
Learning Infinite Layer Networks Without the Kernel Trick

Roi Livni¹ Daniel Carmon² Amir Globerson²

Abstract

Infinite Layer Networks (ILN) have been proposed as an architecture that mimics neural networks while enjoying some of the advantages of kernel methods. ILN are networks that integrate over infinitely many nodes within a single hidden layer. It has been demonstrated by several authors that the problem of learning ILN can be reduced to the kernel trick, implying that whenever a certain integral can be computed analytically they are efficiently learnable. In this work we give an online algorithm for ILN, which avoids the *kernel trick assumption*. More generally and of independent interest, we show that kernel methods in general can be exploited even when the kernel cannot be efficiently computed but can only be estimated via sampling. We provide a regret analysis for our algorithm, showing that it matches the sample complexity of methods which have access to kernel values. Thus, our method is the first to demonstrate that the kernel trick is not necessary, as such, and random features suffice to obtain comparable performance.

1. Introduction

With the increasing success of highly non-convex and complex learning architectures such as neural networks, there is an increasing effort to further understand and explain the limits of training such hierarchical structures.

Recently there have been attempts to draw mathematical insight from kernel methods in order to better understand deep learning, as well as come up with new computationally learnable architectures. One such line of work consists of learning classifiers that are linear functions of a very large or infinite collection of non-linear functions (Bach,

¹University of Princeton, Princeton, New Jersey, USA ²Tel-Aviv University, Tel-Aviv, Israel. Correspondence to: Roi Livni <rlivni@cs.princeton.edu>, Daniel Carmon <carmonda@mail.tau.ac.il>, Amir Globerson <gamir@mail.tau.ac.il>.

2014; Daniely et al., 2016; Cho & Saul, 2009; Heinemann et al., 2016; Williams, 1997). Such models can be interpreted as a neural network with infinitely many nodes in a hidden layer, and we thus refer to them as “Infinite Layer Networks” (ILN). They are of course also related to kernel based classifiers, as will be discussed later.

A target function in an ILN class will be of the form:

$$\mathbf{x} \rightarrow \int \psi(\mathbf{x}; \mathbf{w})f(\mathbf{w})d\mu(\mathbf{w}), \quad (1)$$

Here ψ is some function of the input \mathbf{x} and parameters \mathbf{w} , and $d\mu(\mathbf{w})$ is a prior over the parameter space. For example, $\psi(\mathbf{x}; \mathbf{w})$ can be a single sigmoidal neuron or a complete convolutional network. The integral can be thought of as an infinite sum over all such possible networks, and $f(\mathbf{w})$ can be thought of as an infinite output weight vector to be trained.

A Standard 1–hidden layer network with a finite set of units can be obtained from the above formalism as follows. First, choose $\psi(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{x} \cdot \mathbf{w})$ where σ is an activation function (e.g., sigmoid or relu). Next, set $d\mu(\mathbf{w})$ to be a discrete measure over a finite set $\mathbf{w}_1, \dots, \mathbf{w}_d$.¹ In this case, the integral results in a network with d hidden units, and the function f is the linear weights of the output layer. Namely:

$$\mathbf{x} \rightarrow \frac{1}{d} \sum_{i=1}^d f(\mathbf{w}_i) \cdot \sigma(\mathbf{x} \cdot \mathbf{w}_i).$$

The main challenge when training 1–hidden layer networks is of course to *find* the $\mathbf{w}_1, \dots, \mathbf{w}_d$ on which we wish to support our distribution. It is known (Livni et al., 2014), that due to hardness of learning intersection of halfspaces (Klivans & Sherstov, 2006; Daniely et al., 2014), 1–hidden layer neural networks are computationally hard for a wide class of activation functions. Therefore, as the last example illustrates, the choice of μ is indeed crucial for performance.

For a fixed prior μ , the class of ILN functions is highly expressive, since f can be chosen to approximate any 1–hidden layer architecture to arbitrary precision (by setting f to delta functions around the weights of the network, as

¹In δ function notation $d\mu(\mathbf{w}) = \frac{1}{d} \sum_{i=1}^d \delta(\mathbf{w} - \mathbf{w}_i)d\mathbf{w}$

we did above for μ). However, this expressiveness comes at a cost. As argued in [Heinemann et al. \(2016\)](#), ILN will generalize well when there is a large probability mass of \mathbf{w} parameters that attain a small loss.

The key observation that makes certain ILN tractable to learn is that Eq. 1 is a linear functional in f . In that sense it is a linear classifier and enjoys the rich theory and algorithmic toolbox for such classifiers. In particular, one can use the fact that linear classifiers can be learned via the kernel trick in a batch ([Cortes & Vapnik, 1995](#)) as well as online settings ([Kivinen et al., 2004](#)). In other words, we can reduce learning ILN to the problem of computing the kernel function between two examples. Specifically the problem reduces to computing integrals of the following form:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \int \psi(\mathbf{x}_1; \mathbf{w}) \cdot \psi(\mathbf{x}_2; \mathbf{w}) d\mu(\mathbf{w}) \quad (2)$$

$$= \mathbb{E}_{\bar{\mathbf{w}} \sim \mu} [\psi(\mathbf{x}_1; \bar{\mathbf{w}}) \cdot \psi(\mathbf{x}_2; \bar{\mathbf{w}})]. \quad (3)$$

In this work we extend this result to the case where no closed form kernel is available, and thus the kernel trick is not directly applicable. We thus turn our attention to the setting where features (i.e., \mathbf{w} vectors) can be randomly sampled. In this setting, our main result shows that for the squared loss, we can efficiently learn the above class. Moreover, we can surprisingly do this with a computational cost comparable to that of methods that have access to the closed form kernel $k(\mathbf{x}_1, \mathbf{x}_2)$.

The observation we begin with is that sampling random features (i.e., \mathbf{w} above), leads to an unbiased estimate of the kernel in Eq. 2. Thus, if for example, we ignore complexity issues and can sample infinitely many \mathbf{w} 's, it is not surprising that we can avoid the need for exact computation of the kernel. However, our results provide a much stronger and practical result. Given T training samples, the lower bound on achievable accuracy is $O(1/\sqrt{T})$ (see [Shamir, 2015](#)). We show that we can in fact achieve this rate, using $\tilde{O}(T^2)$ calls² to the random feature generator. For comparison, note that $O(T^2)$ is the size of the kernel matrix, and is thus likely to be the cost of any algorithm that uses an explicit kernel matrix, where one is available. As we discuss later, our approach improves on previous random features based learning ([Dai et al., 2014](#); [Rahimi & Recht, 2009](#)) in terms of sample/computational complexity, and expressiveness.

