of the Perceptron [29]. The Perceptron seems a natural starting point for online SVM. Algorithms derived from the Perceptron share common characteristics. Each iteration consists of two steps. First one decides if the new point x_t should influence the decision rule and then one updates the decision rule. The Perceptron, for instance, only updates its parameter w_{t-1} if the point x_t is misclassified. The updated parameters are obtained by performing a step along direction x_t, that is $w_t = w_{t-1} + y_t x_t$. Compared to maximum margin classifiers such as SVM, the Perceptron runs much faster but does not deliver as good a generalization performance.

Many authors [12, 13, 15, 23, 11] have modified the Perceptron algorithm to ensure a margin. Older variants of the Perceptron, such as *minover* and *adatron* in [27], are also very close to SVMs.

1.4 Active learning

Active learning addresses problems where obtaining labels is expensive [9]. The learner has access to a vast pool of unlabeled examples, but is only allowed to obtain a limited number of labels. Therefore it must carefully choose which examples deserve the labeling expense.

Even when all labels are available beforehand, active learning is useful because it leads to sparser solutions [32, 3]. Moreover, the criteria for asking or not a label may be cheaper to compute than trying a labeled point.

1.5 Outline of the paper

Section 2 briefly presents the SVMs and describes how the LASVM algorithm combines online and active characteristics. Section 3 discusses invariance and presents selective sampling strategies to address them. Finally, section 4 reports experiments and results on a large scale invariant problem.

2 Online algorithm with selective sampling

This section first discusses the geometry of the quadratic optimization problem and its suitability to algorithms, such as SMO [31], that iterate feasible direction searches. Then we describe how to organize feasible direction searches into an online learning algorithm amenable to selective sampling [3].

Consider a binary classification problem with training patterns $x_1 \dots x_n$ and associated classes $y_1 \dots y_n \in \{+1, -1\}$. A soft margin SVM [10] classifies a pattern x according to the sign of a decision function

$$f(x) = \sum_i \alpha_i \langle x, x_i \rangle + b \tag{1}$$

where the notation $\langle x, x' \rangle$ represents the dot-product of feature vectors associated with the patterns x and x'. Such a dot-product is often defined implicitly by means of a Mercer kernel [10]. The coefficients α_i in (1) are obtained by solving the following quadratic optimization (QP)[1]problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \; W(\boldsymbol{\alpha}) \;&=\; \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i\,j} \alpha_i \alpha_j \langle x_i, x_j \rangle \\ \text{with} \quad &\begin{cases} \sum_i \alpha_i = 0 \\ 0 \le y_i \alpha_i \le C \end{cases} \end{aligned} \tag{2}$$

A pattern x_i is called "support vector" when the corresponding coefficient α_i is non zero. The number s of support vectors asymptotically grows linearly with the number n of examples [38].

2.1 Feasible direction algorithms

The geometry of the SVM QP problem (2) is summarized in figure 1. The box constraints $0 \le y_i \alpha_i \le C$ restrict the solutions to a n–dimensional hypercube. The equality constraint $\sum \alpha_i = 0$ further restricts the solutions to a $(n-1)$–dimensional polytope $\mathcal{F}$.

Consider a feasible point $\boldsymbol{\alpha}_t \in \mathcal{F}$. A direction $\boldsymbol{u}_t$ indicates a *feasible direction* if the half-line $\{\boldsymbol{\alpha}_t + \lambda \boldsymbol{u}_t, \lambda \ge 0\}$ intersects the polytope in points other than $\boldsymbol{\alpha}_t$. Feasible direction algorithms iteratively update $\boldsymbol{\alpha}_t$ by first choosing a feasible direction $\boldsymbol{u}_t$ and searching the half-line for the feasible

[1]Note that α_i is positive when $y_i = +1$ and negative when $y_i = -1$.

point $\boldsymbol{\alpha}_{t+1}$ that maximizes the cost function. The optimum is reached when no further improvement is possible [41].

The quadratic cost function restricted to the half-line search might reach its maximum inside or outside the polytope (see figure 2). The new feasible point $\boldsymbol{\alpha}_{t+1}$ is easily derived from the differential information in $\boldsymbol{\alpha}_t$ and from the location of the bounds A and B induced by the box constraints.

$$\begin{aligned} &\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \max\{A,\ \min\{B, C\}\}\ \boldsymbol{u}_t \\ &\text{with}\ \ C = \frac{d\,W(\boldsymbol{\alpha}_t + \lambda \boldsymbol{u}_t)}{d\,\lambda} \left(\frac{d^2\,W(\boldsymbol{\alpha}_t + \lambda \boldsymbol{u}_t)}{d\,\lambda^2} \right)^{-1} \end{aligned} \tag{3}$$

Computing these derivatives for arbitrary directions $\boldsymbol{u}_t$ is potentially expensive because it involves all n^2 terms of the dot-product matrix $\langle x_i, x_j \rangle$. However, it is sufficient to pick $\boldsymbol{u}_t$ from a well chosen finite set of directions [3, appendix], preferably with many zero coefficients. The SMO algorithm [31] exploits this opportunity by only considering feasible directions that only modify two coefficients α_i and α_j by opposite amounts. The most common variant selects the pairs (i, j) that define the successive search directions using a first order criterion:

$$\begin{aligned} i &= \arg\max_s \ \{\ \frac{\partial W}{\partial \alpha_s} \quad \text{s.t.}\ \ \alpha_s\ <\ \max(0,\ y_s\, C)\} \\ j &= \arg\min_s \ \{\ \frac{\partial W}{\partial \alpha_s} \quad \text{s.t.}\ \ \alpha_s\ >\ \min(0,\ y_s\, C)\} \end{aligned} \tag{4}$$

