Nyström-SGD: Fast Learning of Kernel-Classifiers with Conditioned Stochastic Gradient Descent

Lukas Pfahler and Katharina Morik

TU Dortmund University,
Artificial Intelligence Group,
Dortmund, Germany
`{lukas.pfahler,katharina.morik}@tu-dortmund.de`

Abstract. Kernel methods are a popular choice for classification problems, but when solving large-scale learning tasks computing the quadratic kernel matrix quickly becomes infeasible. To circumvent this problem, the Nyström method that approximates the kernel matrix using only a smaller sample of the kernel matrix has been proposed. Other techniques to speed up kernel learning include stochastic first order optimization and conditioning. We introduce Nyström-SGD, a learning algorithm that trains kernel classifiers by minimizing a convex loss function with conditioned stochastic gradient descent while exploiting the low-rank structure of a Nyström kernel approximation. Our experiments suggest that the Nyström-SGD enables us to rapidly train high-accuracy classifiers for large-scale classification tasks.

1 Introduction

Kernel methods are a very powerful family of learning algorithms for classification problems. In addition to their empirical success, we can give strong statistical guarantees for the generalization error [?,?]. However, learning kernel classifiers traditionally involves computing a $N \times N$ kernel matrix and solving linear algebra problems like matrix inversion or eigen-decomposition requiring $\mathcal{O}(N^3)$ operations, which quickly becomes infeasible for large-scale learning.

Recently, Ma and Belkin [?] have proposed an algorithm for large scale kernel learning based on conditioned stochastic gradient descent. Conditioning is an established technique from first-order optimization where we change the coordinate system of the optimization problem to achieve faster convergence to a low-loss classifier. Another recent work by Rudi et al. [?] applies a combination of conditioning and Nyström sampling to learn kernel classifiers. Nyström sampling is a technique to approximate a kernel matrix by evaluating only a smaller sample of m rows of the full kernel matrix. This reduces both the time and space requirement to $\mathcal{O}(mN)$. Both these approaches are limited to minimizing mean-squared error loss (RMSE), where the empirical risk minimization problem reduces to solving a linear system. RMSE is not an ideal loss function for classification problems, as it punishes model outputs that lead to correct classifications. Since different loss function exhibit different convergence behavior of the classification error, by choosing a good loss function it is possible to obtain even faster convergence to high-accuracy solutions [?].

We extend the work of Ma and Belkin to allow general convex loss functions that are better suited for learning classifiers and present *Nyström-SGD*, an algorithm that combines stochastic gradient descent, conditioning and Nyström sampling. We summarize our contributions as follows:

– We show that the conditioned SGD algorithm by Ma and Belkin can be extended to convex loss functions other than RMSE. While their derivation is based on rewriting the optimization problem as solving a linear system, we view it as convex optimization problem and extend results for optimization in Euclidean spaces [?] to reproducing kernel Hilbert spaces.
– We present Nyström-SGD, a learning algorithm that computes a Nyström approximation of the kernel matrix and runs conditioned SGD with this approximated kernel. We show that we can derive a useful conditioner from the approximation and that the resulting update rule of the iterative optimization algorithm can be implemented extremely efficiently.
– We show our method achieves competitive results on benchmark datasets. While computing the Nyström-approximation and the conditioner are computationally expensive steps, our experiments suggest that our approach yields lower-loss solutions in less time than previous algorithms.

The remainder of the paper is structured as follows: We begin by revisiting reproducing kernel Hilbert spaces and introducing necessary notation. We continue by discussing related research, particularly covering techniques used to speed up kernel learning. In Section 4 we introduce conditioned stochastic gradient descent in reproducing kernel Hilbert spaces. In Section 5 we show how to speed up this approach by replacing the kernel matrix with the Nyström approximation, which results in very efficient update rules. In Section 6 we demonstrate the effectiveness of our approach in various experiments. This paper is concluded in Section 7.

2 Preliminaries on Reproducing Kernel Hilbert Spaces

We begin this section by reviewing *reproducing kernel Hilbert spaces* (RKHS) in which we seek to find linear classification models. The necessary notations will be introduced in this section.

We are interested in learning classification models given labeled data points $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ with $i = 1, ..., N$. We denote by $X = [x_1, ..., x_N]$ the data matrix.

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive-definite kernel. A reproducing kernel Hilbert space associated to a kernel k is a Hilbert space with associated scalar product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. Data points $x \in \mathcal{X}$ are embedded in $\mathcal{H}$ via $x \mapsto k(x, \cdot)$. The reproducing property $\langle f, k(x, \cdot) \rangle_{\mathcal{H}} = f(x)$ holds for all $f \in \mathcal{H}$. Particularly it holds that $\langle k(x, \cdot), k(x', \cdot) \rangle_{\mathcal{H}} = k(x, x')$.

