Importance Sampling Tree for Large-scale Empirical Expectation

Olivier Canévet[*] OLIVIER.CANEVET@IDIAP.CH
Cijo Jose[*] CIJO.JOSE@IDIAP.CH
François Fleuret FRANCOIS.FLEURET@IDIAP.CH
Idiap Research Institute, Martigny, Switzerland
École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Abstract

We propose a tree-based procedure inspired by the Monte-Carlo Tree Search that dynamically modulates an importance-based sampling to prioritize computation, while getting unbiased estimates of weighted sums. We apply this generic method to learning on very large training sets, and to the evaluation of large-scale SVMs.

The core idea is to reformulate the estimation of a score – whether a loss or a prediction estimate – as an empirical expectation, and to use such a tree whose leaves carry the samples to focus efforts over the problematic "heavy weight" ones.

We illustrate the potential of this approach on three problems: to improve Adaboost and a multi-layer perceptron on $2D$ synthetic tasks with several million points, to train a large-scale convolution network on several millions deformations of the CIFAR data-set, and to compute the response of a SVM with several hundreds of thousands of support vectors. In each case, we show how it either cuts down computation by more than one order of magnitude and/or allows to get better loss estimates.

1. Introduction

Virtually every single machine learning algorithm relies on data-based empirical expectations, either to estimate a loss, or the response of a predictor (see the examples in § 3.1). The larger the data-set, the more accurate the prediction, and many state-of-the-art results have been obtained by enriching already very large sets using synthetic perturbations, resulting in hundreds of millions of labelled samples (Krizhevsky et al., 2012).

An empirical expectation takes the form of a sum of a quantity evaluated on many data-points, and in practice most of these summed terms are negligible. In training it is because most of the samples are far from the boundary between populations, and get a "trivially correct answer", in test it is because the prediction on a test point is modulated only by its immediate neighbors in the training set. For example in the case of Gaussian kernel SVM trained on large data-sets, even though the model has very large number of support vectors only few of them will actually matter in the final decision score.

Despite this well known state of affairs, algorithms still rely on an exhaustive loop through the samples. Some approaches have been developed to prioritize samples *after they have already been seen* (Kalal et al., 2008; Fleuret & Geman, 2008), or in subsets sampled uniformly (Loosli et al., 2007), but they do not use structures given *a priori* over the said samples, combining it with statistical observations made over those already observed to reject groups without looking at them.

In some sense, the problem at hand is related to the choice of an optimal move in a strategy game. This has been tackled traditionally with branch-and-bound approaches, going down the tree of possible choices, and discarding sub-trees that can be proven to be bad. This type of methods however founds its limits with games of very large combinatorial complexity. State-of-the-art performance for the game of Go for instance is obtained with Monter-Carlo Tree Search (Gelly et al., 2006) which at the same time samples and optimizes the sampling over configurations.

The technique we propose, dubbed as IST for "Importance Sampling Tree", similarly, at the same time, samples leaves, provides a correcting factor to compensate for its sampling bias, and optimizes inner statistical estimates to improve sampling over time.

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.
[*] Authors contributed equally to the paper

2. Related works

Batch learning becomes very difficult or even infeasible when it comes to training predictors on very large datasets. Therefore, sub-sampling appears to be one solution to make it practical and it has been shown that a smart sampling, as opposed to a uniform sampling, has an impact on the performance of the final classifier. For instance the Boosting procedures proposed by Fleuret & Geman (2008), and Kalal et al. (2008) select samples to train a weak learner based on their boosting weights. The classifier is thus presented with highly misclassified samples, with a high individual weight, but also with representatives of populations of low weighted individual samples which have have a large cumulative weight.

Bordes et al. (2005) proposed LASVM, an online algorithm with importance sampling to train kernel support vector machines. They show that importance sampling reduces the training time and also results in models which are compact with fewer support vectors, while retaining equivalent or superior accuracy compared to standard algorithms.

Recently, importance sampling has been studied in the context of stochastic gradient descent (SGD) algorithms (Zhao & Zhang, 2014). Their key observation is that, though sampling observations uniformly at random from the training set results in a stochastic gradient which is an unbiased estimate of the true gradient, this resulting estimator may have high variance. In order to mitigate the convergence issue with high variance they propose an importance sampling scheme. Their theoretical results shows that under certain conditions, importance sampling can improve the convergence rate of SGD algorithms.

More generally a sampling approach called Monte Carlo Tree Search (Browne et al., 2012) has gain much interest in the past years for the tremendous improvement for games such as Computer Go. The purpose of MCTS is to find the optimal solution in a potential huge space organised as a tree by sampling this tree. The tree is traversed from top to bottom by recursively applying a multi-armed bandit on the children of the current node until reaching a leaf. A reward related to the optimal solution is then obtained and the outcome is backpropagated up the the root. The next samplings will use the accumulated statistics to prioritize the sampling towards promising branches and eventually find the optimal value.

Contrary to what is the core purpose of MCTS, we are not interested in finding the best leaf or leaves, but to sample among all the leaves, according to their weights. These two objectives are very distinct when an important fraction of the total weight comes from a large population of low-weighting leaves.

