

# Provable limitations of deep learning

Emmanuel Abbe  
EPFL

Colin Sandon  
MIT

## Abstract

As the success of deep learning reaches more grounds, one would like to also envision the potential limits of deep learning. This paper gives a first set of results proving that deep learning algorithms fail at learning certain efficiently learnable functions. Parity functions form the running example of our results and the paper puts forward a notion of low cross-predictability that defines a more general class of functions for which such failures tend to generalize (with examples in community detection and arithmetic learning discussed).

Recall that it is known that the class of neural networks (NNs) with polynomial network size can express any function that can be implemented in polynomial time, and that their sample complexity scales polynomially with the network size. Thus NNs have favorable approximation and estimation errors. The challenge is with the optimization error, as the ERM is NP-hard and there is no known efficient training algorithm with provable guarantees. The success behind deep learning is to train deep NNs with descent algorithms, such as coordinate, gradient or stochastic gradient descent.

The failures shown in this paper apply to training poly-size NNs on function distributions of low cross-predictability with a descent algorithm that is either run with limited memory per sample or that is initialized and run with enough randomness (such as exponentially small Gaussian noise for GD). We further claim that such types of constraints are necessary to obtain failures, in that exact SGD with careful non-random initialization can be shown to learn parities. The cross-predictability notion has some similarity with the statistical dimension used in statistical query (SQ) algorithms, however the two definitions differ due to the fact that our results apply to weak rather than exact learning and failure is proved for statistical rather than adversarial noise; further we obtain negative results for SGD algorithms that are not related to the SQ framework. The proof techniques are based on exhibiting algorithmic constraints that imply a statistical indistinguishability between the algorithm's output on the test model v.s. a null model, using information measures to bound the total variation distance. This is then translated into an algorithmic failure based on the limitations of the null model.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Learning parities . . . . .	3
1.2	An illustrative experiment . . . . .	4
1.3	Insights about failures and successes of deep learning . . . . .	6
1.4	Related works . . . . .	6
<b>2</b>	<b>Results</b>	<b>8</b>
2.1	Definitions and models . . . . .	8
2.2	Negative results . . . . .	10
2.3	Necessary limitations of negative results . . . . .	12
<b>3</b>	<b>Other functions that are difficult for deep learning</b>	<b>12</b>
<b>4</b>	<b>Proof techniques</b>	<b>14</b>
<b>5</b>	<b>Cross-predictability and sequential learning</b>	<b>16</b>
5.1	Cross-predictability . . . . .	16
5.2	Learning from a bit . . . . .	17
5.3	Sequential learning algorithm . . . . .	20
<b>6</b>	<b>Deep learning and parity</b>	<b>25</b>
6.1	Neural nets and deep learning . . . . .	25
6.2	Proof of Theorem 1 . . . . .	28
6.3	Proof of Theorem 2 . . . . .	29
6.4	Proof of Theorem 3 . . . . .	34
6.4.1	Uniform noise and SLAs . . . . .	35
6.4.2	Gaussian noise, noise accumulation, and blurring . . . . .	36
6.4.3	Means, SLAs, and Gaussian distributions . . . . .	41
<b>7</b>	<b>Universality of deep learning</b>	<b>47</b>
7.1	Emulation of Arbitrary Algorithms 1 . . . . .	48
7.2	Emulation of Arbitrary Algorithms 2 . . . . .	49
<b>8</b>	<b>Acknowledgements</b>	<b>49</b>

# 1 Introduction

It is known that the class of neural networks (NNs) with polynomial network size can express any function that can be implemented in a given polynomial time, and that their sample complexity scales polynomially with the network size. Thus NNs have favorable approximation and estimation errors. The main challenge is with the optimization error, as there is no known efficient training algorithm for NNs with provable guarantees, in particular, it is NP-hard to implement the ERM rule [KS09, DSS16].

The success behind deep learning is to train *deep* NNs with *descent* algorithms (e.g., coordinate, gradient or stochastic gradient descent); this gives record performances in image [KSH12], speech [HDY<sup>+</sup>12] and document recognitions [LBBH98], and the scope of applications is increasing on a daily basis [LBH15, GBC16]. While deep learning operates in an overparametrized regime, and while SGD optimizes a highly non-convex objective function, the training by SGD gives astonishingly low generalization errors in these applications. A major research effort is devoted to explaining these successes, with various components claimed responsible, such as the compositional structure of neural networks matching that of real signals, the implicit regularizations behind SGD (e.g., its stochastic component), the increased size of data sets and the augmented computational power, among others.

With the fast expansion of the field, one would like to also envision the potential limits of deep learning. This is the focus of this paper. To understand the limitations of deep learning, we look for a class of functions that are efficiently learnable by some algorithm, but that deep learning fails at learning.

The function that computes the parity of a Boolean vector is a well-known candidate [LeC16], as most functions that SGD is likely to try would be essentially uncorrelated with it, making it difficult to get close enough to the right function in a manageable time. However, any Boolean function that can be computed in time  $O(T(n))$  can also be expressed by a neural network of size  $O(T(n)^2)$  [Par94, SSBD14], and so one could always start with a neural net that is set to compute the desired function, such as the parity function. The problem is thus meaningful only if one constraints the type of initialization (e.g., random initializations) or if one deals with a class of functions (concept class) rather than a specific one, as commonly done for parities [Sha18, Raz16]. We next discuss parities before going back to other function distributions in Sections 1.3 and 2.

## 1.1 Learning parities

The problem of learning parities is then formulated as follows. Define the class of all parity functions by  $\mathcal{F} = \{p_s : s \subseteq [n]\}$ , where  $p_s : \{+1, -1\}^n \rightarrow \{+1, -1\}$  is such that

$$p_s(x) = \prod_{i \in s} x_i.$$

Nature picks  $S$  uniformly at random in  $2^{[n]}$ , and with access to  $\mathcal{F}$  but not to  $S$ , the problem is to run a descent algorithm for a polynomial number of steps  $t$  (in  $n$ ) to obtain  $w^{(t)}$  (e.g., coordinate, gradient or stochastic gradient descent using labeled samples  $(X_i, P_S(X_i))$  where the  $X_i$ 's are independently and uniformly drawn in  $\{+1, -1\}^n$ ).

The goal is to have the neural network output a label  $eval_{w^{(t)}}(X)$  on a uniformly random input  $X$  that (at least) correlates with the true label  $p_S(X)$ , such as

$$I(eval_{w^{(t)}}(X); p_S(X)) = \Omega_n(1),$$

for some notion of mutual information (e.g., TV, KL or Chi-squared mutual information), or

$$\mathbb{P}\{eval_{w(t)}(X) = p_S(X)\} = 1/2 + \Omega_n(1),$$

if the output is made binary.

Note that this is a weak learning requirement, thus failing at this is discarding any stronger requirements related to PAC-learning as mentioned in Section 2. Note also that this objective can be achieved if we do not restrict ourselves to using a NN trained with a descent algorithm. In fact, one can simply take an algorithm that builds a basis from enough samples (e.g.,  $n + \Omega(\log(n))$ ) and solves the resulting system of linear equations to reconstruct  $S$ . This seems however far from how deep learning proceeds. For instance, SGD is “memoryless” in that it updates the weights of the NN at each step with a sample but does not a priori explicitly remember the previous samples. Since each sample gives very little information about the true  $S$ , it thus seems unlikely for SGD to make any progress on a polynomial time horizon. However, it is far from trivial to argue this formally if we allow the NN to be arbitrarily large and with arbitrary initialization (albeit of polynomial complexity), and in particular inspecting the gradient is typically not sufficient. In fact, we claim that this is wrong, and deep learning can learn the parity function with a careful (though poly-time) initialization — See Sections 2.3 and 7.

On the other hand, if the initialization is done at random, as commonly assumed [SSBD14], and the descent algorithm is run with perturbations, as sometimes advocated in different forms [GHJY15, WT11, RRT17], or if one does not move with the full gradient such as in (block-)coordinate descent or more generally bounded-memory update rules, then we show that it is in fact hard to learn parities. We will provide results in such settings showing the failure of deep learning.

Note also that having GD run with little noise is not equivalent to having noisy labels for which learning parities can be hard irrespective of the algorithm used [BKW03, Reg05]; in fact, the amount of noise that we need for running GD and obtain failure is exponentially small, which would effectively represent no noise if that noise was added on the labels itself (and Gaussian elimination would still work).

## 1.2 An illustrative experiment

To illustrate the phenomenon, we consider the following data set and numerical experiment in PyTorch [PGC<sup>+</sup>17]. The elements in  $\mathcal{X}$  are images with a white background and either an even or odd number of black dots, with the parity of the dots determining the label — see Figure 1. The dots are drawn by building a  $k \times k$  grid with white background and activating each square with probability 1/2.

We then train a neural network to learn the parity label of these images. The architecture is a 3 hidden linear layer perceptron with 128 units and ReLU non linearities trained using binary cross entropy. The training and testing dataset are composed of 1000 images of grid-size  $k = 13$ . We used PyTorch implementation of SGD with step size 0.1 and i.i.d. rescaled uniform weight initialization [HZRS15].

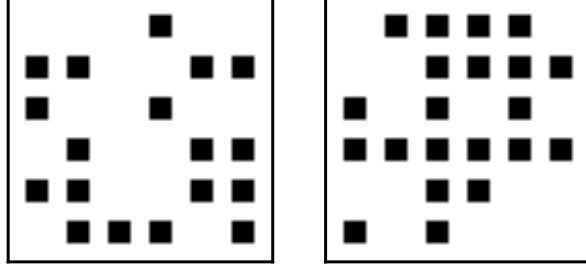


Figure 1: Two images of  $13^2 = 169$  squares colored black with probability  $1/2$ . The left (right) image has an even (odd) number of black squares. The experiment illustrates the incapability of deep learning to learn the parity.

Figure 2 show the evolution of the training loss, testing and training errors. As can be seen, the net can learn the training set but does not generalize better than random guessing.

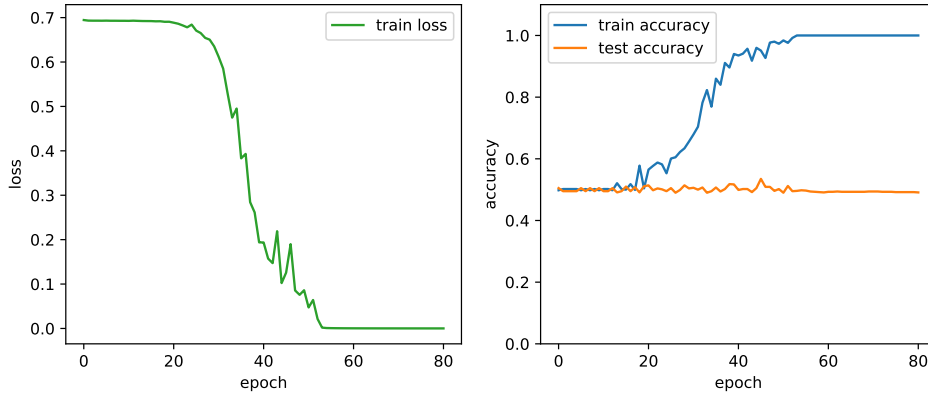


Figure 2: Training loss (left) and training/testing errors (right) for up to 80 SGD epochs.

Note that this is not exactly the model of Section 1.1. In the experiment, each image can be viewed as a Boolean vector of dimension  $k^2$ , but the parity is taken over the entire vector, rather than a subset  $S$ . This is however similar to our setup due to the following observation.<sup>1</sup> First consider the same experiment where one only takes the parity on the set  $S$  consisting of the first half of the image; this would have the same performance outcome. Now, since the net is initialized with i.i.d. random weights, taking  $S$  has the first half or any other random subset of  $\approx k^2/2$  components leads to the same outcome by symmetry. Therefore, we expect failure for the same reason as we expect failure in our model with enough randomness in the initialization.

<sup>1</sup>Another minor distinction is to sample from a pre-set training set v.s. sampling fresh samples, but both can be implemented similarly for the experiment.

### 1.3 Insights about failures and successes of deep learning

Our negative results reveal a measure that captures when deep learning fails. For a probability measure  $P_{\mathcal{X}}$  on the data domain  $\mathcal{X}$ , and a probability measure  $P_{\mathcal{F}}$  on the class of functions  $\mathcal{F}$  from  $\mathcal{X}$  to  $\mathcal{Y}$ , we define the cross-predictability of  $P_{\mathcal{F}}$  with respect to  $P_{\mathcal{X}}$  by

$$\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}}) = \mathbb{E}_{F,F'}(\mathbb{E}_X F(X)F'(X))^2, \quad (X, F, F') \sim P_{\mathcal{X}} \times P_{\mathcal{F}} \times P_{\mathcal{F}}. \quad (1)$$

This measures how predictable a sampled function is from another one on a typical input, and our results primarily exploit the low cross-predictability to obtain negative learning results. In particular, one can obtain failure of SGD with bounded memory per sample and GD with exponentially small noise for any distribution of cross-predictability that vanishes at a super-polynomial rate. One can further express this property in terms of the Fourier-Walsh expansion of the functions, with parity functions as a worst-case example. We also obtain failure of SGD with noise for the case of parities.

The main insight is as follows. All of the algorithms that we consider essentially take a neural net, attempt to compute how well the functions computed by the net and slightly perturbed versions of the net correlate with the target function, and adjust the net in the direction of higher correlation. If none of these functions have significant correlation with the target function, this will generally make little or no progress. The descent algorithm evolves in a flat minima where no significant progress is made in a polynomial time horizon.