We learn functions from the model family

$$x \mapsto \langle f, k(x,\cdot)\rangle_{\mathcal{H}} \tag{1}$$

where $f, k(x,\cdot)$ are elements of the RKHS $\mathcal{H}$. We can think of f as a hyperplane in $\mathcal{H}$. By the representer theorem for kernel Hilbert spaces, we can express these functions as

$$\langle f, k(x,\cdot)\rangle_{\mathcal{H}} = \sum_{i=1}^{n} \alpha_i k(x_i, x). \tag{2}$$

The kernel analogue of the Gram matrix $X^T X$ is called kernel matrix and we write $K = k(x_i, x_j)|_{i,j=1,\dots,N}$. For notational convenience we define $K_{Ni} = K_i$ and $K_{NB} = [K_{Ni}]_{i \in B}$. Similarly $K_{BB} = k(x_i, x_j)|_{i,j \in B}$. Furthermore we define $K_{N\cdot} = [k(x_i, \cdot)]_{i=1,\dots,N}$.

The kernel analogue of the covariance matrix $C = \frac{1}{N}\sum\limits_{i=1}^{N} x_i x_i^T$ is the covariance operator

$$\mathcal{C} = \frac{1}{N}\sum_{i=1}^{N} k(x_i,\cdot) \otimes k(x_i,\cdot). \tag{3}$$

Interestingly, kernel matrix and covariance operator share the same spectrum of eigenvalues and have closely connected eigenvectors and eigenfunctions [?,?]. If λ is an eigenvalue of K with normalized eigenvector $u \in \mathbb{R}^N$, i.e. $\lambda u = K u$, the corresponding normalized eigenfunction of $\mathcal{C}$ is

$$\ell = \frac{1}{\sqrt{\lambda}}\sum_{i=1}^{N} u_i k(x_i,\cdot). \tag{4}$$

It then holds that $\frac{\lambda}{N}\ell = \mathcal{C}\ell$.

3 Related Work

Stochastic gradient descent is probably the most used optimization algorithm used in machine learning recently, particularly for deep network training. SGD training of kernel classifiers has been studied in the past. We separate two approaches: First, as the representer theorem allows us to represent kernel classifiers as a linear combination of kernel evaluations, it is possible to take the derivative of the loss with respect to those coefficients α and optimize using SGD. This was pioneered by Chapelle [?]. The second approach is to take the derivative with respect to the parameter vector in the reproducing kernel Hilbert space, as proposed by Shalev-Shwartz et al. [?]. The resulting SGD update rule can be expressed in terms of α as well and we will rely on this formulation in this work.

Conditioning is a technique to speed up first order optimization methods by transforming the geometry of the optimization problem using a conditioning matrix [?]. Recently it has also been applied to optimization for kernel methods, mostly for kernel ridge regression. In kernel ridge regression, learning reduces to solving a linear system. This linear system can be solved with iterative optimization methods. Avron et al. [?] speed up this optimization using a conditioner based on random feature maps. Cutajar et al. [?] evaluate a variety of different conditioners and achieve fast convergence. Recently Ma and Belkin [?] proposed to use a conditioner to speed up SGD training of kernel classifiers. Their conditioning operator changes the geometry of the RKHS and is very similar to the conditioner for Euclidean spaces proposed by Gonen and Shalev-Shwartz [?]. They prove that their conditioning operator accelerates convergence and that the speedup depends on the decay of eigenvalues of the kernel matrix. Furthermore they show that without conditioning, only a fraction of classifiers is reachable in a polynomial number of epochs, which limits the expressive power of classifiers learned with vanilla SGD.

With Nyström sampling [?,?,?] we avoid computing the full kernel matrix and instead compute a low-rank approximation based on a sample of rows of the kernel matrix. It is subjected to rigorous analysis for the mean squared error loss [?]. Recently Rudi et al. [?] show that it suffices to sample $\mathcal{O}(\sqrt{N})$ rows of the kernel matrix to achieve optimal statistical accuracy if we use Nyström sampling for kernel ridge regression. Their approach iteratively solves a linear system using conditioned conjugate gradient descent where the conditioner is obtain via Nyström sampling as well.

An alternative approach for making large-scale kernel learning tractable is sampling the kernel feature space instead of sampling the kernel matrix. By approximating the RKHS in finite dimensional Euclidean spaces, we reduce the kernel learning task to the linear case where efficient learning is easy. The most popular choice of features are random Fourier features [?,?].

4 Kernel Learning with Conditioned SGD

We continue by reviewing the conditioned stochastic gradient descent for Euclidean vector spaces, on which we will base our conditioned stochastic gradient descent algorithm in RKHS.

