Improper Deep Kernels

Uri Heinemann
The Hebrew University

Roi Livni
The Hebrew University

Elad Eban
Google Inc.

Gal Elidan
The Hebrew University

Amir Globerson
Tel-Aviv University

Abstract

Neural networks have recently re-emerged as a powerful hypothesis class, yielding impressive classification accuracy in multiple domains. However, their training is a non-convex optimization problem which poses theoretical and practical challenges. Here we address this difficulty by turning to "improper" learning of neural nets. In other words, we learn a classifier that is not a neural net but is competitive with the best neural net model given a sufficient number of training examples. Our approach relies on a novel kernel construction scheme in which the kernel is a result of integration over the set of all possible instantiation of neural models. It turns out that the corresponding integral can be evaluated in closed-form via a simple recursion. Thus we translate the non-convex, hard learning problem of a neural net to a SVM with an appropriate kernel. We also provide sample complexity results which depend on the stability of the optimal neural net.

1 Introduction

Deep learning architectures have re-surfaced in the last decade as a powerful hypothesis class that can capture complex mappings from inputs to target classes via multiple layers of non-linear transformations. Using several core training and modeling innovations, applications relying on deep architectures have brought these models into the focus of the machine learning community, achieving state-of-the-art performance in varied domains ranging from machine vision [12] to natural language processing and speech recognition [9].

While the practical potential of deep learning is undeniable, training such models involves difficult non-convex optimization, requires the use of a range of heuristics, and typically relies on architectures that are quite complex in nature. This is in stark contrast to the previous *trend* in machine learning, namely support vector machines, that achieve non-linearity using the so called *kernel trick*, and that are trained using quadratic programming [17].

Consequently, an obvious intriguing question is whether the power of deep architectures can be leveraged within the context of kernel methods. In an elegant approach to this problem, Cho and Saul [4] suggested a new family of kernels that "mimic the computation in large neural networks". Briefly, they provide a kernel whose features are the output of all possible hidden units, for the continuum of weight vectors. A nice trick allows them to apply this kernel recursively resulting in a deep infinite network. Their work is also similar in spirit to other continuous neural net formalism (e.g., see [16]). The above works have employed kernels by considering infinite deep nets. A key question, which we address here, is whether kernels can be employed in the context of finite architectures (i.e., a discrete number of hidden units). We show that this can be achieved via an "improper learning" approach (e.g., see [18, 6]). In the improper learning approach, the learner is not required to output a function which belongs to a given hypothesis class. Instead, the learner can return a function from an arbitrary class, and the goal is that the function will perform at least as well as any function from the given class. The benefit from this approach is that the learning problem may, in some cases, become computationally tractable [20].

Here we follow an improper learning approach by extending the class of neural net classifiers to weighted combinations of such classifiers. The weighting function is high dimensional and continuous and therefore seems hard to optimize at first. However, we show that this problem can be overcome via a closed-form expression which specifies a kernel, that can be used within a standard SVM optimizer. We complement our algorithm with sample complexity bounds which illustrate the trade-off between data size and tractability.

Finally, we evaluate our improper deep kernel on simulated data as well as object recognition benchmarks.

Appearing in Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 41.

2 Parametric Improper Learning

Our improper learning approach will take a parametric hypothesis class of interest (e.g., neural networks) and extend it, such that the new extended class can be learned with kernels. We begin by describing this general approach. In the next section we will make this idea concrete, and provide an efficient algorithm to compute the kernel for the hypothesis class of deep neural networks.

For simplicity in what follows we consider the binary classification task, where a label $y \in \{0,1\}$ is predicted from an input $\boldsymbol{x}$. Given a parametric family of functions $f(\boldsymbol{x},\boldsymbol{w})$ (e.g., an L layered neural network with a scalar output), the predicted label is:

$$y = \Theta\left(f(\boldsymbol{x},\boldsymbol{w})\right) \qquad \Theta\left(z\right) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases} \tag{1}$$

In the case of a neural network, $\boldsymbol{w}$ are the weights assigned to the different neurons at all levels of the network.

Denote the training set with M samples by $\{\boldsymbol{x}^m, y^m\}_{m=1}^M$. In our setting, we aim to discriminatively learn a set of parameters $\boldsymbol{w}$ which minimizes the classification loss on the training set, namely to minimize the zero-one loss:

$$\sum_{m=1}^{M} \mathbb{1}\left(y^m \neq \Theta\left(f(\boldsymbol{x}^m,\boldsymbol{w})\right)\right),$$

where $\mathbb{1}()$ is the indicator function and the summation is over training instances.

The above combinatorial, non-convex loss is computationally hard to minimize. The standard approach to circumventing this problem is to use a convex surrogate of the zero-one loss, and solve the resulting convex minimization problem. Here we will consider the hinge loss:[1]

$$\sum_{m=1}^{M} \max\left[1 - 2(y^m - 0.5) f(\boldsymbol{x}^m,\boldsymbol{w}), 0\right]. \tag{2}$$

When the function $f(\cdot,\boldsymbol{w})$ is linear in $\boldsymbol{w}$ (as in standard SVM), the above would be convex in $\boldsymbol{w}$, making the optimization problem tractable. However, for $f(\cdot,\boldsymbol{w})$ derived from a general parametric model, and in particular from a neural network, this will generally not be the case. We thus adopt a different approach, which will lead to a convex optimization problem.

Our approach is to define a new feature space and a hypothesis-class, such that the resulting prediction rules are a superset of those obtained in Equation 1, and thus of stronger expressive power. Therefore, we think of this approach as an instance of *improper learning* of the discriminative classifier (e.g., see [6] for a review of improper learning and related sample and computational complexity results).