3. Method

3.1. Weighted sums for prediction

Given N positive weights $w_n \in \mathbb{R}_+,\ n = 1, \dots, N$ and a function $f : \{1, \dots, N\} \to \mathbb{R}^D$, we are interested in the weighted average

$$\sum_{n=1}^{N} w_n f(n) \tag{1}$$

when N is too large to allow an exhaustive visit of the weights. Our motivation is that we can express under that form most of the data-driven important quantities in machine learning, such as

– The edge of a weak-learner in Adaboost

$$\sum_n \underbrace{\exp(-y_n \psi(x_n))}_{w_n} \underbrace{y_n h(x_n)}_{f(n)}. \tag{2}$$

– The gradient for training a neural network

$$\sum_n \nabla_\alpha l(\psi(x_n;\alpha), y_n) = \sum_n \underbrace{\|\nabla_\alpha l(\psi(x_n;\alpha), y_n)\|}_{w_n} \underbrace{\frac{\nabla_\alpha l(\psi(x_n;\alpha), y_n)}{\|\nabla_\alpha l(\psi(x_n;\alpha), y_n)\|}}_{f(n)}. \tag{3}$$

– The evaluation of a SVM in its dual form

$$\sum_n \alpha_n y_n k(x_n, x) = \sum_n \underbrace{\alpha_n k(x_n, x)}_{w_n} \underbrace{y_n}_{f(n)}. \tag{4}$$

In these examples, a weight can be interpreted as the "importance" of a sample, and in many practical situations, the vast majority of them are negligible. We aim at devising an approach – relying on a prior tree structure on the weights – that (1) balances computation proportionally to the weights themselves, and (2) does so by looking at a fraction of the full family of weights, opening the way to extremely large samples sets.

3.2. Importance sampling for Monte-carlo simulations

Given an arbitrary distribution μ on $\{1, \dots, N\}$ which puts non-zero probabilities on all the values, we can rewrite Equation (1) as

$$\sum_n \mu(n) \frac{w_n}{\mu(n)} f(n) = E_{\mathbf{N}\sim\mu}\left[\frac{w_{\mathbf{N}}}{\mu(\mathbf{N})} f(\mathbf{N})\right] \tag{5}$$

which we can approximate by generating $\mathbf{N}_1, \dots, \mathbf{N}_K$ i.i.d $\sim \mu$, and using the empirical expectation

$$\hat{E}_{n\sim\mu}\left[\frac{w_n}{\mu(n)} f(n)\right] = \frac{1}{K}\sum_{k=1}^{K} \frac{w_{\mathbf{N}_k}}{\mu(\mathbf{N}_k)} f(\mathbf{N}_k). \tag{6}$$

Table 1. Notation

– $w_n \in \mathbb{R}_+$ positive weights we want to approximate through sampling
– $\mathcal{T}, \mathcal{L} \subset \mathcal{T}$ nodes and leaves of our sample tree
– D the depth of the tree
– $c_0(x), c_1(x)$ children of node x
– U minimum number of observations we impose before biasing the θ_x
– $\mathcal{L}(x) \subset \mathcal{L}$ set of leaves of the sub-tree whose root is x
– $n(x) \in \{1, \dots, N\}$ index of the weight at leaf x
– $\Theta = (\theta_x)_{x \in \mathcal{T} \setminus \mathcal{L}}$ probabilities to chose the right sub-tree at each node during sampling
– $\mu_\Theta(n; x)$ probability to reach leaf n given that the sampling passes through node x
– $w(x) = \sum_{y \in \mathcal{L}(x)} w_{n(y)}$ sum of the weights of the leaves in the sub-tree whose root is x
– $\nu(x)$ number of times the sampling has been through node x
– $s(x)$ sum of the individual estimates of $w(x)$
– $\hat{w}_x = s(x)/\nu(x)$ estimate of $w(x)$
– $\mathcal{S}(x)$ statistics accumulated at node x

The most natural choice for μ would be to use $\mu(n) = \frac{w_n}{\sum_k w_k}$, in which case we would have our desired property of investing the computation proportionally to the weights, and minimize optimally the variance of our estimator.

This choice makes sense if computing $\sum_k w_k$ is tractable, or so cheap to compute compared to the computation of the $f(n)$s that it still provides a substantial gain. However, we are interested in situations where not only w_n is more expensive to compute than $f(n)$, but we also aim at scaling N up to values far greater than the number of CPU operations we have at our disposal.

3.3. Importance Sampling Tree (IST)

We propose a novel structure dubbed as Importance Sampling Tree (IST), which is a binary tree carrying the weights $w_1, \dots, w_N$ at its leaves, and having at each internal node statistics about the weights in the leaves below it. Given such a tree, we use a recursive sampling procedure that results in a sampling of the leaves, and – in a manner similar to the MCTS – we update the estimates at the nodes every time a sampling is done, and modulate the sampling policy accordingly.