2. Problem Setup

We consider algorithms that learn a mapping from input instances $\mathbf{x} \in \mathcal{X}$ to labels $y \in \mathcal{Y}$. We focus on the regression case where \mathcal{Y} is the interval $[-1, 1]$. Our starting point is a class of feature functions $\psi(\mathbf{w}; \mathbf{x}) : \Omega \times \mathcal{X} \rightarrow \mathbb{R}$,

parametrized by vectors $\mathbf{w} \in \Omega$. The functions $\psi(\mathbf{w}; \mathbf{x})$ may contain highly complex non linearities, such as multi-layer networks consisting of convolution and pooling layers. Our only assumption on $\psi(\mathbf{w}; \mathbf{x})$ is that for all $\mathbf{w} \in \Omega$ and $\mathbf{x} \in \mathcal{X}$ it holds that $|\psi(\mathbf{w}; \mathbf{x})| < 1$.

Given a distribution μ on Ω , we denote by $L_2(\Omega, \mu)$ the class of square integrable functions over Ω .

$$L_2(\Omega, \mu) = \left\{ f : \int f^2(\mathbf{w}) d\mu(\mathbf{w}) < \infty \right\}.$$

We will use functions $f \in L_2(\Omega, \mu)$ as mixture weights over the class Ω , where each f naturally defines a new regression function from \mathbf{x} to \mathbb{R} as follows:

$$\mathbf{x} \rightarrow \int \psi(\mathbf{w}; \mathbf{x}) f(\mathbf{w}) d\mu(\mathbf{w}). \quad (4)$$

Our key algorithmic assumption is that the learner can efficiently sample random \mathbf{w} according to the distribution μ . Denote the time to generate one such sample by ρ .

In what follows it will be simpler to express the integrals as scalar products. Define the following scalar product on functions $f \in L_2(\Omega, \mu)$.

$$\langle f, g \rangle = \int f(\mathbf{w}) g(\mathbf{w}) d\mu(\mathbf{w}) \quad (5)$$

We denote the corresponding ℓ_2 norm by $\|f\| = \sqrt{\langle f, f \rangle}$. Also, given features \mathbf{x} denote by $\Phi(\mathbf{x})$ the function in $L_2(\Omega, \mu)$ given by $\Phi(\mathbf{x})[\mathbf{w}] = \psi(\mathbf{w}; \mathbf{x})$. The regression functions we are considering are then of the form $\mathbf{x} \rightarrow \langle f, \Phi(\mathbf{x}) \rangle$.

A subclass of norm bounded elements in $L_2(\Omega, \mu)$ induces a natural subclass of regression functions. Namely, we consider the following class:

$$\mathcal{H}_\mu^B = \{ \mathbf{x} \rightarrow \langle f, \Phi(\mathbf{x}) \rangle : \|f\| < B \}.$$

Our ultimate goal is to output a predictor $f \in L_2(\Omega, \mu)$ that is competitive, in terms of prediction, with the best target function in the class \mathcal{H}_μ^B .

We will consider an online setting, and use it to derive generalization bounds via standard online to batch conversion. In our setting, at each round a learner chooses a target function $f_t \in L_2(\Omega, \mu)$ and an adversary then reveals a sample \mathbf{x}_t and label y_t . The learner then incurs a loss of

$$\ell_t(f_t) = \frac{1}{2} (\langle f_t, \Phi(\mathbf{x}_t) \rangle - y_t)^2. \quad (6)$$

The use of squared loss might seem restrictive if one is interested in classification. However, L_2 loss is common by now in classification with support vector machines and kernel methods since ([Suykens & Vandewalle, 1999](#); [Suykens](#)

²We use \tilde{O} notation to suppress logarithmic factors

et al., 2002). More recently Zhang et al. (2016) showed that when using a large number of features regression achieves performance comparable to the corresponding linear classifiers (see Section 5 therein).

The objective of the learner is to minimize her T round regret w.r.t norm bounded elements in $L_2(\Omega, \mu)$. Namely:

$$\sum_{t=1}^T \ell_t(f_t) - \min_{f^* \in \mathcal{H}_\mu^B} \sum_{t=1}^T \ell_t(f^*). \quad (7)$$

In the statistical setting we assume that the sequence $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^T$ is generated IID according to some unknown distribution \mathbb{P} . We then define the expected loss of a predictor as

$$L(f) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathbb{P}} \left[\frac{1}{2} (\langle f, \Phi(\mathbf{x}) \rangle - y)^2 \right]. \quad (8)$$

3. Main Results

Theorem 1 states our result for the online model. The corresponding result for the statistical setting is given in Corollary 1. We will elaborate on the structure of the Algorithm later, but first provide the main result.

Algorithm 1: The SHRINKING_GRADIENT algorithm.

Data: $T, B > 1, \eta, m$
Result: Weights $\alpha^{(1)}, \dots, \alpha^{(T+1)} \in \mathbb{R}^T$. Functions $f_t \in L_2(\Omega, \mu)$ defined as $f_t = \sum_{i=1}^t \alpha_i^{(t)} \Phi(\mathbf{x}_i)$;
 Initialize $\alpha^{(1)} = \bar{0} \in \mathbb{R}^T$;
for $t = 1, \dots, T$ **do**
 Observe \mathbf{x}_t, y_t ;
 Set $E_t =$
 EST_SCALAR_PROD($\alpha^{(t)}, \mathbf{x}_{1:t-1}, \mathbf{x}_t, m$);
 if $|E_t| < 16B$ **then**
 $\alpha^{(t+1)} = \alpha^{(t)}$;
 $\alpha_t^{(t+1)} = -\eta(y_t - E_t)$;
 else
 $\alpha^{(t+1)} = \frac{1}{4}\alpha^{(t)}$;

Theorem 1. Run Algorithm 1 with parameters $T, B \geq 1, \eta = \frac{B}{\sqrt{T}}$ and $m = O(B^4 T \log(BT))$. Then:

1. For every sequence of squared losses ℓ_1, \dots, ℓ_T observed by the algorithm we have for f_1, \dots, f_T :

$$\mathbb{E} \left[\sum_{t=1}^T \ell_t(f_t) - \min_{f^* \in \mathcal{H}_\mu^B} \sum_{t=1}^T \ell_t(f^*) \right] = O(B\sqrt{T})$$

