On the Power of Differentiable Learning versus PAC and SQ Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We study the power of learning via mini-batch stochastic gradient descent (SGD) on the loss of a differentiable model or neural network, and ask what learning problems can be learnt using this paradigm. We show that SGD can always simulate learning with statistical queries (SQ), but its ability to go beyond that depends on the precision ρ of the gradient calculations and the minibatch size b. With fine enough precision relative to minibatch size, namely when $b\rho$ is small enough, SGD can go beyond SQ learning and simulate any sample-based learning algorithm and thus its learning power is equivalent to that of PAC learning; this extends prior work that achieved this result for $b = 1$. Moreover, with polynomially many bits of precision (i.e. when ρ is exponentially small), SGD can simulate PAC learning regardless of the batch size. On the other hand, when $b\rho^2$ is large enough, the power of SGD is equivalent to that of SQ learning.

1 Introduction

A leading paradigm that has become the predominant approach to learning is that of *differentiable learning*, namely using a parametric function class $f_{\boldsymbol{w}}(x)$, and learning by performing mini-**b**atch **S**tochastic **G**radient **D**escent (bSGD) updates on its loss. Feed-forward neural networks are a particularly popular choice for the parametric model $f_{\boldsymbol{w}}$. One approach to understanding differentiable learning is to think of it as a method for minimizing the empirical error of $f_{\boldsymbol{w}}$ with respect to $\boldsymbol{w}$, i.e. as an empirical risk minimization (ERM). ERM is indeed well understood, and is in a sense a universal learning rule, in that any hypothesis class that is PAC learnable is also learnable using ERM. Furthermore, since poly-sized feed-forward neural networks can represent any poly-time computable function, we get that ERM on neural networks can efficiently learn any tractable problem. But this view of differentiable learning ignores two things.

First, many ERM problems, including ERM on any non-trivial neural network, are highly non-convex and bSGD might not find the global minimizer. As we just discussed, pretending that it does would mean we can learn all poly-time functions, which is known to be impossible[1] [e.g. 11, 12]). In fact, even minimizing the empirical risk on a neural net with two hidden units, and even if we assume data is exactly labeled by such a network, is already NP-hard, and cannot be done efficiently with bSGD [5]. We already see that bSGD is not the same as ERM, as asking "what can be learned by bSGD" is quite different from asking "what can be learned by ERM".

Furthermore, bSGD might also be more powerful than ERM. Consider using a highly over-parametrized function class $f_{\boldsymbol{w}}$, with many more parameters than the number of training examples, as is often the case in modern deep learning. In such a situation there would typically be many

[1] Subject to mild cryptographic assumptions, such as the existence of one-way functions, i.e. the existence of cryptography itself.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

empirical risk minimizers (zero training error solutions), and most of them would generalize horribly, and so the ERM principal on its own is not sufficient for learning [17]. Yet we now understand how gradient descent and bSGD incorporate also intricate implicit bias that lead to a particular empirical risk minimizers which might well ensure learning [19, 16].

All this justifies studying differentiable learning as a different and distinct paradigm from empirical risk minimization. The question we ask is therefore: **what can be learned by performing mini-batch stochastic gradient descent on some parametric function class $f_{\boldsymbol{w}}$, and in particular on a feed-forward neural network**. Answering this question is important not only for understanding the limits of what we could possibly expect from differentiable learning, but even more so in guiding us as to how we should study differentiable learning, and ask questions, e.g., about its power relative to kernel methods [e.g. 22, 2, 3, 13, 7–9, 15], or the theoretical benefits of different architectural innovations [e.g. 14].

A significant, and perhaps surprising, advance toward answering this question was recently presented by Abbe and Sandon [1], who showed that bSGD with a single example per iteration (i.e. a minibatch of size 1) can simulate any poly-time learning algorithm, and hence is as powerful as PAC learning. On the other hand, training using population gradients (infinite mini-batch size) and low-precision (polynomial accuracy, i.e. a logarithmic number of bits of precision), is only as powerful as learning with Statistical Queries (SQ), which is known to be strictly less powerful than PAC learning [10, 6]; see [18] for a comprehensive survey about the SQ paradigm.

Our Contribution: In this paper, we take a more refined view at this dichotomy, and consider learning with bSGD also with larger mini-batch sizes $b > 1$, as is more typically done in practice. We ask whether the ability to simulate PAC learning is indeed preserved also with larger mini-batch sizes, or is a particular quirk of using single examples, or perhaps very small mini-batches. We discover that this depends crucially on the relationship of the mini-batch size b and the precision ρ of the gradient calculations. We first show that regardless of the mini-batch size, bSGD is always able to simulate any SQ method, and so bSGD (with polynomial precision, i.e. a logarithmic number of bits of precision) is at least as powerful as SQ learning. When the mini-batch size b is large relative to the precision, namely $b = \omega(\log(n)/\rho^2)$, where n is the input dimension, bSGD is not any more powerful than SQ. But when $b < 1/(8\rho)$, bSGD can again simulate any sample-based learning algorithm, and is as powerful as PAC learning (see formal results in Section 3).

We see, then, that not only the mini-batch size, but also the precision of the gradient calculations is crucial. With good precision, mini-batch SGD can simulate PAC even with very large mini-batch sizes, and with a linear number of bits of precision (i.e. $\rho = 2^{-n}$), bSGD can simulate PAC with any polynomial mini-batch size. With large mini-batches and low precision, i.e. when ρ is relatively large, which is perhaps a more realistic view of differentiable learning, bSGD is strictly less powerful, but still as powerful as SQ. Either way, except for the intermediate regime between $1/\rho$ and $\log(n)/\rho^2$, we can precisely capture the power of bSGD.

Another difference versus the work of Abbe and Sandon is that we discuss both computationally tractable and intractable learning. We show that poly-time PAC and SQ are equivalent, in the sense described above, to bSGD on a poly-sized neural network, whereas computationally unbounded PAC and SQ (i.e. limited only by the number of samples or number of statistical queries) is equivalent to bSGD on an arbitrary model $f_{\boldsymbol{w}}$. In fact, to simulate PAC and SQ, we first construct an arbitrary $f_{\boldsymbol{w}}$, and then note that if the PAC or SQ method is poly-time computable, then the computations within it can be expressed as poly-size circuits, which can in turn be simulated as poly-size sub-networks, allowing us to implement $f_{\boldsymbol{w}}$ as a neural net.