Let $\mathcal{T}$ be the set of tree nodes, $x^* \in \mathcal{T}$ the root node, $\mathcal{L} \subset \mathcal{T}$ the leaves. Since the leaves carry the weights $w_1, \dots, w_N$, for any leaf $x \in \mathcal{L}$ let $n(x) \in \{1, \dots, N\}$ be the index of the weight there. Note that we will often make a confusion between $x \in \mathcal{L}$ and $n(x)$, identifying a leaf with its index.

For any internal node $x \in \mathcal{T} \setminus \mathcal{L}$, let $c_0(x), c_1(x) \in \mathcal{T}$ be its two child nodes. Finally let $\mathcal{L}(x)$ be the leaves of the sub-tree starting at x and $w(x)$ the sum of their weights. Hence, in particular $\forall\, x \in \mathcal{L}, w(x) = w_{n(x)}$.

Given a family of "bifurcation probabilities" $\Theta = (\theta_x)_{x \in \mathcal{T} \setminus \mathcal{L}} \in]0,1[^{\|\mathcal{T} \setminus \mathcal{L}\|}$, that is for each node the probability to "go down on the right", we can derive for each node x and each leaf n a probability $\mu_\Theta(n; x)$ to reach the leaf n if we start from x and follow the θs at each node we meet. This probability is the product of the probabilities of the bifurcations to go from x to n. Given a leaf n, the $\mu_\Theta(n; x)$ for all the parents x of n can be computed in $O(D)$.

3.3.1. Adaptive sampling

We could use many different policies to modulate the θ_x and bias the sampling according to what we have observed. We propose two strategies, both based on accumulating at every node statistics $\mathcal{S}(x)$ about the weights observed during the previous sampling:

Using empirical weights – Here $\mathcal{S}(x) = (s(x), \nu(x))$ where $s(x)$ is the sum of the weight estimates $w_n/\mu_\Theta(n; x)$, and $\nu(x)$ is the number of times the sampling went through x. For $\nu(x) > 0, \hat{w}(x) = s(x)/\nu(x)$ is an unbiased estimator of $w(x)$. We set θ_x to the ratio of the number of leaves in the child sub-trees if we do not have enough sampling for proper estimations, and to the ratio of the estimations of the weights otherwise. Formally with U a meta-parameter setting the minimum number of observations we request for biasing:

$$\theta_x = \begin{cases} \dfrac{\|\mathcal{L}(c_1(x))\|}{\|\mathcal{L}(c_0(x))\| + \|\mathcal{L}(c_1(x))\|} & \text{if } \nu(c_0(x)) < U \\ & \text{or } \nu(c_1(x)) < U, \\ \dfrac{\hat{w}(c_1(x))}{\hat{w}(c_0(x)) + \hat{w}(c_1(x))} & \text{otherwise.} \end{cases} \tag{7}$$

Using weight intervals – Here $\mathcal{S}(x) = (l(x), u(x))$, corresponding to lower and upper bounds on $w(x)$, which are updated and getting tighter after every sampling. We define $\Phi(x)$, a function of the upper bound $u(x)$ and lower bound $l(x)$ at node x and we set θ_x to be

$$\theta_x = \frac{\Phi(c_1(x))}{\Phi(c_0(x)) + \Phi(c_1(x))} \tag{8}$$

In the experiment section § 4.3, we compare several definitions of Φ.

Hence, if we need T samples, for $t = 1, \dots, T$:

1. Recursively sample a path down the tree to a leaf n^t, according to $\mu_{\Theta^t}(\,.\,; x^*)$.

2. For every node x visited on that path compute $\mathcal{S}^{t+1}(x)$ and θ_x^{t+1} according to Equation (7) or (8) depending on the application. All other nodes remain unchanged.

Then, following § 3.2, we can use

$$\sum_{n=1}^{N} w_n f(n) \simeq \frac{1}{T} \sum_{t=1}^{T} \frac{w_{n^t}}{\mu_{\Theta^t}(n^t; x^*)} f(n^t). \qquad (9)$$

Note that the Θ^ts in that expression – and the resulting μs – are the ones that were in the tree when each sampling was done. Also, note that the update of $\mathcal{S}$ and Θ can be delayed, suppressed, or done in an arbitrary order if implementation constraints impose it.

4. Experiments and results

4.1. Adaboost on a $2D$ synthetic problem

To get a first idea of the behavior of the IST in practice, we use Adaboost on a simple synthetic problem. This algorithm is a very good candidate since the exponential loss is known to induce strongly unbalanced sample weights. Our objective is to assess if the IST is able to focus the sampling efficiently enough to cope with the divergence of the loss that is classically observed in validation.

The synthetic task is a classification problem where the signal X is uniform in the unit square $[0, 1]^2$ and the class Y is $+1$ in a disc centered in that square and -1 elsewhere. The total sample set we consider contains 8192^2 points located on a regular grid in the unit square. The binary tree structure we use for the IST (as described in § 3.3) recursively splits the x and the y axes, and has a depth of 27, corresponding to 13 splits in each directions and one level for the leaves.