Of course, for any specific target function, we could initialize the net to correlate with it. However, if the target function is randomly drawn from a class with negligible cross-predictability, then no function is significantly correlated with the target function with nonnegligible probability. As such, we believe that training a neural network with a descent algorithm will generally fail to learn the function in question in polynomial time.

In contrast, on typical images or sounds, many of the functions we would want to learn are correlated both with each other and with functions a random neural net is likely to compute. For instance, the objects in an image will correlate with whether the image is outside, which will in turn correlate with whether the top left pixel is sky blue. A randomly initialized neural net is likely to compute a function that is nontrivially correlated with the last of these, and some perturbations of it will correlate with it more, which means the network is in position to start learning the functions in question.

Intuitively, this is due to the fact that images and image classes have more compositional structures (i.e., their labels are well explained by combining ‘local’ features). Instead, parity functions are not well explained by the composition of local features of the vectors, and require global operations on the input.

### 1.4 Related works

The difficulty of learning parities with NNs is not new. The parity was already known to be hard based on the early works on the perceptron [MP87], see also [Hås87, All96].

The lack of correlations between two parity functions and its implication in learning parities appear also in the context of statistical query learning algorithms [Kea98, BFJ<sup>+</sup>94], which are algorithms that access estimates of the expected value of some query function over the underlying data distribution (e.g., the first moment statistics of inputs’ coordinates). In particular [FGV17] shows that gradient-based algorithms with approximate oracle access are realizable as statistical query algorithms, which gives strong support to the fact that parities should be hard to learn for deep learning. Note however that [FGV17] makes a convexity assumption that is not satisfied by non-trivial neural nets. More generally, statistical query bounds may give impossibility results for

full gradient descent but with adversarial noise rather than statistical noise as we consider here in Theorem 2. Recall that adversarial is more convenient to handle to obtain negative results. Therefore one cannot use such results in our setting where the noise is statistical. Another key difference is that our results show failures for weak rather than exact learning; this explains the discrepancy between the statistical distance and cross-predictability notions. Finally, the SQ framework does not apply to noisy SGD (even for adversarial noise). In fact, we believe that it is possible to learn parities with better noise-tolerance and complexity than any SQ algorithm will do — see further discussion below and in Sections 2.3 and 7.

In [Raz16], it is shown that one needs either quadratic memory or an exponential number of samples in order to learn parities, settling a conjecture from [SVW15]. This gives a first non-trivial lower bound on the number of samples needed for a learning problem and a first complete negative result in this context. Applications to bounded-storage cryptography are also given in [Raz16]. Other works have extended the results of [Raz16]; in particular [KRT17] applies to  $k$ -sparse sources, [Raz17] to other functions than parities, and [GRT18] exploits properties of two-source extractors<sup>2</sup> to obtain comparable memory v.s. sample complexity trade-offs, with similar results obtained in [BOY17].

In contrast to [Raz16] and follow-up papers, one of our results below, Theorem 6, shows that one needs exponentially many samples to learn parities if less than  $n/24$  pre-assigned bits of memory are used *per sample*. These are thus different models and results. Our result does not say anything interesting about our ability to learn parity with an algorithm that has free access to memory, while the Raz’ result says that it would need to have  $\Omega(n^2)$  total memory or an exponential number of samples. On the flip side, our result shows that an algorithm with unlimited amounts of memory will still be unable to learn a random parity function from a subexponential number of samples if there are sufficiently tight limits on how much it can edit the memory while looking at each sample. The latter is relevant to study SGD with bounded memory as discussed in this paper.

Finally, [SSS17], with an earlier version in [Sha18] from the first author, also give strong support to the impossibility of learning parities. In particular the latter discusses whether specific assumptions on the “niceness” of the input distribution or the target function (for example based on notions of smoothness, non-degeneracy, incoherence or random choice of parameters), are sufficient to guarantee learnability using gradient-based methods, and evidences are provided that neither class of assumptions alone is sufficient.

[SSS17] gives further theoretical insights and practical experiments on the failure of learning parities in such context. More specifically, it proves that the gradient of the loss function of a neural network will be essentially independent of the parity function used. This is achieved by a variant of our Lemma 2 below with the requirement in [SSS17] that the loss function is 1-Lipschitz<sup>3</sup>. This provides a strong intuition of why one should not be able to learn a random parity function using gradient descent or one of its variants, and this is backed up with theoretical and experimental evidence, bringing up the issue of the flat minima. However, it is not proved that one cannot learn parity using stochastic gradient descent or the like. The implication is far from trivial, as with the right algorithm, it is indeed possible to reconstruct the parity function from the gradients of the loss function on a list of random inputs. In fact, as mentioned above and further discussed in this paper, we believe that it is possible to learn a random parity function in polynomial time by using GD or SGD with a careful poly-time initialization of the net (that is of course also agnostic to the parity function). As we show further here, obtaining formal negative results requires more specific

---

<sup>2</sup>The cross-predictability has also similarity with notions of almost orthogonal matrices used in  $L_2$ -extractors for two independent sources [CG88, GRT18], although it does not appear to be exactly the same [Raz18].

<sup>3</sup>The proofs are both simple but slightly different, in particular Lemma 2 does not make regularity assumptions.

assumptions and elaborate proofs, already for GD and particularly for SGD.

We are not aware of meaningful results for noisy SGD (that compare or not with Theorem 3).

## 2 Results

### 2.1 Definitions and models

Before we can talk about the effectiveness of deep learning at learning parity functions, we have to establish some basic notions about deep learning. First of all, in this paper we will be using the following definition for a neural net.

**Definition 1.** *A neural net is a pair of a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  and a weighted directed graph  $G$  with some special vertices and the following properties. First of all,  $G$  does not contain any cycle. Secondly, there exists  $n > 0$  such that  $G$  has exactly  $n + 1$  vertices that have no edges ending at them,  $v_0, v_1, \dots, v_n$ . We will refer to  $n$  as the input size,  $v_0$  as the constant vertex and  $v_1, v_2, \dots, v_n$  as the input vertices. Finally, there exists a vertex  $v_{out}$  such that for any other vertex  $v'$ , there is a path from  $v'$  to  $v_{out}$  in  $G$ . We also use denote by  $w(G)$  the weights on the edges of  $G$ .*

**Definition 2.** *Given a neural net  $(f, G)$  with input size  $n$ , and  $x \in \mathbb{R}^n$ , the evaluation of  $(f, G)$  at  $x$ , written as  $eval_{(f, G)}(x)$ , is the scalar computed by means of the following procedure:*

1. Define  $y \in \mathbb{R}^{|G|}$  where  $|G|$  is the number of vertices in  $G$ , set  $y_{v_0} = 1$ , and set  $y_{v_i} = x_i$  for each  $i$ .
2. Find an ordering  $v'_1, \dots, v'_m$  of the vertices in  $G$  other than the constant vertex and input vertices such that for all  $j > i$ , there is not an edge from  $v'_j$  to  $v'_i$ .
3. For each  $1 \leq i \leq m$ , set

$$y_{v'_i} = f \left( \sum_{v: (v, v'_i) \in E(G)} w_{v, v'_i} y_v \right)$$

4. Return  $y_{v_{out}}$ .

Generally, we want to find a neural net that computes a certain function, or at least a good approximation of that function. A reasonable approach to doing that is to start with some neural network and then attempt to adjust its weights until it computes a reasonable approximation of the desired function. The trademark of deep learning is to do this by defining a loss function in terms of how much the network's outputs differ from the desired outputs, and then use a descent algorithm to try to adjust the weights. More formally, if our loss function is  $L$ , the function we are trying to learn is  $h$ , and our net is  $(f, G)$ , then the net's loss at a given input  $x$  is  $L(h(x) - eval_{(f, G)}(x))$  (or more generally  $L(h(x), eval_{(f, G)}(x))$ ). Given a probability distribution for the function's inputs, we also define the net's expected loss as  $\mathbb{E}[L(h(X) - eval_{(f, G)}(X))]$ . We now discuss three common descent algorithms.

The gradient descent (GD) algorithm does the following. Its input includes the initial neural net, a learning rate, and the number of time steps the algorithm runs for. At each time step, the algorithm computes the derivative of the net's expected loss with respect to each of its edge weights, and then decreases each edge weight by the derivative with respect to that weight times the learning rate. After the final time step, it returns the current neural net.

One problem with GD is that it requires computing an expectation over every possible input, which is generally impractical. One possible fix to that is to use stochastic gradient descent (SGD)



instead of gradient descent. The input to SGD includes the initial neural net, a learning rate, and the number of time steps the algorithm runs for. However, instead of computing an expectation over all possible inputs in each time step, SGD randomly selects a single input in each time step, computes the derivative of the net’s loss at that input with respect to each edge weight, and decreases every edge weight by the corresponding derivative times the learning rate. Note that the stochasticity in SGD is sometimes also claimed to help with the generalization error of deep learning; with the insight that it helps with stability, implicit regularization or bad critical points [HRS16, ZBH<sup>+</sup>16, PP17, KLY18].

A third option is to use coordinate (or block-coordinate) descent (CD) instead of gradient descent. This works as SGD, except that in each time step CD only updates a small number of edge weights, with all other weights remaining fixed. There are multiple options for deciding which edge weights to change, some of which would base the decision on the chosen input, but our result involving CD will be generic and not depend on the details of how the edges are chosen.

It is also possible to use a noisy version of any of the algorithm mentioned above. This would be the same as the noise-free version, except that in each time step, the algorithm independently draws a noise term for each edge from some probability distribution and adds it to that edge’s weight. Adding noise can help avoid getting stuck in local minima or regions where the derivatives are small [GHJY15], however it can also drown out information provided by the gradient, and some learning algorithms are extremely sensitive to noise.

Finally, each of these algorithms needs to start with an initial set of weights before initiating the descent. A priori, the initialization could be done according to any rule, albeit with manageable complexity since we will focus in this paper on the efficiency of algorithms. However, in practice, SGD implementations in deep learning typically start with random initializations, see [SSBD14], or variants such as [HZRS15] that involve different types of probability distributions for the initial weights. Note that random initializations have also been shown to help with escaping certain bad extremal points [LSJR16].

We want to answer the question of whether or not training a neural net with these algorithms is a universal method of learning, in the sense that it can learn anything that is reasonably learnable. We next define exactly what this means.

**Definition 3.** Let  $n > 0$ ,  $\epsilon > 0$ ,  $P_{\mathcal{X}}$  be a probability distribution on  $\{0, 1\}^n$ , and  $P_{\mathcal{F}}$  be a probability distribution on the set of functions from  $\{0, 1\}^n$  to  $\{0, 1\}$ . Also, let  $X_0, X_1, \dots$  be independently drawn from  $P_{\mathcal{X}}$  and  $F \sim P_{\mathcal{F}}$ . An algorithm learns  $(P_{\mathcal{F}}, P_{\mathcal{X}})$  with accuracy  $1/2 + \epsilon$  if the following holds. There exists  $T > 0$  such that if the algorithm is given the value of  $(X_i, F(X_i))$  for each  $i < T$  and it is given the value of  $X_T$ , it returns  $Y_T$  such that  $P[F(X_T) = Y_T] \geq 1/2 + \epsilon$ .

In particular, we talk about “learning parities” in the case where  $P_{\mathcal{F}}$  picks a parity function uniformly at random and  $P_{\mathcal{X}}$  is uniform on  $\{+1, -1\}^n$ , as defined in Section 1.1.

**Remark 1.** As we have defined them, neural nets generally give outputs in  $\mathbb{R}$  rather than  $\{0, 1\}$ . As such, when talking about whether training a neural net by some method learns a given function, we will implicitly be assuming that the output of the net on the final input is thresholded at some predefined value or the like. None of our results depend on exactly how we deal with this part.

**Definition 4.** For each  $n > 0$ , let<sup>4</sup>  $P_{\mathcal{X}}$  be a probability distribution on  $\{0, 1\}^n$ , and  $P_{\mathcal{F}}$  be a probability distribution on the set of functions from  $\{0, 1\}^n$  to  $\{0, 1\}$ . We say that  $(P_{\mathcal{F}}, P_{\mathcal{X}})$  is efficiently learnable if there exists  $\epsilon > 0$ ,  $N > 0$ , and an algorithm with running time polynomial in  $n$  such that for all  $n \geq N$ , the algorithm learns  $(P_{\mathcal{F}}, P_{\mathcal{X}})$  with accuracy  $1/2 + \epsilon$ .

---

<sup>4</sup>Note that these are formally sequences of distributions.

## 2.2 Negative results

In order to disprove the universality of learning of these algorithms, we need an efficiently learnable function that they fail to learn. In this paper, we will use a random parity function with input that is uniformly distributed in  $\{0, 1\}^n$ . One can easily learn such a function by taking a linear sized sample of its values on random inputs, finding a basis of  $\{0, 1\}^n$  in the inputs sampled, and then using the fact that these parity functions are linear. However, as we will show, some of the algorithms listed above are unable to learn such a function. The fundamental problem is that any two different parity functions are uncorrelated, so no function is significantly correlated with a nonnegligible fraction of them. As a result, the neural nets will generally fail to even come close enough to computing the desired function for the gradient to provide useful feedback on how to improve it. We will formalize this idea by defining a quantity called cross-predictability and showing that it is exponentially small for random parity functions. Similar negative results would hold for other families of functions with comparably low cross-predictability. The results in question are the following:

**Theorem 1.** *Let  $\epsilon > 0$ . For each  $n > 0$ , let  $(f, g)$  be a neural net of polynomial size in  $n$  such that each edge weight is recorded using  $O(\log(n))$  bits of memory. Run stochastic gradient descent on  $(f, g)$  with at most  $2^{n/24}$  time steps and with  $o(n/\log(n))$  edge weights updated per time step. For all sufficiently large  $n$ , this algorithm fails at learning parities with accuracy  $1/2 + \epsilon$ .*

**Corollary 1.** *Coordinate descent with a polynomial number of steps and precision fails at learning parities with non-trivial accuracy.*

If one applies the same reasoning used to prove this theorem to a general distribution of functions instead of the standard distribution of parity functions, one gets the following.