The time required for solving the SVM QP problem grows like n^β with $2 \leq \beta \leq 3$ [3, section 2.1] when the number of examples n increases. Meanwhile the kernel matrix $\langle x_i, x_j \rangle$ becomes too large to store in memory. Computing kernel values on the fly vastly increases the computing time. Modern SVM solvers work around this problem using a cache of recently computed kernel matrix coefficients.

2.2 Learning is easier than optimizing

The SVM quadratic optimization problem (2) is only a sophisticated statistical proxy, defined on the finite training set, for the actual problem, that is classifying future patterns correctly. Therefore it makes little sense to optimize with an accuracy that vastly exceeds the uncertainty that arises from the use of a finite number of training examples.

Online learning algorithms exploit this fact. Since each example is only processed once, such algorithms rarely can compute the optimum of the

objective function. Nevertheless, many online learning algorithms come with formal generalization performance guarantees. In certain cases, it is even possible to prove that suitably designed online algorithms outspeed the direct optimization of the corresponding objective function [5]: they do not optimize the objective function as accurately, but they reach an equivalent test error more quickly.

Researchers therefore have sought efficient online algorithms for kernel machines. For instance, the Budget Perceptron [11] demonstrates that online kernel algorithms should be able to both insert and remove support vectors from the current kernel expansion. The Huller [2] shows that both insertion and deletion can be derived from incremental increases of the SVM objective function $W(\boldsymbol{\alpha})$.

2.3 The LASVM online SVM algorithm

(author?) [3] describe an online SVM algorithm that incrementally increases the dual objective function $W(\boldsymbol{\alpha})$ using feasible direction searches. LASVM maintains a current coefficient vector $\boldsymbol{\alpha}_t$ and the associated set of support vector indices $\mathcal{S}_t$. Each LASVM iteration receives a fresh example $(x_{\sigma(t)}, y_{\sigma(t)})$ and updates the current coefficient vector $\boldsymbol{\alpha}_t$ by performing two feasible direction searches named "*process*" and "*reprocess*".

- *Process* is a SMO direction search (3) along the direction defined by the pair formed with the current example index $\sigma(t)$ and another index chosen among the current support vector indices $\mathcal{S}_t$ using the first order criterion (4). Example $\sigma(t)$ might be a new support vector in the resulting $\boldsymbol{\alpha}'_t$ and $\mathcal{S}'_t$.
- *Reprocess* is a SMO direction search (3) along the direction defined by a pair of indices (i, j) chosen among the current support vectors $\mathcal{S}'_t$ using the first order criterion (4). Examples i and j might no longer be support vectors in the resulting $\boldsymbol{\alpha}_{t+1}$ and $\mathcal{S}_{t+1}$.

Repeating LASVM iterations on randomly chosen training set examples provably converges to the SVM solution with arbitrary accuracy. Empirical evidence indicates that a single presentation of each training example is sufficient to achieve training errors comparable to those achieved by the SVM solution. After presenting all training examples in random order, it is useful to tune the final coefficients by running *reprocess* until convergence.

Online LASVM

1: Initialize $\boldsymbol{\alpha}_0$
2: **while** there are training examples left **do**
3: Select an unseen training example $(x_{\sigma(t)}, y_{\sigma(t)})$
4: *Process*$(\sigma(t))$
5: *Reprocess*
6: **end while**
7: *Finish*: repeat *reprocess* until convergence

This single pass algorithm runs faster than SMO and needs much less memory. Useful orders of magnitude can be obtained by evaluating how large the kernel cache must be to avoid the systematic recomputation of kernel values. Let n be the number of examples and s be the number of support vectors which is smaller than n. Furthermore, let $r \leq s$ be the number of *free support vectors*, that is, support vectors such that $0 < y_i \alpha_i < C$. Whereas the SMO algorithm requires $n\,r$ cache entries to avoid the systematic recomputation of kernel values, the LASVM algorithm only needs $s\,r$ entries [3].

2.4 Selective sampling

Each iteration of the above algorithm selects a training example $(x_{\sigma(t)}, y_{\sigma(t)})$ randomly. More sophisticated example selection strategies yield further scalability improvements. **(author?)** [3] suggest four example selection strategies:

- *Random Selection* : Pick a random unseen training example.
- *Gradient Selection* : Pick the most poorly classified example (smallest value of $y_k\, f(x_k)$) among 50 randomly selected unseen training examples. This criterion is very close to what is done in [25].
- *Active Selection* : Pick the training example that is closest to the decision boundary (smallest value of $|f(x_k)|$) among 50 randomly selected unseen training examples. This criterion chooses a training example independently of its label.
- *Autoactive selection* : Randomly sample at most 100 unseen training examples but stop as soon as 5 of them fall inside the margins (any of these examples would be inserted in the set of support vectors). Pick among these 5 examples the one closest to the decision boundary (smallest value of $|f(x_k)|$.)