Our analysis relies on introduction of a variant of SQ learning which we refer to as mini-**b**atch **S**tatistical **Q**ueries (bSQ). In this variant, which is related to *Honest SQ* [20, 21], Statistical Queries are answered using a mini-batch of samples drawn from the source distribution, up to some precision. We first show that bSQ methods can always be simulated by bSGD, by constructing a differentiable model where at each step the derivatives with respect to some of the parameters contain the answers to the statistical queries. We then relate bSQ to SQ and PAC, based on the relationship between the mini-batch size and precision. In order to simulate PAC using bSQ, we develop a novel "sample extraction" that uses mini-batch statistical queries (on independently sampled mini-batches) to extract a single sample drawn from the source distribution. This procedure might be of independent interest, perhaps also in studying privacy, where such an extraction is not desirable. Our study of the relationship of bSQ to SQ and PAC, summarized in Section 4, also sheds light on how well the SQ

framework captures learning by answering queries using limited data, which is arguably one of the main motivations for the inverse-polynomial tolerance parameter in the SQ framework.

2 Learning Paradigms

We consider learning a predictor $f : \mathcal{X} \to \mathbb{R}$ over an *input space* $\mathcal{X}$, so as to minimize its *population loss* $\mathcal{L}_{\mathcal{D}}(f) := \mathbb{E}_{(x,y)\sim\mathcal{D}}\, \ell(f(x), y)$ with respect to a *source distribution* $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, where $\ell : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ is a loss function. Unless noted otherwise, we take the loss function to be the square-loss $\ell_2(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$. For concreteness, we always take $\mathcal{Y} = \{0, 1\}$ and $\mathcal{X} = \{0, 1\}^n$.

Differentiable Learning. We study learning by (approximate) gradient descent on a differentiable model. Formally, a *differentiable model* of size p is a mapping $f : \mathbb{R}^p \times \mathcal{X} \to \mathbb{R}$, denoted $f_{\boldsymbol{w}}(x)$, where $\boldsymbol{w} \in \mathbb{R}^p$ are the parameters, or "weights", of the model, and for every $x \in \mathcal{X}$ there exists a gradient $\nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(x)$ for almost every $\boldsymbol{w} \in \mathbb{R}^p$ (i.e, outside of a measure-zero set of weights). If $f_{\boldsymbol{w}}(x)$ is not differentiable at $\boldsymbol{w}$, we let $\nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(x)$ be an arbitrary vector in $[-1, 1]^p$.

A notable special case of differentiable models is that of neural networks, defined by a directed acyclic graph with a single output node (i.e. with zero out-degree) and $n + 1$ input nodes (i.e. with zero in-degree) corresponding to the n bits of $\mathcal{X}$ and the constant 1. Every edge e corresponds to a weight parameter w_e. Computation proceeds recursively with each vertex v returning the value o_v obtained by applying an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ on the linear combination of the values computed at its predecessors as specified by the edge weights $\boldsymbol{w} = (w_e)_e$, that is, $o_v := \sigma\left(\sum_{e=(u\to v)} w_e o_u\right)$. Unless stated otherwise, we take the activation function to be the ReLU activation, namely $\sigma(x) = \max(x, 0)$. The final output of the model on input x is the value returned by the output node. In general, we allow weights to take constant values, i.e., have untrainable fixed values. This simplifies the presentation of our results. However, we note that we also prove our main results for the case where all the weights of the network are allowed to train. We denote the total number of trainable weights by p. The size of the network is the total number of edges in the graph, denoted by s (i.e., if all weights in the network are trainable, then $p = s$).

For a differentiable model $f_{\boldsymbol{w}}(x)$, a distribution $\mathcal{W}$ over $\mathbb{R}^p$, mini-batch size b, gradient precision ρ, stepsize[2] γ, the mini-**b**atch **S**tochastic **G**radient **D**escent (bSGD) algorithm operates by computing iterates of the form:

$$\begin{aligned} \boldsymbol{w}^{(0)} &\sim \mathcal{W} \\ \boldsymbol{w}^{(t+1)} &\leftarrow \boldsymbol{w}^{(t)} - \gamma g_t \end{aligned} \tag{1}$$

where g_t is a *ρ-approximate rounding* of the empirical *clipped* gradient $\overline{\nabla}\mathcal{L}_{S_t}(f_{\boldsymbol{w}^{(t)}}) := \mathbb{E}_{(x,y)\sim S_t}[\nabla\ell(f_{\boldsymbol{w}^{(t)}}(x), y)]_1$ for $S_t \sim \mathcal{D}^b$ (independently sampled for each iteration), where $[\alpha]_1$ denotes entry-wise clipping of α to $[-1, 1]$. We say that $u \in [-1, 1]^p$ is a ρ-approximate rounding of $v \in [-1, 1]^p$ if $u \in \rho\cdot\mathbb{Z}^p$ (namely, each entry of u is an integral multiple of ρ), and $\|u - v\|_\infty \leq 3\rho/4$. Thus, overall g_t is a valid gradient estimate if it satisfies

$$g_t \in \rho \cdot \mathbb{Z}^p \quad \text{and} \quad \|g_t - \overline{\nabla}\mathcal{L}_{S_t}(f_{\boldsymbol{w}^{(t)}})\|_\infty \leq 3\rho/4 \tag{2}$$

We always assume that $\rho = 2^{-d}$, where $d \in \mathbb{N}$ is the number of bits of precision.

Remark 1. We consider *clipped* gradients because the precision ρ makes sense only when considered relative to the scale of the gradients. We note however that all the bSGD algorithms we develop in this paper are such that the norm of the gradients of the loss $\|\nabla\ell(f_{\boldsymbol{w}^{(t)}}(x), y)\|_\infty$ for all (x, y) and t would always be at most 1 and hence clipping will not play a role.

Remark 2. In our rounding error model, for any integer q, values in $(q\rho - \frac{\rho}{4}, q\rho + \frac{\rho}{4})$ get rounded to $q\rho$, whereas, values in $[q\rho + \rho/4, q\rho + 3\rho/4]$ can get rounded either to $q\rho$ or $(q + 1)\rho$. We also assume that all the values are clipped, so that each value can be represented by d bits (and a sign bit). This error model is a good proxy for rounding errors in real finite precision arithmetic, where we have an uncertainty about the $(d + 1)$-th bit when representing real numbers in $[-1, 1]$ with d bits of precision.

[2] The stepsize doesn't play an important role in our analysis. We can also allow variable or adaptive stepsize sequences—for simplicity of presentation we stick with a fixed stepsize.