We consider a standard Adaboost based on the exponential loss as in Equation (2), and linear stumps. Each stump is trained by sampling uniformly 100 directions in the $2D$ plan, and optimizing exactly its bias, and its weight in the strong predictor.

We test in our experiments three algorithms: **Boost** is the baseline. It samples $1,000$ samples uniformly initially, and uses them as a standard training set for all the stumps. **Boost-U** re-samples uniformly $1,000$ new training samples for each stump. **Boost-IST** samples $1,000$ training samples with the IST for each stump, and updates the IST with their Adaboost weights. The same tree is used for all the stumps, and not reinitialized for each.

For all variants, we also sample $1,000$ samples uniformly initially as a validation set. The results are illustrated in Figure 1. To be consistent with the rest of the article, losses correspond to estimates on the total sample set of 2^{26} samples, hence the initial value of $\simeq 6.7e7$.

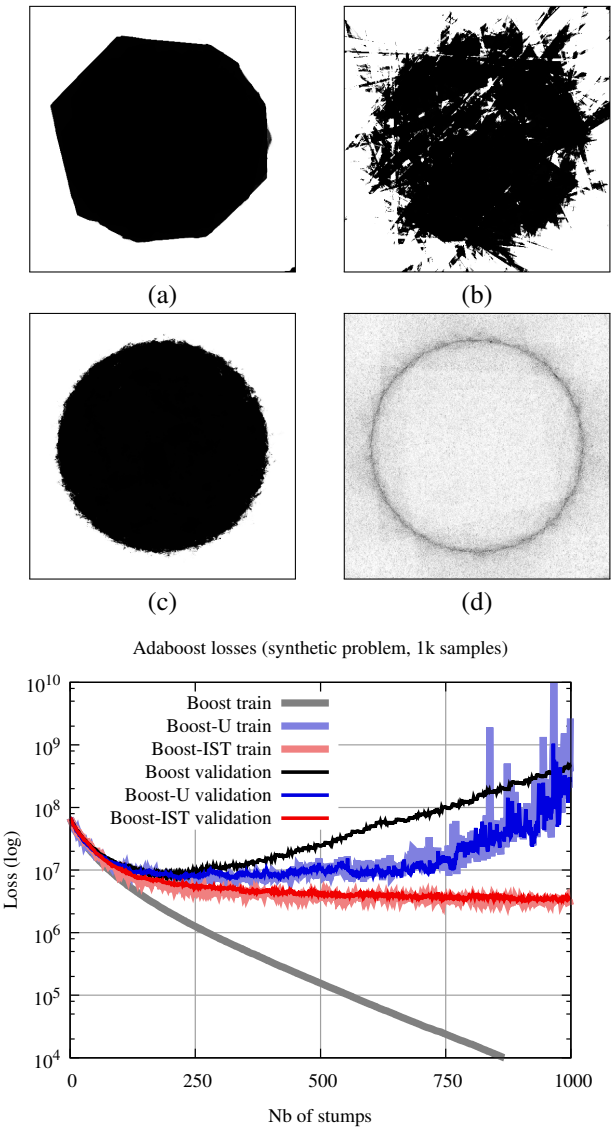

Figure 1. Adaboost on a synthetic $2D$ problem (see § 4.1). Images (a), (b), and (c) depict respectively the prediction obtained with **Boost**, **Boost-U** and **Boost-IST**. The exponential loss behaves pathologically on samples at the frontier which have not been seen during the stump optimization, which impacts both **Boost** and **Boost-U** variants, and leads to a diverging training for the latter. Picture (d) shows the sampling intensity with **Boost-IST**. The bottom graph shows the evolution of the training and validation losses for the three methods vs. the number of stumps

The **Boost** baseline converges in training and leads to a very sharp and clean prediction decision, albeit only roughly approximating the proper domain, which leads to a diverging validation loss, as the validation samples near the boundary may be misclassified by the predictor, with an "in-

creasingly wrong" prediction when the learning progresses. The **Boost-U** re-sampling suffers strongly from the same problem, as each re-sampled set contains points strongly misclassified by the current strong learner, which induce very strong sample weights and correspondingly a choice of stumps that gets more and more inconsistent while the process goes on. This results in diverging training and validation losses, and a pathological final strong predictor. The **Boost-IST** based approach focuses on the boundary population, and by sampling more and more intensively there, it does not diverge and leads to a more accurate decision rule. Remarkably, the validation loss is very well reflected by the training one, even if the former is estimated on a fixed sample set and the latter on one re-sampled at every iteration.

4.2. Neural Networks

We assess experimentally in this section how IST can be used to improve the gradient-descent procedure to train an artificial neural network.

4.2.1. Multi-layer Neural Network on a $2D$ synthetic problem

As for the Boosting example of the previous section, we consider a $2D$ synthetic problem, depicted as Figure 2(a), where the frontier between the two classes is a oscillation of variable frequency. This problem exhibits the difficulty of many real-world data-sets in which the core issue is to capture fine details of the boundary.