Empirical evidence shows that the *active* and *autoactive* criterion yield comparable or better performance level using a smaller number of support

vectors. This is understandable because the linear growth of the number of support vectors is related to fact that soft margin SVMs make a support vector with every misclassified training example. Selecting training examples near the boundary excludes a large number of examples that are uninformative outliers. The reduced number of support vectors further improves both the learning speed and the memory footprint.

The following section presents the challenge set by invariance and discusses how the low memory requirements and the selective sampling capabilities of LASVM are well suited to the task.

3 Invariance

Many pattern recognition problems have invariance properties: the class remains largely unchanged when specific transformations are applied to the pattern. Object recognition in images is invariant under lighting changes, translations and rotation, mild occlusions, etc. Although humans handle invariance very naturally, computers do not.

In machine learning, the a priori knowledge of such invariance properties can be used to improve the pattern recognition accuracy. Many approaches have been proposed [36, 40, 34, 22]. We first propose an illustration of the influence of invariance. Then we describe our selective sampling approach to invariance, and discuss practical implementation details.

3.1 On the influence of invariance

Let us first illustrate how we propose to handle invariance. Consider points in the plane belonging to one of two classes and assume that there is an uncertainty on the point coordinates corresponding to some rotation around the origin. The class labels are therefore expected to remain invariant when one rotates the input pattern around the origin. Figure 3(*a*) shows the points, their classes and a prospective decision boundary. Figure 3(*b*) shows the example orbits, that is, the sets of all possible positions reached by applying the invariant transformation to a specific example. All these positions should be given the same label by the classifier. Figure 3(*c*) shows a decision boundary that takes into account all the potential transformations of the training examples. Figure 3(*d*) shows that this boundary can be obtained by selecting adequate representatives for the orbits corresponding to each training example.

This simple example illustrates the complexity of the problem. Learning orbits leads to some almost intractable problems [17]. Adding virtual examples

[34] requires considerable memory to simply store the transformed examples in memory. However, figure 3(*d*) suggests that we do not need to store all the transformed examples forming an orbit. We only need to add a few well chosen transformed examples.

The LASVM algorithm (section 2) displays interesting properties for this purpose. Because LASVM is an online algorithm, it does not require storing all the transformed examples. Because LASVM uses selective sampling strategies, it provides the means to select the few transformed examples that we think are sufficient to describe the invariant decision boundaries. We therefore hope to solve problems with multiple invariance with milder size and complexity constraints.

Our first approach was inspired by [24]. Each iteration randomly picks the next training example, generates a number of transformed examples describing the orbit of the original example, selects the best transformed example (see section 2.4), and performs the LASVM *process*/*reprocess* steps.

Since the online algorithm never revisits a previously seen training example, this first approach cannot pick more than one representative transformed example from each orbit. Problems with multiple invariance are likely to feature complicated orbits that are poorly summarized using a single transformed example. This major drawback can be remedied by revisiting training examples that have generated interesting variations in previous iterations. Alas this remedy requires to either recompute the example transformations, or to store all of them in memory.

Our final approach simply considers a huge virtual training set composed of all examples and all their transformation. Each iteration of the algorithm picks a small sample of randomly transformed training examples, selects the best one using one of the criteria described in section 2.4, and performs the LASVM *process*/*reprocess* steps.

This approach can obviously select multiple examples for each orbit. It also provides great flexibility. For instance, it is interesting to boostrap the learning process by first learning from untransformed examples. Once we have a baseline decision function, we can apply increasingly ambitious transformations.

3.2 Invariance in practice

Learning with invariance is a well studied problem. Several papers explain how to efficiently apply pattern transformations [36, 40, 34]. We use the MNIST database of handwritten digit images because many earlier results have been reported [21]. This section explains how we store the original

images and how we efficiently compute random transformations of these digits images on the fly.

3.2.1 Tangent vectors

(author?) [35] explains how to use Lie algebra and tangent vectors to apply arbitrary affine transformations to images. Affine transformations can be described as the composition of a six elementary transformations: horizontal and vertical translations, rotations, horizontal and vertical scale transformations, and hyperbolic transformations. For each image, the method computes a tangent vector for each elementary transformation, that is, the normalized pixel-wise difference between an infinitesimal transformation of the image and the image itself.

Small affine transformation are then easily approximated by adding a linear combination of these six elementary tangent vectors:

$$x_{aff}(i,j) = x(i,j) + \sum_{T \in \mathcal{T}} \alpha_T t_T(i,j)$$

where $\mathcal{T}$ is the set of elementary transformations, $x(i,j)$ represents the initial image, $t_T(i,j)$ is its tangent vector for transformation T, and α_T represents the coefficient for transformation T.