Algorithm 2: EST_SCALAR_PROD

Data: $\alpha, \mathbf{x}_{1:t-1}, \mathbf{x}, m$
Result: Estimated scalar product E
if $\alpha = \bar{0}$ **then**
 Set $E = 0$
else
 for $k=1, \dots, m$ **do**
 Sample i from the distribution $q(i) = \frac{|\alpha_i|}{\sum |\alpha_i|}$
 ;
 Sample parameter $\bar{\mathbf{w}}$ from μ . Set
 $E^{(k)} = \text{sgn}(\alpha_i) \psi(\mathbf{x}_i; \bar{\mathbf{w}}) \psi(\mathbf{x}; \bar{\mathbf{w}})$;
 Set $E = \frac{\|\alpha\|_1}{m} \sum_{k=1}^m E^{(k)}$

2. The run-time of the algorithm is $\tilde{O}(\rho B^4 T^2)$.³
3. For each $t = 1 \dots T$ and a new test example \mathbf{x} , we can with probability $\geq 1 - \delta$ estimate $\langle f_t, \Phi(\mathbf{x}) \rangle$ within accuracy ϵ_0 by running Algorithm 2 with parameters $\alpha^{(t)}, \{\mathbf{x}_i\}_{i=1}^t, \mathbf{x}$ and $m = O(\frac{B^4 T}{\epsilon_0^2} \log 1/\delta)$. The resulting running time for a test point is then $O(\rho m)$.

We next turn to the statistical setting, where we provide bounds on the expected performance. Following standard online to batch conversion and Theorem 1 we can obtain the following Corollary (e.g., see Shalev-Shwartz, 2011):

Corollary 1 (Statistical Setting). *The following holds for any $\epsilon > 0$. Run Algorithm 1 as in Theorem 1, with $T = O(\frac{B^2}{\epsilon^2})$. Let $S = \{(x_t, y_t)\}_{t=1}^T$, be an IID sample drawn from some unknown distribution \mathbb{P} . Let $f_S = \frac{1}{T} \sum f_t$. Then the expected loss satisfies:*

$$\mathbb{E}_{S \sim \mathbb{P}} [L(f_S)] < \inf_{f^* \in \mathcal{H}_\mu^B} L(f^*) + \epsilon.$$

The runtime of the algorithm, as well as estimation time on a test example are as defined in Theorem 1.

Proofs of the results are provided in Section 5.1 and the appendix.

4. Related Work

Learning with random features can be traced to the early days of learning (Minsky & Papert, 1988), and infinite networks have also been introduced more than 20 years ago (Williams, 1997; Hornik, 1993). More recent works have considered learning neural nets (also multi-layer) with infinite hidden units using the kernel trick (Cho & Saul, 2009; Deng et al., 2012; Hazan & Jaakkola, 2015; Heineemann et al., 2016). These works take a similar approach

³Ignoring logarithmic factors in B and T .

to ours but focus on computing the kernel for certain feature classes in order to invoke the kernel trick. Our work in contrast avoids using the kernel trick and applies to any feature class that can be randomly generated. All the above works are part of a broader effort of trying to circumvent hardness in deep learning by mimicking deep nets through kernels (Mairal et al., 2014; Bouvrie et al., 2009; Bo et al., 2011; 2010), and developing general duality between neural networks and kernels (Daniely et al., 2016).

From a different perspective the relation between random features and kernels has been noted by Rahimi & Recht (2007) who show how to represent translation invariant kernels in terms of random features. This idea has been further studied (Bach, 2015; Kar & Karnick, 2012) for other kernels as well. The focus of these works is mainly to allow scaling down of the feature space and representation of the final output classifier.

Dai et al. (2014) focus on tractability of large scale kernel methods, and their proposed *doubly stochastic* algorithm can also be used for learning with random features as we have here. In Dai et al. (2014) the objective considered is of the regularized form: $\frac{\gamma}{2}\|f\|^2 + R(f)$, with a corresponding sample complexity of $O(1/(\gamma^2\epsilon^2))$ samples needed to achieve ϵ approximation with respect to the risk of the optimum of the regularized objective.

To relate the above results to ours, we begin by emphasizing that the bound in (Dai et al., 2014) holds for fixed γ , and refers to optimization of the regularized objective. Our objective is to minimize the risk $R(f)$ which is the expected squared loss, for which we need to choose $\gamma = O(\frac{\epsilon}{B^2})$ in order to attain accuracy ϵ (Sridharan et al., 2009). Plugging this γ into the generalization bound in Dai et al. (2014) we obtain that the algorithm in Dai et al. (2014) needs $O(\frac{B^4}{\epsilon^4})$ samples to compete with the optimal target function in the B -ball. Our algorithm needs $O(\frac{B^2}{\epsilon^2})$ examples which is considerably better. We note that their method does extend to a larger class of losses, whereas our is restricted to the quadratic loss.

In Rahimi & Recht (2009), the authors consider embedding the domain into the feature space $\mathbf{x} \rightarrow [\psi(\mathbf{w}_1; \mathbf{x}), \dots, \psi(\mathbf{w}_m; \mathbf{x})]$, where \mathbf{w}_i are IID random variables sampled according to some prior $\mu(\mathbf{w})$. They show that with $O(\frac{B^2 \log 1/\delta}{\epsilon^2})$ random features estimated on $O(\frac{B^2 \log 1/\delta}{\epsilon^2})$ samples they can compete with the class:

$$\mathcal{H}_{\mu}^B = \left\{ \mathbf{x} \rightarrow \int \psi(\mathbf{w}; \mathbf{x}) f(\mathbf{w}) d\mu(\mathbf{w}) : |f(\mathbf{w})| \leq B \right\}$$

Our algorithm relates to the mean square error cost function which does not meet the condition in Rahimi & Recht (2009), and is hence formally incomparable. Yet we can invoke our algorithm to compete against a larger class

of target functions. Our main result shows that Algorithm 1, using $\tilde{O}(\frac{B^8}{\epsilon^4})$ estimated features and using $O(\frac{B^2}{\epsilon^2})$ samples will, in expectation, output a predictor that is ϵ close to the best in \mathcal{H}_{μ}^B . Note that $|f(\mathbf{w})| < B$ implies $\mathbb{E}_{\mathbf{w} \sim \mu}(f^2(\mathbf{w})) < B^2$. Hence $\mathcal{H}_{\mu}^B \subseteq \mathcal{H}_{\mu}^{B^2}$. Note however, that the number of estimated features (as a function of B) is worse in our case.