4.1 Conditioned Stochastic Gradient Descent

Let us review a preconditioning solution in Euclidean vector spaces proposed by Gonen and Shalev-Shwartz [?]. We want to minimize the empirical loss of a linear classifier with a convex loss function l via

$$\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} l(\langle w, x_i \rangle, y_i) \tag{5}$$

using a stochastic (sub)gradient descent (SGD) approach. The iterative algorithm updates an intermediate solution by randomly choosing a minibatch of examples

$B \subseteq \{1, ..., N\}$ and applying the iteration

$$\begin{aligned} w^{t+1} = \arg\min_{w} & \frac{|B|}{2\eta} ||w - w^t||_\star^2 + \sum_{i \in B} l\left(\langle w, x_i \rangle, y_i\right) \\ & + \sum_{i \in B} \left\langle w - w^t, l'(\langle w, x_i \rangle, y_i) \cdot x_i \right\rangle \end{aligned} \tag{6}$$

where $|| \cdot ||_\star$ is a vector norm and $l'(\langle w^t, x_i \rangle, y_i)$ is the (sub)derivative of $l(\hat{y}, y)$ with respect to the first argument $\hat{y}$ evaluated at $\langle w^t, x_i \rangle$. If we use the Euclidean norm, we get the classic stochastic gradient descent update rule

$$w^{t+1} = w^t - \frac{\eta}{|B|} \sum_{i \in B} l'(\langle w^t, x_i \rangle, y_i) \cdot x_i \tag{7}$$

If instead we use the norm $||w||_A = \sqrt{w^T A w}$ for a positive-definite matrix A, the update rule becomes

$$w^{t+1} = w^t - \frac{\eta}{|B|} \sum_{i \in B} l'(\langle w^t, x_i \rangle, y_i) \cdot A^{-1} x. \tag{8}$$

We call A the conditioning matrix. We want to select A such that the optimization algorithm converges to a minimum as fast as possible, while still allowing efficient updates. To this end Gonen and Shalev-Shwartz [**?**] propose to use a conditioning matrix with a low-rank structure. This allows efficient matrix-vector multiplication, hence the update rule has little overhead over the traditional SGD update. We denote by $C = U \Lambda U^T$ the eigen-decomposition of C where U is the orthonormal matrix of eigenvectors and $\Lambda = \text{diag}(\lambda_1, ..., \lambda_N)$ contains the eigenvalues in non-increasing order. Golen and Shalev-Shwartz propose the following conditioner

$$A^{-1} = I - \sum_{i=1}^{k} \left(1 - \frac{a}{\sqrt{\lambda_i}}\right) u_i u_i^T \tag{9}$$

$$a = \sqrt{\frac{1}{N-k} \sum_{i=k+1}^{N} \lambda_i} \tag{10}$$

that only uses the first k eigenvalues and -vectors of C. However computing a requires knowing all the eigenvalues. We settle for an upper bound: By bounding $\lambda_i \leq \lambda_{k+1}$ for all $i \geq k+1$ we have $a \leq \sqrt{\lambda_{k+1}} =: \tilde{a}$. This leads to the conditioner

$$A^{-1} = I - \sum_{i=1}^{k} \left(1 - \sqrt{\frac{\lambda_{k+1}}{\lambda_i}}\right) u_i u_i^T. \tag{11}$$

which is the foundation of our approach.

4.2 Kernel Learning with Conditioned Stochastic Gradient Descent

In the kernel setting, the empirical risk minimization objective becomes

$$\min_{f\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N} l(\langle f, k(x_i,\cdot)\rangle_{\mathcal{H}}, y_i) \tag{12}$$

Shalev-Shwartz et al. show that the SGD update (7) carries over to the kernel setting [?] where $f^t \in \mathcal{H}$. We obtain

$$f^{t+1} = f^t - \frac{\eta}{|B|}\sum_{i\in B} l'(\langle f^t, k(x_i,\cdot)\rangle_{\mathcal{H}}, y_i) k(x_i,\cdot) \tag{13}$$

By setting $f^0 = \mathbf{0} = \sum_i 0 \cdot k(x_i,\cdot)$ we obtain an update rule that can be expressed in terms of α as in (2). By induction, every f^t can be written as a linear combination of kernel basis functions $k(x_i,\cdot)$. By substituting $k(x_i,\cdot) \mapsto e_i$ with the ith standard basis vector e_i, we obtain the update rule for α.