3.2.2 Deformation fields

It turns out that the tangent vector for the six elementary affine transformations can be derived from the tangent vectors $t_x(i,j)$ and $t_y(i,j)$ corresponding to the horizontal and vertical translations. Each elementary affine transformation can be described by a vector field $[f_x^T(i,j), f_y^T(i,j)]$ representing the displacement direction of each pixel (i,j) when one performs an infinitesimal elementary transformation T of the image. The tangent vector for transformation T is then:

$$t_T(i,j) = f_x^T(i,j)\, t_x(i,j) + f_y^T(i,j)\, t_y(i,j)$$

This property provides for extending the tangent vector approach from affine transformations to arbitrary *elastic transformations*, that is, transformations that can be represented by a deformation field $[\ f_x(i,j),\ f_y(i,j)\ ]$. Small transformations of the image $x(i,j)$ are then easily approximated using a linear operation:

$$x_{deformed}(i,j) = x(i,j) + \alpha * (f_x(i,j) * t_x(i,j) + f_y(i,j) * t_y(i,j))$$

Plausible deformation fields $[f_x(i,j), f_y(i,j)]$ are easily generated by randomly drawing an independent motion vector for each pixel and applying a smoothing filter. Figure 4 shows examples of such deformation fields. Horizontal and vertical vector components are generated independently and therefore can be used interchangeably.

We also generate transformation fields by adding a controlled amount of random smoothed noise to a transformation field representing a pure affine transformation. This mainly aims at introducing more rotations in the transformations.

3.2.3 Thickening

The horizontal and vertical translation tangent vectors can also be used to implement a thickening transformation [35] that erodes or dilates the digit images.

$$x_{thick}(i,j) = x(i,j) + \beta * \sqrt{t_x(i,j)^2 + t_y(i,j)^2},$$

where β is a coefficient that controls the strength of the transformation. Choosing $\beta < 0$ makes the strokes thinner. Choosing $\beta > 0$ makes them thicker.

3.2.4 The infinite virtual training set

As discussed before, we cannot store all transformed example in memory. However we can efficiently generate random transformations by combining the above methods:

$$\begin{aligned} x_{trans}(i,j) &= x(i,j) + \alpha_x * f_x(i,j) * t_x(i,j) \\ &+ \alpha_y * f_y(i,j) * t_y(i,j) \\ &+ \beta * \sqrt{t_x(i,j)^2 + t_y(i,j)^2} \end{aligned}$$

We only store the initial images $x(i,j)$ along with its translation tangent vectors $t_x(i,j)$ and $t_y(i,j)$. We also store a collection of pre-computed deformation fields that can be interchangeably used as $f_x(i,j)$ or $f_y(i,j)$. The scalar coefficients α_x, α_y and β provide further transformation variability.

Figure 6 shows examples of all the combined transformations. We can generate as many examples as we need this way, playing on the choice of deformation fields and scalar coefficients.

3.2.5 Large translations

All the transformations described above are small sub-pixel transformations. Even though the MNIST digit images are roughly centered, experiments indicate that we still need to implement invariance with respect to translations of magnitude one or two pixels. Thus we also apply randomly chosen translations of one or two pixels. These full-pixel translations come on top of the sub-pixel translations implemented by the random deformation fields.

4 Application

This section reports experimental results achieved on the MNIST database using the techniques described in the previous section. We have obtained state-of-the-art results using 10 SVM classifiers in one-versus-rest configuration. Each classifier is trained using 8 million transformed examples using the standard RBF kernel $< x, x' >= \exp(-\gamma\|x - x'\|^2)$. The soft-margin C parameter was always 1000.

As explained before, the untransformed training examples and their two translation tangent vectors are stored in memory. Transformed exemples are computed on the fly and cached. We allowed 500MB for the cache of transformed examples, and 6.5GB for the cache of kernel values. Indeed, despite the favorable characteristics of our algorithm, dealing with millions of examples quickly yields tens of thousands support vectors.

4.1 Deformation settings

One could argue that handling more transformations always increases the test set error. In fact we simply want a classifier that is invariant to transformations that reflect the typical discrepancies between our training examples and our test examples. Making the system invariant to stronger transformations might be useful in practice, but could reduce the performance on our particular test set.

We used cross-validation to select both the kernel bandwidth parameter and to select the strength of the transformations applied to our initial examples. The different parameters of the cross validation are the deformation strength α and the RBF kernel bandwidth γ for the RBF kernel. During the same time, we have also estimated whether the thickening transform and the 1 or 2 pixel translations are desirable.

Figure 7 reports the SVM error rates measured for various configurations on a validation set of 10,000 points taken from the standard MNIST training

set. The training set was composed by picking 5,000 other points from the MNIST training set and applying 10 random transformations to each point. We see that thickening is not a relevant transformation for the MNIST problem. Similarly, we observe that 1 pixel translations are very useful, and that it is not necessary to use 2 pixels translations.

4.2 Example selection

One of the claims of our work is the ability to implement invariant classifiers by selecting a moderate amount of transformed training examples. Otherwise the size of the kernel expansion would grow quickly and make the classifier impractically slow.

We implemented the example selection criteria discussed in section 2.4. Figure 8 compares error rates (left), number of support vectors (center), and training times (right) using three different selection criteria: random selection, active selection, and auto-active selection. These results were obtained using 100 random transformations of each of the 60000 MNIST training examples. The graphs also show the results obtained on the 60000 MNIST training examples without random transformations.

The random and auto-active selection criteria give the best test errors. The auto-active criterion however yields a much smaller number of support vectors and trains much faster. Therefore we chose the auto-active selection criterion for the following experiments.