Our approach to the problem is to consider learning with a noisy estimate of the kernel. A related setting was studied in Cesa-Bianchi et al. (2011b), where the authors considered learning with kernels when the data is corrupted. Noise in the data and noise in the scalar product estimation are not equivalent when there is non-linearity in the kernel space embedding. There is also extensive research on linear regression with actively chosen attributes (Cesa-Bianchi et al., 2011a; Hazan & Koren, 2012). The convergence rates and complexity of the algorithms are dimension dependent. It would be interesting to see if their method can be extended from finite set of attributes to a continuum set of attributes.

5. Algorithm

We next turn to present Algorithm 1, from which our main result is derived. The algorithm is similar in spirit to Online Gradient Descent (OGD) (Zinkevich, 2003), but with some important modifications that are necessary for our analysis.

We first introduce the problem in the terminology of online convex optimization, as in Zinkevich (2003). At iteration t our algorithm outputs a hypothesis f_t . It then receives as feedback (\mathbf{x}_t, y_t) , and suffers a loss $\ell_t(f_t)$ as in Eq. 6. The objective of the algorithm is to minimize the regret against a benchmark of B -bounded functions, as in Eq. 7.

A classic approach to the problem is to exploit the OGD algorithm. Its simplest version would be to update $f_{t+1} \rightarrow f_t - \eta \nabla_t$ where η is a step size, and ∇_t is the gradient of the loss w.r.t. f at f_t . In our case, ∇_t is given by:

$$\nabla_t = (\langle f_t, \Phi(\mathbf{x}_t) \rangle - y_t) \Phi(\mathbf{x}_t) \quad (9)$$

Applying this update would also result in a function $f_t = \sum_{i=1}^t \alpha_i \Phi(\mathbf{x}_i)$ as we have in Algorithm 1 (but with different α_i from ours). However, in our setting this update is not applicable since the scalar product $\langle f_t, \Phi(\mathbf{x}_t) \rangle$ is not available. One alternative is to use a stochastic unbiased estimate of the gradient that we denote by $\bar{\nabla}_t$. This induces an update step $f_{t+1} \rightarrow f_t - \eta \bar{\nabla}_t$. One can show that OGD with such an estimated gradient enjoys the following upper bound on the regret $\mathbb{E}[\sum \ell_t(f_t) - \ell_t(f^*)]$ for every $\|f^*\| \leq B$ (e.g., see Shalev-Shwartz, 2011):

$$\frac{B^2}{\eta} + \eta \sum_{i=1}^T \mathbb{E}[\|\nabla_i\|^2] + \eta \sum_{i=1}^T V[\bar{\nabla}_i], \quad (10)$$

where $V[\bar{\nabla}_t] = \mathbb{E}[\|\bar{\nabla}_t - \nabla_t\|^2]$. We can bound the first two terms using standard techniques applicable for the squared loss (e.g., see Zhang, 2004; Srebro et al., 2010). The third term depends on our choice of gradient estimate. There are various choices for such an estimate, and we use a version which facilitates our analysis, as explained below.

Assume that at iteration t , our function f_t is given by $f_t = \sum_{i=1}^t \alpha_i^{(t)} \Phi(\mathbf{x}_t)$. We now want to use sampling to obtain an unbiased estimate of $\langle f_t, \Phi(\mathbf{x}_t) \rangle$. This will be done via a two step sampling procedure, as described in Algorithm 2. First, sample an index $i \in [1, \dots, t]$ by sampling according to the distribution $q(i) \propto |\alpha_i^{(t)}|$. Next, for the chosen i , sample $\bar{\mathbf{w}}$ according to μ , and use $\psi(\mathbf{x}; \bar{\mathbf{w}})\psi(\mathbf{x}_i; \bar{\mathbf{w}})$ to construct an estimate of $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_t) \rangle$. The resulting unbiased estimate of $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_t) \rangle$ is denoted by E_t and given by:

$$E_t = \frac{\|\alpha^{(t)}\|_1}{m} \sum_{i=1}^m \text{sgn}(\alpha_i^{(t)}) \psi(\mathbf{x}_i; \bar{\mathbf{w}}) \psi(\mathbf{x}_t; \bar{\mathbf{w}}) \quad (11)$$

The corresponding unbiased gradient estimate is:

$$\bar{\nabla}_t = (E_t - y_t) \mathbf{x}_t \quad (12)$$

The variance of $\bar{\nabla}$ affects the convergence rate and depends on both $\|\alpha\|_1$ and the number of estimations m . We wish to maintain $m = O(T)$ estimations per round, while achieving $O(\sqrt{T})$ regret.

To effectively regularize $\|\alpha\|_1$, we modify the OGD algorithm so that whenever E_t is larger than $16B$, we do not perform the usual update. Instead, we perform a shrinking step that divides $\alpha^{(t)}$ (and hence f_t) by 4. Treating B as constant, this guarantees that $\|\alpha\|_1 = O(\eta T)$, and in turn $\text{Var}(\bar{\nabla}_t) = O(\frac{\eta^2 T^2}{m})$. Setting $\eta = O(1/\sqrt{T})$, we have that $m = O(T)$ estimations are sufficient.

The rationale for the shrinkage is that whenever E_t is large, it indicates that f_t is ‘‘far away’’ from the B -ball, and a shrinkage step, similar to projection, brings f_t closer to the optimal element in the B -ball. However, due to stochasticity, the shrinkage step does add a further term to the regret bound that we would need to take care of.

5.1. Analysis

In what follows we analyze the regret for Algorithm 1, and provide a high level proof of Theorem 1. The appendix provides the necessary lemmas and a more detailed proof. We begin by modifying the regret bound for OGD in Eq. 10 to accommodate for steps that differ from the standard gradient update, such as shrinkage. We use the following notation for the regret at iteration t :

$$R_t(f^*) = \mathbb{E} \left[\sum_{t=1}^T \ell_t(f_t) - \ell_t(f^*) \right] \quad (13)$$

Lemma 1. *Let ℓ_1, \dots, ℓ_T be an arbitrary sequence of convex loss functions, and let f_1, \dots, f_T be random vectors, produced by an online algorithm. Assume $\|f_i\| \leq B_T$ for all $i \leq T$. For each t let $\bar{\nabla}_t$ be an unbiased estimator of $\nabla \ell_t(f_t)$. Denote $\hat{f}_t = f_{t-1} - \eta \bar{\nabla}_{t-1}$ and let*

$$P_t(f^*) = \mathbb{P} \left[\|f_t - f^*\| > \|\hat{f}_t - f^*\| \right]. \quad (14)$$

For every $\|f^*\| \leq B$ it holds that :

$$R_t(f^*) \leq \frac{B^2}{\eta} + \eta \sum_{t=1}^T \mathbb{E}[\|\nabla_t\|^2] + \eta \sum_{t=1}^T V[\bar{\nabla}_t] + \sum_{t=1}^T \frac{(B_T + B)^2}{\eta} \mathbb{E}[P_t(f^*)] \quad (15)$$

See Appendix B.1 for proof of the lemma. As discussed earlier, the first three terms on the RHS are the standard bound for OGD from Eq. 10. Note that in the standard OGD it holds that $f_t = \hat{f}_t$, and therefore $P_t(f^*) = 0$ and the last term disappears.