We train a neural network with two units as input standing for the coordinate in the $[0, 1]^2$ domain, two fully connected hidden layers with 40 units each, and one output unit. The transfer function is the hyperbolic tangent, and the weights are initialized layer after layer so that the response of every unit before non-linearity is centered, of standard deviation 0.5. We use the quadratic loss for training, and a pure stochastic gradient descent, one sample at a time. Every $1,000$ gradient steps, we compute a validation loss and adapt the step size.

We compare three strategies: **ANN** is the baseline, using uniform sampling in the plan, **ANN-IST** samples with IST using the gradient norm as importance function, following Equation (3) and the same tree structure we use for Boosting in § 4.1. **ANN-IST-L** is the same but uses the loss per sample instead of the gradient norm as the importance function.

We benchmark the three methods through ten train/test runs, with 3 millions gradient steps, and obtain a test error of $2.64\%(\pm 0.29\%)$ for **ANN**, $0.62\%(\pm 0.14\%)$ for **ANN-IST**, and $1.58\%(\pm 0.60\%)$ for **ANN-IST-L**. The losses are consistent with the error rates in all the runs. As shown on Figure 2, the greater performance of **ANN-IST** is explained by its ability to capture the thin structure on the left. This behavior is extremely consistent through the runs, and **ANN-IST** always ranks first, **ANN-IST-L** second, and **ANN** third.

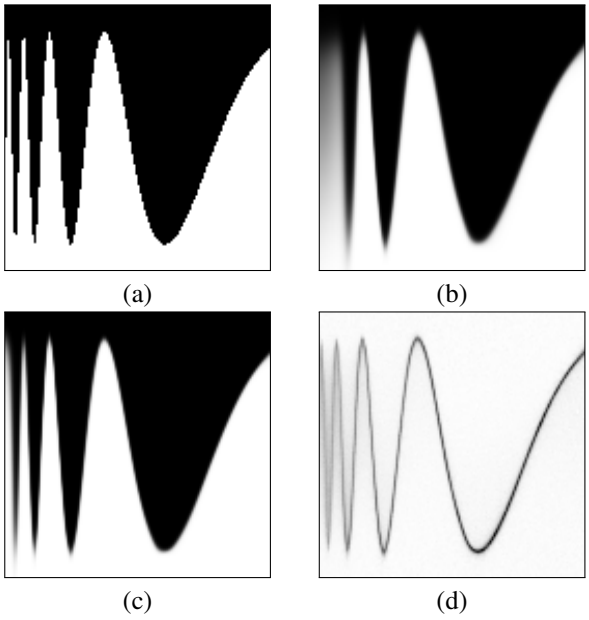

Figure 2. A multi-layer neural network on a synthetic $2D$ problem (see § 4.2.1). Images (a) is the binary labeling to learn, (b) is the prediction of the baseline **ANN**, (c) is the prediction of **ANN-IST**, and (d) is the sampling density during training with **ANN-IST.**

4.2.2. Deep Convolution Network on CIFAR10

For the synthetic examples of the previous sections, we have exploited the Euclidean structure of the problem to build the IST. The main (potential) weakness for using it in practice is the availability of a tree structure consistent with the gradient norm.

We show in this section that this is not the case, and that a very natural tree structure leads to consistent sampling of training points with large gradient norms on a state-of-the-art large-scale problem of image classification.

Our experiments replicate the training of a network[1] designed for a Kaggle competition on the CIFAR10 data-set (Krizhevsky & Hinton, 2009), which relies on synthetic deformations of the original $50,000$ images with translations and scalings to create a total of 1.8 millions images.

The IST for this data-set has the structure depicted on Figure 3: we first split the classes, then the images in each class separately with a top down clustering using the image gradient maps. That is, we recursively apply a K-means

[1] https://github.com/nagadomi/kaggle-cifar10-torch7

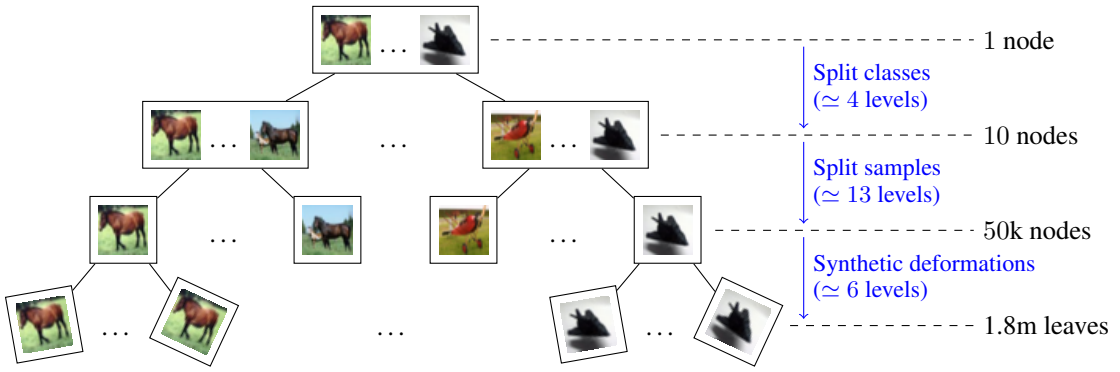

Figure 3. Structure of the sample tree we use to train a CNN on the CIFAR10 data-set. The first levels of the tree split the classes uniformly, the next levels split the images of each class according to the L^2 metric, until we get to individual images, from which the tree splits synthesized images using the same "deformation tree" replicated for each single original image.

with $K = 2$ at each node: starting from the full set of images at the top, the set is clustered into two sub-sets, which are themselves each clustered in two, etc. until reaching a single image, where we then stop and create a leaf. From that point we split the synthetically generated images by clustering the deformations themselves so that similar deformations are close in the tree.