Formally, denote by $\mathcal{F}$ the set of functions from the parameter vector $\boldsymbol{w}$ to the reals. Define the map $\psi : X \to \mathcal{F}$ as follows:

$$\psi(\boldsymbol{x}) = f(\boldsymbol{x},\boldsymbol{w}). \tag{3}$$

The mapping ψ transforms the input vector $\boldsymbol{x}$ into a feature *function* from the domain of $\boldsymbol{w}$ to the reals. We now consider linear classifiers in this feature space. Each such classifier is defined via a weight function $\alpha(\boldsymbol{w})$, and the output label for a given input $\boldsymbol{x}$ is:

$$y = \Theta\left(\int f(\boldsymbol{x},\boldsymbol{w})\alpha(\boldsymbol{w})\, d\boldsymbol{w}\right). \tag{4}$$

Denote the set of classifiers of the form in Equation 1 by $\mathcal{A}$ and the set of classifiers as in Equation 4 by $\mathcal{A}^+$. Then clearly $\mathcal{A}^+ \supseteq \mathcal{A}$ since the classifier $f(\boldsymbol{x},\boldsymbol{w}_0)$ can be obtained from Equation 4 by choosing $\alpha(\boldsymbol{w}) = \delta(\boldsymbol{w} - \boldsymbol{w}_0)$, where $\delta()$ is the Dirac delta function.[2]

The classifiers in $\mathcal{A}^+$ can be thought of as mixtures of classifiers in $\mathcal{A}$. We could have further constrained $\alpha(\boldsymbol{w})$ to be a density, in which case it could have been interpreted as a prior over $\mathcal{A}$. However, we do not introduce this additional constraint, as it limits expressive power and at the same time complicates optimization.

We now turn to show how classifiers in $\mathcal{A}^+$ can be learned efficiently. Note that the expression in the sum of Equation 2 is linear so that the following regularized hinge loss minimization problem is convex in the function $\alpha(\boldsymbol{w})$:

$$\min_{\alpha(\boldsymbol{w})} \quad \sum_{m=1}^{M} \max\left[1 - y^m \int f(\boldsymbol{x}^m,\boldsymbol{w})\alpha(\boldsymbol{w})\, d\boldsymbol{w}\ , 0\right] + \frac{C}{2}\int \alpha^2(\boldsymbol{w})\, d\boldsymbol{w}. \tag{5}$$

Despite the convexity of this objective in $\alpha(\boldsymbol{w})$ it is still not clear how to optimize it efficiently, since $\alpha(\boldsymbol{w})$ is a function over a high dimensional parameter set $\boldsymbol{w}$. However, it turns out that we can now use the *kernel trick* [17], and in turn optimize the above problem efficiently, as shown next.

We start be defining the inner product or kernel function in this case to be:[3]

$$K(\boldsymbol{x},\boldsymbol{x}') = \int f(\boldsymbol{x},\boldsymbol{w}) f(\boldsymbol{x}',\boldsymbol{w}) d\boldsymbol{w}. \tag{6}$$

[1] Note that we are using $\{0,1\}$ labels, and hence the expression is different from that for $\{-1,+1\}$ labels. The former is more appropriate for our later derivation.

[2] There is a technical subtlety here since δ is a generalized function, but it is a limit of continuous differentiable functions, so that f can be approximated arbitrarily well by such a continuous $\alpha(\boldsymbol{w})$ as well.

[3] We assume that the integral in Equation 6 is bounded so that the kernel exists. In later sections we propose an approach for ensuring finiteness.

The representer theorem [17] in this context states that if $\alpha^*(\boldsymbol{w})$ is the solution to Equation 5, then there exist coefficients $\beta_1, \ldots, \beta_M$ such that:

$$\int f(\boldsymbol{x}, \boldsymbol{w})\alpha^*(\boldsymbol{w})d\boldsymbol{w} = \sum_i \beta_i y^i K(\boldsymbol{x}, \boldsymbol{x}_i).$$

The coefficients $\boldsymbol{\beta}$ can be found as follows (e.g., see [17]). Define the $M \times M$ kernel matrix $\mathcal{K}$ such that $\mathcal{K}_{lm} = y^l y^m K(\boldsymbol{x}^l, \boldsymbol{x}^m)$. The $\boldsymbol{\beta}$ are then the solution to the following dual quadratic program:

$$\max_{\boldsymbol{\beta}} \sum_i \beta_i - \frac{1}{2}\boldsymbol{\beta}^T \mathcal{K}\boldsymbol{\beta} \qquad s.t. 0 \leq \beta_i \leq C.$$

In other words we have obtained a standard SVM dual which uses the kernel defined in Equation 6. What remains is to show how the general form of our kernel can be evaluated for the case when $f(\boldsymbol{x}; \boldsymbol{w})$ is a finite deep neural network. This is discussed in the next section.

3 A Kernel for Deep Networks

One of the key insights of the current work is that for deep neural networks with threshold activation functions, the kernel in Equation 6 can be calculated in closed form. In this section, we derive the resulting kernel.

We begin with some notations. The architecture of a depth M neural network will be defined via integers $N_0, N_1, \ldots, N_M$ representing the number of neurons in each layer. Here $N_0 = d$, the dimension of the input, and $N_{M+1} = 1$ since we are considering a scalar output. A weight matrix $W_k \in \mathbb{R}^{N_k \times N_{k-1}}$ parameterizes the transition of outputs from layer $k-1$ into the input of layer k. Since the last layer is a scalar we denote the last weight vector by $\boldsymbol{w}_{M+1}$. We use $\boldsymbol{w}_k^i$ do denote the ith row of the matrix W_k which corresponds to input to the $i'th$ neuron at level k. The set of weights $W_1, \ldots, W_k$ will be denoted by $\mathcal{W}_{1:k}$, and the overall set of weights by $\mathcal{W}$.

Recall that $\Theta(\cdot)$ is the threshold activation function defined in Equation 1. We also apply it to vectors in an element-wise fashion. The input into the kth layer will be a vector $\boldsymbol{z}_k \in \mathbb{R}^{N_k}$, defined recursively as:

$$\boldsymbol{z}_k(\boldsymbol{x}, \mathcal{W}_{1:k}) = \Theta\left(W_k \boldsymbol{z}_{k-1}(\boldsymbol{x}, \mathcal{W}_{1:k-1})\right) \qquad k > 1,$$

where $\boldsymbol{z}_1 = \Theta(W_1 \boldsymbol{x})$ is the input into the first layer. The $\boldsymbol{z}$ depend recursively on $\boldsymbol{x}$ and the other parameters, but this dependence will be dropped when clear from context.