**Theorem 1'.** *Let  $\epsilon > 0$ , and  $P_{\mathcal{F}}$  be a probability distribution over functions with a cross-predictability of  $c_p = o(1)$ . For each  $n > 0$ , let  $(f, g)$  be a neural net of polynomial size in  $n$  such that each edge weight is recorded using  $O(\log(n))$  bits of memory. Run stochastic gradient descent on  $(f, g)$  with at most  $c_p^{-1/24}$  time steps and with  $o(|\log(c_p)|/\log(n))$  edge weights updated per time step. For all sufficiently large  $n$ , this algorithm fails at learning functions drawn from  $P_{\mathcal{F}}$  with accuracy  $1/2 + \epsilon$ .*

**Theorem 2.** *For each  $n > 0$ , let  $(f, g)$  be a neural net of polynomial size in  $n$ . Run gradient descent on  $(f, g)$  with less than  $2^{n/10}$  time steps, a learning rate of at most  $2^{n/10}$ , Gaussian noise with variance at least  $2^{-n/10}$  and overflow range of at most  $2^{n/10}$ . For all sufficiently large  $n$ , this algorithm fails at learning parities with accuracy  $1/2 + 2^{-n/10}$ .*

**Remark 2.** *An overflow range of  $B$  means that any value (e.g., derivatives of the loss function for a certain input) potentially exceeding  $B$  is kept at  $B$ .*

**Remark 3.** *We could alternately have defined the algorithm such that if there is any input for which one of the derivatives is larger than the overflow range, we give up and return random predictions. In this case, the result above would still hold.*

*As a third option, we could define  $\epsilon$  to be the probability that there is a time step in which the derivative of the loss function with respect to some edge weight is greater than the overflow range for some input. In this case, given that no such overflow occurs, the algorithm would fail to learn parities with accuracy  $1/2 + 2^{-n/10}/(1 - \epsilon)$ .*

**Remark 4.** *Note that having GD run with a little noise is not equivalent to having noisy labels for which learning parities can be hard irrespective of the algorithm used [BKW03, Reg05]. The*

amount of noise needed for GD in the above theorem is exponentially small, and if such amount of noise were added to the sample labels themselves, then the noise would essentially be ineffective (e.g., Gaussian elimination would still work with rounding, or if the noise were Boolean with such variance, no flip would take place with high probability).

**Remark 5.** As a consequence of Remark 11, the result extends to other function/input distributions having a cross-predictability of at least super-polynomial decay.

**Remark 6.** In various ways, this result is similar to some of the results proven for statistical query algorithms. However, there are some key differences. First of all, our results use noise randomly drawn from a given distribution, whereas statistical query algorithms tend to use adversarial noise. As such, in order to prove that a SQ algorithm fails to learn, it suffices to show that the allowable noise is greater than the signal in each step. Under our model, that would still leave the possibility that the signal will add up over several steps and the noise will largely cancel out. Also, we express the degree to which a probability distribution of functions is difficult to learn by defining the cross-predictability, whereas SQ results typically use the statistical dimension [Kea98, FGR<sup>+</sup>17]. The cross-predictability is a measure of the average correlation between functions drawn from the distribution, whereas the statistical dimension measures the maximum probability subdistribution with a sufficiently high correlation among its members. As a result, any probability distribution with a low cross predictability must have a high statistical dimension. However, a distribution of functions that are all moderately correlated with each other could have an arbitrarily high statistical dimension despite having a reasonably high cross-predictability. Finally, we consider an algorithm as succeeding at learning a function if it can compute it with accuracy  $1/2 + \epsilon$ , while the SQ results tend to define learning in a significantly stronger sense — see [FGR<sup>+</sup>17].

In the case of full gradient descent, the gradients of the losses with respect to different inputs mostly cancel out, so an exponentially small amount of noise is enough to drown out whatever is left. With stochastic gradient descent, that does not happen, and we have the following instead.

**Definition 5.** Let  $(f, g)$  be a NN, and recall that  $w(g)$  denotes the set of weights on the edges of  $g$ . Define the  $\tau$ -neighborhood of  $(f, g)$  as

$$N_\tau(f, g) = \{(f, g') : E(g') = E(g), |w_{u,v}(g) - w_{u,v}(g')| \leq \tau, \forall (u, v) \in E(g)\}. \quad (2)$$

**Theorem 3.** For each  $n > 0$ , let  $(f, g)$  be a neural net with size  $m$  polynomial in  $n$ , and let  $B, \gamma, T > 0$  such that  $B, 1/\gamma$ , and  $T$  are polynomial in  $n$ . There exist  $\sigma = O(m^2\gamma^2B^2/n^2)$  and  $\sigma' = O(m^3\gamma^3B^3/n^2)$  such that the following holds. Perturb the weight of every edge in the net by a Gaussian distribution of variance  $\sigma$  and then train it with a noisy stochastic gradient descent algorithm with learning rate  $\gamma$ ,  $T$  time steps, and Gaussian noise with variance  $\sigma'$ . Also, let  $p$  be the probability that at some point in the algorithm, there is a neural net  $(f, g')$  in  $N_\tau(f, g)$ ,  $\tau = O(m^2\gamma B/n)$ , such that at least one of the first three derivatives of the loss function on the current sample with respect to some edge weight(s) of  $(f, g')$  has absolute value greater than  $B$ . Then this algorithm fails to learn parities with an accuracy greater than  $1/2 + 2p + O(Tm^4B^2\gamma^2/n)$ .

**Remark 7.** Normally, we would expect that if training a neural net by means of SGD works, then the net will improve at a rate proportional to the learning rate, as long as the learning rate is small enough. As such, we would expect that the number of time steps needed to learn a function would be inversely proportional to the learning rate. This theorem shows that if we set  $T = c/\gamma$  for any constant  $c$  and slowly decrease  $\gamma$ , then the accuracy will approach  $1/2 + 2p$  or less. If we also let  $B$  slowly increase, we would expect that  $p$  will go to 0, so the accuracy will go to  $1/2$ . It is also worth

noting that as  $\gamma$  decreases, the typical size of the noise terms will scale as  $\gamma^{3/2}$ . So, for sufficiently small values of  $\gamma$ , the noise terms that are added to edge weights will generally be much smaller than the signal terms.

**Remark 8.** *The bound on the derivatives of the loss function is essentially a requirement that the behavior of the net be stable under small changes to the weights. It is necessary because otherwise one could effectively multiply the learning rate by an arbitrarily large factor simply by ensuring that the derivative is very large. Alternately, excessively large derivatives could cause the probability distribution of the edge weights to change in ways that disrupt our attempts to approximate this probability distribution using Gaussian distributions. For any given initial value of the neural net, and any given  $M > 0$ , there must exist some  $B$  such that as long as none of the edge weights become larger than  $M$  this will always hold. However, that  $B$  could be very large, especially if the net has many layers.*

**Definition 6.** [MRT12] *A class of function  $\mathcal{F}$  is said to be PAC-learnable if there exists an algorithm  $A$  and a polynomial function  $\text{poly}(\cdot, \cdot, \cdot)$  such that for any  $\varepsilon > 0$  and  $\delta > 0$ , for all distributions  $D$  on  $\mathcal{X}$  and for any target function  $f \in \mathcal{F}$ , the following holds for any sample size  $m \geq \text{poly}(1/\varepsilon, 1/\delta, n, \text{size}(f))$ :*

$$\mathbb{P}_{S \sim D^m} \{ \mathbb{P}_{X \sim D} \{ h_S(X) \neq f(X) \} \leq \varepsilon \} \geq 1 - \delta. \quad (3)$$

*If  $A$  further runs in  $\text{poly}(1/\varepsilon, 1/\delta, n, \text{size}(f))$ , then  $\mathcal{F}$  is said to be efficiently PAC-learnable. When such an algorithm  $A$  exists, it is called a PAC-learning algorithm for  $\mathcal{F}$ .*

Therefore, picking  $D$  to be uniform on  $\mathbb{B}^n$ ,  $\varepsilon = 1/10$  and  $\delta = 1/10$ , Theorems 1, 2, 3 imply that a neural network trained by one of the specified descent algorithms on a polynomial number of samples will not compute the parity function in question with accuracy  $1 - \varepsilon$  with probability  $1 - \delta$ . Thus, we have the following.

**Theorem 4.** *Deep learning algorithms as described in Theorems 1, 2, 3, fail at PAC-learning the class of parity functions  $\mathcal{F} = \{p_s : s \subseteq [n]\}$  in  $\text{poly}(n)$ -time.*

### 2.3 Necessary limitations of negative results

We show that deep learning fails at learning parities in polynomial time if the descent algorithm is either run with limited memory or initialized and run with enough randomness. However, if initialized carefully and run with enough memory and precision, we claim that it is in fact possible to learn parities:

*One can construct in polynomial time in  $n$  a neural net  $(f, g)$  that has polynomial size in  $n$  such that for a learning rate  $\gamma$  that is at most polynomial in  $n$  and an integer  $T$  that is at most polynomial in  $n$ ,  $(f, g)$  trained by SGD with learning rate  $\gamma$  and  $T$  time steps learns parities with accuracy  $1 - o(1)$ .*

In fact, we claim that a more general result holds for any efficiently learnable distribution, and that deep learning can universally learn any such distribution (though the initialization will be unpractical) — Section 7. Full proofs will follow in subsequent versions of the paper.

## 3 Other functions that are difficult for deep learning

This paper focuses on the difficulty of learning parities, with proofs extending to other function/input distributions having comparably low cross-predictability. Parities are not the most common type of

functions used to generate real signals, but they are central to the construction of good codes (in particular the most important class of codes, i.e., linear codes, that rely heavily on parities). We mention now a few additional examples of functions that we believe would be also difficult to learn with deep learning.

**Arithmetic.** First of all, consider trying to teach a neural net arithmetic. More precisely, consider trying to teach it the following function. The function takes as input a list of  $n$  numbers that are written in base  $n$  and are  $n$  digits long, combined with a number that is  $n + 1$  digits long and has all but one digit replaced by question marks, where the remaining digit is not the first. Then, it returns whether or not the sum of the first  $n$  numbers matches the remaining digit of the final number. So, it would essentially take expressions like the following, and check whether there is a way to replace the question marks with digits such that the expression is true.

$$\begin{array}{r} 120 \\ +112 \\ +121 \\ =??0? \end{array}$$

Here, we can define a class of functions by defining a separate function for every possible ordering of the digits. If we select inputs randomly and map the outputs to  $\mathbb{R}$  in such a way that the average correct output is 0, then this class will have a low cross predictability. Obviously, we could still initialize a neural net to encode the function with the correct ordering of digits. However, if the net is initialized in a way that does not encode the digit's meanings, then deep learning will have difficulties learning this function comparable to its problems learning parity. Note that one can sort out which digit is which by taking enough samples where the expression is correct and the last digit of the sum is left, using them to derive linear equation in the digits  $(\text{mod } n)$ , and solving for the digits.

We believe that if the input contained the entire alleged sum, then deep learning with a random initialization would also be unable to learn to determine whether or not the sum was correct. However, in order to train it, one would have to give it correct expressions far more often than would arise if it was given random inputs drawn from a probability distribution that was independent of the digits' meanings. As such, our notion of cross predictability does not apply in this case, and the techniques we use in this paper do not work for the version where the entire alleged sum is provided. The techniques instead apply to the above version.

**Connectivity and community detection.** Another example of a problem that we believe deep learning would have trouble with consists of determining whether or not some graphs are connected. This could be difficult because it is a global property of the graph, and there is not necessarily any function of a small number of edges that is correlated with it. Of course, that depends on how the graphs are generated. In order to make it difficult, we define the following probability distribution for random graphs.

**Definition 7.** *Given  $n, m, r > 0$ , let  $AER(n, m, r)$  be the probability distribution of  $n$ -vertex graphs generated by the following procedure. First of all, independently add an edge between each pair of vertices with probability  $m/n$  (i.e., start with an Erdős-Rényi random graph). Then, randomly select a cycle of length less than  $r$  and delete one of its edges at random. Repeat this until there are no longer any cycles of length less than  $r$ .*

Now, we believe that deep learning with a random initialization will not be able to learn to distinguish a graph drawn from  $AER(n, 10 \ln(n), \sqrt{\ln(n)})$  from a pair of graphs drawn from  $AER(n/2, 10 \ln(n), \sqrt{\ln(n)})$ , provided the vertices are randomly relabeled in the latter case. That is, deep learning will not distinguish between a patching of two such random graphs (on half of the vertices) versus a single such graph (on all vertices). Note that a simple depth-first search algorithm would learn the function in poly-time. More generally, we believe that deep learning would not solve community detection on such variants of random graph models<sup>5</sup> (with edges allowed between the clusters as in a stochastic block model with similar loop pruning), as connectivity v.s. disconnectivity is an extreme case of community detection.