A learning algorithm $\mathcal{A}$ is said to be a $\mathsf{bSGD}(T, \rho, b, p, r)$ algorithm if it operates by computing bSGD iterates for some differentiable model $f : \mathbb{R}^p \times \mathcal{X} \to \mathbb{R}$ (i.e. with p parameters), starting from some (randomized) initialization $\mathcal{W}$ which uses r random bits,[3] with some stepsize γ, given ρ-approximate gradients over batches of size b; the output of $\mathcal{A}$ is the final iterate $f_{\boldsymbol{w}^{(T)}} : \mathcal{X} \to \mathbb{R}$. We say that $\mathcal{A}$ ensures error ε on a source distribution $\mathcal{D}$ if $\mathbb{E}[\sup \mathcal{L}_D(f_{\boldsymbol{w}^{(T)}})] \leq \varepsilon$, where the expectation is over both the initialization $\boldsymbol{w}^{(0)}$ and the mini-batches $S_t \sim \mathcal{D}^b$, and the $\sup$ is over all gradient estimates g_t satisfying (2).[4] We denote by $\mathsf{bSGD}_{\mathsf{NN}}(T, \rho, b, p, r, s)$ the family of bSGD algorithms where the differentiable model is implemented by a neural network of size s, and the initialization distribution $\mathcal{W}$ can be sampled from in $\mathrm{poly}(n)$ time.

PAC and SQ Learning. A learning algorithm $\mathcal{A}$ is said to be a $\mathsf{PAC}(m, r)$ algorithm[5] if it takes in a set of samples $S \subseteq \mathcal{X} \times \mathcal{Y}$ of size m, uses r bits of randomness and returns a predictor $f : \mathcal{X} \to \mathbb{R}$. We say that $\mathcal{A}$ ensures error ε on a source distribution $\mathcal{D}$ if $\mathbb{E}[\mathcal{L}_{\mathcal{D}}(f)] \leq \varepsilon$, where the expectation is over $S \sim \mathcal{D}^m$ and the randomness in $\mathcal{A}$. Similarly, we say that $\mathcal{A}$ is a $\mathsf{PAC}_{\mathsf{TM}}(m, r, \mathrm{TIME})$ algorithm if it is a $\mathsf{PAC}(m, r)$ algorithm which can be implemented using a Turing machine that runs in time TIME.

A learning algorithm $\mathcal{A}$ is said to be a *statistical-query* $\mathsf{SQ}(k, \tau, r)$ algorithm, if it operates in k rounds where in round t the algorithm produces a *statistical-query* $\Phi_t : \mathcal{X} \times \mathcal{Y} \to [-1, 1]$ for which it receives a response v_t, and finally outputs a predictor $f : \mathcal{X} \to \mathbb{R}$. Formally, each Φ_t is a function of a random string $R \in \{0, 1\}^r$ and past responses $v_1, \ldots, v_{t-1}$. We say that $\mathcal{A}$ ensures error ε on a source distribution $\mathcal{D}$ if $\mathbb{E}_R[\sup \mathcal{L}_{\mathcal{D}}(f)] \leq \varepsilon$, where the $\sup$ is over all "valid" $v_t \in [-1, 1]$, namely $|v_t - \mathbb{E}_{\mathcal{D}} \Phi_t(x, y)| \leq \tau$. Similarly, we say that $\mathcal{A}$ is a $\mathsf{SQ}_{\mathsf{TM}}(k, \tau, r, \mathrm{TIME})$ if $\mathcal{A}$ is a $\mathsf{SQ}(k, \tau, r)$ that can be implemented using a Turing machine that runs in time TIME, using queries that can be computed in time TIME.

For any algorithm $\mathcal{A}$ and for any source distribution $\mathcal{D}$, let $\mathrm{err}(\mathcal{A}, \mathcal{D})$ be the infimum over ε such that $\mathcal{A}$ ensures error ε on $\mathcal{D}$.

3 Main Results : bSGD versus PAC and SQ

Our main results show the power of bSGD learning algorithms relative to that of PAC and SQ algorithms. The exact relation depends on the mini-batch size b and gradient precision ρ. Namely, for some fixed precision ρ, we show that for large enough mini-batch size b, bSGD is equivalent to SQ, i.e. anything that can be learned using SQ algorithms can be learned using bSGD, and vice-versa. On the other hand, if b is not too large, bSGD is equivalent to PAC. To present these results, it is useful to define the following notion of when algorithms from class $\mathcal{C}'$ can simulate all algorithms from class $\mathcal{C}$:

Definition 1. *For two classes of algorithms $\mathcal{C}, \mathcal{C}'$ we write $\mathcal{C}' \preceq \mathcal{C} + \delta$ if for every algorithm $\mathcal{A} \in \mathcal{C}$ there exists an algorithm $\mathcal{A}' \in \mathcal{C}'$ such that for every source distribution $\mathcal{D}$ we have $\mathrm{err}(\mathcal{A}', \mathcal{D}) \leq \mathrm{err}(\mathcal{A}, \mathcal{D}) + \delta$.*

That is, $\mathcal{C}' \preceq \mathcal{C} + \delta$ means that $\mathcal{C}'$ *is at least as powerful as* $\mathcal{C}$. Observe that for all classes of algorithms $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$, if $\mathcal{C}_1 \preceq \mathcal{C}_2 + \delta_1$ and $\mathcal{C}_2 \preceq \mathcal{C}_3 + \delta_2$ then $\mathcal{C}_1 \preceq \mathcal{C}_3 + \delta_1 + \delta_2$.

We now state our main results, establishing the relationship between bSGD and PAC/SQ.

Theorem 1 (PAC equivalence)**.** *For all $\delta > 0$, it holds that*

1. ***For all b, ρ if $b < 1/(8\rho)$**, then for all m, r, δ, it holds for $T' = O(mn/\delta)$ and $p' = r + O(T'(n + \log b))$ and $r' = r + O(T' \log b)$ that*

$$\mathsf{bSGD}(T', \boldsymbol{\rho}, \boldsymbol{b}, p, r') \preceq \mathsf{PAC}(m, r) + \delta\,.$$

[3] Namely, $\mathcal{W} : \{0, 1\}^r \to \mathbb{R}^p$ such that $\boldsymbol{w}^{(0)} \sim \mathcal{W}(s)$ for $s \sim$ uniform over $\{0, 1\}^r$.