The output of the final, linear, layer is:

$$f(\boldsymbol{x}; \mathcal{W}) = \boldsymbol{w}_{M+1}^T \boldsymbol{z}_M(\boldsymbol{x}, \mathcal{W}_{1:M}).$$

As discussed in the previous section, the output class is then $\Theta(f(\boldsymbol{x}; \mathcal{W}))$.

Using a well known integral [8, 4], the following function of two vectors $\boldsymbol{v}, \boldsymbol{v}'$ will be useful throughout:

$$\begin{aligned} H(\boldsymbol{v}, \boldsymbol{v}') &\equiv \int \Theta\left(\boldsymbol{w}^T \boldsymbol{v}\right) \Theta\left(\boldsymbol{w}^T \boldsymbol{v}'\right) d\boldsymbol{w} \\ &= \frac{1}{2} - \frac{1}{2\pi} \arccos \frac{\boldsymbol{v} \cdot \boldsymbol{v}'}{\|\boldsymbol{v}\|_2 \|\boldsymbol{v}'\|_2}. \end{aligned} \tag{7}$$

We shall also make use of the following function:

$$J(k, l, m) \equiv 0.5 - \frac{1}{2\pi} \arccos \frac{m}{\sqrt{k}\sqrt{l}}. \tag{8}$$

Finally, we use the binomial coefficient:

$$\mathcal{B}(N_n, k, s, s') = \binom{N_n}{k, s-k, s'-k, N_n - s - s' + k}.$$

The following theorem provides a recursive expression for the kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ in Equation 6.

Theorem 3.1. *Consider a neural network with architecture $N_0, \ldots, N_M$. Assume that the W_ks are independent and that the distribution of $\boldsymbol{w}_k^i$ is uniform on the ball. The kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ in Equation 6 is given by:*

$$K(\boldsymbol{x}, \boldsymbol{x}') = V_{M,0}(\boldsymbol{x}, \boldsymbol{x}'),$$

where $V_{n,q}$ is defined recursively using:

$$\begin{aligned} V_{n,q}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{s=1}^{N_{n-1}} \sum_{s'=1}^{N_{n-1}} \sum_{k=[s+s'-N_{n-1}]_+}^{\min\{s,s'\}} \Big(& \\ J^{N_n - q}(s, s', k)(0.5 - J(s, s', k))^q \mathcal{B}(N_{n-1}, k, s, s') & \\ V_{n-1, s+s'-2k}(\boldsymbol{x}, \boldsymbol{x}') \Big) & \end{aligned}$$

The base of the recursion is:

$$V_{1,q}(\boldsymbol{x}, \boldsymbol{x}') = H^{N_1 - q}(\boldsymbol{x}, \boldsymbol{x}')(0.5 - H(\boldsymbol{x}, \boldsymbol{x}'))^q.$$

Proof. For compactness we will use the following shorthands: $\boldsymbol{z}_M \equiv \boldsymbol{z}_M(\boldsymbol{x}', \mathcal{W}_{1:M})$, and similarly $\boldsymbol{z}'_M \equiv \boldsymbol{z}_M(\boldsymbol{x}', \mathcal{W}_{1:M})$. The kernel is defined as:

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{x}') &= \int f(\boldsymbol{x}, \mathcal{W}) f(\boldsymbol{x}', \mathcal{W}) d\mathcal{W} \\ &= \int \boldsymbol{w}_{M+1}^T \boldsymbol{z}_M \boldsymbol{w}_{M+1}^T \boldsymbol{z}'_M d\mathcal{W} \end{aligned}$$

Using the fact that $\mathbb{E}\left[\boldsymbol{w}_{M+1} \boldsymbol{w}_{M+1}^T\right] = \frac{1}{N_M} I$ we have:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{N_M} \int \boldsymbol{z}_M^T \boldsymbol{z}'_M d\mathcal{W}_{1:M}$$

The integral can be broken down as follows:

$$
\begin{aligned}
&N_M K(\boldsymbol{x}, \boldsymbol{x}') = \\
&\int \left(\int \Theta\left(W_M \boldsymbol{z}_{M-1}\right)^T \Theta\left(W_M \boldsymbol{z}'_{M-1}\right) dW_M \right) d\mathcal{W}_{1:M-1} \\
&\int \left(\int \sum_{i=1}^{N_M} \Theta\left(\boldsymbol{w}_M^i \boldsymbol{z}_{M-1}\right) \Theta\left(\boldsymbol{w}_M^i \boldsymbol{z}'_{M-1}\right) dW_M \right) d\mathcal{W}_{1:M-1} \\
&\int \left(\sum_{i=1}^{N_M} \int \Theta\left(\boldsymbol{w}_M^i \boldsymbol{z}_{M-1}\right) \Theta\left(\boldsymbol{w}_M^i \boldsymbol{z}'_{M-1}\right) d\boldsymbol{w}_M^i \right) d\mathcal{W}_{1:M-1} \\
&N_M \int \left(\int \Theta\left(\boldsymbol{w}_M \boldsymbol{z}_{M-1}\right) \Theta\left(\boldsymbol{w}_M \boldsymbol{z}'_{M-1}\right) d\boldsymbol{w}_M \right) d\mathcal{W}_{1:M-1} \\
&N_M \int H(\boldsymbol{z}_{M-1}, \boldsymbol{z}'_{M-1}) d\mathcal{W}_{1:M-1}
\end{aligned}
$$

In order to compute this, we will make use of the following auxiliary integral:

$$
\begin{aligned}
&V_{j,q}(\boldsymbol{x}, \boldsymbol{x}') \equiv \\
&\int H^{N_j - q}(\boldsymbol{z}_{j-1}, \boldsymbol{z}'_{j-1})(0.5 - H(\boldsymbol{z}_{j-1}, \boldsymbol{z}'_{j-1})^q d\mathcal{W}_{1:j-1}
\end{aligned} \tag{9}
$$

Note that from the above $K(\boldsymbol{x}, \boldsymbol{x}') = V_{M,0}$. Thus, to prove our result, it remains to develop the recursive computation of $V_{j,q}$.