The third term will be bounded by controlling $\|\alpha\|_1$. The last term $P_t(f^*)$ is a penalty that results from updates that stir f_t away from the standard update step \hat{f}_t . This will indeed happen for the shrinkage step. The next lemma bounds this term. See Appendix B.2 for proof.

Lemma 2. *Run Algorithm 1 with parameters $T, B \geq 1$ and $\eta < 1/8$. Let $\bar{\nabla}_t$ be the unbiased estimator of $\nabla \ell_t(f_t)$ of the form $\bar{\nabla}_t = (E_t - y_t) \Phi(\mathbf{x}_t)$. Denote $\hat{f}_t = f_t - \eta \bar{\nabla}_t$ and define $P_t(f^*)$ as in Eq. 14. Then:*

$$P_t(f^*) \leq 2 \exp \left(-\frac{m}{(3\eta t)^2} \right)$$

The following lemma (see Appendix B.3 for proof) bounds the second and third terms of Eq. 15.

Lemma 3. *Consider the setting as in Lemma 2. Then $V[\bar{\nabla}_t] \leq \frac{((16B+1)\eta t)^2}{m}$ and $\mathbb{E}[\|\nabla_t\|^2] \leq 2\mathbb{E}[\ell_t(f_t)]$.*

Proof of Theorem 1 Combining Lemmas 1, 2 and 3 and rearranging we get:

$$(1 - 2\eta) \mathbb{E}[R_t(f^*)] \leq \frac{B^2}{\eta} + 2\eta \sum_{t=1}^T \ell_t(f^*) + (16) \eta \frac{((16B+1)\eta T)^2 T}{m} + \frac{(B_T + B)^2}{\eta} \sum_{t=1}^T P_t(f^*)$$

To bound the second term in Eq. 16 we note that:

$$\min_{\|f^*\| < B} \sum_{t=1}^T \ell_t(f^*) \leq \sum_{t=1}^T \ell_t(0) \leq T. \quad (17)$$

We next set η and m as in the statement of the theorem. Namely: $\eta = \frac{B}{2\sqrt{T}}$, and $m = ((16B+1)B)^2 T \log \gamma$, where $\gamma = \max\left(\frac{((16B+1)\eta T + B)^2}{\eta^2}, e\right)$. This choice of m implies that $m > ((16B+1)\eta T)^2$, and hence the third term in Eq. 16 is upper bounded by T .

Next we have that $m > (3\eta t)^2 \log \gamma$ for every t , and by the bound on B_T we have that $\gamma > \frac{(B+B_T)^2}{\eta^2}$. Taken together with Lemma 2 we have that:

$$\frac{(B_T + B)^2}{\eta} \sum_{t=1}^T P_t(f^*) \leq \eta T. \quad (18)$$

The above bounds imply that:

$$(1 - 2\eta)\mathbb{E}[R_t(f^*)] \leq \frac{B^2}{\eta} + 2\eta T + \eta T + \eta T$$

Finally by choice of η , and dividing both sides by $(1 - 2\eta)$ we obtain the desired result.

6. Experiments

In this section we provide a toy experiment to compare our Shrinking Gradient algorithm to other random feature based methods. In particular, we consider the following three algorithms: **Fixed-Random**: Sample a set of r features $\mathbf{w}_1, \dots, \mathbf{w}_r$ and evaluate these on all the train and test points. Namely, all \mathbf{x} points will be evaluated on the same features. This is the standard random features approach proposed in [Rahimi & Recht \(2007; 2009\)](#). **Doubly Stochastic Gradient Descent (Dai et al., 2014)**: Here each training point \mathbf{x} samples k features $\mathbf{w}_1, \dots, \mathbf{w}_k$. These features will from that point on be used for evaluating dot products with \mathbf{x} . Thus, different \mathbf{x} points will use different features. **Shrinking Gradient**: This is the approach proposed here in Section 3. Namely, each training point \mathbf{x} samples m features in order to calculate the dot product with the current regression function.

In comparing the algorithms we choose r, k, m so that the same overall number of features is calculated. For all methods we explored different initial step sizes and schedules for changing the step size.

The key question in comparing the three algorithms is how well they use a given budget of random features. To explore this we perform an experiments to simulate the high dimensional feature case. We consider vectors $\mathbf{x} \in \mathbb{R}^D$, where a random feature w corresponds to a uniform choice of coordinate w in \mathbf{x} . We work in the regime where D is large in the sense that $D > T$, where T is the size of the training data. Thus random sampling of T features will not reveal all coordinates of \mathbf{x} . The training set is generated as follows. First, a training set $\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathbb{R}^D$ is

sampled from a standard Gaussian. We furthermore clip negative values to zero, in order to make the data sparser and more challenging for feature sampling. Next a weight vector $a \in \mathbb{R}^D$ is chosen as a random sparse linear combination of the training points. This is done in order for the true function to be in the corresponding RKHS. Finally, the training set is labeled using $y_i = a \cdot \mathbf{x}_i$.

During training we do not assume that the algorithms have access to \mathbf{x} . Rather they can uniformly sample coordinates from it, which mimics our setting of random features. For the experiment we take $D = 550, 600, \dots, 800$ and $T = 200$. All algorithms perform one pass over the data, to emulate the online regret setting. The results shown in Figure 1 show that our method indeed achieves a lower loss while working with the same feature budget.

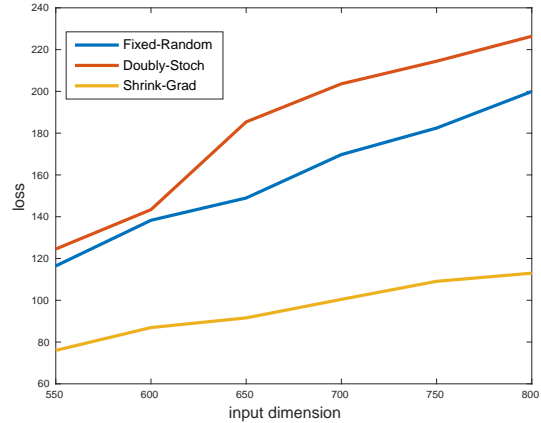


Figure 1. Comparison of three random feature methods. See Section 6 for details.