We propose to use a similar conditioner matrix as (11) for kernel reproducing Hilbert spaces. Let λ_i, ℓ_i be the eigenvalues and eigenfunctions of $\mathcal{C}$. We define the conditioning operator

$$\mathcal{A}^{-1} = \mathcal{I} - \sum_{i=1}^{k}\left(1 - \sqrt{\frac{\lambda_{k+1}}{\lambda_i}}\right) \ell_i(\cdot) \otimes \ell_i(\cdot) \tag{14}$$

which is very similar to the conditioner proposed by Ma and Belkin [?], but has an additional square-root[1]. We obtain the update rule for conditioned SGD in kernel reproducing Hilbert spaces

$$f^{t+1} = f^t - \frac{\eta}{|B|}\sum_{i\in B} l'(\langle f^t, k(x_i,\cdot)\rangle, y_i)\mathcal{A}^{-1} k(x_i,\cdot) \tag{15}$$

In this scenario, we can still derive efficient updates for α. We decompose $\mathcal{A}^{-1} = \mathcal{I} - \mathcal{D}$ and focus on $\mathcal{D}$ as $\mathcal{I}$ results in standard SGD updates. For

[1] Inspection of the source-code released with [?] shows that the experiments were actually conducted with a conditioner that uses the same square-root.

notational convenience we define $\tilde{\lambda}_i = 1 - \sqrt{\lambda_{k+1} {\lambda_i}^{-1}}$. We see that

$$\begin{aligned}
\mathcal{D}k(x_i, \cdot) &= \sum_{j=1}^{k} \tilde{\lambda}_j \ell_j(\cdot) \langle \ell_j(\cdot), k(x_i, \cdot) \rangle_{\mathcal{H}} && (16)\\
&= \sum_{j=1}^{k} \tilde{\lambda}_j \ell_j(\cdot) \sum_{l=1}^{N} \frac{1}{\sqrt{N\lambda_j}} u_{lj} \cdot k(x_l, x_i) && (17)\\
&= \sum_{j=1}^{k} \frac{\tilde{\lambda}_j}{N\lambda j} K_{Ni}^T u_j \sum_{l=1}^{N} u_{lj} k(x_l, \cdot) && (18)\\
&= K_{Ni}^T \underbrace{\sum_{j=1}^{k} \frac{\tilde{\lambda}_j}{N\lambda j} u_j u_j^T}_{=:D} K_{N\cdot} =: K_{Ni}^T D K_{N\cdot} && (19)
\end{aligned}$$

For notational convenience we define $D := U\tilde{\Lambda}U$ with $\tilde{\Lambda}$ is a diagonal matrix with coefficients as in (19). Thus we can rewrite the conditioned SGD iteration in terms of α as

$$\alpha^{t+1} = \alpha^t - \frac{\eta}{|B|} \sum_{i \in B} l'(\langle \alpha^t, K_{Ni} \rangle, y_i) \left(e_i - DK_{Ni} \right) \tag{20}$$

The update can still be computed efficiently, as it does not need additional kernel evaluations and D is a low-rank matrix which allows efficient matrix-vector multiplication.

5 Faster Training via Nyström Sampling

So far we have derived a conditioned stochastic gradient descent algorithm that operates in reproducing kernel Hilbert spaces. Unfortunately we either have to store $\mathcal{O}(N^2)$ elements of the kernel matrix or compute $\mathcal{O}(N^2)$ kernel evaluations with each pass over the data. We address this by applying Nyström sampling to obtain an approximation of the kernel matrix that only requires $\mathcal{O}(Nm)$ kernel evaluations and storage for a constant sample-size $m < N$. While Ma and Belkin [**?**] use a Nyström sampling approach to estimate the conditioning matrix, we additionally use it to speed up the learning.

The Nyström approximation is computed as follows [**?**,**?**,**?**]: We draw a sample of m landmark points L uniformly at random from X and compute K_{NL} and extract K_{LL}. Now the approximation is defined as

$$\tilde{K} = K_{NL} K_{LL}^{-1} K_{NL}^T \tag{21}$$

To obtain the conditioning matrix D, we compute the exact eigenvalues and eigenvectors of $\tilde{K}$. Let $V \Sigma V^T = K_{LL}$ be the eigen-decomposition of K_{LL}. We decompose

$$K_{NL} V \Sigma^{-1} =: QR \tag{22}$$

Algorithm 1 Nyström-SGD(Data-Set $(x_i, y_i)_{i=1,...,N}$, kernel k, and hyperparameters as described above)

```
Sample landmark points $L$ and materialize $K_{NL}$
Decompose $V\Sigma V^T := K_{LL}$.
Decompose $QR := K_{NL}V\Sigma^{-1}$
Decompose $\tilde{U}\Lambda\tilde{U}^T := R\Sigma R^T$
Multiply $U := Q\tilde{U}$.
Set $\tilde{\Lambda}$ according to (19).
Initialize $w = 0 \in \mathbb{R}^m$.
for $i = 1, ..., t$ do
    $\hat{y} = U_{B\cdot}w$ for random minibatch $B$.
    $w = w - \frac{\eta}{|B|} \sum_{i \in B} l'(\hat{y}_i, y_i) \cdot (\Lambda - \tilde{\Lambda}\Lambda^2)U_{i\cdot}^T$
return w
```