[4] Defining the error as $\mathbb{E}[\sup \mathcal{L}_D(f_{\boldsymbol{w}^{(T)}})]$ can be interpreted as allowing the rounding errors on g_t to also depend on random samples in future steps. A more conservative definition would involve $\mathbb{E}_{\boldsymbol{w}^{(0)}} \mathbb{E}_{S_1} \sup_{g_1} \ldots \mathbb{E}_{S_T} \sup_{g_T} \mathcal{L}_{\mathcal{D}}(f_{\boldsymbol{w}^{(T)}})$. However, this distinction does not change any of our results and we stick to the simpler $\mathbb{E}\sup$ definition for convenience.

[5] The usage of the acronym PAC is technically improper as it is used to refer to an *algorithm* and not a class of learning problems. We use it for notational convenience and historical reasons.

Furthermore, for every runtime TIME*, it holds for* $s' = \mathrm{poly}(n, m, r, \mathrm{TIME}, \rho^{-1}, b, \delta^{-1})$ *that*

$$\mathsf{bSGD}_{\mathsf{NN}}(T', \boldsymbol{\rho}, \boldsymbol{b}, p', r', s') \preceq \mathsf{PAC}_{\mathsf{TM}}(m, r, \mathrm{TIME}) + \delta$$

2. ***For all b,*** $\boldsymbol{\rho}$ *and* T, p, r*, it holds that*

$$\mathsf{PAC}(m' = Tb, r' = r) \preceq \mathsf{bSGD}(T, \boldsymbol{\rho}, \boldsymbol{b}, p, r)\,.$$

Furthermore, for every s*, it holds that*

$$\mathsf{PAC}_{\mathsf{TM}}(m' = T\boldsymbol{b}, r', \mathrm{TIME}' = \mathrm{poly}(T, b, s, r, n)) \preceq \mathsf{bSGD}_{\mathsf{NN}}(T, \boldsymbol{\rho}, \boldsymbol{b}, p, r, s)\,.$$

Theorem 2 (SQ equivalence)**.** *There exists a constant* C *such that for all* $\delta > 0$*, it holds that*

1. ***For all*** T*,* $\boldsymbol{\rho}$*,* p*,* r*,* $\boldsymbol{b}$ ***s.t.*** $\boldsymbol{b} > \frac{\mathbf{16}C\log(Tp/\delta)}{\boldsymbol{\rho}^2}$*, it holds for* $k' = Tp$ *and* $\tau' = \frac{\rho}{4} - \sqrt{\frac{C\log(Tp/\delta)}{b}}$ *that*

$$\mathsf{SQ}(k', \tau', r' = r) \preceq \mathsf{bSGD}(T, \boldsymbol{\rho}, \boldsymbol{b}, p, r) + \delta\,.$$

Furthermore, for all s*, it holds for* $\mathrm{TIME}' = \mathrm{poly}(T, \rho^{-1}, b, s, r, \delta^{-1})$ *that*

$$\mathsf{SQ}_{\mathsf{TM}}(k', \tau', r' = r, \mathrm{TIME}') \preceq \mathsf{bSGD}_{\mathsf{NN}}(T, \boldsymbol{\rho}, \boldsymbol{b}, p, r, s) + \delta.$$

2. ***For all b*** *and all* k, τ, r*, it holds for* $\boldsymbol{\rho} = \boldsymbol{\tau}/\mathbf{16}$*,* $T' = k \cdot \max\left\{\frac{C\log(k/\delta)}{b\tau^2}, 1\right\}$ *and* $p' = r + 2T$ *that*

$$\mathsf{bSGD}(T', \boldsymbol{\rho}, \boldsymbol{b}, p', r' = r) \preceq \mathsf{SQ}(k, \tau, r) + \delta\,.$$

Furthermore, it holds for $s' = \mathrm{poly}(k, 1/\tau, r, \mathrm{TIME}, 1/\delta)$ *that*

$$\mathsf{bSGD}_{\mathsf{NN}}(T', \boldsymbol{\rho}, \boldsymbol{b}, p', r' = r, s') \preceq \mathsf{SQ}_{\mathsf{TM}}(k, \tau, r, \mathrm{TIME}) + \delta\,.$$

In the above Theorems, the reduction holds with parameters on the left-hand-side (the parameters of the model being reduced to) that are polynomially related to the dimension and the parameters on the right-hand-side. But the mini-batch size b and gradient precision ρ play an important role. In Theorem 1 Part 1, we may choose b and ρ as we wish, as long as they satisfy $b\rho < 1/8$—they do no need to be chosen based on the parameters of the PAC algorithm, and we can always simulate PAC with any $b\rho < 1/8$. Similarly, in Theorem 2 Part 2, we may chose $b \geq 1$ arbitrarily, and can always simulate SQ, although ρ does have to be chosen according to τ. Finally, the reduction from bSGD to SQ in part 1 of Theorem 2 is only valid when b is sufficiently large.

Focusing on the mini-batch size b, and allowing all other parameters to be chosen to be polynomially related, Theorems 1 and 2 can be informally summarized as:

$$\mathsf{PAC} \preceq \mathsf{bSGD}(b) \preceq \mathsf{SQ} \tag{3}$$

where the left relationship is tight if $b < 1/(8\rho)$ and the right relationship is tight if $b \gg \omega((\log n)/\rho^2)$. Furthermore, even if do not allow ρ to be chosen as an arbitrary polynomial, e.g. even if it is constant, we still have for all b, ρ:

$$\mathsf{PAC} \preceq \mathsf{bSGD}(b, \rho) \quad \text{and if } b\rho < 1/8, \text{ then } \mathsf{bSGD}(b, \rho) \preceq \mathsf{PAC} \tag{4}$$

These results can also be viewed as a relationship between classes of learning problems. Formally, a *learning problem* is a sequence $(\mathcal{P}_n)_{n\in\mathbb{N}}$, where each $\mathcal{P}_n$ is a set of distributions $\mathcal{D}_n$ over $\{0,1\}^n \times \mathcal{Y}$. For a parametrized class of methods $\mathcal{C}(\theta)$, we say a learning problem is poly-learnable with $\mathcal{C}$ if for every polynomial $\varepsilon(n)$ there is a polynomial $\theta(n)$ s.t. for every n there is a method in $\mathcal{C}(\theta(n))$ that ensures error at most $\varepsilon(n)$ on all distributions in $\mathcal{P}_n$. We slightly abuse notation and use $\mathcal{C}$ to denote the set of learning problems poly-learnable with $\mathcal{C}$ algorithms, so that, e.g. PAC, $\mathsf{PAC}_{\mathsf{TM}}$, SQ and $\mathsf{SQ}_{\mathsf{TM}}$, are the familiar classes of PAC and SQ (poly) learnable problems. For bSGD we also define $\mathsf{bSGD}[b(n), \rho(n)]$, where b and ρ are only allowed to depend on n via the specified (polynomially bounded, possibly constant) dependence, and not arbitrarily (the other parameters may grow as an arbitrary polynomial), and $\mathsf{bSGD}[b(n, \rho)]$, where $1/\rho$ can grow as an arbitrary polynomial, but the choice of b as a function of n and ρ is constrained. With this notation, our results imply the following relationships:

Corollary 1. *For any (poly bounded, possible constant) dependence $b(n, \rho)$ it holds that*

$$\begin{aligned} \mathsf{SQ} &\subseteq^{(1)} \mathsf{bSGD}[b] &&\subseteq^{(2)} \mathsf{PAC} \\ \mathsf{SQ}_{\mathsf{TM}} &\subseteq^{(3)} \mathsf{bSGD}_{\mathsf{NN}}[b] &&\subseteq^{(4)} \mathsf{PAC}_{\mathsf{TM}} \end{aligned}$$

Moreover, if $\forall_{n,\rho} \boldsymbol{b(n,\rho) < 1/(8\rho)}$, then inclusions (2) and (4) are tight, and if $\boldsymbol{b(n,\rho) \geq \omega(\log n)/\rho^2}$, then inclusions (1) and (3) are tight.

Corollary 2. *For any (poly bounded, possibly constant) $b(n), \rho(n)$:*

$$\begin{aligned} &\textit{If } \forall_n\ b\rho < 1/8 \textit{ then} && \mathsf{bSGD}[b,\rho] = \mathsf{PAC} && \textit{and} && \mathsf{bSGD}_{\mathsf{NN}}[b,\rho] = \mathsf{PAC}_{\mathsf{TM}} \\ &\textit{If } b\rho^2 \geq \omega(\log n) \textit{ then} && \mathsf{bSGD}[b,\rho] \subseteq \mathsf{SQ} \subsetneq \mathsf{PAC} && \textit{and} && \mathsf{bSGD}_{\mathsf{NN}}[b,\rho] \subseteq \mathsf{SQ}_{\mathsf{TM}} \subsetneq \mathsf{PAC}_{\mathsf{TM}} \end{aligned}$$

In Corollaries 1 and 2, for the sake of simplicity, we focused on *realizable* learning problems, where $\inf_f \mathcal{L}_{\mathcal{D}_n}(f) = 0$ for each $\mathcal{D}_n \in \mathcal{P}_n$. However, we note that Theorems 1 and 2 are more general, as they preserve the performance of learning algorithms (up to an additive δ) on all source distribution $\mathcal{D}$. So, a result similar to Corollary 1 could be stated for other forms of learning, such as agnostic learning, weak learning etc.

The proofs for Theorems 1 and 2 are presented in Appendix D.1, building upon the tools developed in Sections 4 and 5. As mentioned, in general we allow neural networks to have fixed constant weights. However, in Appendix E, we provide an alternate approach to prove similar results for neural networks where all the weights are allowed to train. This has the advantage of being a direct explicit construction, without going through the approach of first constructing a (sophisticated) differentiable model as in Section 5; this is done for a different loss function ($\ell(\hat{y}, y) := \max\{0, |\hat{y} - y| - 1/2\}/2$) and uses a different (piecewise linear) activation. However, this construction requires a restriction in that it only works for source distributions $\mathcal{D}$ in which $\mathbb{E}_{\mathcal{D}}[y]$ is bounded away from both 0 and 1.

4 The Mini-Batch Statistical Query Model

En route to proving Theorems 1 and 2, we introduce the model of mini-**b**atch **S**tatistical **Q**ueries (bSQ), an intermediate model between PAC and SQ. This model is similar to the *honest-SQ* model studied by Yang [20, 21], but with some important distinctions (see Appendix A for a detailed comparison).

A learning algorithm $\mathcal{A}$ is said to be a $\mathsf{bSQ}(k, \tau, b, p, r)$, if it operates in k rounds where in round t, the algorithm produces a query $\Phi_t : \mathcal{X} \times \mathcal{Y} \to [-1, 1]^p$ for which it receives a response v_t and finally outputs a predictor $f : \mathcal{X} \to \mathbb{R}$. Formally, each Φ_t is a function of a random string $R \in \{0, 1\}^r$ and past responses $v_1, \ldots, v_{t-1}$. We say that $\mathcal{A}$ ensures error ε on a source distribution $\mathcal{D}$ if $\mathbb{E}[\sup \mathcal{L}_{\mathcal{D}}(f)] \leq \varepsilon$, where the expectation is over R and mini-batches $S_t \sim \mathcal{D}^b$ and the sup is over all "valid" $v_t \in [-1, 1]^p$, namely

$$\|v_t - \mathbb{E}_{S_t} \Phi_t(x, y)\|_\infty \leq \tau\,. \tag{5}$$

for independently sampled set $S_t \sim \mathcal{D}^b$. The key difference between bSQ and SQ is that the answers are statistical averages over a mini-batch and not over the population. We also allow p simultaneous queries to the same mini-batch; this was irrelevant when discussing SQ algorithms since allowing p sequential queries is only more powerful as each query is statistically averaged over the entire distribution $\mathcal{D}$. The following lemmas establish the relationship between bSQ and PAC/SQ.

Theorem 3. (bSQ vs. PAC) *For all $\delta > 0$,*

1. *If $b < 1/(2\tau)$, then for all m, r, it holds for $k = 10m(n+1)/\delta$, $p = n+1$, $r' = r + k\log_2 b$ that*
$$\mathsf{bSQ}(k, \tau, b, p, r') \preceq \mathsf{PAC}(m, r) + \delta\,.$$
Furthermore, for every runtime TIME*, it holds for* $\mathrm{TIME}' = \mathrm{poly}(n, m, r, \mathrm{TIME}, 1/\delta)$ *that*
$$\mathsf{bSQ}_{\mathsf{TM}}(k, \tau, b, p, r', \mathrm{TIME}') \preceq \mathsf{PAC}_{\mathsf{TM}}(m, r, \mathrm{TIME})\,.$$

2. *For all b, τ and k, p, r, it holds that*
$$\mathsf{PAC}(m = kb, r) \preceq \mathsf{bSQ}(k, \tau, b, p, r)\,.$$
Furthermore, for every runtime TIME *it holds for* $\mathrm{TIME}' = \mathrm{poly}(n, b, \mathrm{TIME})$ *that*
$$\mathsf{PAC}_{\mathsf{TM}}(m, r, \mathrm{TIME}') \preceq \mathsf{bSQ}_{\mathsf{TM}}(k, \tau, b, p, r, \mathrm{TIME})\,.$$