7. Discussion

We presented a new online algorithm that employs kernels implicitly but avoids the kernel trick assumption. Namely, the algorithm can be invoked even when one has access to only estimations of the scalar product. The problem was motivated by kernels resulting from neural nets, but it can of course be applied to any scalar product of the form we described. As an example of an interesting extension, consider a setting where a learner can observe an unbiased estimate of a coordinate in a kernel matrix, or alternatively the scalar product between any two observations. Our results imply that in this setting the above rates are applicable, and at least for the square loss, having no access to the true values in the kernel matrix is not necessarily prohibitive during training.

The results show that with sample size T we can achieve error of $O(\frac{B}{\sqrt{T}})$. As demonstrated in [Shamir \(2015\)](#) these

rates are optimal, even when the scalar product is computable. To achieve this rate our algorithm needs to perform $\tilde{O}(B^4 T^2)$ scalar product estimations. When the scalar product can be computed, existing kernelized algorithms need to observe a fixed proportion of the kernel matrix, hence they observe order of $\Omega(T^2)$ scalar products. In [Cesa-Bianchi et al. \(2015\)](#) it was shown that when the scalar product can be computed exactly, one would need access to at least $\Omega(T)$ entries to the kernel matrix. It is still an open problem whether one has to access $\Omega(T^2)$ entries when the kernel can be computed exactly. However, as we show here, for fixed B even if the kernel can only be estimated $\tilde{O}(T^2)$ estimations are enough. It would be interesting to further investigate and improve the performance of our algorithm in terms of the norm bound B .

To summarize, we have shown that the *kernel trick* is not strictly necessary in terms of sample complexity. Instead, simply sampling random features via our proposed algorithm results in a similar sample complexity. Recent empirical results by [Zhang et al. \(2016\)](#) show that using a large number of random features and regression comes close to the performance of the first successful multilayer CNNs ([Krizhevsky et al., 2012](#)) on CIFAR-10. Although deep learning architectures still substantially outperform random features, it is conceivable that with the right choice of random features, and scalable learning algorithms like we present here, considerable improvement in performance is possible.

A. Estimation Concentration Bounds

In this section we provide concentration bounds for the estimation procedure in [Algorithm 2](#).

Lemma 4. *Run [Algorithm 2](#) with α and, $\{\mathbf{x}_i\}_{i=1}^T$, \mathbf{x} , and m . Let $f = \sum \alpha_i \Phi(\mathbf{x}_i)$. Assume that $|\psi(\mathbf{x}; \mathbf{w})| < 1$ for all \mathbf{w} and \mathbf{x} . Let E be the output of [Algorithm 2](#). Then E is an unbiased estimator for $\langle f, \Phi(\mathbf{x}) \rangle$ and:*

$$\mathbb{P}[|E - \langle f, \Phi(\mathbf{x}) \rangle| > \epsilon] \leq \exp\left(-\frac{m\epsilon^2}{\|\alpha\|_1^2}\right) \quad (19)$$

Proof. Consider the random variables $\|\alpha\|_1 E^{(k)}$ (where $E^{(k)}$ is as defined in [Algorithm 2](#)) and note that they are IID. One can show that $\mathbb{E}[\|\alpha\|_1 E^{(k)}] = \sum \alpha_i \mathbb{E}[\psi(\mathbf{x}_i; \mathbf{w})\psi(\mathbf{x}; \mathbf{w})] = \langle f, \Phi(\mathbf{x}) \rangle$. By the bound on $\psi(\mathbf{x}; \mathbf{w})$ we have that $|\|\alpha\|_1 E^{(k)}| < \|\alpha\|_1$ with probability 1. Since $E = \frac{1}{m} \sum E^{(k)}$ the result follows directly from Hoeffding's inequality. \square

Next, we bound the $\alpha^{(t)}$ coefficients and obtain a concentration bound for the estimated dot product E_t .

Lemma 5. *The $\alpha^{(t)}$ obtained in [Algorithm 1](#) satisfies:*

$$\|\alpha^{(t)}\|_1 \leq (16B + 1)\eta t.$$

As a corollary of this and [Lemma 4](#) we have that the function f_t satisfies:

$$\mathbb{P}[|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| > \epsilon] \leq \exp\left(-\frac{\epsilon^2 m}{((16B + 1)\eta t)^2}\right) \quad (20)$$

Proof. We prove the statement by induction. We separate into two cases, depending on whether the shrinkage step was performed or not.

If $|E_t| \geq 16B$ the algorithm sets $\alpha^{(t+1)} = \frac{1}{4}\alpha^{(t)}$, and:

$$\|\alpha^{(t+1)}\|_1 = \frac{1}{4}\|\alpha^{(t)}\|_1 \leq (16B + 1)\eta(t + 1)$$

If $|E_t| < 16B$ the gradient update is performed. Since $|y_t| \leq 1$ we have that $|E_t - y_t| < 16B + 1$ and:

$$\|\alpha^{(t+1)}\|_1 \leq \|\alpha^{(t)}\|_1 + \eta|E_t - y_t| \leq (16B + 1)\eta(t + 1). \quad \square$$

B. Proofs of Lemmas

B.1. Proof of [Lemma 1](#)

First, by convexity we have that

$$2(\ell_t(f_t) - \ell_t(f^*)) \leq 2\langle \nabla_t, f_t - f^* \rangle. \quad (21)$$

Next we upper bound $\langle \nabla_t, f_t - f^* \rangle$. Denote by \mathcal{E} the event $\|\hat{f}_{t+1} - f^*\| > \|\hat{f}_{t+1} - f^*\|$. Note that:

$$\begin{aligned} \mathbb{E}[\|f_{t+1} - f^*\|^2] &\leq \mathbb{E}[\|\hat{f}_{t+1} - f^*\|^2] + \\ &\mathbb{E}[\|f_{t+1} - f^*\|^2 | \mathcal{E}] \cdot P_{t+1}(f^*) \\ &\leq \mathbb{E}[\|\hat{f}_{t+1} - f^*\|^2] + (B + B_T)^2 P_{t+1}(f^*) \end{aligned}$$

Plugging in $\hat{f}_{t+1} = f_t - \eta \bar{\nabla}_t$, summing over t and using [Eq. 21](#) and $\mathbb{E}[\|\bar{\nabla}_t\|^2] = \mathbb{E}[\|\nabla_t\|^2] + V[\bar{\nabla}_t]$, we obtain the desired result.