where Q is unitary and R is upper-triangular and then decompose

$$R\Sigma R^T =: \tilde{U}\Lambda\tilde{U}^T \tag{23}$$

where $\tilde{U}$ is unitary and Λ is diagonal. Now we can write

$$\tilde{K} = Q\tilde{U}\Lambda\tilde{U}^T Q^T =: U\Lambda U^T \tag{24}$$

where $U := Q\tilde{U}$ is unitary. Thus U contains the orthonormal eigenvectors of $\tilde{K}$ and Λ contains the corresponding eigenvalues. We can store $\tilde{K}$ in $\mathcal{O}(Nm)$ storage because of its low-rank structure. The computational cost of computing the eigenvalues and -vectors is dominated by computing the QR-decomposition that needs $\mathcal{O}(Nm^2)$ operations and the two eigen-decomposition operations that need $\mathcal{O}(m^3)$ operations with standard linear algebra routines. Since we know all the eigenvalues and vectors, we can set $k = m$. We now show that this has no negative effects on the runtime of SGD udpates.

We note that predictions are computed as $\tilde{K}\alpha = U\Lambda U^T\alpha$, hence we never need to know $\alpha \in \mathbb{R}^N$ explicitly, but it suffices to provide $\Lambda U^T\alpha \in \mathbb{R}^m$. Thus we can express the stochastic gradient updates not in α, but in $\Lambda U^T\alpha$. This dramatically accelerates the stochastic gradient update, which simplifies to

$$\begin{aligned} \Lambda U^T\alpha^{t+1} &= \Lambda U^T\alpha t - \frac{\eta}{|B|}\sum_{i\in B} l'_i \Lambda U^T(e_i - D\tilde{K}_{Ni}) \\ &= \Lambda U^T\alpha^t - \frac{\eta}{|B|}\sum_{i\in B} l'_i \Lambda U^T(e_i - U\tilde{\Lambda}U^T U\Lambda U_i^T) \\ &= \Lambda U^T\alpha^t - \frac{\eta}{|B|}\sum_{i\in B} l'_i (\Lambda - \tilde{\Lambda}\Lambda^2)U_{i\cdot}^T \end{aligned} \tag{25}$$

The update reduces to computing the product of a diagonal matrix with selected rows of U which costs $\mathcal{O}(|B|m)$ operations. The update rule without conditioning can be derived analogously. The full learning algorithm is depicted in Algorithm 1.

To obtain predictions for previously unseen data x, we extend the Nyström approximation kernel matrix by one row and compute $\hat{y}(x) = K_L(x)K_{LL}^{-1}K_{NL}^T\alpha$ where $K_L(x) = k(x_i, x)|_{i\in L}$. We can express this in terms of $\Lambda U^T\alpha$ as

$$\hat{y}(x) = K_L(x)(VR^T\tilde{U}\Lambda^{-1})(\Lambda U^T\alpha) \tag{26}$$

where $VR^T\tilde{U}\Lambda^{-1} \in \mathbb{R}^{m\times m}$ is a constant matrix that we compute only once and $\Lambda U^T\alpha$ is the output of the learner.

6 Experiments

In this section we empirically investigate the advantages of using conditioning and a Nyström kernel approximation for training kernel classifiers. We first present the setup of our experiments. Then we investigate convergence behavior of conditioned SGD with general loss functions. We present results on classification performance of our method and finally evaluate the runtime benefits of Nyström-SGD.

6.1 Design choices

We first describe the basic setup of our experiment, including the datasets and kernels used.

Datasets We conduct our experiments mostly on standard datasets used in many machine learning publications. We use standard datasets MNIST, CIFAR10, IMDB and SUSY. The FACT dataset [?] contains 16 high-level features derived from simulated sensor measurements of a telescope recording cosmic rays in two classes, gamma and proton. Important properties of the datasets are summarized in Table 1.

We apply standard pre-processing to all datasets: We reduce color-channels to a single greyscale value and normalize these to a $[0, 1]$ range. Text data is tokenized and transformed into a bag-of-words representation with relative word frequencies. The most-frequent 30 words are omitted, the following 10k most-frequent words are used as features.

Table 1. Datasets used in our evaluations

Dataset	Type	N	dim	$\lvert\mathcal{Y}\rvert$	Val.-N
MNIST	Image	60k	784	10	10k
CIFAR10	Image	50k	1024	10	10k
IMDB	BoW	25k	10,000	2	25k
SUSY	Physics	4m	18	2	1m
FACT	Physics	1.10m	16	2	369k

Kernels The most-important user-choice for running the proposed learning algorithm is the choice of a suitable kernel function. For our evaluations, we rely on the following three kernel functions:

– **RBF-Kernel:** Probably the most-frequently used kernel for classification tasks is the radial basis function kernel defined as $k(x, x') = \exp(-\gamma\frac{1}{2}||x - x'||_2^2)$ where $\gamma > 0$ is a hyperparameter. We use $\gamma = \frac{1}{d}$ where d is the dimension of x in all our experiments.
– **Inverted-Polynomial-Kernel:** $k(x, x') = (2-\langle\bar{x}, \bar{x}'\rangle)^{-1}$ where $\bar{x} = ||x||_2^{-1} \cdot x$ denotes the normalized input vector [?].
– **Arc-Cosine-Kernel:** The arc-cosine kernel assesses what fraction of half-spaces both x and x' lie in. $k(x, x') = \frac{1}{\pi}||x|| \cdot ||x'|| \cdot (\sin\theta + (\pi - \theta)\cos\theta)$ where $\theta = \arccos\frac{\langle x,x'\rangle}{||x||\cdot||x'||}$ [?].