We use gradient norm as the importance for each sample in the IST, and we update it after every step of mini-batch gradient descent. We exploit the property of the matrix products as described in Goodfellow (2015) to compute the gradient norms efficiently for the fully connected layers. For the convolution layers we have the per-sample gradients in the intermediate computations which we use to compute the gradient norms.

The results are depicted on Figure 4 and show that the IST-based sampling is able to leverage the structure of the tree to sample training points with large gradient norm. In particular, after five epochs, the 0.75-quantile of the gradient norm is 2.64×10^{-4} with the uniform sampling, and 0.52 with the IST.

To assess the stability of the convergence of IST, we also computed the correlation between the $\hat{w}$ after each epoch between two randomized runs using it. The correlation goes from -0.017 after the first epoch, to 0.9897 after 10 epochs and remains above this value after that, showing that the two runs lead to virtually identical weight values.

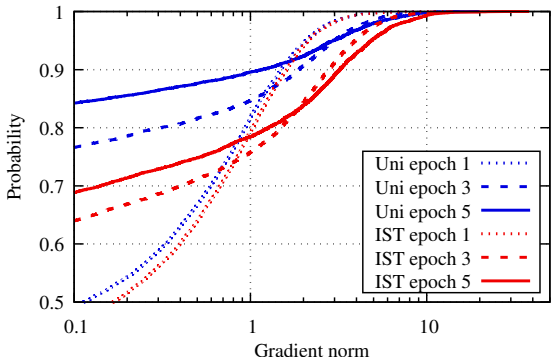

Figure 4. Gradient norms of the sampled training points on three different epochs. This plot shows that as the training progresses (1) most of the examples get a small gradient norm, and a small proportion of "hard" examples appears, with greater gradient norm, and (2) the sampling based on IST (red curves) is always better than uniform (blue curves), and improves in time, as reflected by the increasing gap between the curves.

4.3. Non-linear SVM Prediction

The prediction cost of non-linear SVM grows linearly with the number of support vectors which in turn grows linearly with the number of training points (Steinwart & Christmann, 2008). Thus prediction using non-linear SVM trained on large data-sets is prohibitively expensive. As noted in § 1 for non-linear SVMs such as Gaussian kernel SVM, the support vectors which are far way from the test point contribute very little to the decision function. We exploit this fact to do very fast prediction of Gaussian SVM with IST.

4.3.1. Building the tree

We first organize the support vectors in a binary tree that reflects the Euclidian distance between the points. We perform a top down clustering of the support vectors as we did in § 4.2.2 for the images. The resulting tree is a binary tree, whose leaves are the individual support vectors and whose internal nodes carries the centroid corresponding to the clusters made of all the support vectors below. In addition, each node carries the radius of the cluster (that is the largest distance between the centroid and the points in that cluster, which is 0 at a leaf).

Our objective is to estimate the predictor response on a test sample without computing the full inner products between the test samples and the support vectors. We propose to use a weighted sampling based on the bounds we have on the total weights of the support vector below each node. Using the triangle inequality, we can derive a bound on the weights below that node: given a test sample z, the centroid ω of a cluster of radius ρ, we have

$$\underbrace{e^{-\gamma\left(||z-\omega||+\rho\right)^2}}_{\beta_l} \leq k(z_n, z) \leq \underbrace{e^{-\gamma\left|||z-\omega||-\rho\right|_+^2}}_{\beta_u}, \quad (10)$$

and since we consider the α_n positive (the sign is carried by $f(n) = y_n$, see Equation 4)

$$\underbrace{\beta_l \sum_n \alpha_n}_{\text{Lower bound } l(x) \text{ due to cluster of centroid } \omega} \leq \sum_n \alpha_n k(z_n, z) \leq \underbrace{\beta_u \sum_n \alpha_n}_{\text{Upper bound } u(x)} \quad (11)$$

The tree is traversed using policy of Equation (8). At each node, the bounds of both children are queried and the sampling goes on until reaching a leaf. Querying a bound requires the computation of one inner product with the corresponding centroid (*i.e.* $||z - \omega||$), but only once as the computation can be reused later. At a leaf, we have the true value $\alpha_n k(z_n, z)$ of the weight. The bounds from the leaf up to the root can now be updated according to the true weight: the upper (resp. lower) bound $u(x)$ (resp. $l(x)$) will now be $u(c_0(x))+u(c_1(x))$ (resp. $l(c_0(x))+l(c_1(x))$). The bounds are thus tighter and in future iteration, the sampling will be based on better bounds. Asymptotically, the sampling is performed on exact bounds, that is on the sum of the weights below. In the case of Gaussian kernel, some upper bounds are drastically reduced after a few samplings which cause many parts of the branches not to be sampled again.