The vectors $\boldsymbol{z}$ have only $\{0, 1\}$ values. Furthermore, the function H depends only on the dot product of $\boldsymbol{z}(\boldsymbol{x}) \cdot \boldsymbol{z}(\boldsymbol{x}')$ and their norms. Thus, we only need to consider all possible values for these dot products and norms. Accordingly, we are interested in integrals over W conditioned on certain $\boldsymbol{z}$ vectors. For two vectors $\boldsymbol{v}, \boldsymbol{v}' \in \{0, 1\}^{N_j}$ define the set of $\mathcal{W}_{1:j}$ such that $\boldsymbol{z}(\boldsymbol{x}, \mathcal{W}_{1:j}) = \boldsymbol{v}, \boldsymbol{z}(\boldsymbol{x}', \mathcal{W}_{1:j}) = \boldsymbol{v}'$:

$$
\begin{aligned}
&C_j(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x}') \equiv \\
&\quad \{\mathcal{W}_{1:j} : \boldsymbol{z}_j(\boldsymbol{x}, \mathcal{W}_{1:j}) = \boldsymbol{v}, \boldsymbol{z}_j(\boldsymbol{x}', \mathcal{W}_{1:j}) = \boldsymbol{v}'\}
\end{aligned}
$$

Using this definition, and recalling the auxiliary function Equation 8, we can rewrite Equation 9 by breaking it down according to the norms and dot products of the $\boldsymbol{z}$ vectors:

$$
\begin{aligned}
V_{j,q}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{s=1}^{N_{j-1}} \sum_{s'=1}^{N_{j-1}} \Bigg(& \\
\sum_{k=[s+s'-N_{j-1}]_+}^{\min\{s,s'\}} & J^{N_j - q}(s, s', k)(0.5 - J(s, s', k))^q \\
\sum_{\substack{\boldsymbol{v}, \boldsymbol{v}': \\ \|\boldsymbol{v}\|=s, \|\boldsymbol{v}'\|=s', \\ \boldsymbol{v} \cdot \boldsymbol{v}'=k}} & \mathcal{V}\left(C_{j-1}(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x}')\right) \Bigg)
\end{aligned}
$$

where $\mathcal{V}()$ denotes the volume of the set of assignments. This still seems hard, since there are exponentially many assignments $\boldsymbol{v}$. However, using Lemma 3.2 below, we can rewrite V_j by counting how many $\boldsymbol{v}, \boldsymbol{v}'$ there are with given s, s', k and multiplying by the corresponding volume element. Using simple combinatorial arguments we have:

$$
\begin{aligned}
V_{j,q}(\boldsymbol{x}, \boldsymbol{x}') = \Bigg(& \\
\sum_{s,s',k} & J^{N_j - q}(s, s', k)(0.5 - J(s, s', k))^q \\
& \mathcal{B}(N_n, k, s, s') V_{j-1, N_{j-1}-s-s'+2k, s+s'-2k}(\boldsymbol{x}, \boldsymbol{x}') \Bigg),
\end{aligned}
$$

which gives the desired result. □

To complete the proof, the following lemma simplifies the form of the volume term $\mathcal{V}\left(C_j(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x}')\right)$:

Lemma 3.2. *Given two vectors $\boldsymbol{v}, \boldsymbol{v}'$ with $s = \|\boldsymbol{v}\|_1, s' = \|\boldsymbol{v}\|_1, k = \boldsymbol{v} \cdot \boldsymbol{v}'$, the volume of $C_j(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x}')$ is only dependent on $s, s', k, \boldsymbol{x}, \boldsymbol{x}'$, and is given by:*

$$
\mathcal{V}\left(C_j(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x}')\right) = V_{j, N_j - s - s' + 2k, s+s'-2k}(\boldsymbol{x}, \boldsymbol{x}').
$$

Proof. We write $\mathcal{V}\left(C_j(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x})\right)$ via an integral whose integrand takes the value of one on points in $C_j(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x}')$ and zero otherwise. This is done via the threshold functions $\Theta(\cdot)$ which are constructed to be 1 whenever $\boldsymbol{v}$ is obtained as the output of the previous layers, and a separation into the four binary cases. Below we abuse notation and use $\boldsymbol{v}$ to represent the indices for which $v_i = 1$. Similarly we use $\boldsymbol{v}^C$ to represent the complement set where $v_i = 0$. We denote the ith row of W_j by $\boldsymbol{w}_j^i$.

$$
\begin{aligned}
&\mathcal{V}\left(C_j(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x}')\right) = \\
&\int \Bigg(\prod_{i \in \boldsymbol{v} \cap \boldsymbol{v}'} \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right) \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right) d\boldsymbol{w}_j^i \\
&\quad \prod_{i \in \boldsymbol{v}^C \cap \boldsymbol{v}'^C} \int (1 - \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right))(1 - \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right)) d\boldsymbol{w}_j^i \\
&\quad \prod_{i \in \boldsymbol{v} \cap \boldsymbol{v}'^C} \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right) \left(1 - \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right)\right) d\boldsymbol{w}_j^i \\
&\quad \prod_{i \in \boldsymbol{v}^C \cap \boldsymbol{v}'} \int \left(1 - \Theta\left(\boldsymbol{w}^i{}_j \boldsymbol{z}_{j-1}\right)\right) \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right) d\boldsymbol{w}_j^i \Bigg) d\mathcal{W}^{j-1}.
\end{aligned}
$$