The latter two kernels have connections to deep networks. These have been used for theoretical analyses of neural network learning: The arc-cosine kernel induces feature maps that correspond to neural networks with infinitely-many ReLu-nodes [?]; the inverted-polynomial kernel induces a class of predictor functions that is a superset of fully-connected networks with bounded weights [?].

Loss Functions We experiment with different classification losses:

– **Mean-Squared Error** Plugging in the mean-squared error loss $l(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$, our proposed algorithm is highly related to the approach proposed by Ma and Belkin [?], as discussed above. We believe that the Mean-Squared-Error is not ideal for classification problems as it unnecessarily punishes outputs that lead to correct classifications when $\hat{y}y > 1$.
– **Hinge-Loss** The loss function used in support vector machines is defined as $l(\hat{y}, y) = \max(0, 1 - \hat{y}y)$ with $y \in \{-1, 1\}$. The hinge-loss does not have a Lipschitz-continuous subgradient, but the smoothed-hinge loss or Huber-loss can be used if a Lipschitz-smooth loss function is desired.
– **Squared Hinge-Loss** $l(\hat{y}, y) = \max(0, 1 - \hat{y}y)^2$. It consistently outperforms other loss functions in the evaluations by Janocha and Czarnecki[?]. It has a Lipschitz-smooth gradient and large gradient when classification is far off.

Hyperparameters Arguably the most important hyperparameter is the stepsize. Following the arguments of Ma and Belkin [?], we set $\eta = \frac{1}{\lambda_1}$ when no conditioner is used. For RMSE and Squared-Hinge, λ_1 is an upper bound of the Lipschitz constant of the gradient, using the inverse of the Lipschitz constant is a popular selection for the stepsize for convex optimization. When conditioning is used, we set $\eta = \frac{1}{\lambda_1}\sqrt{\frac{\lambda_1}{\lambda_k}}$ to account for the conditioning operator. We briefly experimented with using larger stepsizes, however for some combinations of kernel and dataset this leads to divergence. For instance, a multiplicative increase of factor 2 leads to divergence on MNIST with squared hinge loss and RMSE. We defer the analysis of other stepsize policies than constant steps to future work.

Table 2. Loss after 10 epochs of training for different datasets and kernel functions. Runs using conditioning (columns marked 'yes') consistently reach lower losses than vanilla SGD ('no').

		RMSE		HINGE		HINGE^2	
	COND.	NO	YES	NO	YES	NO	YES
MNIST	RBF	2.97	1.46	2.06	0.64	2.97	0.79
	ARC	0.72	0.25	0.40	0.06	0.44	0.05
	INV	0.67	0.11	0.39	0.01	0.41	0.01
CIFAR	RBF	3.60	3.36	2.28	2.13	3.60	3.35
	ARC	3.28	2.69	2.11	1.89	3.24	2.45
	INV	3.21	2.23	2.08	1.67	3.15	1.95
IMDB	RBF	2.02	1.78	2.00	1.85	2.02	1.79
	ARC	1.71	0.72	1.63	0.47	1.55	0.50
	INV	1.55	0.63	1.63	0.46	1.55	0.49

In most of our experiments, we use $m = 10,000$ as the sample size to estimate the eigenvalues or approximate the kernel matrix. For the SUSY dataset we settle for $m = 5000$ due to main memory constraints. This does not negatively impact classification performance. We use a batch size of 64 in all our experiments. When we use Nyström sampling, we set $k = m$, otherwise we set $k = 160$ following the results of Ma and Belkin.

6.2 Convergence Results

In this section we compare the optimization progress with conditioning to the standard SGD updates. We show that the conditioned SGD updates substantially accelerate convergence to low-loss solutions. Ma and Belkin demonstrated the effectiveness of conditioning for RMSE loss, we hypothesize that these results carry over to other convex losses.

To this end we run SGD and the conditioned SGD algorithm from Section 4.2 with all combinations of kernel, loss and the smaller three datasets. We run the optimization for 10 training epochs. As we can see in Table 2, conditioning consistently yields better loss values than unconditioned SGD.