The key issue is that no subgraph induced by fewer than  $\sqrt{\ln(n)}$  vertices provides significant information on which of these cases apply. Generally, the function computed by a node in the net can be expressed as a linear combination of some expressions in small numbers of inputs and an expression that is independent of all small sets of inputs. The former cannot possibly be significantly correlated with the desired output, while the later will tend to be uncorrelated with any specified function with high probability. As such, we believe that the neural net would fail to have any nodes that were meaningfully correlated with the output, or any edges that would significantly alter its accuracy if their weights were changed. Thus, the net would have no clear way to improve. ‘

## 4 Proof techniques

Our main approach to showing the failure of an algorithm (e.g., SGD) using data from a test model (e.g., parities) with limited resources (e.g., samples and memory) for a desired task (e.g., non-trivial accuracy of prediction), will be to establish an *indistinguishable to null condition (INC)*, namely, a condition on the resources that implies failure to statistically distinguish the trace of the algorithm on the test model from a null model, where the null model fails to provide the desired performance for trivial reasons. The INC is obtained by manipulating information measures, bounding the total variation distance of the two posterior measures between the test and null models. The failure of achieving the desired algorithmic performance on the test model is then a consequence of the INC, either by converse arguments – if one could achieve the claimed performance, one would be able to use the performance gap to distinguish the null and test models and thus contradict the INC – or directly using the total variation distance between the two probability distributions to bound the difference in the probabilities that the nets drawn from those distributions compute the function correctly (and we know that it fails to do so on the null model).

With more details:

- Let  $D_1$  be the distribution of the data for the parity learning model, i.e., i.i.d. samples with labels from the parity model in dimension  $n$ ;
- Let  $R = (R_1, R_2)$  be the resource in question, i.e., the number  $R_1$  of edge weights of  $\text{poly}(n)$  memory that are updated and the number of steps  $R_2$  of the algorithm;
- Let  $A$  be the SGD (or coordinate descent) algorithm used with a constraint  $C$  on the resource  $R$ ;
- Let  $T$  be the task, i.e, achieving an accuracy of  $1/2 + \Omega_n(1)$  on a random input.

Our program then runs as follows:

---

<sup>5</sup>It would be interesting to investigate the approach of [CLB17] on such models.



1. Chose  $D_0$  as the null distribution that generates i.i.d. pure noise labels, such that the task  $T$  is obviously not achievable for  $D_0$ .
2. Find a INC on  $R$ , i.e., a constraint  $C$  on  $R$  such that the trace of the algorithm  $A$  is indistinguishable under  $D_1$  and  $D_0$ ; to show this,
  - (a) show that the total variation distance between the posterior distribution of the trace of  $A$  under  $D_0$  and  $D_1$  vanishes if the INC holds; to obtain this,
  - (b) show that any  $f$ -mutual information between the algorithm's trace and the model hypotheses  $D_0$  or  $D_1$  (chosen equiprobably) vanishes.
3. Conclude that the INC on  $R$  prohibits the achievement of  $T$  on the test model  $D_0$ , either by contradiction as one could use  $T$  to distinguish between  $D_1$  and  $D_0$  if only the latter fails at  $T$  or using the fact that for any event Success and any random variables  $Y(D_i)$  that depend on data drawn from  $D_i$  (and represent for example the algorithms outputs), we have  $\mathbb{P}\{Y(D_1) \in \text{Success}\} \leq \mathbb{P}\{Y(D_0) \in \text{Success}\} + TV(D_0, D_1) = 1/2 + TV(D_0, D_1)$ .

Most of the work then lies in part 2(a)-(b), which consist in manipulating information measures to obtain the desired conclusion. In particular, the Chi-squared mutual information will be convenient for us, as its “quadratic” form will allow us to bring the cross-predictability as an upper-bound, which is then easy to evaluate and is small for the parity model. This is carried out in Section 5 in the general context of so-called “sequential learning algorithms”, and then applied to SGD with bounded memory (or coordinate descent) in Section 2. For Theorem 2, one needs also to take into account the fact that information can be carried in the pattern of *which* weights can be updated, and these are taken into account with a proper SLA implementation, with Theorem 2 concluding from a contradiction argument as discussed in step 3. above.

In the case of noisy GD (Theorem 2), the program is more direct from step 2, and runs with the following specifications. When computing the full gradient, the losses with respect to different inputs mostly cancel out, which makes the gradient updates reasonably small, and a small amount of noise suffices to cover it. In this case, working<sup>6</sup> with Gaussian noise allows us to bound the total variation distance in terms of the  $\ell_2$  distance of the weight updates between the test vs. null models. We then to bound that  $\ell_2$  distance in terms of the gradient norm using classical orthogonality properties of the parity (Fourier-Walsh) basis.

In the case of the failure of SGD under noisy initialization and updates (Theorem 3), we rely on a more sophisticated version of the above program. We use again a step used for GD that consists in showing that the average value of any function on samples generated by a random parity function will be approximately the same as the average value of the function on true random samples.<sup>7</sup> This is essentially a consequence of the low cross-predictability of parities, which can be proved more directly in this context. Most of the work consist then is using this to show that if we draw a set of weights in  $\mathbb{R}^m$  from a sufficiently noisy probability distribution and then perturb it slightly in a manner dependent on a sample generated by a random parity function, the probability distribution of the result is essentially indistinguishable from what it would be if the samples were truly random. Then, we argue that if we do this repeatedly and add in some extra noise after each step, the probability distribution stays noisy enough that the previous result continues to apply, with the result that the final probability distribution when this is done using samples generated by a random parity function is similar to the final probability distribution using true random samples. After that,

---

<sup>6</sup>Similar results should hold for non-Gaussian distributions; the Gaussian case is more convenient.

<sup>7</sup>This gives also a variant of the result in [SSS17] applying with 1-Lipschitz loss function.

we use that result to show that the probability distribution of the weights in a neural net trained by noisy stochastic gradient descent on a random parity function is indistinguishable from the probability distribution of the weights in a neural net trained by noisy SGD (NSGD) on random samples, which represent most of the work. Finally, we conclude that a neural net trained by NSGD on a random parity function will fail to learn the function (step 3).

## 5 Cross-predictability and sequential learning

### 5.1 Cross-predictability

We denote by  $\mathcal{X}$  the domain of the data (e.g., Boolean vectors, matrices of pixels) and by  $\mathcal{Y}$  the domain of the labels; for simplicity, we assume that  $\mathcal{Y} = \{-1, +1\}$ . A hypothesis is a function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  that labels data points in  $\mathcal{X}$  with elements in  $\mathcal{Y}$ . We define  $\mathcal{F} := \mathcal{Y}^{\mathcal{X}}$ .

**Definition 8.** Let  $P_{\mathcal{X}}$  be a probability measure on  $\mathcal{X}$ , and  $P_{\mathcal{F}}$  be a probability measure on  $\mathcal{F}$  (the set of functions from  $\mathcal{X}$  to  $\mathcal{Y}$ ). Define the cross-predictability of  $P_{\mathcal{F}}$  with respect to  $P_{\mathcal{X}}$  by

$$\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}}) = \mathbb{E}_{F, F'}(\mathbb{E}_X F(X)F'(X))^2, \quad (X, F, F') \sim P_{\mathcal{X}} \times P_{\mathcal{F}} \times P_{\mathcal{F}}. \quad (4)$$

Note that

$$\mathbb{E}_X F(X)F(X') = \mathbb{P}_X\{F(X) = F'(X)\} - \mathbb{P}_X\{F(X) \neq F'(X)\} \quad (5)$$

$$= 2\mathbb{P}_X\{F(X) = F'(X)\} - 1 \quad (6)$$

and we also have

$$\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}}) = \mathbb{E}_{X, X'}(\mathbb{E}_F F(X)F(X'))^2. \quad (7)$$

Therefore a low cross-predictability can be interpreted as having a low correlation between two sampled functions on a sampled input, or, as a low correlation between two sampled input on a sampled function. This suggests that two samples - such as those of two consecutive steps of SGD - may not have common information about a typical function being learned.

We discuss in Remark 6 the analogy and difference between the cross-predictability and the statistical dimension. We next cover some examples.

**Example 1.** If  $P_{\mathcal{X}} = \delta_x$ ,  $x \in \mathcal{X}$ , then  $\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}}) = 1$  no matter what  $P_{\mathcal{F}}$  is. That is, if the world produces always the same data point, then any labelling function has maximal cross-predictability.

**Example 2.** If  $P_{\mathcal{F}}$  is the uniform probability measure on  $\mathcal{F}$ , then  $\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}}) = \|P_{\mathcal{X}}\|_2^2$  no matter what  $P_{\mathcal{X}}$  is. In fact,

$$\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}}) = \mathbb{E}_{F, F'}(\mathbb{E}_{X \sim P_{\mathcal{X}}} F(X)F'(X))^2 \quad (8)$$

$$= (1/|\mathcal{F}|)^2 \sum_{f, f'} \sum_{x, x'} f(x)f'(x)f(x')f'(x')P_{\mathcal{X}}(x)P_{\mathcal{X}}(x') \quad (9)$$

$$= (1/|\mathcal{F}|)^2 \sum_{x, x'} \left( \sum_f f(x)f(x') \right)^2 P_{\mathcal{X}}(x)P_{\mathcal{X}}(x') \quad (10)$$

$$= (1/|\mathcal{F}|)^2 \sum_x \left( \sum_f f^2(x) \right)^2 P_{\mathcal{X}}^2(x) \quad (11)$$

$$= \sum_x P_{\mathcal{X}}^2(x). \quad (12)$$



In particular,  $\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}}) = 1/|\mathcal{X}|$  if  $P_{\mathcal{F}}$  and  $P_{\mathcal{X}}$  are uniform. That is, if the labelling function is “completely random,” then the cross-predictability depends on how “random” the input distribution is, measured by the  $L_2$  norm of  $P_{\mathcal{X}}$ , also called the collision entropy.

**Example 3.** Let  $\mathcal{X} = \mathbb{B}^n$ , and for  $s \in [n]$ , define  $f_s : \mathbb{B}^n \rightarrow \mathbb{B}$  by  $f_s(x) = \prod_{i \in S} x_i$ . Let  $P_{\mathcal{F}} = P_n$  be the uniform probability measure on  $\{f_s\}_{s \in \mathbb{B}^n}$ , and let  $U_n$  be the uniform probability measure on  $\mathbb{B}^n$ . Then  $\text{Pred}(P_n|U_n) = 2^{-n}$ . In fact,  $\mathbb{E}_{S,T} f_S(X) f_T(X) = \mathbb{1}(S = T)$ , thus  $\text{Pred}(P_n|U_n) = \mathbb{P}_{S,T}\{S = T\} = 2^{-n}$ .

Note that uniformly random parity functions have the same cross-predictability as uniformly random generic functions, with respect to uniform inputs. We will crucially exploit this property to prove the forthcoming results. We obtain in fact generalizations of two our results to other function distributions having cross-predictability that scales super-polynomially.

## 5.2 Learning from a bit

We now consider the following setup:

$$(X, F) \sim P_{\mathcal{X}} \times P_{\mathcal{F}} \quad (13)$$

$$Y = F(X) \text{ (denote by } P_{\mathcal{Y}} \text{ the marginal of } Y) \quad (14)$$

$$W = g(X, Y) \text{ where } g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{B} \quad (15)$$

$$(\tilde{X}, \tilde{Y}) \sim P_{\mathcal{X}} \times U_{\mathcal{Y}} \text{ (independent of } (X, F)) \quad (16)$$

That is, a random input  $X$  and a random hypothesis  $F$  are drawn from the working model, leading to an output label  $Y$ . We store a bit  $W$  after observing the labelled pair  $(X, Y)$ . We are interested in estimating how much information can this bit contain about  $F$ , no matter how “good” the function  $g$  is. We start by measuring the information using the variance of the MSE or Chi-squared mutual information<sup>8</sup>, i.e.,

$$I_2(W; F) = \text{Var } \mathbb{E}(W|F) \quad (17)$$

which gives a measure on how random  $W$  is given  $F$ . We provide below a bound in terms of the cross-predictability of  $P_{\mathcal{F}}$  with respect to  $P_{\mathcal{X}}$ , and the marginal probability that  $g$  takes value 1 on two independent inputs, which is a “inherent bias” of  $g$ .

The Chi-squared is convenient to analyze and is stronger than the classical mutual information, which is itself stronger than the squared total-variation distance by Pinsker’s inequality. More precisely<sup>9</sup>, for an equiprobable  $W$ ,

$$TV(W; F) \lesssim I(W; F)^{1/2} \leq I_2(W; F)^{1/2}. \quad (18)$$

Here we will need to obtain such inequalities for arbitrary marginal distributions of  $W$  and in a self-contain series of lemmas. We then bound the latter with the cross-predictability which allows us to bound the error probability of the hypothesis test deciding whether  $W$  is dependent on  $F$  or not, which we later use in a more general framework where  $W$  relates to the updated weights of the descent algorithm. We will next derive the bounds that are needed.<sup>10</sup>

<sup>8</sup>The Chi-squared mutual information should normalize this expression with respect to the variance of  $W$  for non equiprobable random variables.

<sup>9</sup>See for example [AB18] for details on these inequalities.

<sup>10</sup>These bounds could be slightly tightened but are largely sufficient for our purpose.