To simplify this, note that the inner integrals in the first line are simply of the form $H(\boldsymbol{z}_{j-1}, \boldsymbol{z}'_{j-1})$. The integrals of the second line are also of this form:

$$
\begin{aligned}
&\int (1 - \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right))(1 - \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right)) d\boldsymbol{w}_j^i \\
&= \quad 1 - \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right) d\boldsymbol{w}_j^i - \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right) d\boldsymbol{w}_j^i \\
&\qquad + \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right) \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right) d\boldsymbol{w}_j^i \\
&= \quad \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right) \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right) d\boldsymbol{w}_j^i \\
&= \quad H(\boldsymbol{z}_{j-1}, \boldsymbol{z}'_{j-1}).
\end{aligned}
$$

Similarly, the integrals of the third and fourth line are

$$\begin{aligned}
&\int (1-\Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right))\Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right) d\boldsymbol{w}_j^i \\
&= \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right) d\boldsymbol{w}_j^i - \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right)\Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right) d\boldsymbol{w}_j^i \\
&= 0.5 - \int \Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}_{j-1}\right)\Theta\left(\boldsymbol{w}_j^i \boldsymbol{z}'_{j-1}\right) d\boldsymbol{w}_j^i \\
&= 0.5 - H(\boldsymbol{z}_{j-1}, \boldsymbol{z}'_{j-1}).
\end{aligned}$$

Now, since the products of each line are the same, all that we need to derive the final result is the size of each product group. The size of $\boldsymbol{v} \cap \boldsymbol{v}'$ is k by definition and the size of $\boldsymbol{v}^C \cap \boldsymbol{v}'^C$ is $N_j - s - s' + k$ while the size of $\boldsymbol{v} \cap \boldsymbol{v}'^C$ and $\boldsymbol{v}^C \cap \boldsymbol{v}'$ is $s-k$ and $s'-k$, respectively. Putting this together we can write:

$$\begin{aligned}
&\mathcal{V}\left(C_j(\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{x}, \boldsymbol{x}')\right) \\
&= \int \left(H(\boldsymbol{z}_{j-1}, \boldsymbol{z}'_{j-1})\right)^{N_j - s - s' + 2k} \\
&\qquad (0.5 - H(\boldsymbol{z}_{j-1}, \boldsymbol{z}'_{j-1}))^{s+s'-2k} d\mathcal{W}_{1:j-1} \\
&= V_{j, s+s'-2k}(\boldsymbol{x}, \boldsymbol{x}')
\end{aligned}$$

which completes our proof. □

4 Generalization Bounds

Our approach extends the hypothesis class of functions $f(\boldsymbol{x}; \boldsymbol{w})$ to a larger class defined by α, as defined in Equation 4. As in Section 2 we refer to these as $\mathcal{A}$ and $\mathcal{A}^+$ respectively. Using a larger class introduces the typical bias-variance tradeoff. On the one hand, the larger class is more expressive; on the other hand it is more prone to over-fitting. In the theoretical analysis below, we ask a simple question. Given that there exists an ϵ_0 accurate hypothesis in $\mathcal{A}$, how many samples are required to find it when learning in $\mathcal{A}^+$.

In what follows, we use the hinge loss to quantify the error of a classifier:[4]

$$\ell(z, y) = \max(1 - 2(y - 0.5)z, 0). \tag{10}$$

Given a function $\alpha(\boldsymbol{w})$ we consider classifiers as in Equation 4. Namely, we define a function:

$$g(\boldsymbol{x}; \alpha) = \int f(\boldsymbol{x}, \boldsymbol{w})\alpha(\boldsymbol{w}) d\boldsymbol{w}, \tag{11}$$

and the classifier is $y = \Theta\left(g(\boldsymbol{x}; \boldsymbol{w})\right)$. The corresponding empirical and generalization hinge losses are:

$$\begin{aligned}
\hat{\mathcal{L}}(\alpha) &= \frac{1}{M}\sum_{m=1}^{M} \ell(g(\boldsymbol{x}^m; \alpha), y^m) \\
\mathcal{L}(\alpha) &= \mathbb{E}_{(\boldsymbol{x}, y)\sim D}\left[\ell(g(\boldsymbol{x}; \alpha), y)\right],
\end{aligned}$$

[4]Recall we are considering labels in $\{0, 1\}$.

Where D is the true underlying distribution. Finally, define the squared norm of α via:

$$\|\alpha\|^2 = \int \alpha^2(\boldsymbol{w}) d\boldsymbol{w}. \tag{12}$$

For simplicity, we let $d\boldsymbol{w}$ be the uniform distribution over the unit ball in $\mathbb{R}^d$.

4.1 Dimension Based Bound

We begin by recalling a standard sample complexity result for linear classifiers, relating the norm of the weights (in our case the function α) to generalization error. The theorem follows Corollary 4 in [21].

Theorem 4.1. *Let $\{(x_m, y_m)\}_{m=1}^M$ be a sample of size M drawn IID from D. Given $\delta > 0$ and α_0, the following holds with probability at least $1-\delta$ over a sample of size M. Assume that C is chosen such that $C = O(\frac{\sqrt{\log 1/\delta}}{\sqrt{\|\alpha_0\|^2 M}})$. Then the α that minimizes Equation 5 satisfies:*

$$\mathcal{L}(\alpha) \le \mathcal{L}(\alpha_0) + O\left(\sqrt{\frac{\|\alpha_0\|^2 \log 1/\delta}{M}}\right),$$

The above is a standard result for learning in $\mathcal{A}^+$. However, our key interest is in relating $\mathcal{A}$ to $\mathcal{A}^+$. Specifically, we would like to identify cases where learning in $\mathcal{A}^+$ will result in similar generalization to learning in $\mathcal{A}$.

Denote the best hypothesis in $\mathcal{A}$ by $\boldsymbol{w}_0$, and denote its generalization error by ϵ_0. When can learning in $\mathcal{A}^+$ result in error ϵ_0? As stated earlier, the hypothesis $\boldsymbol{w}_0 \in \mathcal{A}$ corresponds to a hypothesis $\alpha \in \mathcal{A}^+$ where α is a delta function centered at $\boldsymbol{w}_0$. However, this α will have large (unbounded) norm $\|\alpha\|^2$, and will thus require an unbounded sample size to discover.