Next we want to test our hypothesis, that there are loss functions better suited for training classifiers than RMSE. Following the results of Janocha and Czarnecki [?], we expect to obtain faster convergence to classifiers with small empirical risk with squared hinge loss than with RMSE. For each dataset and kernel, we compare the training accuracies achieved with each loss function after 10 epochs and use the lowest accuracy as reference. Now we check after how many epochs the training accuracy exceeded this worst accuracy. Figure 1 shows the results for the arccosine kernel; we see that the squared hinge loss reaches the reference accuracy using only 6-7 of the 10 epochs. This suggests that using general convex loss functions is beneficial. Squared hinge loss causes large

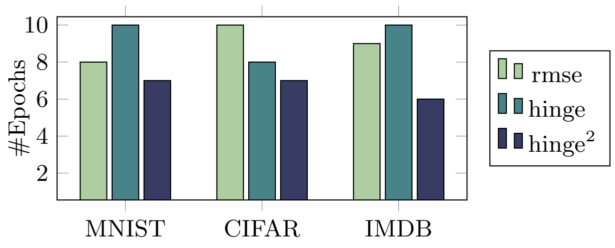

Fig. 1. Number of epochs needed to reach accuracy of slowest learner (lower is better)

Table 3. Classification performance of different configurations. We tried different kernels (ARC: arccosine,INV: inverted polynomial) and loss functions (RMSE: root mean-squared error, HINGE: Hinge loss, HINGE^2: squared hinge loss) and report the number of epochs used for training (EP).

	Nyström SGD		Conditioned SGD		Eigen Pro	
	Acc.	Config	Acc.	Config	Acc.	Config
FACT	86.77	ARC, HINGE 10EP	-	-	-	-
SUSY	79.84	ARC, HINGE^2 5EP	-	-	**80.20**	[?]
MNIST	**98.77**	INV, HINGE^2 7EP	98.68	INV, HINGE^2 7EP	98.51	INV, RMSE 7EP
CIFAR	46.08	INV, HINGE^2 8EP	**48.19**	INV, HINGE^2 70EP	48.17	INV, RMSE 72EP
IMDB	88,08	ARC, HINGE^2 3EP	**88.52**	INV, HINGE^2 16EP	88.29	INV, RMSE 8EP

gradients for far-off predictions and zero gradient for correct predictions, thereby combining benefits of RMSE loss and hinge loss.

6.3 Classification Performances

In this section we evaluate the classification performance achieved by running Nyström-SGD on the datasets. All our experiments do not use explicit regularization. However when using SGD-like methods where the initial solution is zero, we can regularize by limiting the number of training iterations. This early-stopping regularization is effective, as the norm of the solution grows with each gradient update, thus early solutions have lower norms. We report the best validation error, of the best choice of kernel and loss function. Nyström-SGD is compared to conditioned SGD with the full kernel matrix as well as Eigen-Pro, the approach by Ma and Belkin [?], that uses RMSE loss and the full kernel matrix. For the larger datasets, due to memory constraints we only run Nyström-SGD ourselves and rely on published results for comparison.

Table 4. Runtime for different combinations of components: Using Conditioning (C) and using Nyström sampling (N). Using Nyström sampling dramatically reduces the cost of training epochs. Computing the exact conditioner for a Nyström approximation is the most expensive initialization.

C N	Phase	FACT	SUSY	MNIST	CIFAR	IMDB
Y N	Setup	-	-	130s	113s	255s
	Epoch	-	-	448s	420s	835s
N Y	Setup	823s	984s	172s	198s	445s
	Epoch	114s	390s	12s	11s	3s
Y Y	Setup	3400s	4941s	442s	485s	648s
	Epoch	114s	390s	12s	11s	3s

We depict our results in Table 3. On the Fact dataset we achieve 86.7% accuracy, which compares to the published 87.1% ROC AUC achieved with random forests [?]. The performance on CIFAR is far worse than state-of-the-art results achieved with convolution networks. We believe that choosing a kernel designed for image tasks will improve this performance.

Over most datasets we see that using the Nyström approximation decreases accuracy by a small margin. We attribute this to the approximation error induced. The notable exception is the MNIST dataset, where Nyström-SGD achieves the best accuracy.We hypothesize that the approximation works as a form of regularization that prevents overfitting.

6.4 Runtime Analysis

In this section, we analyze the trade-off between the initialization phase of the algorithm and the actual training, i.e. the SGD iterations.

Obviously computing the Nyström approximation of the kernel matrix and its eigenvalues and vectors are costly operations. We need to verify that it is worthwhile to do so. In Table 4 we report the runtime of the two phases in seconds for the different datasets. We use arccosine kernel and squared hinge loss, but these choices have very little impact on runtime. We use a machine with two Intel Xeon E5-2697v2 CPUs with 12 cores each and ∼300GB of memory.