**Lemma 1.**

$$\text{Var } \mathbb{E}(g(X, Y)|F) \leq \min_{i \in \{0,1\}} \mathbb{P}\{g(\tilde{X}, \tilde{Y}) = i\} \sqrt{\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}})} \quad (19)$$

*Proof.* Note that

$$\text{Var } \mathbb{E}(W|F) = \mathbb{E}_F(\mathbb{P}\{W = 1|F\} - \mathbb{P}\{W = 1\})^2 \quad (20)$$

$$\leq \mathbb{E}_F(\mathbb{P}\{W = 1|F\} - c)^2 \quad (21)$$

for any  $c \in \mathbb{R}$ . Moreover,

$$\mathbb{P}\{W = 1|F = f\} = \sum_x \mathbb{P}\{W = 1|F = f, X = x\} P_{\mathcal{X}}(x) \quad (22)$$

$$= \sum_x \mathbb{P}\{W = 1|F = f, X = x, Y = f(x)\} P_{\mathcal{X}}(x) \quad (23)$$

$$= \sum_{x,y} \mathbb{P}\{W = 1|X = x, Y = y\} P_{\mathcal{X}}(x) \mathbb{1}(f(x) = y). \quad (24)$$

Pick now

$$c := \sum_{x,y} \mathbb{P}\{W = 1|X = x, Y = y\} P_{\mathcal{X}}(x) U_{\mathcal{Y}}(y) \quad (25)$$

Therefore,

$$\mathbb{P}\{W = 1|F = f\} - c = \sum_{x,y} A_g(x, y) B_f(x, y) =: \langle A_g, B_f \rangle \quad (26)$$

where

$$A_g(x, y) := \mathbb{P}\{W = 1|X = x, Y = y\} \sqrt{P_{\mathcal{X}}(x) U_{\mathcal{Y}}(y)} \quad (27)$$

$$= \mathbb{P}\{g(X, Y) = 1|X = x, Y = y\} \sqrt{P_{\mathcal{X}}(x) U_{\mathcal{Y}}(y)} \quad (28)$$

$$B_f(x, y) := \frac{\mathbb{1}(f(x) = y) - U_{\mathcal{Y}}(y)}{U_{\mathcal{Y}}(y)} \sqrt{P_{\mathcal{X}}(x) U_{\mathcal{Y}}(y)}. \quad (29)$$

We have

$$\langle A_g, B_F \rangle^2 = \langle A_g, B_F \rangle \langle B_F, A_g \rangle = \langle A_g^{\otimes 2}, B_F^{\otimes 2} \rangle \quad (30)$$

and therefore

$$\mathbb{E}_F \langle A_g, B_F \rangle^2 = \langle A_g^{\otimes 2}, \mathbb{E}_F B_F^{\otimes 2} \rangle \quad (31)$$

$$\leq \|A_g^{\otimes 2}\|_2 \|\mathbb{E}_F B_F^{\otimes 2}\|_2. \quad (32)$$

Moreover,

$$\|A_g^{\otimes 2}\|_2 = \|A_g\|_2^2 \quad (33)$$

$$= \sum_{x,y} \mathbb{P}\{W = 1|X = x, Y = y\}^2 P_{\mathcal{X}}(x) U_{\mathcal{Y}}(y) \quad (34)$$

$$\leq \sum_{x,y} \mathbb{P}\{W = 1|X = x, Y = y\} P_{\mathcal{X}}(x) U_{\mathcal{Y}}(y) \quad (35)$$

$$= \mathbb{P}\{W(\tilde{X}, \tilde{Y}) = 1\} \quad (36)$$

and

$$\|\mathbb{E}_F B_F^{\otimes 2}\|_2 = \left( \sum_{x,y,x',y'} \left( \sum_f B_f(x,y) B_f(x',y') P_{\mathcal{F}}(f) \right)^2 \right)^{1/2} \quad (37)$$

$$= \left( \sum_{f,f'} \sum_{x,y} B_f(x,y) B_{f'}(x,y) \sum_{x',y'} B_f(x',y') B_{f'}(x',y') P_{\mathcal{F}}(f) P_{\mathcal{F}}(f') \right)^{1/2} \quad (38)$$

$$= (\mathbb{E}_{F,F'} \langle B_F, B_{F'} \rangle^2)^{1/2}. \quad (39)$$

Moreover,

$$\langle B_f, B_{f'} \rangle = \sum_{x,y} \frac{\mathbb{1}(f(x) = y) - U_{\mathcal{Y}}(y)}{U_{\mathcal{Y}}(y)} \frac{\mathbb{1}(f'(x) = y) - U_{\mathcal{Y}}(y)}{U_{\mathcal{Y}}(y)} P_{\mathcal{X}}(x) U_{\mathcal{Y}}(y) \quad (40)$$

$$= (1/2) \sum_{x,y} (2\mathbb{1}(f(x) = y) - 1)(2\mathbb{1}(f'(x) = y) - 1) P_{\mathcal{X}}(x) \quad (41)$$

$$= \mathbb{E}_X f(X) f'(X). \quad (42)$$

Therefore,

$$\text{Var } \mathbb{P}\{W = 1|F\} \leq \mathbb{P}\{\tilde{W} = 1\} \sqrt{\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}})}. \quad (43)$$

The same expansion holds with  $\text{Var } \mathbb{P}\{W = 1|F\} = \text{Var } \mathbb{P}\{W = 0|F\} \leq \mathbb{P}\{\tilde{W} = 0\} \sqrt{\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}})}$ .  $\square$

Consider now the new setup where  $g$  is valued in  $[m]$  instead of  $\{0, 1\}$ :

$$(X, F) \sim P_{\mathcal{X}} \times P_{\mathcal{F}} \quad (44)$$

$$Y = F(X) \quad (45)$$

$$W = g(X, Y) \text{ where } g : \mathbb{B}^n \times \mathcal{Y} \rightarrow [m]. \quad (46)$$

We have the following theorem.

**Theorem 5.**

$$\|P_{W,F} - P_W P_F\|_2^2 \leq \|P_{\mathcal{F}}\|_{\infty} \sqrt{\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}})}. \quad (47)$$

*Proof.* From Lemma 1, for any  $i \in [m]$ ,

$$\text{Var } \mathbb{P}\{W = i|F\} \leq \mathbb{P}\{g(\tilde{X}, \tilde{Y}) = i\} \sqrt{\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}})}, \quad (48)$$

therefore,

$$\|P_{W,F} - P_W P_F\|_2^2 = \sum_{i \in [m]} \sum_{f \in F} \mathbb{P}\{F = f\}^2 (\mathbb{P}\{W = i|F = f\} - \mathbb{P}\{W = i\})^2 \quad (49)$$

$$\leq \max_{f \in \mathcal{F}} \mathbb{P}\{F = f\} \sum_{i \in [m]} \sum_{f \in F} \mathbb{P}\{F = f\} (\mathbb{P}\{W = i|F = f\} - \mathbb{P}\{W = i\})^2 \quad (50)$$

$$\leq \max_{f \in \mathcal{F}} \mathbb{P}\{F = f\} \sum_{i \in [m]} \mathbb{P}\{g(\tilde{X}, \tilde{Y}) = i\} \sqrt{\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}})} \quad (51)$$

$$= \max_{f \in \mathcal{F}} \mathbb{P}\{F = f\} \sqrt{\text{Pred}(P_{\mathcal{F}}|P_{\mathcal{X}})}. \quad (52)$$

$\square$

We next specialize the bound in Theorem 5 to the case of uniform parity functions on uniform inputs, adding a bound on the  $L_1$  norm due to Cauchy-Schwarz.

**Corollary 2.** *Let  $m, n > 0$ . If we consider the setup of (44), (45), (46) for the case where  $P_{\mathcal{F}} = P_n$ , the uniform probability measure on parity functions, and  $P_{\mathcal{X}} = U_n$ , the uniform probability measure on  $\mathbb{B}^n$ , then*

$$\|P_{W,F} - P_W P_F\|_2^2 \leq 2^{-(3/2)n}, \quad (53)$$

$$\|P_{W,F} - P_W P_F\|_1 \leq \sqrt{m} 2^{-n/4}. \quad (54)$$

In short, the value of  $W$  will not provide significant amounts of information on  $F$  unless its number of possible values  $m$  is exponentially large.

**Corollary 3.** *Consider the same setup as in previous corollary, with in addition  $(\tilde{X}, \tilde{Y})$  independent of  $(X, F)$  such that  $(\tilde{X}, \tilde{Y}) \sim P_{\mathcal{X}} \times U_{\mathcal{Y}}$  where  $U_{\mathcal{Y}}$  is the uniform distribution on  $\mathcal{Y}$ , and  $\tilde{W} = g(\tilde{X}, \tilde{Y})$ . Then,*

$$\sum_{i \in [m]} \sum_{s \subseteq [n]} (P[W = i | f = p_s] - P[\tilde{W} = i])^2 \leq 2^{n/2}.$$

*Proof.* In the case where  $P_{\mathcal{F}} = P_n$ , taking the previous corollary and multiplying both sides by  $2^{2n}$  yields

$$\sum_{i \in [m]} \sum_{s \subseteq [n]} (P[W = i | f = p_s] - P[W = i])^2 \leq 2^{n/2}.$$

Furthermore, the probability distribution of  $(X, Y)$  and the probability distribution of  $(\tilde{X}, \tilde{Y})$  are both  $U_{n+1}$  so  $P[\tilde{W} = i] = P[W = i]$  for all  $i$ . Thus,

$$\sum_{i \in [m]} \sum_{s \subseteq [n]} (P[W = i | f = p_s] - P[\tilde{W} = i])^2 \leq 2^{n/2}. \quad (55)$$

□

Notice that for fixed values of  $P_{\mathcal{X}}$  and  $g$ , changing the value of  $P_{\mathcal{F}}$  does not change the value of  $P[W = i | f = p_s]$  for any  $i$  and  $s$ . Therefore, inequality (55) holds for any choice of  $P_{\mathcal{F}}$ , and we also have the following.

**Corollary 4.** *Consider the general setup of (44), (45), (46) with  $P_{\mathcal{X}} = U_n$ , and  $(\tilde{X}, \tilde{Y})$  independent of  $(X, F)$  such that  $(\tilde{X}, \tilde{Y}) \sim P_{\mathcal{X}} \times U_{\mathcal{Y}}$ ,  $\tilde{W} = g(\tilde{X}, \tilde{Y})$ . Then,*

$$\sum_{i \in [m]} \sum_{s \subseteq [n]} (P[W = i | f = p_s] - P[\tilde{W} = i])^2 \leq 2^{n/2}.$$

### 5.3 Sequential learning algorithm

Next, we would like to analyze the effectiveness of an algorithm that repeatedly receives an ordered pair,  $(X, F(X))$ , records some amount of information about that pair, and then forgets it. To formalize this concept, we define the following.

**Definition 9.** *A sequential learning algorithm  $A$  on  $(\mathcal{Z}, \mathcal{W})$  is an algorithm that for an input of the form  $(Z, (W_1, \dots, W_{t-1}))$  in  $\mathcal{Z} \times \mathcal{W}^{t-1}$  produces an output  $A(Z, (W_1, \dots, W_{t-1}))$  valued in  $\mathcal{W}$ . Given a probability distribution  $D$  on  $\mathcal{Z}$ , a sequential learning algorithm  $A$  on  $(\mathcal{Z}, \mathcal{W})$ , and  $T \geq 1$ , a  $T$ -trace of  $A$  for  $D$  is a series of pairs  $((Z_1, W_1), \dots, (Z_T, W_T))$  such that for each  $i \in [T]$ ,  $Z_i \sim D$  independently of  $(Z_1, Z_2, \dots, Z_{i-1})$  and  $W_i = A(Z_i, (W_1, W_2, \dots, W_{i-1}))$ .*

If  $|\mathcal{W}|$  is sufficiently small relative to  $2^n$ , then a sequential learning algorithm that outputs elements of  $\mathcal{W}$  will be unable to effectively distinguish between a random element of  $P_n$  and a true random function in the following sense.

**Theorem 6.** *Let  $n > 0$ , and  $A$  be a sequential learning algorithm on  $(\mathbb{B}^{n+1}, \mathcal{W})$ . Let  $\star$  be the uniform distribution on  $\mathbb{B}^{n+1}$ , and for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen uniformly at random in  $\mathbb{B}^n$ . Next, let  $P_Z$  be a probability distribution on  $\mathbb{B}^{n+1}$  that is chosen by means of the following procedure: with probability  $1/2$ , set  $P_Z = \star$ , otherwise select  $S$  uniformly at random in  $[n]$  and set  $P_Z = \rho_S$ . If  $|\mathcal{W}| \leq 2^{n/24}$ ,  $m$  is a positive integer with  $m < 2^{n/24}$ , and  $((Z_1, W_1), \dots, (Z_m, W_m))$  is a  $m$ -trace of  $A$  for  $P_Z$ , then*

$$\|P_{W^m|P_Z=\star} - P_{W^m|P_Z \neq \star}\|_1 = O(2^{-n/24}). \quad (56)$$

*Proof.* Note that by the triangular inequality,

$$\begin{aligned} & \|P_{W^m|P_Z=\star} - P_{W^m|P_Z \neq \star}\|_1 \\ &= \sum_{w_1, \dots, w_m \in \mathcal{W}} |P[W^m = w^m | P_Z \neq \star] - P[W^m = w^m | P_Z = \star]| \\ &\leq 2^{-n} \sum_{s \subseteq [n]} \sum_{w^m \in \mathcal{W}^m} |P[W^m = w^m | P_Z = \rho_s] - P[W^m = w^m | P_Z = \star]| \end{aligned}$$

and we will bound the last term by  $O(2^{-n/24})$ .