While this policy requires to compute inner products with the centroids as well as with the true support vectors in the leaves, the products can be cached once computed and re-accessed when required, without further computation. The true number of inner products for N support vectors of dimension M is $\mathcal{O}(NM)$ whereas for IST, when sampling T times, the number of inner products is at most $\mathcal{O}(TM \log_2 N)$. $\log_2 N$ is the depth of the tree (the number of centroids visited when performing one sampling). This is an upper bound since many computation can be reused.

4.3.2. Defining the probability of bifurcation

We investigate three strategies to define θ_x (*i.e.* the probability to "go down on the right", or to select child $c_1(x)$). To simplify notations, we call u_0 (resp. l_0) the upper (resp. lower) bound of child 0 of node x (*i.e.* $u_0 = u(c_0(x))$).

Mean bound (`mean`) – We define the score of a node to be the mean of the upper and lower bounds.

$$\Phi(x) = \frac{u(x) + l(x)}{2} \text{ so } \theta_x = \frac{l_1 + u_1}{l_0 + u_0 + l_1 + u_1} \quad (12)$$

Max bound (`max`) – We simply use the upper bound to bias the sampling.

$$\Phi(x) = u(x) \text{ so } \theta_x = \frac{u_1}{u_0 + u_1} \quad (13)$$

Close bound (`close`) – We call $O_v(x)$ the indicator function of the overlap between the bound interval of both children of node x:

$$O_v(x) = \mathbb{1}_{\min[u_0(x),u_1(x)] \geq \max[l_0(cx),l_1(x)]} \quad (14)$$

At node x, the probability θ_x to select child 1 is defined by

$$\theta_x = \begin{cases} 0.5 & \text{if } O_v(x) = 1 \\ \frac{l_1(x)}{u_0(x)+l_1(x)} & \text{if } O_v(x) = 0 \text{ and } u_0(x) < l_1(x) \\ \frac{u_1(x)}{l_0(x)+u_1(x)} & \text{if } O_v(x) = 0 \text{ and } u_0(x) \geq l_1(x) \end{cases} \quad (15)$$

The `close` bound is illustrated in the supplementary material. This strategy is less aggressive than the other two because as long as the bounds of both children overlap, the sampling is uniform. And when they no longer overlap, the bounds used to compute Φ are the closest to each other.

4.3.3. SVM experiment on a synthetic problem

To get an idea of the behaviour of the IST policy for SVM evaluation with a Gaussian kernel, we use a synthetic synthetic problem, a 10×10 checkerboard. The data-set is a 2-class problem made of $2D$-Gaussian clouds of points centred at each integer coordinate (from 0 to 9) and of standard deviation 0.3 alternating classes.

Three parameters are involved in the IST SVM evaluation: C and γ, the usual SVM parameters and T, the number of samplings with replacement that are performed in the evaluation. We chose these parameters through cross validation such that the validation accuracy and the actual number of kernel computation are Pareto optimal (see supplementary material for details).

The results are illustrated on the top plot of Figure 5. The `exact` method is the usual SVM policy (Equation 4). For the sampling methods, we vary the number of samplings T to do the estimation of the SVM prediction from 10 to $100,000$.

4.3.4. SVM experiment on real data

We applied this IST method to the Gaussian kernel SVM trained on the Covertype data-set (Bache & Lichman). It contains $522,910$ samples of dimension 54 for training

and $58,102$ for testing. As in Collobert et al. (2002) we consider the binary classification problem (class 2 versus 6 others). We carefully selected the parameters through cross-validation to train the model which in the end contains $105,492$ support vectors. The bottom plot of Figure 5 shows the test accuracy versus the average number of inner products performed during sampling. The exact computation of the SVM decision (see Equation (4)) yields an error rate of 1.765% and requires $105,492$ inner products.

We have also tried the IST method on other data-sets such as `webspam` or `a9a` but the computation cost of IST was greater than for the default policy (*i.e.* evaluating all the dot products), because not only IST required to compute many of the inner products with the support vector but also almost all of the inner products with the centroids which lead to twice as many computations.

The Covertype data-set has this property that at test time, only *very* few support vectors matter in the decision (~100 out of the $100,000$ support vectors) which makes the clustering and the sampling very efficient. As demonstrated by Jose et al. (2013) or Hsieh et al. (2014), this data-set is prone to huge gain in prediction.