B.2. Proof for [Lemma 2](#)

To prove the bound in the lemma, we first bound the event $P_t(f^*)$ w.r.t to two possible events:

Lemma 6. *Consider the setting as in [Lemma 2](#). Run [Algorithm 1](#) and for each t consider the following two events:*

- \mathcal{E}_1^t : $|E_t| > 16B$ and $|E_t| > \frac{1}{4\eta}\|f_t\|$.
- \mathcal{E}_2^t : $|E_t| > 16B$ and $\|f_t\| < 8B$.

For every $\|f^\| < B$ we have that $P_t(f^*) < \mathbb{P}[\mathcal{E}_1^t \cup \mathcal{E}_2^t]$.*

Proof. Denote the event $|E_t| > 16B$ by \mathcal{E}_0^t . Note that if \mathcal{E}_0^t does not happen, then $f_t = \hat{f}_t$. Hence trivially

$$P_t(f^*) = \mathbb{P} \left[\|f_t - f^*\| > \|\hat{f}_t - f^*\| \wedge \mathcal{E}_0^t \right]$$

We will assume that: (1) $|E_t| > 16B$, (2) $|E_t| < \frac{1}{4\eta}\|f_t\|$, (3) $\|f_t\| > 8B$. We then show $\|f_{t+1} - f^*\| \leq \|\hat{f}_{t+1} - f^*\|$.

In other words, we will show that if \mathcal{E}_0^t happens and $\|f_{t+1} - f^*\| > \|\hat{f}_{t+1} - f^*\|$, then either \mathcal{E}_2^t or \mathcal{E}_1^t happened. This will conclude the proof.

Fix t , note that since $|\psi(\mathbf{x}; \mathbf{w})| < 1$ we have that $\|\Phi(\mathbf{x})\| < 1$. We then have:

$$\begin{aligned} \|\hat{f}_{t+1}\| &= \|f_t - \eta(E_t - y)\Phi(\mathbf{x}_t)\| \\ &\geq \|f_t\| - \eta|E_t| - \eta \geq \frac{3}{4}\|f_t\| - \eta \end{aligned} \quad (22)$$

where the last inequality is due to assumption (2) above. We therefore have the following for every $\|f^*\| < B$:

$$\|\hat{f}_{t+1} - f^*\| \geq \frac{3}{4}\|f_t\| - \eta - B$$

On the other hand, if $f_{t+1} \neq \hat{f}_{t+1}$ then by construction of the algorithm $f_{t+1} = \frac{1}{4}f_t$:

$$\|f_{t+1} - f^*\| \leq \|f_{t+1}\| + \|f^*\| \leq \frac{\|f_t\|}{4} + B.$$

Next note that $\eta < 2B$ and assumption (3) states $\|f_t\| > 8B$. Therefore: $\frac{1}{2}\|f_t\| > 4B > \eta + 2B$, and:

$$\begin{aligned} \|\hat{f}_{t+1} - f^*\| &\geq \frac{3}{4}\|f_t\| - \eta - B \\ &= \frac{1}{4}\|f_t\| + \left(\frac{1}{2}\|f_t\| - \eta - 2B\right) + B \\ &\geq \frac{1}{4}\|f_t\| + B \geq \|f_{t+1} - f^*\| \end{aligned}$$

□

Next we upper bound $\mathbb{P}[\mathcal{E}_1^t \cup \mathcal{E}_2^t]$. In what follows the superscript t is dropped.

A bound for $\mathbb{P}[\mathcal{E}_1 \cap \mathcal{E}_2^c]$: Assume that

$$|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| < \left(\frac{1}{4\eta} - 1\right)8B.$$

We assume T is sufficiently large and $\eta < \frac{1}{8}$. We have $\frac{1}{4\eta} - 1 > 1$. Since we assume \mathcal{E}_2 did not happen we must have $\|f_t\| > 8B$ and $|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| < \left(\frac{1}{4\eta} - 1\right)\|f\|$, and therefore:

$$E_t - \|f\| < |E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| < \left(\frac{1}{4\eta} - 1\right)\|f\|.$$

Which implies $E_t < \frac{1}{4\eta}\|f\|$, and we get that \mathcal{E}_1 did not happen. We conclude that if \mathcal{E}_1 and not \mathcal{E}_2 then:

$$|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| \geq \left(\frac{1}{4\eta} - 1\right)8B.$$

Since $\frac{1}{4\eta} - 1 > 1$ we have that: $|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| \geq 8B$, leading to:

$$\mathbb{P}[\mathcal{E}_1 \cap \mathcal{E}_2^c] \leq \mathbb{P}[|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| \geq 8B]. \quad (23)$$

A bound for $\mathbb{P}[\mathcal{E}_2]$: If $|E_t| > 16B$ and $\|f_t\| < 8B$ then by normalization of $\Phi(\mathbf{x}_t)$ we have that $\langle f_t, \Phi(\mathbf{x}_t) \rangle < 8B$ and trivially we have that $|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| \geq 8B$, and therefore:

$$\mathbb{P}[\mathcal{E}_2] \leq \mathbb{P}[|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| \geq 8B]. \quad (24)$$

Taking Eq. 23 and Eq. 24 we have that

$$\mathbb{P}[\mathcal{E}_2 \cup \mathcal{E}_1] \leq 2\mathbb{P}[|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| \geq 8B]. \quad (25)$$

By Lemma 5 we have that:

$$\begin{aligned} P(|E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle| > 8B) &< \\ \exp\left(-\frac{m(8B)^2}{((16B+1)\eta t)^2}\right) &< \exp\left(-\frac{m}{(3\eta t)^2}\right) \end{aligned}$$

Taking the above upper bounds together with Lemma 6 we can prove Lemma 2.