To overcome this difficulty, we add an assumption that $\boldsymbol{w}_0$ is not an isolated good solution, but is rather part of a ball of good solutions. Formally, assume there exists an L such that $\|\boldsymbol{w}_0\| < 1 - 1/L$ and:

$$\mathop{\mathbb{E}}_{\|w - w_0\| < 1/L}\left[\mathcal{L}(f(\boldsymbol{x}, \boldsymbol{w}))\right] < \epsilon_0. \tag{13}$$

In other words, $\boldsymbol{w}_0$ is the center of a ball of radius $1/L$ where the expected loss is at most ϵ_0. Intuitively, this means that the quality of the solution $\boldsymbol{w}_0$ is stable with respect to perturbations of radius $1/L$. As the following lemma states, this assumption implies that there is a bounded norm α with error ϵ_0.

Lemma 4.2. *Denote the overall number of parameters in the network by N. Under the assumptions on $\boldsymbol{w}_0$ above, there exists an α_0 with $\|\alpha_0\|^2 = L^N$ such that*

$$\mathcal{L}(\alpha_0) < \epsilon_0 \tag{14}$$

Proof. Consider the following function:

$$\alpha_0(\boldsymbol{w}) = \begin{cases} L^N & \|\boldsymbol{w} - \boldsymbol{w}_0\| < 1/L \\ 0 & \text{else} \end{cases}. \tag{15}$$

Note that $\int \alpha_0(\boldsymbol{w}) d\boldsymbol{w} = 1$ and that:

$$\|\alpha_0\|^2 = \int \alpha_0^2(\boldsymbol{w}) dw = \int_{\|\boldsymbol{w}-\boldsymbol{w}_0\|<1/L} L^{2N} d\boldsymbol{w} = L^N. \tag{16}$$

Next, we relate the performance of α_0 to the performance of $\boldsymbol{w}$ in the vicinity of $\boldsymbol{w}_0$:

$$\begin{aligned} \mathcal{L}(\alpha_0) &= \mathbb{E}\left[\ell\left(\int f(\boldsymbol{x}, \boldsymbol{w})\alpha_0(\boldsymbol{w}), y\right)\right] \\ &\leq \mathbb{E}\left[\int \ell\left(f(\boldsymbol{x}, \boldsymbol{w}), y\right)\alpha_0(\boldsymbol{w}) d\boldsymbol{w}\right] \\ &= \mathop{\mathbb{E}}_{\|\boldsymbol{w}-\boldsymbol{w}_0\|<1/L}[\mathcal{L}(f(\boldsymbol{x}; \boldsymbol{w}))] < \epsilon_0, \end{aligned}$$

where the first inequality follows from Jensen and the fact that $\int \alpha_0(\boldsymbol{w}) d\boldsymbol{w} = 1$, and the last inequality follows from the assumption Equation 13. Thus we see that the performance of α_0 is the expected error of the solutions in a neighborhood of $\boldsymbol{w}_0$. □

The above Theorem 4.1 and Lemma 4.2 imply a sample complexity result linking $\mathcal{A}$ and $\mathcal{A}^+$.

Corrolary 4.3. *Given $\delta > 0, \epsilon > 0$ and number of samples $M = O\left(\frac{L^N \log 1/\delta}{\epsilon^2}\right)$, the α that minimizes Equation 5 attains a generalization error of at most $\epsilon_0 + \epsilon$ with probability at least $1 - \delta$..*

The corollary has the following intuitive interpretation: the larger the volume of good solutions in $\mathcal{A}$ is, the better the sample complexity of learning in $\mathcal{A}^+$. The complexity is exponential in N, but improves as L approaches 1 (i.e., as $\boldsymbol{w}_0$ becomes more stable). It should however be noted that the learning algorithm itself is polynomial in the number of samples, rendering the method practical for a given training set.

4.2 Margin Based Bound

The previous section used the number of parameters N in the sample complexity bound. A common alternative notion of complexity in learning theory is that of a margin. Here we consider a margin based result for the case of a one hidden layer network with k hidden units. As before, we denote the weight vectors of the first layer by $\boldsymbol{w}_0^i$ for $i = 1, \ldots, k$ and an output vector $\boldsymbol{w}_1$.

This network can for example implement an intersection of halfspaces (if $\boldsymbol{w}_1$ is set accordingly). Previous works have studied learning in this setting under margin assumption [1, 10]. We make a similar assumption and study its consequences. Specifically, we assume existence of a solution $\boldsymbol{w}_0, \boldsymbol{w}_1$ such that $\frac{|(\boldsymbol{w}_0^i)^\top \boldsymbol{x}|}{\|\boldsymbol{x}\|} > \gamma$ for each hidden neuron i. We further add an assumption of robustness for the last output layer. Namely, that for every $\boldsymbol{w} \in \mathbb{R}^k$ such that $\|\boldsymbol{w} - \boldsymbol{w}_1\| < \gamma$ we have that

$$\mathbb{E}\left[\ell(\boldsymbol{w} \cdot \Theta\left(W_0 \boldsymbol{x}\right), y)\right] < \epsilon_0. \tag{17}$$

Together, the above assumptions state that each hyperplane in the first layer has a margin of γ and that there is a ball of *good* output vectors $\boldsymbol{w}_1$. It easily follows that the assumption in the previous section is satisfied with $L = \gamma^{-1}$. This in turn implies a sample complexity of $O(1/\gamma)^{dk+k}$.

The result can be improved further via a random projection argument as in [1, 2]. Consider for example, Theorem 5 in [3]. It states that if a linear classifier separates with margin γ then we can project to dimension $O(\frac{1}{\gamma^2} \log \frac{1}{\epsilon_1 \delta})$ such that the resulting feature space can be separated with error ϵ_1 at margin $\frac{\gamma}{4}$. We need the result to hold for all the k classifiers in the first layer. Applying a union bound yields a projection dimension of $O(\frac{1}{\gamma^2} \log \frac{k}{\epsilon_1 \delta})$.