4.3 The finishing step

After presenting the last training example, **(author?)** [3] suggest to tune the final solution by running *reprocess* until convergence. This amounts to optimizing the SVM objective function restricted to the remaining support vectors. This operation is known as the "finishing step". In our case, we never see the last training examples since we can always generate more.

At first, we simply eliminated this finishing step. However we noticed that after processing a large amount of examples (between 5 and 6 millions depending on the class) the number of support vectors decreases slowly. There is in fact an implicit finishing step. After a sufficient time, the *process* operation seldom does anything because hardly any of the selected examples needs to be added to the expansion. Meanwhile the *reprocess* step remains active.

We then decided to perform a finishing step every 600,000 points. We observed the same the reduction of the number of support vectors, but

earlier during the learning process (see figure 9). These additional finishing steps seem useful for the global task. We achieved the best results using this setup. However this observation raises several questions. Is it enough to perform a single *reprocess* step after each example selection? Can we get faster and better results?

4.4 The number of *reprocess*

As said before, the LASVM algorithm does not explicitly defines how much optimization should be performed after processing each example. To explore this issue, we ran several variants of the LASVM algorithms on 5 random transformations of 10,000 MNIST examples.

	1R/1P	**2R/1P**	**3R/1P**	**4R/1P**	**5R/1P**
Max size	24531	21903	21436	20588	20029
Removed pts	1861	1258	934	777	537
Proportion	7.6%	5.7%	4.3%	3.7%	2.6%
Train time (s)	1621	1548	1511	1482	1441
Error rate	2.13%	2.08%	2.19%	2.09%	2.07%

	1R each	**2R each**	**3R each**
Max size	23891	21660	20596
Removed pts	1487	548	221
Proportion	6.2%	2.5%	1.0%
Train time (s)	1857	1753	1685
Error rate	2.06%	2.17%	2.13%

Table 1: Effects of transformations on the performance, with an rbf kernel of bandwidth $\gamma = 0.006$. The table shows a comparison for different trade-off between Process and Reprocess. We change the number of consecutive Reprocess after a Process, and also after each coming point, even if it is not Processed.

The variants denoted "nR/1P" consist of performing n *reprocess* steps after selecting a transformed training example and performing a *process* step. The variants denoted "nR each" consist of performing n *reprocess* steps after each examples, regardless of whether the example was selected for a *process* step.

Table 1 shows the number of support vectors before and after the finishing step, the training times, and the test performance measured on an independent validation set of 10,000 examples. Neither optimizing a lot (last column) or optimizing very little (first column) are good setups. In terms of training time, the best combinations for this data set are "4R/1P" and "5R/1P". This results certainly shows that achieving the right balance remains an obscure aspect of the LASVM algorithm.

4.5 Final results

Number of binary classifiers	10
Number of examples for each binary classifier	8,100,000
Thickening transformation	no
Additional translations	1 pixel
RBF Kernel bandwidth (γ)	0.006
Example selection criterion	auto-active
Finishing step	every 600,000 examples
Full training time	**8 days**
Test set error	**0.67%**

Table 2: Summary of our final experiment.

Table 2 summarizes our final results. We first bootstrapped the system using the original MNIST training and 4 random deformation of each example. Then we expanded the database with 130 further random transformations, performing a finishing step every 600,000 examples. The final accuracy matches the results obtained using virtual support vectors [33] on the original MNIST test set. Slightly better performances have been reported using convolution networks [37], or using a deskewing algorithm to make the test set easier [33].

5 Conclusion

We have shown how to address large invariant pattern recognition problems using selective sampling and online algorithms. We also have demonstrated that these techniques scale remarkably well. It is now possible to run SVM on millions of examples in a relatively high dimension input space (here 784), using a single processor. Because we only keep a few thousands of support vectors per classifier that we can handle millions of training examples.

References

[1] K. P. Bennett and C. Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explorations*, 2(2):1–13, 2000.

[2] A. Bordes and L. Bottou. The Huller: a simple and efficient online svm. In *Machine Learning: ECML 2005*, Lecture Notes in Artificial Intelligence, LNAI 3720, pages 505–512. Springer Verlag, 2005.

[3] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

[4] L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[5] L. Bottou and Y. LeCun. Large scale online learning. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[6] C. Campbell, N. Cristianini, and A. Smola. Query learning with large margin classifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 111–118. Morgan Kaufmann, San Francisco, CA, 2000.

[7] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[8] O. Chapelle and B. Schölkopf. Incorporating invariances in nonlinear svms. In Dietterich T. G.and Becker S. and Ghahramani Z., editors, *Advances in Neural Information Processing Systems*, volume 14, pages 609–616, Cambridge, MA, USA, 2002. MIT Press.

[9] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 705–712. The MIT Press, 1995.

[10] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:pp 1–25, 1995.

[11] K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[12] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In J. Shavlik, editor, *Machine Learning: Proceedings of the Fifteenth International Conference*, San Francisco, CA, 1998. Morgan Kaufmann.

[13] T. Frieß, N. Cristianini, and C. Campbell. The kernel Adatron algorithm: a fast and simple learning procedure for support vector machines. In J. Shavlik, editor, *15th International Conf. Machine Learning*, pages 188–196. Morgan Kaufmann Publishers, 1998.

[14] K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988.