First of all we note that the SGD epochs are significantly accelerated by using the Nyström approximation. The factor depends on the runtime of computing the kernel function and is higher when the dimension of d is higher. This speedup comes at the cost of computing the approximation once. This takes only unsubstantially longer than computing the approximate conditioner for the exact kernel matrix. We see that computing the exact conditioner for the Nyström operation is the most expensive initialization step. Depending on the dataset, it is as expensive as only computing the Nyström approximation and running between 20 and > 60 epochs of SGD.

Table 5. Progress of the optimization compared between Nyström-SGD with and without conditioning. After 200 epochs of SGD training without conditioning (Vanilla), the accuracy of Nyström-SGD still is not matched.

	Conditioning 20 Epochs		Vanilla 20 Epochs		Vanilla 200 Epochs	
	Train	Test	Train	Test	Train	Test
FACT	**86.77**	**86.84**	84.28	84.3	84.99	85.10
SUSY	**80.04**	**80.27**	80.05	80.12	80.18	80.18
MNIST	**100.00**	**98.47**	96.26	95.91	98.78	97.98
CIFAR	**73.80**	**46.05**	35.52	33.16	44.53	38.06
IMDB	**97.88**	85.67	79.47	80.24	87.58	**86.26**

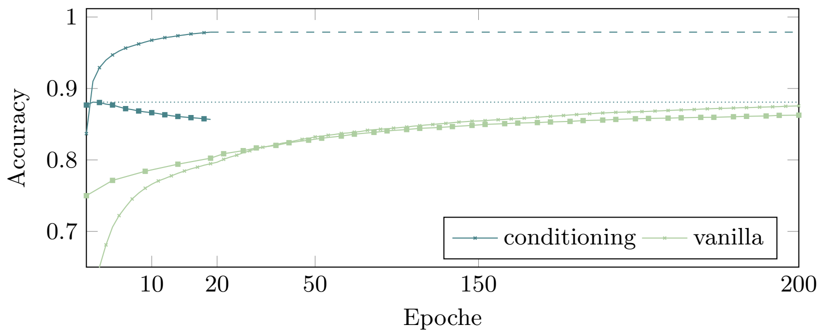

Fig. 2. Convergence behavior for training (x) and testing (squares) on IMDB data. In 200 epochs, vanilla SGD does not reach training accuracy of Nyström-SGD after 20 epochs. Nyström-SGD reaches solution with best validation accuracy after 1 epoch and subsequently begins to overfit. Vanilla SGD is still learning after 200 epochs and does not overfit yet.

Thus for learning with Nyström approximation, we check if models trained only few epochs with conditioning outperform models trained more iterations without conditioning. As we can see in Table 5 this is generally the case. To achieve the same accuracy on the training data as the learner with conditioning after 20 epochs, the learner without conditioning requires more than 200 epochs of training. Thus in total, the cost of learning a high-accuracy prediction model with Nyström and conditioning is lower than without conditioning or without Nyström. All but one classifiers trained with conditioning have better validation accuracy after 20 epochs than vanilla SGD after 200 epoches. The IMDB dataset is an exception: As we can see in Figure 2, with conditioning we reach the solution with best validation accuracy in the second epoch and from there on start to overfit. The vanilla learner does not reach the overfitting stage in 200 epochs.

Overall, these results are in line with the result of Ma and Belkin [**?**], that vanilla SGD can only reach a fraction of the function space in polynomial time.

7 Conclusion and Outlook

Building on results of Ma and Belkin [**?**], we have derived a conditioned stochastic gradient descent algorithm for learning kernel classifiers that minimize arbitrary convex loss functions. Then we have presented Nyström-SGD, a learning algorithm that combines conditioned SGD with Nyström approximation. This way we reduce the number of kernel evaluations needed to train a model; overall we need only $m \cdot N$ kernel evaluations. We compute a useful conditioner for optimizing models using the approximated kernel matrix by efficiently computing its exact eigen-decomposition. Exploiting the structure of conditioner and kernel approximation has allowed us to speed-up the stochastic gradient updates significantly.

In our experiments we have shown the benefits of conditioning and the advantages of choosing a suitable loss function for fast convergence to a high-accuracy classifier. Nyström-SGD has a computationally expensive setup phase that computes the approximation and conditioner in $\mathcal{O}(m^2N + m^3)$ and fast iterations with cost per epoch of $\mathcal{O}(Nm)$. Our experiments suggest that this expensive setup is worthwhile, as we rapidly converge to a high-accuracy solutions on a number of benchmark datasets.

In the future, we want to derive generalization bounds that quantify the influence of the sample size m similar to the ones presented by Rudi et al. [**?**]. Furthermore we want to investigate using approximate conditioning operators for the Nyström kernel approximation that are cheaper to compute by avoiding the computation of the full QR decomposition and the second eigen-decomposition.

Currently the sample for the Nyström approximation is drawn uniformly at random. More advanced sampling schemes have been proposed. These allow to draw smaller samples while maintaining the same level of accuracy. Incorporating these into our algorithm is certainly beneficial.

Acknowledgment Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project C3.

References