The above implies that we can use the projected dimension in place of the input dimension d. Since that margin γ is preserved, the assumption of the previous section is still preserved with $L = O(\frac{1}{\gamma})$. Putting all these components together we arrive at a sample complexity of

$$O(\frac{1}{\epsilon^2} \gamma^{-\frac{k}{\gamma^2} \log \frac{k}{\delta}} \log \frac{1}{\delta}).$$

The dependence on the input dimension has been replaced by a dependence the margin γ. The number of hidden units k remains, since we have not reduced the dimensionality of the hidden layer.

Our sample complexity result is similar to existing results on intersection of hyperplanes [1, 10] in the sense of exponential dependence on the margin and the number of hyperplanes (i.e., k in our case), although the bound in [10] has a better dependence.

5 Experimental Evaluation

In this section we evaluate our method on both synthetic and object recognition benchmarks. To compare our approach to baseline kernels, we use an identical learning setup for all methods and only vary the kernel function. Concretely, we compare the following: an RBF kernel, our improper deep learning kernel (IDK), and the kernel of [4]. For the latter, we consider two variants: one with a threshold activation function ($CS0$) and one with a rectified linear unit ($CS1$).

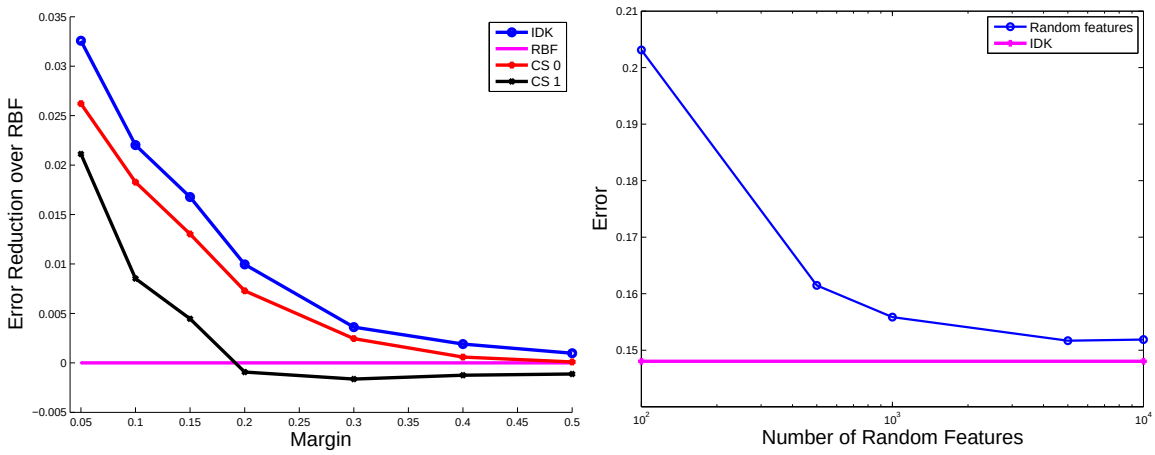

Figure 1: (left) Comparison of test prediction accuracy as a function of the margin for our kernel (IDK) and those of Choi and Saul ($CS0$ and $CS1$) relative to the performance of the RBF on synthetic data generated from a network with two hidden layers. The y axis shows accuracy advantage over RBF so that larger numbers correspond to larger reduction in error. Results are averaged over 700 repetitions. (right) Comparison of test prediction accuracy when using our IDK kernel to a numerical estimation of the kernel integral using random features, as a function of the number of features used for estimation.

5.1 Synthetic Experiments

We start by considering a synthetic setting. Training data is generated from a network with two hidden layers, and a threshold activation function $\Theta(\cdot)$, as in our kernel derivation. The input to the network is two dimensional and the number of hidden neurons is 40 and 20 for the first and second layer (performance was not sensitive to these settings). The weight of each unit is sampled uniformly in the range $[-1, 1]$ and normalized to 1. Inputs were uniformly sampled form the two dimensional unit square. Input samples were also required to have a balanced label distribution, so that cases where one of the label probabilities was below 0.4 were discarded.

Finally, our theoretical analysis predicts that data with a large margin should be easier to learn. We thus vary the margin of the training data by removing training points that are γ close to the decision boundary.

Comparison to Other Kernels: To fairly compare the accuracy of the different kernels, we tune the hyperparameters of all kernels on a holdout set. For RBF, the kernel width is chosen from $[0.001, 0.01, 0.1, 1, 10, 100]$. For IDK, we consider network structures $[40], [40, 20], [4, 4, 4, 4]$. Similarly, for $CS0, CS1$, we choose between $1-5$ hidden layers.

Figure 1(left) shows the performance of the classifiers as a function of the margin parameter. It can be seen that our IDK kernel outperforms the other methods across all margin values. It can also be seen that as the margin grows, all methods improve, as expected.

Comparison to Random Features: Recall that our kernel is based on a closed form solution of the integral Equation 6. An alternative to evaluating this integral is to sample $\boldsymbol{w}$ vectors randomly, and numerically evaluate the integral via an empirical average. This approach is similar to the kitchen sinks of [15], and has the advantage of being solved via a linear SVM (where the dimension is the number of sampled features). Here we test this approach for different numbers of random features. For this comparison, both our closed form IDK and the random features use the correct model structure. Results are shown in Figure 1(right). It can be seen that the random features approach improves as more features are added (note the logarithmic scale of the x-axis) but there is still a gap between it and the closed form IDK kernel.

5.2 Object Recognition Benchmarks

One of the great success stories of deep learning is the task of object recognition [11]. Namely, labeling an image with a set of categories (e.g., building, frog, paper clip). Here we evaluate IDK on two such standard benchmarks. We use the CIFAR-10 and STL-10 datasets, with the same preprocessing as in [7].

For the IDK hyperparameters we test the structures $[4], [4, 4, 4, 4], [16], [32], [32, 16]$ and $[32, 16, 4]$. For both $CS0$ and $CS1$, we test up to eight hidden layers. For RBF we test widths of $[0.01, 0.1, 1, 10, 100]$.