Proof Sketch. The main ingredient for Part (1) is a bSQ algorithm that extracts samples by performing mini-batch statistical queries over independently sampled mini-batches. At a high level this algorithm works by sampling (x, y) one bit at a time, each time drawing from the conditional distribution, conditioned on the previously sampled bit. The details of this algorithm are delicate, but the main reason why this works when $b\tau < 1/2$ is that for any query of the form $\Phi : \mathcal{X} \times \mathcal{Y} \to \{0,1\}^p$, given any valid response v such that $\|v - \mathbb{E}_S \Phi(x,y)\|_\infty \le \tau$, it is possible to exactly recover $\mathbb{E}_S \Phi(x,y)$, by simply rounding each entry to the nearest integral multiple of $1/b$. Once we can simulate sample access, it is straightforward to simulate the entire PAC algorithm. Part (2) is straightforward since a PAC algorithm can easily produce valid responses to bSQ queries by mini-batch averaging and simulating the algorithm. The full proof is presented in Appendix B.1. □

Theorem 4. (bSQ vs. SQ) *There exist universal constants $C \ge 0$ such that for all $\delta > 0$,*

1. *If $b > \frac{C\log(kp/\delta)}{\tau^2}$ for any k, τ, p, r, then it holds for $k' = kp$ and $\tau' = \tau - \sqrt{\frac{C\log(kp/\delta)}{b}}$ that*
$$\mathsf{SQ}(k', \tau', r) \preceq \mathsf{bSQ}(k, \tau, b, p, r) + \delta\,.$$
Furthermore, for any runtime TIME *it holds for* $\mathrm{TIME}' = \mathrm{poly}(\mathrm{TIME})$ *that*
$$\mathsf{SQ}_{\mathsf{TM}}(k', \tau', r, \mathrm{TIME}') \preceq \mathsf{bSQ}_{\mathsf{TM}}(k, \tau, b, p, r, \mathrm{TIME}) + \delta\,.$$

2. *For all b and all k, τ, r, it holds for $\tau' < \tau$, $k' = k \cdot \max\left\{\frac{C\log(k/\delta)}{b(\tau-\tau')^2}, 1\right\}$ and $p = 1$ that*
$$\mathsf{bSQ}(k', \tau', b, p, r) \preceq \mathsf{SQ}(k, \tau, r) + \delta\,.$$
Furthermore, for every runtime TIME *it holds for* $\mathrm{TIME}' = \mathrm{poly}(\mathrm{TIME}, k, (\tau - \tau')^{-1})$ *that*
$$\mathsf{bSQ}_{\mathsf{TM}}(k', \tau', b, p, r, \mathrm{TIME}') \preceq \mathsf{SQ}_{\mathsf{TM}}(k, \tau, r, \mathrm{TIME})\,.$$

Proof Sketch. The main insight for Part (1) here is that when $b \gg 1/\tau^2$, the variation observed between the sample averages $\mathbb{E}_{S_t} \Phi_t(x,y)$ and the population average $\mathbb{E}_{\mathcal{D}} \Phi_t(x,y)$ becomes much smaller than τ with high probability and hence bSQ algorithms can be simulated by SQ algorithms. The main idea for Part (2) is that a bSQ algorithm can simulate responses to an SQ queries by averaging responses across multiple batches if b is not large enough already. The full proof is presented in Appendix B.2. □

5 Equivalence Results for Differentiable Models

In this section, we build tools in order to prove Theorems 1 and 2, for the case of bSGD on general differentiable models, as well as neural networks with some fixed weights; the case of neural networks with all trainable weights is handled separately in Appendix E.

At a high level, we show that bSQ and bSGD algorithms are essentially equally powerful (either one can simulate the other) for all range of parameters b and ρ (or τ). This along with Theorems 3 and 4 allows us to prove Theorems 1 and 2.

As a technical step, we consider a further restriction of bSQ algorithms to what we call *alternating* algorithms, denoted as $\mathsf{bSQ}^{0/1}$. Formally, we say that a (mini-batch) statistical query $\Phi : \mathcal{X} \times \mathcal{Y} \to [-1,1]^p$ is a $\overline{y}$-query for $\overline{y} \in \mathcal{Y}$ if $\Phi(x,y) = 0$ for all $y \ne \overline{y}$, or equivalently $\Phi(x,y) = \mathbb{1}\{y = \overline{y}\} \cdot \Phi_{\mathcal{X}}(x)$ for some $\Phi_{\mathcal{X}} : \mathcal{X} \to [-1,1]^p$. A $\mathsf{bSQ}^{0/1}$ algorithm is a bSQ algorithm such that for all odd rounds t, Φ_t is a 1-query, and at all even rounds t, Φ_t is a 0-query.

Simulating PAC and SQ with $\mathsf{bSQ}^{0/1}$. As minor extensions of Part 1 of Theorem 3 and Part 2 of Theorem 4, we show that PAC and SQ algorithms can be simulated by $\mathsf{bSQ}^{0/1}$ algorithms. We present the full details in Appendix B.

Lemma 1. ($\mathsf{bSQ}^{0/1}$ vs. PAC) *If $b < 1/(2\tau)$, then for all $\delta > 0$ and for all m, r, it holds for $k = 20m(n+1)/\delta$, $p = n+1$, $r' = r + k\log_2 b$ that*
$$\mathsf{bSQ}^{0/1}(k, \tau, b, p, r') \preceq \mathsf{PAC}(m, r) + \delta\,.$$
Furthermore, for every runtime TIME *it holds for* $\mathrm{TIME}' = \mathrm{poly}(n, m, r, \mathrm{TIME}, 1/\delta)$ *that*
$$\mathsf{bSQ}^{0/1}_{\mathsf{TM}}(k, \tau, b, p, r', \mathrm{TIME}') \preceq \mathsf{PAC}_{\mathsf{TM}}(m, r, \mathrm{TIME})\,.$$

Lemma 2. ($\mathsf{bSQ}^{0/1}$ vs. SQ) *There exists a constant $C > 0$, such that for all $\delta > 0$, for all b and all k, τ, r, it holds for $\tau' < \tau/2$, $k' = k \cdot \max\left\{\frac{C\log(k/\delta)}{b(\tau-2\tau')^2}, 1\right\}$ and $p = 1$ that*

$$\mathsf{bSQ}^{0/1}(k', \tau', b, p, r) \;\preceq\; \mathsf{SQ}(k, \tau, r) + \delta\,.$$

Furthermore, for every runtime TIME*. it holds for* $\mathrm{TIME}' = \mathrm{poly}(\mathrm{TIME}, k, (\tau - \tau')^{-1})$ *that*

$$\mathsf{bSQ}^{0/1}_{\mathsf{TM}}(k', \tau', b, p, r, \mathrm{TIME}') \preceq \mathsf{SQ}_{\mathsf{TM}}(k, \tau, r, \mathrm{TIME})\,.$$

Simulating $\mathsf{bSQ}^{0/1}$ with bSGD. Our next step is to show that every $\mathsf{bSQ}^{0/1}$ algorithm can be simulated by a bSGD algorithm.