We need to prove that  $P[W^m = w^m | P_Z = \rho_s] \approx P[W^m = w^m | P_Z = \star]$  most of the time. In order to do that, we will use the fact that

$$\frac{P[W^m = w^m | P_Z = \rho_s]}{P[W^m = w^m | P_Z = \star]} = \prod_{i=1}^m \frac{P[W_i = w_i | W^{i-1} = w^{i-1}, P_Z = \rho_s]}{P[W_i = w_i | W^{i-1} = w^{i-1}, P_Z = \star]}$$

So, as long as  $P[W_i = w_i | W^{i-1} = w^{i-1}, P_Z = \rho_s] \approx P[W_i = w_i | W^{i-1} = w^{i-1}, P_Z = \star]$  and  $P[W_i = w_i | W^{i-1} = w^{i-1}, P_Z = \star]$  is reasonably large for all  $i$ , this must hold for the values of  $w^m$  and  $s$  in question. As such, we plan to define a good value for  $(w^m, s)$  to be one for which this holds, and then prove that the set of good values has high probability measure.

First, call a sequence  $w^m \in \mathcal{W}^m$  *typical* if for each  $1 \leq i \leq m$ , we have that

$$t(w^i) := P[W_i = w_i | W^{i-1} = w^{i-1}, P_Z = \star] \geq 2^{-n/8},$$

and denote by  $\mathcal{T}$  the set of typical sequences

$$\mathcal{T} := \{w^m : \forall i \in [m], t(w^i) \geq 2^{-n/8}\}. \quad (57)$$

We have

$$1 = \mathbb{P}\{W^m \in \mathcal{T} | P_Z = \star\} + \mathbb{P}\{W^m \notin \mathcal{T} | P_Z = \star\} \quad (58)$$

$$= \mathbb{P}\{W^m \in \mathcal{T} | P_Z = \star\} + \mathbb{P}\{\exists i \in [m] : t(W^i) < 2^{-n/8} | P_Z = \star\} \quad (59)$$

$$\leq \mathbb{P}\{W^m \in \mathcal{T} | P_Z = \star\} + \sum_{i=1}^m \mathbb{P}\{t(W^i) < 2^{-n/8} | P_Z = \star\} \quad (60)$$

$$\leq \mathbb{P}\{W^m \in \mathcal{T} | P_Z = \star\} + m2^{-n/8}|\mathcal{W}|. \quad (61)$$

Thus

$$\mathbb{P}\{W^m \in \mathcal{T} | P_{\mathcal{Z}} = \star\} \geq 1 - m2^{-n/8} |\mathcal{W}| \geq 1 - 2^{-n/24}. \quad (62)$$

Next, call an ordered pair of a sequence  $w^m \in \mathcal{W}^m$  and an  $s \subseteq [n]$  *good* if  $w^m$  is typical and

$$\left| \frac{P[W_i = w_i | W^{i-1} = w^{i-1}, P_{\mathcal{Z}} = \rho_s]}{P[W_i = w_i | W^{i-1} = w^{i-1}, P_{\mathcal{Z}} = \star]} - 1 \right| \leq 2^{-n/12}, \quad \forall i \in [m], \quad (63)$$

and denote by  $\mathcal{G}$  the set of good pairs. A pair which is not good is called bad.

Note that for any  $i$  and any  $w_1, \dots, w_{i-1} \in \mathcal{W}$ , there exists a function  $g_{w_1, \dots, w_{i-1}}$  such that  $W_i = g_{w_1, \dots, w_{i-1}}(Z_i)$ . So, corollary 3 implies that

$$\sum_{w_i \in \mathcal{W}} \sum_{s \subseteq [n]} (P[W_i = w_i | W^{i-1} = w^{i-1}, P_{\mathcal{Z}} = \rho_s] - P[W_i = w_i | W^{i-1} = w^{i-1}, P_{\mathcal{Z}} = \star])^2 \quad (64)$$

$$= \sum_{w_i \in \mathcal{W}} \sum_{s \subseteq [n]} (P[g_{w_1, \dots, w_{i-1}}(Z_i) = w_i | P_{\mathcal{Z}} = \rho_s] - P[g_{w_1, \dots, w_{i-1}}(Z_i) = w_i | P_{\mathcal{Z}} = \star])^2 \quad (65)$$

$$\leq 2^{n/2} \quad (66)$$

Also, given any  $w^m$  and  $s \subseteq [n]$  such that  $w^m$  is typical but  $w^m$  and  $s$  are not good, there must exist  $1 \leq i \leq m$  such that

$$r(w^i, s) := |P[W_i = w_i | W^{i-1} = w^{i-1}, P_{\mathcal{Z}} = \rho_s] - P[W_i = w_i | W^{i-1} = w^{i-1}, P_{\mathcal{Z}} = \star]| \quad (67)$$

$$\geq 2^{-5n/24}. \quad (68)$$

Thus, for  $w^m \in \mathcal{T}$

$$\sum_{s \subseteq [n]: (w^m, s) \notin \mathcal{G}} 2^{-n} = \mathbb{P}\{(w^m, S) \notin \mathcal{G}\} \quad (69)$$

$$\leq \mathbb{P}\{\exists i \in [m] : r(w^i, S) \geq 2^{-5n/24}\} \quad (70)$$

$$\leq \sum_{i=1}^m \mathbb{P}\{r(w^i, S) \geq 2^{-5n/24}\} \quad (71)$$

$$\leq \sum_{i=1}^m \sum_{s \subseteq [n]: r(w^i, s) \geq 2^{-5n/24}} 2^{-n} \quad (72)$$

$$= 2^{-n} \sum_{i=1}^m \sum_{s \subseteq [n]: r(w^i, s)^2 (2^{5n/24})^2 \geq 1} 1 \quad (73)$$

$$= 2^{-n} (2^{5n/24})^2 \sum_{i=1}^m \sum_{s \subseteq [n]} r(w^i, s)^2 \quad (74)$$

$$\leq 2^{-n} (2^{5n/24})^2 \sum_{i=1}^m \sum_{w'_i \in \mathcal{W}} \sum_{s \subseteq [n]} r((w'_i, w^{i-1}), s)^2 \quad (75)$$

$$\leq 2^{-n} (2^{5n/24})^2 m 2^{n/2} \quad (76)$$

$$(77)$$

This means that for a given typical  $w^m$  there are at most  $m(2^{5n/24})^2 2^{n/2}$  possible  $s \subseteq [n]$  for which  $w^m$  and  $s$  are not good.

Therefore, if  $P_Z = \star$  and  $S$  is a random subset of  $[n]$ , the probability that  $W^m$  is typical but  $W^m$  and  $S$  is not good is at most  $m(2^{5n/24})^2 2^{n/2} / 2^n \leq 2^{-n/24}$ ; in fact:

$$\mathbb{P}\{W^m \in \mathcal{T}, (W^m, S) \notin \mathcal{G} | P_Z = \star\} \quad (78)$$

$$= 2^{-n} \sum_{s \subseteq [n], w^m \in \mathcal{T}: (w^m, s) \notin \mathcal{G}} \mathbb{P}\{W^m = w^m | P_Z = \star\} \quad (79)$$

$$= 2^{-n} \sum_{w^m \in \mathcal{T}} \mathbb{P}\{W^m = w^m | P_Z = \star\} \sum_{s \subseteq [n]: (w^m, s) \notin \mathcal{G}} 1 \quad (80)$$

$$\leq 2^{-n} m(2^{5n/24})^2 2^{n/2} \sum_{w^m \in \mathcal{T}} \mathbb{P}\{W^m = w^m | P_Z = \star\} \quad (81)$$

$$\leq 2^{-n} m(2^{5n/24})^2 2^{n/2} \leq 2^{-n/24}. \quad (82)$$

We already knew that  $W^m$  is typical with probability  $1 - 2^{-n/24}$  under these circumstances, so  $W^m$  and  $S$  is good with probability at least  $1 - 2 \cdot 2^{-n/24}$  since

$$1 - 2^{-n/24} \leq \mathbb{P}\{W^m \in \mathcal{T} | P_Z = \star\} \quad (83)$$

$$= \mathbb{P}\{W^m \in \mathcal{T}, (W^m, S) \in \mathcal{G} | P_Z = \star\} + \mathbb{P}\{W^m \in \mathcal{T}, (W^m, S) \notin \mathcal{G} | P_Z = \star\} \quad (84)$$

$$\leq \mathbb{P}\{(W^m, S) \in \mathcal{G} | P_Z = \star\} + 2^{-n/24}. \quad (85)$$

Next, recall that

$$\frac{P[W^m = w^m | P_Z = \rho_s]}{P[W^m = w^m | P_Z = \star]} = \prod_{i=1}^m \frac{P[W_i = w_i | W^{i-1} = w^{i-1}, P_Z = \rho_s]}{P[W_i = w_i | W^{i-1} = w^{i-1}, P_Z = \star]}$$

So, if  $w^m$  and  $s$  is good (and each term in the above product is close to 1 by  $2^{-n/12}$ ), we have

$$\left| \frac{P[W^m = w^m | P_Z = \rho_s]}{P[W^m = w^m | P_Z = \star]} - 1 \right| \leq e^{m2^{-n/12}} - 1 \leq (1 + o(1))2^{-n/24}. \quad (86)$$

That implies that

$$\begin{aligned} & 2^{-n} \sum_{(w^m, s) \in \mathcal{G}} |P[W^m = w^m | P_Z = \rho_s] - P[W^m = w^m | P_Z = \star]| \\ & \leq 2^{-n} \sum_{(w^m, s) \in \mathcal{G}} (1 + o(1))2^{-n/24} \cdot P[W^m = w^m | P_Z = \star] \\ & \leq \sum_{w^m} (1 + o(1))2^{-n/24} \cdot P[W^m = w^m | P_Z = \star] \\ & \leq (1 + o(1))2^{-n/24}. \end{aligned}$$

Also,

$$\begin{aligned}
& 2^{-n} \sum_{(w^m, s) \notin \mathcal{G}} (P[W^m = w^m | P_{\mathcal{Z}} = \rho_s] - P[W^m = w^m | P_{\mathcal{Z}} = \star]) \\
&= P[(W^m, S) \notin \mathcal{G} | P_{\mathcal{Z}} \neq \star] - P[(W^m, S) \notin \mathcal{G} | P_{\mathcal{Z}} = \star] \\
&= P[(W^m, S) \in \mathcal{G} | P_{\mathcal{Z}} = \star] - P[(W^m, S) \in \mathcal{G} | P_{\mathcal{Z}} \neq \star] \\
&= 2^{-n} \sum_{(w^m, s) \in \mathcal{G}} (P[W^m = w^m | P_{\mathcal{Z}} = \star] - P[W^m = w^m | P_{\mathcal{Z}} = \rho_s]) \\
&\leq 2^{-n} \sum_{(w^m, s) \in \mathcal{G}} |P[W^m = w^m | P_{\mathcal{Z}} = \star] - P[W^m = w^m | P_{\mathcal{Z}} = \rho_s]| \\
&\leq (1 + o(1))2^{-n/24}.
\end{aligned}$$

That means that

$$\begin{aligned}
& 2^{-n} \sum_{(w^m, s) \notin \mathcal{G}} |P[W^m = w^m | P_{\mathcal{Z}} = \rho_s] - P[W^m = w^m | P_{\mathcal{Z}} = \star]| \\
&\leq 2^{-n} \sum_{(w^m, s) \notin \mathcal{G}} (P[W^m = w^m | P_{\mathcal{Z}} = \rho_s] + P[W^m = w^m | P_{\mathcal{Z}} = \star]) \\
&= 2^{-n} \sum_{(w^m, s) \notin \mathcal{G}} 2P[W^m = w^m | P_{\mathcal{Z}} = \star] \\
&\quad + 2^{-n} \sum_{(w^m, s) \notin \mathcal{G}} (P[W^m = w^m | P_{\mathcal{Z}} = \rho_s] - P[W^m = w^m | P_{\mathcal{Z}} = \star]) \\
&\leq 4 \cdot 2^{-n/24} + (1 + o(1))2^{-n/24}.
\end{aligned}$$

Therefore,

$$2^{-n} \sum_{s \subseteq [n]} \sum_{w^m \in \mathcal{W}} |P[W^m = w^m | P_{\mathcal{Z}} = \rho_s] - P[W^m = w^m | P_{\mathcal{Z}} = \star]| = O(2^{-n/24}), \quad (87)$$

which gives the desired bound.  $\square$

**Corollary 5.** *Consider a data structure with a polynomial amount of memory that is divided into variables that are each  $O(\log n)$  bits long, and define  $m$ ,  $\mathcal{Z}$ ,  $\star$ , and  $P_{\mathcal{Z}}$  the same way as in Theorem 6. Also, let  $A$  be an algorithm that takes the data structure's current value and an element of  $\mathbb{B}^{n+1}$  as inputs and changes the values of at most  $o(n/\log(n))$  of the variables. If we draw  $Z_1, \dots, Z_m$  independently from  $P_{\mathcal{Z}}$  and then run the algorithm on each of them in sequence, then no matter how the data structure is initialized, it is impossible to determine whether or not  $P_{\mathcal{Z}} = \star$  from the data structure's final value with accuracy greater than  $1/2 + O(2^{-n/24})$ .*

*Proof.* Let  $W_0$  be the initial state of the data structure's memory, and let  $W_i = A(W_{i-1}, Z_i)$  for each  $0 < i \leq m$ . Next, for each such  $i$ , let  $W'_i$  be the list of all variables that have different values in  $W_i$  than in  $W_{i-1}$ , and their values in  $W_i$ . There are only polynomially many variables in memory, so it takes  $O(\log(n))$  bits to specify one and  $O(\log(n))$  bits to specify a value for that variable.  $A$  only changes the values of  $o(n/\log(n))$  variables at each timestep, so  $W'_i$  will only ever list  $o(n/\log(n))$  variables. That means that  $W'_i$  can be specified with  $o(n)$  bits, and in particular that there exists some set  $\mathcal{W}$  such that  $W'_i$  will always be in  $\mathcal{W}$  and  $|\mathcal{W}| = 2^{o(n)}$ . Also, note that we can determine



the value of  $W_i$  from the values of  $W_{i-1}$  and  $W'_i$ , so we can reconstruct the value of  $W_i$  from the values of  $W'_1, W'_2, \dots, W'_i$ .