Results are reported in Table 1 where we also two additional literature baselines, namely Sum Product Networks (SPN) [7] and Convolutional Kernels Networks (CKN)[14]. On CIFAR-10 the $CS1$ outperforms IDK

	IDK	RBF	CS0	CS1	SPN	CKN
CIFAR-10	81.8	81.8	81.63	82.49	83.96	82.18
STL-10	62.6	61.7	62.3	52	62.3	62.32

Table 1: Classification accuracy (in %) for the CIFAR-10 and STL-10 benchmarks. Compared are our IDK kernel, as well as the $CS0$,$CS1$ and RBF kernels, Sum Product Networks (SPN) [7], and Convolutional Kernels Networks (CKN)[14].

by 0.7%, and SPN outperforms all methods. For STL-10 $CS1$ performs quite badly, and the IDK method outperforms the other methods, although by a small margin.

6 Discussion

We presented a method for learning a class that extends deep neural networks. Learning in the extended class is equivalent to solving an SVM with the kernel derived in Theorem 3.1. The neural nets we consider use a threshold activation function, and a fully connected architecture with different parameters for each weight. In this case the outputs of hidden layers are binary, a fact which lets us enumerate over the possible outputs and use symmetries in the integral. Furthermore, the fact that each weight has its own parameter further decouples the integral, and facilitates our recursive close form kernel.

Modern deep learning architectures are different from our architecture in several respects. First, they typically use a rectified linear unit (ReLU) for activation (e.g., see [12]), which yields better models.[5] It is not clear whether our integral can be solved in closed form for ReLUs, as we can no longer use the discrete nature of the outputs. A second difference is the use of convolutional networks, which essentially tie different weights in the network. Such tying does complicate our recursive derivation, and it is not clear whether it will allow a closed form solution. Finally, a commonly used component is max-pooling, which again changes the structure of the integral. An exciting avenue for future research is to study the kernel resulting from these three components, and seeing whether it can be evaluated in closed-form or approximated.

As mentioned in Section 5.1, it is natural to try and evaluate the kernel numerically by sampling a finite set of parameters $\boldsymbol{w}$, and approximating the integral in Equation 6 as a finite average over these. As our experiments show, this does not perform as well as using our closed form expression for the integral, even with a large number of random features. However, for cases where the integral cannot be found in closed form, there may be intermediate versions that combine partial closed form and sampling. This may have interesting algorithmic implications, since random features have recently been shown to result in fast kernel based learning algorithms [5].

[5]Note that it is not clear whether this is due to improved optimization or better modeling.

Recent work [13] has shown that replacing the activation function with a quadratic unit results in improper learning that is poly time both algorithmically and in sample complexity. It would be interesting to study such activation functions with our kernel approach.

Another interesting recent work employing kernels is [14]. However, there the focus is on explicitly constructing a kernel that has certain invariances. Our empirical results show comparable results to [14].

The algorithm we present is polynomial in the number of samples, and globally optimal due to convexity. Our analysis in Section 4 shows that the cost of convexity is an increase in sample complexity. Namely, to guarantee finding a model that generalizes as well as the original neural architecture, we need $O(L^N)$ samples. This is perhaps not unexpected given the recently proved hardness of improper learning for related hypothesis classes such as intersection of hyperplanes [6]. As we also show in 4, the input dimension d can be replaced with the inverse margin $\frac{1}{\gamma^2}$. Again, exponential dependence on margin for such problems is manifested in related works [1, 10, 19, 13].

The key open problem in this context, and indeed for the deep learning field, is to understand what alternative distributional assumptions may lead to both algorithmic tractability and polynomial sample complexity. Our kernel approach attains tractability at the cost of increased sample complexity. It will be very interesting to study which assumptions will improve its sample complexity.

Acknowledgments: This work was supported by the ISF Centers of Excellence grant 1789/11, by the Intel Collaborative Research Institute for Computational Intelligence (ICRI- CI), and by a Google Research Award. Roi Livni is a recipient of the Google Europe Fellowship in Learning Theory, and this research is supported in part by this Fellowship.

References

[1] Rosa Arriaga, Santosh Vempala, et al. An algorithmic theory of learning: Robust concepts and random projection. In *Foundations of Computer Science*, pages 616–623. IEEE, 1999.

[2] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. Kernels as features: On kernels, margins, and low-dimensional mappings. *Machine Learning*, 65(1):79–94, 2006.

[3] Avrim Blum. Random projection, margins, kernels, and feature-selection. In *Subspace, Latent Structure and Feature Selection*, pages 52–68. Springer, 2006.

[4] Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems 22*, pages 342–350. 2009.

[5] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina F Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 3041–3049, 2014.

[6] Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. From average case complexity to improper learning complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 441–448, New York, NY, USA, 2014. ACM.

[7] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 3248–3256, 2012.

[8] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[9] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Tara Sainath. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[10] Adam R Klivans and Rocco A Servedio. Learning intersections of halfspaces with a margin. *Journal of Computer and System Sciences*, 74(1):35–48, 2008.

[11] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[13] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pages 855–863, 2014.

[14] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2627–2635. 2014.

[15] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, pages 1313–1320, 2009.

[16] Nicolas L Roux and Yoshua Bengio. Continuous neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 404–411, 2007.

[17] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[18] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[19] Shai Shalev-Shwartz, Ohad Shamir, and Karthik Sridharan. Learning kernel-based halfspaces with the 0-1 loss. *SIAM Journal on Computing*, 40(6):1623–1646, 2011.

[20] Shai Shalev-Shwartz, Ohad Shamir, and Eran Tromer. Using more data to speed-up training time. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, pages 1019–1027, 2012.

[21] Karthik Sridharan, Shai Shalev-Shwartz, and Nathan Srebro. Fast rates for regularized objectives. In *Advances in Neural Information Processing Systems*, pages 1545–1552, 2009.