B.3. Proof of Lemma 3

Begin by noting that since $\|\Phi(\mathbf{x})\| < 1$, it follows from the definitions of $\nabla, \bar{\nabla}$ that $V[\bar{\nabla}_t] = \mathbb{E}[\|\bar{\nabla}_t - \nabla_t\|^2]$ and therefore

$$V[\bar{\nabla}_t] \leq \mathbb{E}[(E_t - \langle f_t, \Phi(\mathbf{x}_t) \rangle)^2] = V[E_t]$$

By construction (see Algorithm 2) we have that:

$$V[E_t] = \frac{1}{m} V[\|\alpha^{(t)}\|_1^2 \psi(\mathbf{x}_i; \mathbf{w}) \psi(\mathbf{x}_t; \mathbf{w})]$$

where the index i is sampled as in Algorithm 2, and $\psi(\mathbf{x}_i; \mathbf{w}) \psi(\mathbf{x}_t; \mathbf{w})$ is bounded by 1. By Lemma 5 we have that

$$V[E_t] \leq \frac{((16B+1)\eta t)^2}{m}.$$

This provides the required bound on $V[\bar{\nabla}_t]$. Additionally, we have that

$$\|\nabla_t\|^2 = (\langle f_t, \Phi(\mathbf{x}_t) \rangle - y_t)^2 \|\Phi(\mathbf{x}_t)\|^2 \leq 2\ell_t(f_t)$$

and the result follows by taking expectation.

Acknowledgements The authors would like to thank Tomer Koren for helpful discussions. Roi Livni was supported by funding from Eric and Wendy Schmidt Fund for Strategic Innovation. This work was supported by the Blavatnik Computer Science Research Fund, the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), and an ISF Centers of Excellence grant.

References

- Bach, Francis. Breaking the curse of dimensionality with convex neural networks. *arXiv preprint arXiv:1412.8690*, 2014.
- Bach, Francis. On the equivalence between kernel quadrature rules and random feature expansions. *arXiv preprint arXiv:1502.06800*, 2015.
- Bo, Liefeng, Ren, Xiaofeng, and Fox, Dieter. Kernel descriptors for visual recognition. In *Advances in neural information processing systems*, pp. 244–252, 2010.
- Bo, Liefeng, Lai, Kevin, Ren, Xiaofeng, and Fox, Dieter. Object recognition with hierarchical kernel descriptors. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 1729–1736. IEEE, 2011.
- Bouvier, Jake, Rosasco, Lorenzo, and Poggio, Tomaso. On invariance in hierarchical models. In *Advances in Neural Information Processing Systems*, pp. 162–170, 2009.
- Cesa-Bianchi, Nicolo, Shalev-Shwartz, Shai, and Shamir, Ohad. Efficient learning with partially observed attributes. *The Journal of Machine Learning Research*, 12:2857–2878, 2011a.
- Cesa-Bianchi, Nicolo, Shalev-Shwartz, Shai, and Shamir, Ohad. Online learning of noisy data. *Information Theory, IEEE Transactions on*, 57(12):7907–7931, 2011b.
- Cesa-Bianchi, Nicolò, Mansour, Yishay, and Shamir, Ohad. On the complexity of learning with kernels. In *Proceedings of The 28th Conference on Learning Theory*, pp. 297–325, 2015.
- Cho, Youngmin and Saul, Lawrence K. Kernel methods for deep learning. In *Advances in neural information processing systems*, pp. 342–350, 2009.
- Cortes, Corinna and Vapnik, Vladimir. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- Dai, Bo, Xie, Bo, He, Niao, Liang, Yingyu, Raj, Anant, Balcan, Maria-Florina F, and Song, Le. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pp. 3041–3049, 2014.
- Daniely, Amit, Linial, Nati, and Shalev-Shwartz, Shai. From average case complexity to improper learning complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pp. 441–448. ACM, 2014.
- Daniely, Amit, Frostig, Roy, and Singer, Yoram. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2253–2261. Curran Associates, Inc., 2016.
- Deng, Li, Tur, Gokhan, He, Xiaodong, and Hakkani-Tur, Dilek. Use of kernel deep convex networks and end-to-end learning for spoken language understanding. In *Spoken Language Technology Workshop (SLT), 2012 IEEE*, pp. 210–215. IEEE, 2012.
- Hazan, Elad and Koren, Tomer. Linear regression with limited observation. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 807–814, 2012.
- Hazan, Tamir and Jaakkola, Tommi. Steps toward deep kernel methods from infinite neural networks. *arXiv preprint arXiv:1508.05133*, 2015.
- Heinemann, Uri, Livni, Roi, Eban, Elad, Elidan, Gal, and Globerson, Amir. Improper deep kernels. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 1159–1167, 2016.
- Hornik, Kurt. Some new results on neural network approximation. *Neural Networks*, 6(8):1069–1072, 1993.
- Kar, Purushottam and Karnick, Harish. Random feature maps for dot product kernels. In *International Conference on Artificial Intelligence and Statistics*, pp. 583–591, 2012.
- Kivinen, Jyrki, Smola, Alexander J, and Williamson, Robert C. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.
- Klivans, Adam R and Sherstov, Alexander A. Cryptographic hardness for learning intersections of halfspaces. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pp. 553–562. IEEE, 2006.
- Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Image classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.
- Livni, Roi, Shalev-Shwartz, Shai, and Shamir, Ohad. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pp. 855–863, 2014.
- Mairal, Julien, Koniusz, Piotr, Harchaoui, Zaid, and Schmid, Cordelia. Convolutional kernel networks. In *Advances in Neural Information Processing Systems*, pp. 2627–2635, 2014.
- Minsky, Marvin and Papert, Seymour. Perceptrons: an introduction to computational geometry (expanded edition), 1988.
- Rahimi, Ali and Recht, Benjamin. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pp. 1177–1184, 2007.
- Rahimi, Ali and Recht, Benjamin. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, pp. 1313–1320, 2009.
- Shalev-Shwartz, Shai. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2): 107–194, 2011.
- Shamir, Ohad. The sample complexity of learning linear predictors with the squared loss. *Journal of Machine Learning Research*, 16(Dec):3475–3486, 2015.
- Srebro, Nathan, Sridharan, Karthik, and Tewari, Ambuj. Smoothness, low noise and fast rates. In *Advances in neural information processing systems*, pp. 2199–2207, 2010.
- Sridharan, Karthik, Shalev-Shwartz, Shai, and Srebro, Nathan. Fast rates for regularized objectives. In *Advances in Neural Information Processing Systems*, pp. 1545–1552, 2009.
- Suykens, Johan AK and Vandewalle, Joos. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

Suykens, Johan AK, Van Gestel, Tony, and De Brabanter, Jos. *Least squares support vector machines*. World Scientific, 2002.

Williams, Christopher. Computing with infinite networks. *Advances in neural information processing systems*, pp. 295–301, 1997.

Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, and Vinyals, Oriol. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Zhang, Tong. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 116. ACM, 2004.

Zinkevich, Martin. Online convex programming and generalized infinitesimal gradient ascent. In *Machine Learning, Proceedings of the Twentieth International Conference*, pp. 928–936, 2003.