[15] C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

[16] G. H. Golub and C. F. Van Loan. *Matrix Computation*. John Hopkins University Press, 1991. Second Edition.

[17] T. Graepel and R. Herbrich. Invariant pattern recognition by semi-definite programming machines. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[18] T. Joachims. Making large-scale SVM learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advanced in Kernel Methods - Support Vector Learning*, pages 169–184. MIT Press, 1999.

[19] J. Kivinen, A. Smola, and R. Williamson. Online learning with kernels, 2002.

[20] K. J. Lang and G. E. Hinton. A time delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, Carnegie-Mellon University, Pittsburgh PA, 1988.

[21] Y. Le Cun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. `http://yann.lecun.com/exdb/mnist/`.

[22] T. K. Leen. From data distributions to regularization in invariant learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 223–230. The MIT Press, 1995.

[23] Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Mach. Learn.*, 46(1-3):361–387, 2002.

[24] G. Loosli, S. Canu, S. V. N. Vishwanathan, and A. J. Smola. Invariances in classification : an efficient svm implementation. In *ASMDA 2005 - Applied Stochastic Models and Data Analysis*, 2005.

[25] G. Loosli, S. Canu, S. V. N. Vishwanathan, A. J. Smola, and M. Chattopadhyay. Une boîte à outils rapide et simple pour les SVM. In Michel Liquière and Marc Sebban, editors, *CAp 2004 - Conférence d'Apprentissage*, pages 113–128. Presses Universitaires de Grenoble, 2004.

[26] N. Murata and S.-I. Amari. Statistical analysis of learning dynamics. *Signal Processing*, 74(1):3–28, 1999.

[27] J.-P. Nadal. *Réseaux de neurones: de la physique à la psychologie*. Armand Colin, Collection 2aI, 1993.

[28] A. Nemirovski. Introduction to convex programming, interior point methods, and semi-definite programming. Machine Learning, Support Vector Machines, and Large-Scale Optimization Pascal Workshop, March 2005.

[29] A. B. J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622. Polytechnic Institute of Brooklyn, 1962.

[30] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm fo support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop*, pages 276–285, 1997.

[31] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

[32] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 839–846. Morgan Kaufmann, June 2000.

[33] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, 2001.

[34] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In J.C. Vorbrüggen C. von der Malsburg, W. von Seelen and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, volume 1112, pages 47–52, Berlin, 1996. Springer Lecture Notes in Computer Science.

[35] P. Simard, Y. Le Cun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. *International Journal of Imaging Systems and Technology*, 11(3), 2000.

[36] P. Simard, Y. Le Cun, and Denker J. efficient pattern recognition using a new transformation distance. In S. Hanson, J. Cowan, and L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufmann, 1993.

[37] P. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, pages 958–962, 2003.

[38] I. Steinwart. Sparseness of support vector machines—some asymptotically sharp bounds. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[39] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48(1-3):165–187, 2002.

[40] J. Wood. Invariant pattern recognition: A review. *Pattern Recognition*, 29 Issue 1:1–19, 1996.

[41] G. Zoutendijk. *Methods of Feasible Directions*. Elsevier, 1960.

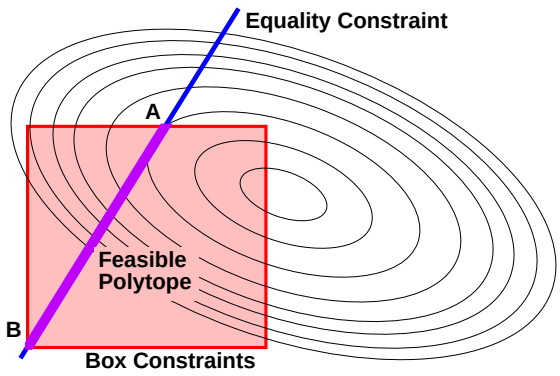

Figure 1: Geometry of the SVM dual QP problem (2). The box constraints $0 \leq y_i \alpha_u \leq C$ restrict the solutions to a n–dimensional hypercube. The equality constraint $\sum \alpha_i = 0$ further restricts the solutions to a $(n-1)$–dimensional polytope, that is segment [AB] in this 2-dimensional figure.

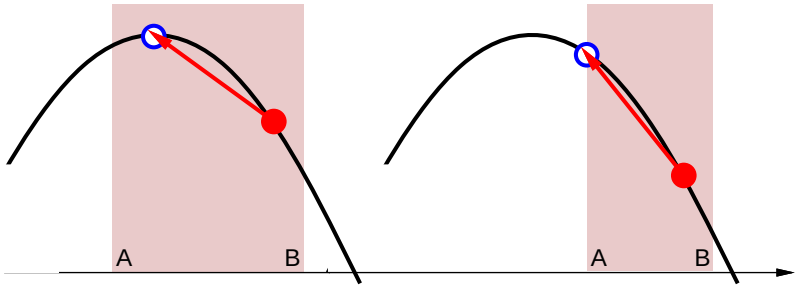

Figure 2: Performing a line search inside the feasible space of the SVM dual QP problem. The quadratic cost function restricted to the search line might reach its maximum inside (left) or outside the box constraints (right.)