Now, let  $A'$  be the algorithm that takes  $(Z_t, (W'_1, \dots, W'_{t-1}))$  as input and does the following. First, it reconstructs  $W_{t-1}$  from  $(W'_1, \dots, W'_{t-1})$ . Then, it computes  $W_t$  by running  $A$  on  $W_{t-1}$  and  $Z_t$ . Finally, it determines the value of  $W'_t$  by comparing  $W_t$  to  $W_{t-1}$  and returns it. This is an SLA, and  $((Z_1, W'_1), \dots, (Z_m, W'_m))$  is an  $m$ -trace of  $A'$  for  $P_Z$ . So, by the theorem

$$\sum_{w_1, \dots, w_m} |P[W'_1 = w_1, \dots, W'_m = w_m | P_Z \neq \star] - P[W'_1 = w_1, \dots, W'_m = w_m | P_Z = \star]| \quad (88)$$

$$= O(2^{-n/24}) \quad (89)$$

Furthermore, since  $W_m$  can be reconstructed from  $(W'_1, \dots, W'_m)$ , this implies that

$$\sum_w |P[W_m = w | P_Z \neq \star] - P[W_m = w | P_Z = \star]| = O(2^{-n/24}). \quad (90)$$

Finally, the probability of deciding correctly between the hypothesis  $P_Z = \star$  and  $P_Z \neq \star$  given the observation  $W_m$  is at most

$$1 - \frac{1}{2} \sum_{w \in \mathcal{W}} P[W_m = w | P_Z = \star] \wedge P[W_m = w | P_Z \neq \star] \quad (91)$$

$$= \frac{1}{2} + \frac{1}{4} \sum_{w \in \mathcal{W}} |P[W_m = w | P_Z \neq \star] - P[W_m = w | P_Z = \star]| \quad (92)$$

$$= \frac{1}{2} + O(2^{-n/24}), \quad (93)$$

which implies the conclusion. □

**Remark 9.** *The theorem and its second corollary state that the algorithm can not determine whether or not  $P_Z = \star$ . However, one could easily transform them into results showing that the algorithm can not effectively learn to compute  $p_s$ . More precisely, after running on  $2^{n/24-1}$  pairs  $(x, p_s(x))$ , the algorithm will not be able to compute  $p_s(x)$  with accuracy  $1/2 + \omega(2^{-n/48})$  with a probability of  $\omega(2^{-n/24})$ . If it could, then we could just train it on the first  $m/2$  of the  $Z_i$  and count how many of the next  $m/2$   $Z_i$  it predicts the last bit of correctly. If  $P_Z = \star$ , each of those predictions will be independently correct with probability  $1/2$ , so the total number it is right on will differ from  $m/4$  by  $O(\sqrt{m})$  with high probability. However, if  $P_Z = \rho_s$  and the algorithm learns to compute  $\rho_s$  with accuracy  $1/2 + \omega(2^{-n/48})$ , then it will predict  $m/4 + \omega(\sqrt{m})$  of the last  $m/2$  correctly with high probability. So, we could determine whether or not  $P_Z = \star$  with greater accuracy than the theorem allows by tracking the accuracy of the algorithm's predictions.*

## 6 Deep learning and parity

### 6.1 Neural nets and deep learning

Before we can talk about the effectiveness of deep learning at learning these parity functions, we will have to establish some basic information about deep learning. First of all, in this paper we will be using the following definition for a neural net.

**Definition 10.** A neural net is a pair of a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  and a weighted directed graph  $G$  with some special vertices and the following properties. First of all,  $G$  does not contain any cycles. Secondly, there exists  $n > 0$  such that  $G$  has exactly  $n + 1$  vertices that have no edges ending at them,  $v_0, v_1, \dots, v_n$ . We will refer to  $n$  as the input size,  $v_0$  as the constant vertex and  $v_1, v_2, \dots, v_n$  as the input vertices. Finally, there exists a vertex  $v_{out}$  such that for any other vertex  $v'$ , there is a path from  $v'$  to  $v_{out}$  in  $G$ . We also use denote by  $w(G)$  the weights on the edges of  $G$ .

**Definition 11.** Given a neural net  $(f, G)$  with input size  $n$ , and  $x \in \mathbb{R}^n$ , the evaluation of  $(f, G)$  at  $x$ , is the number computed by means of the following procedure.

1. Define  $y \in \mathbb{R}^{|G|}$  where  $|G|$  is the number of vertices in  $G$ , set  $y_{v_0} = 1$ , and set  $y_{v_i} = x_i$  for each  $i$ .
2. Find an ordering  $v'_1, \dots, v'_m$  of the vertices in  $G$  other than the constant vertex and input vertices such that for all  $j > i$ , there is not an edge from  $v'_j$  to  $v'_i$ .
3. For each  $1 \leq i \leq m$ , set

$$y_{v'_i} = f \left( \sum_{v: (v, v'_i) \in E(G)} w_{v, v'_i} y_v \right)$$

4. Return  $y_{v_{out}}$ .

We will write the evaluation of  $(f, G)$  at  $x$  as  $eval_{(f, G)}(x)$ .

Generally, we want to find a neural net that computes a certain function, or at least a good approximation of that function. A reasonable approach to doing that is to start with some neural network and then attempt to adjust its weights until it computes a reasonable approximation of the desired function. A common way to do that is to define a loss function in terms of how much the network's outputs differ from the desired outputs, and then use gradient descent to try to adjust the weights. Of course, that may not be well defined if  $f$  is not differentiable, it may not be possible to find weights for which the network approximates the desired function if it has too few vertices or is missing some key edges, and the gradient descent algorithm has an increased risk of getting stuck in a local minimum if  $f$  has local minima. Allowing the loss function or the derivative of  $f$  to take on arbitrarily large values under some circumstances can also cause problems, which can be mitigated by redefining our function to have input in  $[0, 1]^n$  and output in  $[0, 1]$  and then using an  $f$  with output in  $[0, 1]$ . We would also like to ensure that  $f$  can take on values arbitrarily close to any desired output in that range. As such, we give the following criteria for a neural net to be considered well behaved.

**Definition 12.** Let  $(f, G)$  be a neural net. Then  $(f, G)$  is normal if it satisfies the following properties.  $f$  must be a smooth function, the derivative of  $f$  must be positive everywhere, the derivative of  $f$  must be bounded, it must be the case that  $\lim_{x \rightarrow -\infty} f(x) = 0$  and  $\lim_{x \rightarrow \infty} f(x) = 1$ , and  $G$  must have an edge from the constant vertex to every other vertex except the input vertices.

More formally, given a target function  $h$ , a probability distribution  $P_{\mathcal{X}}$  for the input of  $h$ , and a loss function  $L : \mathbb{R} \rightarrow \mathbb{R}$ , we would like to find a neural net  $(f, G)$  that has a small value of

$$E_{X \sim P_{\mathcal{X}}} [L(eval_{(f, G)}(X) - h(X))]$$

In order to do that, we could try starting with some neural net  $(f, G_0)$ , and then using the following algorithm to assign new weights to the graph's edges.

*GradientDescentStep*( $f, G, h, P_{\mathcal{X}}, L, \gamma, B$ ):

1. For each  $(v, v') \in E(G)$ :

- (a) Set

$$w'_{v,v'} = w_{v,v'} - \gamma \frac{\partial E_{X \sim P_{\mathcal{X}}} [L(\text{eval}_{(f,G)}(X) - h(X))]}{\partial w_{v,v'}}$$

- (b) If  $w'_{v,v'} < -B$ , set  $w'_{v,v'} = -B$ .

- (c) If  $w'_{v,v'} > B$ , set  $w'_{v,v'} = B$ .

2. Return the graph that is identical to  $G$  except that its edge weights are given by the  $w'$ .

*GradientDescentAlgorithm*( $f, G, h, P_{\mathcal{X}}, L, \gamma, B, t$ ):

1. Set  $G_0 = G$ .
2. If any of the edge weights in  $G_0$  are less than  $-B$ , set all such weights to  $-B$ .
3. If any of the edge weights in  $G_0$  are greater than  $B$ , set all such weights to  $B$ .
4. For each  $0 \leq i < t$ , set  $G_{i+1} = \text{GradientDescentStep}(f, G_i, h, P_{\mathcal{X}}, L, \gamma, B)$ .
5. Return  $G_t$ .

The hope is that if we set  $G' = \text{GradientDescentAlgorithm}(f, G, h, P_{\mathcal{X}}, L, \gamma, B, t)$  for a small enough  $\gamma$  and large enough  $t$  then  $\text{eval}_{(f,G')}$  will be a good approximation of  $h$ . Of course, actually running this algorithm requires us to compute  $\text{eval}_{(f,G)}(X)$  for every possible value of  $X$  in every step, which is generally impractical. As a result, we are more likely to pick a single value of  $X$  at each step, and adjust the net to give a better output on that input. More formally, we would use the following algorithm.

*SampleGradientDescentStep*( $f, G, Y, X, L, \gamma, B$ ):

1. For each  $(v, v') \in E(G)$ :

- (a) Set

$$w'_{v,v'} = w_{v,v'} - \gamma \frac{\partial L(\text{eval}_{(f,G)}(X) - Y)}{\partial w_{v,v'}}$$

- (b) If  $w'_{v,v'} < -B$ , set  $w'_{v,v'} = -B$ .

- (c) If  $w'_{v,v'} > B$ , set  $w'_{v,v'} = B$ .

2. Return the graph that is identical to  $G$  except that its edge weights are given by the  $w'$ .

*StochasticGradientDescentAlgorithm*( $f, G, P_{\mathcal{Z}}, L, \gamma, B, t$ ):

1. Set  $G_0 = G$ .

2. If any of the edge weights in  $G_0$  are less than  $-B$ , set all such weights to  $-B$ .
3. If any of the edge weights in  $G_0$  are greater than  $B$ , set all such weights to  $B$ .
4. For each  $0 \leq i < t$ :
  - (a) Draw  $(X_i, Y_i) \sim P_Z$ , independently of all previous values.
  - (b) Set  $G_{i+1} = \text{SampleGradientDescentStep}(f, G_i, Y_i, X_i, L, \gamma, B)$
5. Return  $G_t$ .

## 6.2 Proof of Theorem 1

One possible variant of this is to only adjust a few weights at each time step, such as the  $k$  that would change the most or a random subset. However, any such algorithm cannot learn a random parity function in the following sense.

**Theorem 7.** *Let  $n > 0$ ,  $k = o(n/\log(n))$ , and  $(f, g)$  be a neural net of size polynomial in  $n$  in which each edge weight is recorded using  $O(\log n)$  bits. Also, let  $\star$  be the uniform distribution on  $\mathbb{B}^{n+1}$ , and for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen randomly from  $\mathbb{B}^n$ . Next, let  $P_Z$  be a probability distribution on  $\mathbb{B}^{n+1}$  that is chosen by means of the following procedure. First, with probability  $1/2$ , set  $P_Z = \star$ . Otherwise, select a random  $S \subseteq [n]$  and set  $P_Z = \rho_S$ . Then, let  $A$  be an algorithm that draws a random element from  $P_Z$  in each time step and changes at most  $k$  of the weights of  $g$  in response to the sample and its current values. If  $A$  is run for less than  $2^{n/24}$  time steps, then it is impossible to determine whether or not  $P_Z = \star$  from the resulting neural net with accuracy greater than  $1/2 + O(2^{-n/24})$ .*

*Proof.* This follows immediately from corollary 5. □

**Remark 10.** *The theorem state that one cannot determine whether or not  $P_Z = \star$  from the final network. However, if we used a variant of corollary 5 we could get a result showing that the final network will not compute  $p_s$  accurately. More precisely, after training the network on  $2^{n/24-1}$  pairs  $(x, p_s(x))$ , the network will not be able to compute  $p_s(x)$  with accuracy  $1/2 + \omega(2^{-n/48})$  with a probability of  $\omega(2^{-n/24})$ .*

We can also use this reasoning to prove theorem 1, which is restated below.

**Theorem 1.** *Let  $\epsilon > 0$ , and for each  $n > 0$ , let  $(f, g)$  be a neural net of size polynomial in  $n$  in which each edge weight is recorded using  $O(\log n)$  bits of memory. Then training  $(f, g)$  with any coordinate descent algorithm that runs for less than  $2^{n/24}$  time steps and updates  $o(n/\log(n))$  edge weights per time step fails to learn a random parity function with accuracy  $1/2 + \epsilon$  for all sufficiently large  $n$ .*

*Proof.* Consider a data structure that consists of a neural net  $(f, g')$  and a boolean value  $b$ . Now, consider training  $(f, g)$  with any such coordinate descent algorithm while using the data structure to store the current value of the net. Also, in each time step, set  $b$  to *True* if the net computed the output corresponding to the sampled input correctly and *False* otherwise. This constitutes a data structure with a polynomial amount of memory that is divided into variables that are  $O(\log n)$  bits long, such that  $o(n/\log(n))$  variables change value in each time step. As such, by corollary 5, one cannot determine whether the samples are actually generated by a random parity function or whether they are simply random elements of  $\mathbb{B}^{n+1}$  from the data structure's final value with

accuracy  $1/2 + \omega(2^{-n/24})$ . In particular, one cannot determine which case holds from the final value of  $b$ . If the samples were generated randomly, it would compute the final output correctly with probability  $1/2$ , so  $b$  would be equally likely to be *True* or *False*. So, when it is trained on a random parity function, the probability that  $b$  ends up being *True* must be at most  $1/2 + O(2^{-n/24})$ . Therefore, it must compute the final output correctly with probability  $1/2 + O(2^{-n/24})$ .  $\square$

### 6.3 Proof of Theorem 2

Before we talk about the effectiveness of noisy gradient descent, we need to formally define it. So, we define the following.