Lemma 3a. ($\mathsf{bSQ}^{0/1}$ to bSGD) *For all $\tau \in (0, 1)$ and $b, k, p, r \in \mathbb{N}$, it holds for $T = k$, $\rho = \tau/4$, $p' = r + (p + 1)k$, that*

$$\mathsf{bSGD}(T, \rho, b, p', r) \;\preceq\; \mathsf{bSQ}^{0/1}(k, \tau, b, p, r)\,.$$

To achieve the above result, we show how a single $\overline{y}$-query can be simulated using a single step of bSGD on a specific differentiable model. For example, given a 0-query $\Phi : \mathcal{X} \times \mathcal{Y} \to [-1, 1]^p$, consider the following model:

$$f_{\boldsymbol{w}}(x) = 1 - \langle \Phi_{\mathcal{X}}(x), \boldsymbol{w}\rangle \tag{6}$$

Assuming $\boldsymbol{w}^{(0)} = 0$, the model $f_{\boldsymbol{w}^{(0)}}$ "guesses" the label to be 1 for all examples, and therefore suffers a loss only for examples with the label 0. Using a simple gradient calculations we get:

$$\nabla_{\boldsymbol{w}} \mathbb{E}_S \ell_2(f_{\boldsymbol{w}^{(0)}}(x), y) = \mathbb{E}_S \mathbb{1}\{y = 0\} \cdot \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}^{(0)}}(x) = -\mathbb{E}_S \mathbb{1}\{y = 0\} \cdot \Phi_{\mathcal{X}}(x) = -\mathbb{E}_S \Phi(x, y)$$

Hence, after a single step of bSGD we have $\boldsymbol{w}^{(1)} = \mathbb{E}_S \Phi(x, y)$ (up to precision ρ), so $\boldsymbol{w}^{(1)}$ stores the answer for the 0-query Φ. Similarly, we can also construct a differentiable model that simulates 1-queries. Now, achieving the simulation of the complete $\mathsf{bSQ}^{0/1}$ algorithm is done using a composition of such differentiable models. The full details of the proof are given in Appendix D.

Simulating $\mathsf{bSQ}^{0/1}_{\mathsf{TM}}$ with $\mathsf{bSGD}_{\mathsf{NN}}$. In the previous paragraph, we showed how to simulate $\mathsf{bSQ}^{0/1}$ using bSGD on an arbitrary differentiable model. We build on this result to show how $\mathsf{bSQ}_{\mathsf{TM}}$, ($\mathsf{bSQ}$ with tractable computations), can be simulated using $\mathsf{bSGD}_{\mathsf{NN}}$ (bSGD on neural networks).

Lemma 3b. ($\mathsf{bSQ}^{0/1}_{\mathsf{TM}}$ to $\mathsf{bSGD}_{\mathsf{NN}}$) *For all $\tau \in (0, 1)$ and $b, k, p, r \in \mathbb{N}$, it holds for $T = k$, $\rho = \tau/4$, $p' = r + (p + 1)k$, and for $s = \mathrm{poly}(k, 1/\tau, b, p, r, \mathrm{TIME})$, that*

$$\mathsf{bSGD}_{\mathsf{NN}}(T, \rho, b, p', r, s) \;\preceq\; \mathsf{bSQ}^{0/1}_{\mathsf{TM}}(k, \tau, b, p, r, \mathrm{TIME})\,.$$

To show the above result, we rely on the well-known fact that neural networks (and in particular, ReLU networks) can simulate any Boolean circuit, and hence any Turing machine, with a network of size that depends polynomially on the runtime of the Turing machine. So, we observe that the model $f_{\boldsymbol{w}}$ described in Equation (6) can be written as a neural network, given that generating the query $\Phi_{\mathcal{X}}$ can be done using a Turing machine with finite runtime. Finally, a composition of such networks results in a deeper neural network, which can then simulate the complete $\mathsf{bSQ}^{0/1}_{\mathsf{TM}}$ algorithm. Details in Appendix D.

Simulating bSGD with bSQ. Finally, we show that any bSGD algorithm can be implemented as a bSQ algorithm (proof in Appendix B).

Lemma 4. (bSGD to bSQ) *For every T, ρ, b, p, r, it holds that*

$$\mathsf{bSQ}\left(k = T, \tau = \frac{\rho}{4}, b, p, r\right) \;\preceq\; \mathsf{bSGD}(T, \rho, b, p, r)\,.$$

Furthermore, it holds for $\mathrm{TIME} = \mathrm{poly}(T, s, b, r)$ *that*

$$\mathsf{bSQ}_{\mathsf{TM}}(k, \tau, b, p, r, \mathrm{TIME}) \preceq \mathsf{bSGD}_{\mathsf{NN}}(T, \rho, b, p, r, s)\,.$$

Putting it together. Thus, at a high level, $\mathsf{bSQ}^{0/1}$ and bSGD algorithms are essentially equally powerful (either one can simulate the other) for all range of parameters b and ρ (τ). Thus, it is easy to see that the tools developed in this and previous sections can be put together to prove Theorems 1 and 2. We defer the details and calculations of the parameters to Appendix D.1.

6 Summary and Discussion

We provided an almost tight characterization of the learning power of mini-batch SGD, relating it to the well-studied learning paradigms PAC and SQ, and thus (nearly) settling the question of "what can be learned using mini-batch SGD?". That single-sample SGD was able to simulate PAC learning was previously known, but we extended this result considerably, studied its limit, and showed that even outside this limit, bSGD can still always simulate SQ. A gap still remains, when the mini-batch size is between $1/\rho$ and $\log(n)/\rho^2$, where we do not know where bSGD sits between SQ and PAC.