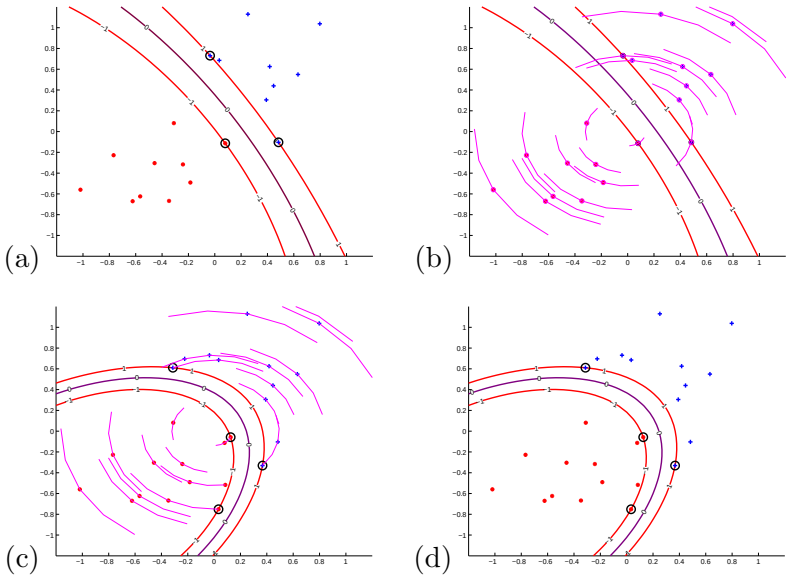

Figure 3: Those figures illustrate the influence of point variations on the decision boundary for a toy example. Plot (*a*) shows a typical decision boundary obtained by only considering the training examples. Plot (*b*) shows "orbits" describing the possible transformations of the training examples. All these variations should be given the same label by the classifier. Plot (*c*) shows a decision boundary that takes into account the variations. Plot (*d*) shows how this boundary can be obtained by selecting adequate representants for each orbit.

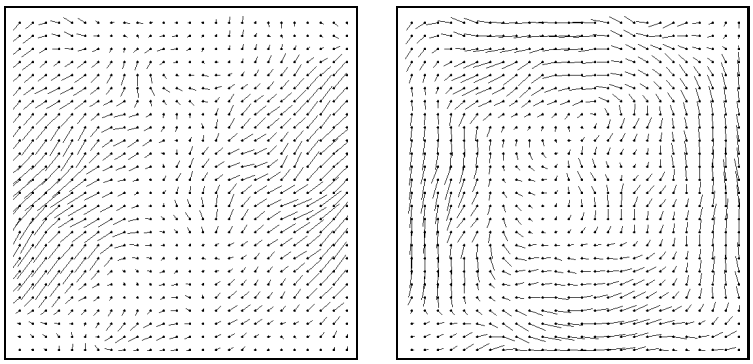

Figure 4: This figure shows examples of deformation fields. We represent the combination of horizontal and vertical fields. The first one is smoothed random and the second one is a rotation field modified by random noise.

Figure 5: This figure shows the original digit, the two translation tangent vectors and the thickening tangent vector.

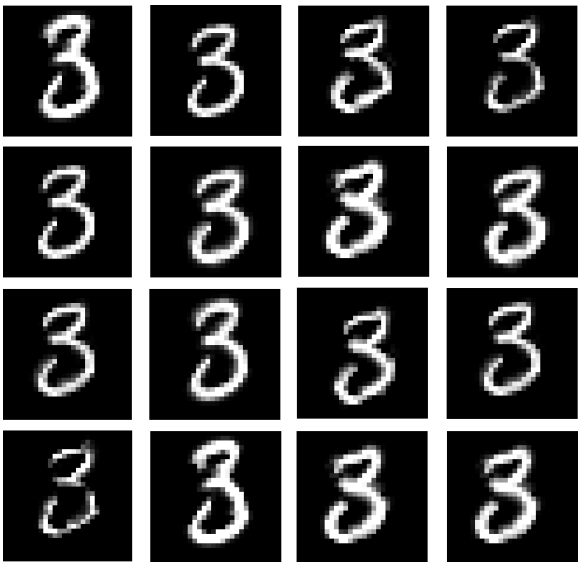

Figure 6: This figure shows 16 variations of a digit with all the transformations cited here.