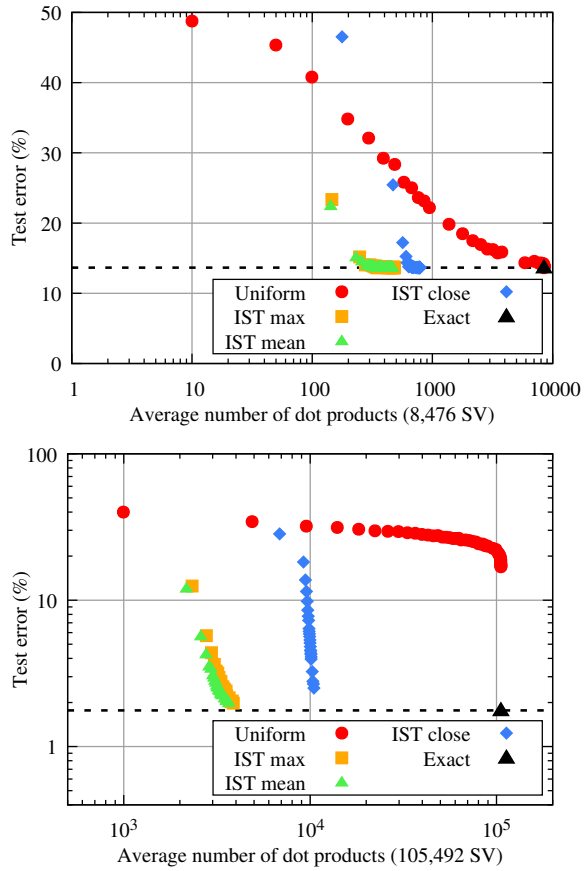

Figure 5. Test error vs. average number of inner products for the synthetic problem (top) and for the Covertype data-set (bottom) – Each point corresponds to a fixed number of samplings to estimate the prediction value. The greater the number of samplings, the better the estimate. IST methods reach top accuracy with less inner product computations. For the Covertype data-set (bottom), the exact point performs all the computations and reaches 1.765% error rate. For the sampling methods, the number of samplings is varied from $1,000$ to $300,000$. For this particular data-set, the distribution of kernel responses is so skewed that IST requires very few inner products and reaches low error, while the uniform approach never reaches good estimates.

5. Conclusion

We have presented a novel approach to cope with very large data-sets. Instead of sampling uniformly, or designing *a priori* a sampling scheme, our strategy adaptively modulates the sampling according to data it has sampled previously. Because it compensates the sampling bias at any time, estimation of the empirical quantities of interest is done at the same time the sampling is improved. Experiments show that this technique can be used both for learning and at test time, and cuts down drastically the number of samples required for a certain level of accuracy.

Two key elements remain to be investigated thoroughly. The first is the construction of the tree itself. In our experiments on real data, it was constructed with a recursive partitioning based on the Euclidean metric. The rationale behind this strategy is that "close samples" should be "similarly weighted". This makes sense in the SVM experiments, since the Euclidean metric is directly related to the structure of the predictor response, but it is an unsatisfactory proxy for the CNN.

The second point is the sampling procedure. We have proposed two strategies, one using empirical weight estimates, the other exact bounds on the said weights. The latter is exact but can be used only with a strong prior information about these bounds (e.g. geometrical information consistent with the weight distribution), which is not the case for very large training sets. What is missing in our approach is a bandit-type use of a confidence interval on the estimate. For instance based on the Hœffding's inequality, to get something similar to the UCB policy used in MCTS.

Acknowledgements

This work was supported by the Swiss National Science Foundation (grants 200021-140912 – DASH and CRSII2-147693 – WILDTRACK).

References

Bache, K. and Lichman, M. Covertype dataset. https://archive.ics.uci.edu/ml/datasets/Covertype.

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

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

Collobert, Ronan, Bengio, Samy, and Bengio, Yoshua. A parallel mixture of svms for very large scale problems. *Neural computation*, 14(5):1105–1114, 2002.

Fleuret, F. and Geman, D. Stationary features and cat detection. *Journal of Machine Learning Research*, 9:2549–2578, 2008.

Gelly, S., Wang, Y., Munos, R., and Teytaud, O. Modification of UCT with Patterns in Monte-Carlo Go. Technical Report RR-6062, INRIA, 2006.

Goodfellow, Ian. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

Hsieh, Cho-Jui, Si, Si, and Dhillon, Inderjit S. Fast prediction for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pp. 3689–3697, 2014.

Jose, Cijo, Goyal, Prasoon, Aggrwal, Parv, and Varma, Manik. Local deep kernel learning for efficient nonlinear svm prediction. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 486–494, 2013.

Kalal, Z., Matas, J., and Mikolajczyk, K. Weighted sampling for large-scale boosting. In *BMVC*, 2008.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1(4):7, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*, pp. 1097–1105, 2012.

Loosli, G., Canu, S., and Bottou, L. Training invariant support vector machines using selective sampling. In Bottou, Léon, Chapelle, Olivier, DeCoste, Dennis, and Weston, Jason (eds.), *Large Scale Kernel Machines*, pp. 301–320. MIT Press, Cambridge, MA., 2007.

Steinwart, I. and Christmann, A. *Support vector machines*. Springer Science & Business Media, 2008.

Zhao, P. and Zhang, T. Stochastic optimization with importance sampling. *arXiv preprint arXiv:1401.2753*, 2014.