Our results were stated for mini-batch SGD, where an independent mini-batch of samples from the population is used at each step. But it is important to note that very similar results can also be obtained when we draw a single sample once, and then use batch gradient descent to minimize the empirical error on the sample. The simulation of $\mathsf{bSQ}^{0/1}$ with bSGD is still valid, except now we must consider a variant of $\mathsf{bSQ}^{0/1}$ where all queries are answered using the same sample. If the number of training examples is large enough, this can still be used to simulate SQ. And the sample extraction is even easier, since we now make repeated queries to the same sample. We can thus extract samples drawn uniformly with replacement from the training set, which should be enough for simulating PAC. Similar ideas can also be used to analyze what is perhaps the most realistic variant: mini-batch SGD where mini-batches are drawn with replacements (or in epochs) from a training-set. Our results should therefore not be interpreted as indicating a separation between mini-batch SGD and batch GD on the empirical error.

It is tempting to view our results, which show the theoretical power of differentiable learning, as explaining the success of this paradigm. But we do not think that modern deep learning behaves similar to the constructions in our work. While we show how any SQ or PAC algorithm can be simulated, this requires a very carefully constructed network, with an extremely particular initialization, which doesn't look anything like deep learning in current practice. Our result certainly does *not* imply that SGD on a *particular* neural net can learn anything learnable by PAC or SQ, as this would imply that such network can learn any computationally tractable function[6], which is known to be impossible (subject to mild cryptographic assumptions).

Rather, we view our work as guiding us as to what questions we should ask toward understanding how *actual* deep learning works. We see that understanding differentiable learning in such a broad generality as we did here is probably too strong, as it results in answers involving unrealistic initialization, and no restriction, and thus no insight, as to what makes a learning problems are learnable using deep learning. Can we define a class of neural networks, or initializations, which is broad enough to capture the power of deep learning, yet disallows such crazy initialization and does provide insight as to when deep learning is appropriate? Perhaps even mild restrictions on the initialization can already severely restrict the power of differentiable learning. E.g., [15] recently showed that even just requiring that the output of the network at initialization is close to zero can significantly change the power of differentiable learning. An interesting direction for future work is understanding the power of differentiable learning under this, or some other, restriction. Does this lead to a different class of learnable problems, distinct from SQ and PAC, which is perhaps more related to deep learning in practice?

[6] Observe that for any tractable function f, there exists a trivial learning algorithm that returns f regardless of its input, which means that the class $\{f\}$ is PAC learnable.

References

[1] E. Abbe and C. Sandon. On the universality of deep learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e7e8f8e5982b3298c8addedf6811d500-Abstract.html.

[2] Z. Allen-Zhu and Y. Li. What can resnet learn efficiently, going beyond kernels? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9015–9025, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/5857d68cd9280bc98d079fa912fd6740-Abstract.html.

[3] Z. Allen-Zhu and Y. Li. Backward feature correction: How deep learning performs deep learning. *arXiv*, abs/2001.04413, 2020. URL https://arxiv.org/abs/2001.04413.

[4] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009. ISBN 0521424267.

[5] A. Blum and R. L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127, 1992. doi: 10.1016/S0893-6080(05)80010-3. URL https://doi.org/10.1016/S0893-6080(05)80010-3.

[6] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

[7] A. Daniely and E. Malach. Learning parities with neural networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/eaae5e04a259d09af85c108fe4d7dd0c-Abstract.html.

[8] B. Ghorbani, S. Mei, T. Misiakiewicz, and A. Montanari. Linearized two-layers neural networks in high dimension. *arXiv*, abs/1904.12191, 2019. URL http://arxiv.org/abs/1904.12191.

[9] B. Ghorbani, S. Mei, T. Misiakiewicz, and A. Montanari. When do neural networks outperform kernel methods? In *Advances in Neural Information Processing Systems*, volume 33, pages 14820–14830. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/a9df2255ad642b923d95503b9a7958d8-Paper.pdf.

[10] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM (JACM)*, 45(6):983–1006, 1998.

[11] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[12] A. R. Klivans and A. A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. *Journal of Computer and System Sciences*, 75(1):2–12, 2009.

[13] Y. Li, T. Ma, and H. R. Zhang. Learning over-parametrized two-layer neural networks beyond NTK. In *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, volume 125 of *Proceedings of Machine Learning Research*, pages 2613–2682. PMLR, 2020. URL http://proceedings.mlr.press/v125/li20a.html.

[14] E. Malach and S. Shalev-Shwartz. Computational separation between convolutional and fully-connected networks. *arXiv preprint arXiv:2010.01369*, 2020.

[15] E. Malach, P. Kamath, E. Abbe, and N. Srebro. Quantifying the benefit of using differentiable learning over tangent kernels. *arXiv*, abs/2103.01210, 2021. URL https://arxiv.org/abs/2103.01210.

[16] M. S. Nacson, N. Srebro, and D. Soudry. Stochastic gradient descent on separable data: Exact convergence with a fixed learning rate. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 3051–3059. PMLR, 2019. URL http://proceedings.mlr.press/v89/nacson19a.html.

[17] B. Neyshabur, R. Tomioka, and N. Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6614.

[18] L. Reyzin. Statistical queries and statistical algorithms: Foundations and applications. *CoRR*, abs/2004.00557, 2020. URL https://arxiv.org/abs/2004.00557.

[19] D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar, and N. Srebro. The implicit bias of gradient descent on separable data. *J. Mach. Learn. Res.*, 19:70:1–70:57, 2018. URL http://jmlr.org/papers/v19/18-188.html.

[20] K. Yang. On learning correlated boolean functions using statistical queries. In *Algorithmic Learning Theory, 12th International Conference, ALT 2001, Washington, DC, USA, November 25-28, 2001, Proceedings*, volume 2225 of *Lecture Notes in Computer Science*, pages 59–76. Springer, 2001. doi: 10.1007/3-540-45583-3_7. URL https://doi.org/10.1007/3-540-45583-3_7.

[21] K. Yang. New lower bounds for statistical query learning. *J. Comput. Syst. Sci.*, 70(4):485–509, 2005. doi: 10.1016/j.jcss.2004.10.003. URL https://doi.org/10.1016/j.jcss.2004.10.003.

[22] G. Yehudai and O. Shamir. On the power and limitations of random features for understanding neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 6594–6604, 2019. URL http://papers.nips.cc/paper/8886-on-the-power-and-limitations-of-random-features-for-understanding-neural-networks.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] *Main results (theorems) are presented in Section 3.*
   (b) Did you describe the limitations of your work? [Yes] *We discuss the context of our results in Section 6, where we explain some limitations, future directions and especially what our results are **not** about.*
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] *As this is a purely theoretical work aimed at getting a better understanding of deep learning, we don't anticipate any negative societal impacts.*
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] *In Appendices.*
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]