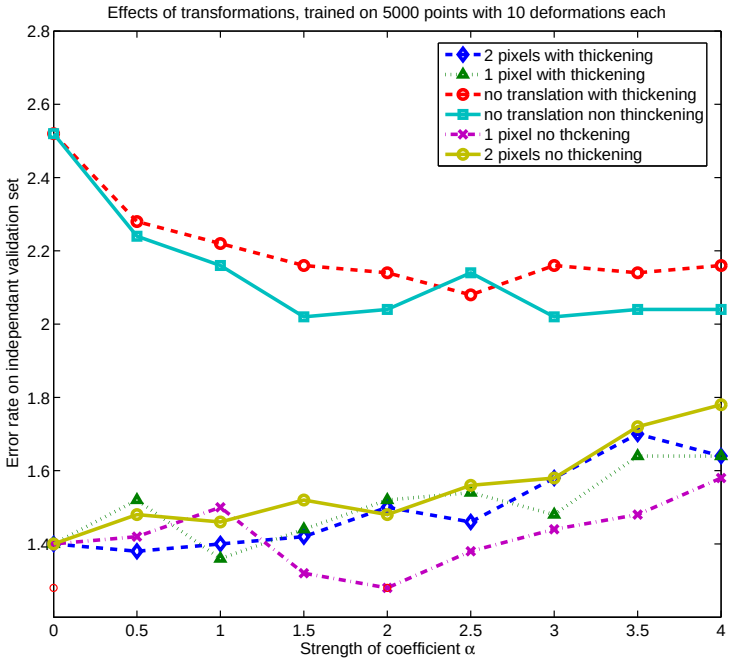

Figure 7: Effects of transformations on the performance. This graph is obtained on a validation set of 10,000 points, trained on 5000 points and 10 transformations for each (55,000 training points in total) with an rbf kernel with bandwidth $\gamma = 0.006$. The best configuration is elastic deformation without thickening, for $\alpha = 2$ and translations of 1 pixel, which gives 1.28%. Note that $\alpha = 0$ is equivalent to no elastic deformation. The baseline results for the validation set is thus 2.52%.

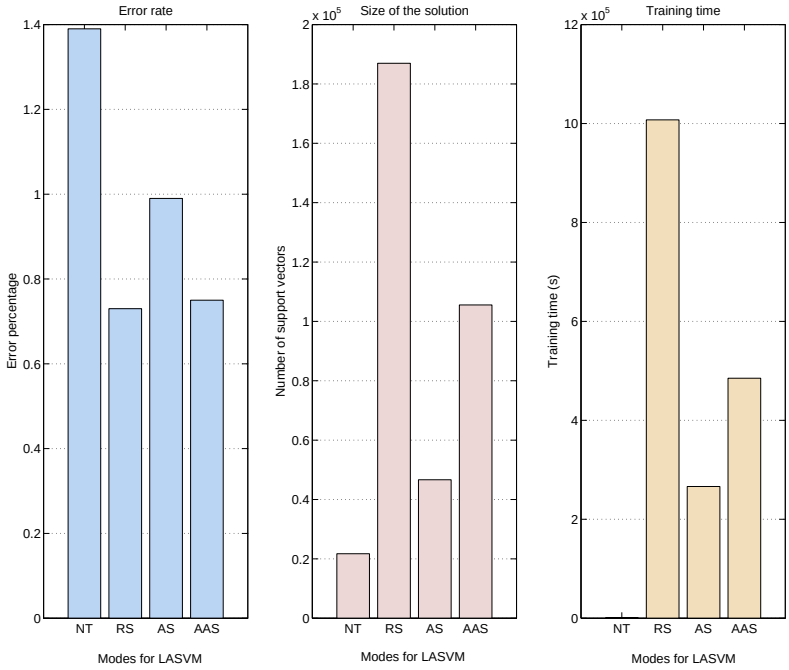

Figure 8: This figures compare the error rates (left), the numbers of support vectors (center) and the training times (right) of different LASVM runs. The first bar of each graph corresponds to the training on the original 60,000 MNIST examples (no transformation - NT). The others three bars were obtained using 100 random deformations of each MNIST example, that is 6 millions points. The second columns reports results for random selection (RS), the thirds for active selection (AS) and the last ones for auto-active selection (AAS) The deformation settings are set according to previous results (figure 7). The auto-active run gives the best compromise.

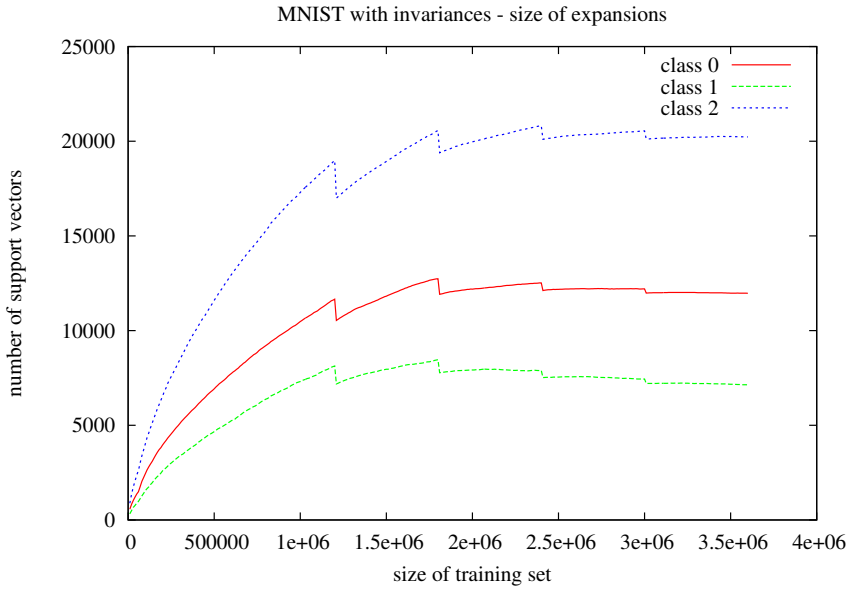

Figure 9: This figure shows the evolution of the expansion size during training. We used auto-active selection and performing a finishing step at regular intervals Each gap corresponds to one finishing step. Here we notice that the number of support vectors eventually decreases without fully optimizing, at least intentionally.