*NoisyGradientDescentStep*( $f, G, P_Z, L, \gamma, \delta$ ):

1. For each  $(v, v') \in E(G)$ , set

$$w'_{v,v'} = w_{v,v'} - \gamma \frac{\partial E_{(X,Y) \sim P_Z} [L(\text{eval}_{(f,G)}(X) - Y)]}{\partial w_{v,v'}} + \delta \quad (94)$$

2. Return the graph that is identical to  $G$  except that its edge weights are given by  $w'$ .

*NoisyGradientDescentAlgorithm*( $f, G, P_Z, L, \gamma, \Delta, t$ ):

1. Set  $G_0 = G$ .
2. For each  $0 \leq i < t$ :
  - (a) Generate  $\delta^{(i)}$  by independently drawing  $\delta^{(i)}_{v,v'}$  from  $\Delta$  for each  $(v, v') \in E(G)$ .
  - (b) Set  $G_{i+1} = \text{NoisyGradientDescentStep}(f, G_i, h, P_Z, L, \gamma, \delta^{(i)})$ .
3. Return  $G_t$ .

The case where one of the derivatives is very large causes enough problems that it is convenient to define versions of the algorithms that treat the derivative of the loss function with respect to a given edge weight on a given input as having some maximum value if it is actually larger. Then we can prove results for these algorithms, and argue that if none of the derivatives are that large using the normal versions must yield the same result. As such, we define the following:

**Definition 13.** For every  $B > 0$  and  $x \in \mathbb{R}$ , let  $\Psi_B(x)$  be  $B$  if  $x > B$ ,  $-B$  if  $x < -B$ , and  $x$  otherwise.

*BoundedNoisyGradientDescentStep*( $f, G, P_Z, L, \gamma, \delta, B$ ):

1. For each  $(v, v') \in E(G)$ , set

$$w'_{v,v'} = w_{v,v'} - \gamma E_{(X,Y) \sim P_Z} \left[ \Psi_B \left( \frac{\partial L(\text{eval}_{(f,G)}(X) - Y)}{\partial w_{v,v'}} \right) \right] + \delta \quad (95)$$

2. Return the graph that is identical to  $G$  except that its edge weights are given by  $w'$ .

*BoundedNoisyGradientDescentAlgorithm*( $f, G, P_Z, L, \gamma, \Delta, B, t$ ):

1. Set  $G_0 = G$ .
2. For each  $0 \leq i < t$ :
  - (a) Generate  $\delta^{(i)}$  by independently drawing  $\delta_{v,v'}^{(i)}$  from  $\Delta$  for each  $(v, v') \in E(G)$ .
  - (b) Set  $G_{i+1} = \text{BoundedNoisyGradientDescentStep}(f, G_i, h, P_Z, L, \gamma, \delta^{(i)}, B)$ .
3. Return  $G_t$ .

In order to prove that noisy gradient descent fails to learn a random parity function, we first show that the gradient will be very small, and then argue that the noise will drown it out. The first steps are the following basic inequalities.

**Lemma 2.** *Let  $n > 0$  and  $f : \mathbb{B}^{n+1} \rightarrow \mathbb{R}$ . Also, let  $X$  be a random element of  $\mathbb{B}^n$  and  $Y$  be a random element of  $\mathbb{B}$  independent of  $X$ . Then*

$$\sum_{s \subseteq [n]} (\mathbb{E}f(X, Y) - \mathbb{E}f(X, p_s(X)))^2 \leq \mathbb{E}f^2(X, Y)$$

*Proof.* For each  $x \in \mathbb{B}^n$ , let  $g(x) = f(x, 1) - f(x, 0)$ .

$$\sum_{s \subseteq [n]} (\mathbb{E}[f(X, Y)] - \mathbb{E}[f(X, p_s(X))])^2 \tag{96}$$

$$= \sum_{s \subseteq [n]} \left( 2^{-n-1} \sum_{x \in \mathbb{B}^n} (f(x, 0) + f(x, 1) - 2f(x, p_s(x))) \right)^2 \tag{97}$$

$$= \sum_{s \subseteq [n]} \left( 2^{-n-1} \sum_{x \in \mathbb{B}^n} g(x)(-1)^{p_s(x)} \right)^2 \tag{98}$$

$$= 2^{-2n-2} \sum_{x_1, x_2 \in \mathbb{B}^n, s \subseteq [n]} g(x_1)(-1)^{p_s(x_1)} \cdot g(x_2)(-1)^{p_s(x_2)} \tag{99}$$

$$= 2^{-2n-2} \sum_{x_1, x_2 \in \mathbb{B}^n} g(x_1)g(x_2) \sum_{s \subseteq [n]} (-1)^{p_s(x_1)}(-1)^{p_s(x_2)} \tag{100}$$

$$= 2^{-2n-2} \sum_{x \in \mathbb{B}^n} 2^n g^2(x) \tag{101}$$

$$= 2^{-n-2} \sum_{x \in \mathbb{B}^n} [f(x, 1) - f(x, 0)]^2 \tag{102}$$

$$\leq 2^{-n-1} \sum_{x \in \mathbb{B}^n} f^2(x, 1) + f^2(x, 0) \tag{103}$$

$$= \mathbb{E}[f^2(X, Y)] \tag{104}$$

where we note that the equality from (98) to (101) is Parseval's identity for the Fourier-Walsh basis (here we used Boolean outputs for the parity functions).  $\square$

Note that by the triangular inequality the above implies

$$\text{Var}_F \mathbb{E}_X f(X, F(X)) \leq 2^{-n} \mathbb{E}_{X,Y} f^2(X, Y). \quad (105)$$

As mentioned earlier, this is similar to Theorem 1 in [SSS17] that requires in addition the function to be the gradient of a 1-Lipschitz loss function.

**Remark 11.** *While this result is stated for a random parity function, a similar result would hold for any other class of functions with sufficiently low cross predictability. To see this, consider computing the value of  $f$  on a random sample generated by a function drawn from the class. Then, let  $b$  be a bit that is set to 1 if the value of  $f$  is above a given threshold and 0 otherwise. By lemma 1, the probability that  $b$  is 1 must be nearly independent of which function was drawn from the class. So, the probability distribution of  $f$  must also be nearly independent of which function was drawn from the class.*

We then obtain the following corollary from Cauchy-Schwarz.

**Corollary 6.** *Let  $n > 0$  and  $f : \mathbb{B}^{n+1} \rightarrow \mathbb{R}$ . Also, let  $X$  be a random element of  $\mathbb{B}^n$  and  $Y$  be a random element of  $\mathbb{B}$  independent of  $X$ . Then*

$$\sum_{s \subseteq [n]} |E[f((X, Y))] - E[f((X, p_s(X)))]| \leq 2^{n/2} \sqrt{E[f^2((X, Y))]}.$$

In other words, the expected value of any function on an input generated by a random parity function is approximately the same as the expected value of the function on a true random input. This in turn implies the following:

**Lemma 3.** *Let  $(f, g)$  be a neural net with  $m$  edges,  $B, \gamma, \sigma > 0$ , and  $L : \mathbb{R} \rightarrow \mathbb{R}$  be a differentiable function. Also, let  $\star$  be the uniform distribution on  $\mathbb{B}^{n+1}$ , and for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen randomly from  $\mathbb{B}^n$ . Next, let  $Q_\star$  be the probability distribution of the output of  $\text{BoundedNoisyGradientDescentStep}(f, g, \star, L, \gamma, \delta, B)$  when  $\delta \sim \mathcal{N}(0, \sigma^2 I)$  and  $Q_s$  be the probability distribution of the output of  $\text{BoundedNoisyGradientDescentStep}(f, g, \rho_s, L, \gamma, \delta, B)$  when  $\delta \sim \mathcal{N}(0, \sigma^2 I)$  for each  $s \subseteq [n]$ . Then*

$$\sum_{s \subseteq [n]} \|Q_\star - Q_s\|_1 \leq \gamma \sqrt{\frac{1}{\pi \sigma^2} \sum_{(x,y) \in \mathbb{B}^{n+1}, (v,v') \in E(g)} \left( \frac{\partial [L(\text{eval}_{(f,g)}(x) - y)]}{\partial w_{v,v'}} \right)^2}.$$

$$\sum_{s \subseteq [n]} \|Q_\star^{(T)} - Q_s^{(T)}\|_1 \leq \gamma B \sqrt{\frac{m 2^{n+1}}{\pi \sigma^2}}$$

*Proof.* Let  $w^{(\star)} \in \mathbb{R}^m$  be the edge weights of the graph output by  $\text{BoundedNoisyGradientDescentStep}(f, g, \star, L, \gamma, 0, B)$  and  $w^{(s)} \in \mathbb{R}^m$  be the edge weights of the graph output by  $\text{BoundedNoisyGradi-$

entDescentStep  $(f, g, \rho_s, L, \gamma, 0, B)$  for each  $s$ . Observe that

$$\begin{aligned}
& \sum_{s \subseteq [n]} \|w^{(s)} - w^{(\star)}\|_2 \\
& \leq 2^{n/2} \sqrt{\sum_{s \subseteq [n]} \|w^{(s)} - w^{(\star)}\|_2^2} \\
& \leq 2^{n/2} \sqrt{\sum_{(v,v') \in E(g)} \sum_{s \subseteq [n]} \left(w_{(v,v')}^{(s)} - w_{(v,v')}^{(\star)}\right)^2} \\
& \leq 2^{n/2} \sqrt{\sum_{(v,v') \in E(g)} 2^{-n-1} \sum_{(x,y) \in \mathbb{B}^{n+1}} \gamma^2 \Psi_B^2 \left( \frac{\partial [L(\text{eval}_{(f,g)}(x) - y)]}{\partial w_{v,v'}} \right)} \\
& \leq 2^{n/2} \sqrt{\sum_{(v,v') \in E(g)} 2^{-n-1} \sum_{(x,y) \in \mathbb{B}^{n+1}} \gamma^2 B^2} \\
& = \gamma B \sqrt{m 2^n}
\end{aligned}$$

where the second to last inequality follows by applying the previous lemma to the formula in BoundedNoisyGradientDescentStep for the changes in edge weight. Next, observe that  $Q_\star$  is a Gaussian distribution with mean  $w^{(\star)}$  and covariance  $\sigma^2 I$ . Similarly,  $Q_s$  is a Gaussian distribution with mean  $w^{(s)}$  and covariance  $\sigma^2 I$  for each  $s$ . As such,

$$\begin{aligned}
& \|Q_\star - Q_s\|_1 \\
& = 2 - 2 \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \min(e^{-x^2/2\sigma^2}, e^{-(x - \|w^{(\star)} - w^{(s)}\|_2)^2/2\sigma^2}) dx \\
& = 2 - 2 \int_{-\infty}^{\|w^{(\star)} - w^{(s)}\|_2/2} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x - \|w^{(\star)} - w^{(s)}\|_2)^2/2\sigma^2} dx - 2 \int_{\|w^{(\star)} - w^{(s)}\|_2/2}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2} dx \\
& = 2 - 2 \int_{-\infty}^{-\|w^{(\star)} - w^{(s)}\|_2/2} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2} dx - 2 \int_{\|w^{(\star)} - w^{(s)}\|_2/2}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2} dx \\
& = 2 \int_{-\|w^{(\star)} - w^{(s)}\|_2/2}^{\|w^{(\star)} - w^{(s)}\|_2/2} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2} dx \\
& \leq 2 \|w^{(\star)} - w^{(s)}\|_2 / \sqrt{2\pi\sigma^2}
\end{aligned}$$

for all  $s$ .

Therefore,

$$\begin{aligned}
& \sum_{s \subseteq [n]} \|Q_\star - Q_s\|_1 \\
& \leq \sum_{s \subseteq [n]} 2 \|w^{(s)} - w^{(\star)}\|_2 / \sqrt{2\pi\sigma^2} \\
& \leq \gamma B \sqrt{m 2^{n+1} / \pi \sigma^2}
\end{aligned}$$

□

This allows us to prove the following.



**Corollary 7.** Let  $(f, g)$  be a neural net with  $m$  edges,  $B, \gamma, \sigma, T > 0$ , and  $L : \mathbb{R} \rightarrow \mathbb{R}$  be a differentiable function. Also, let  $\star$  be the uniform distribution on  $\mathbb{B}^{n+1}$ , and for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen randomly from  $\mathbb{B}^n$ . Next, for each  $0 \leq t \leq T$ , let  $Q_\star^{(t)}$  be the probability distribution of the output of  $\text{BoundedNoisyGradientDescentAlgorithm}(f, g, \star, L, \gamma, \mathcal{N}(0, \sigma^2 I), B, t)$  and  $Q_s^{(t)}$  be the probability distribution of the output of  $\text{BoundedNoisyGradientDescentAlgorithm}(f, g, \rho_s, L, \gamma, \mathcal{N}(0, \sigma^2 I), B, t)$  for each  $s \subseteq [n]$ . Then

$$\sum_{s \subseteq [n]} \|Q_\star^{(T)} - Q_s^{(T)}\|_1 \leq \gamma B T \sqrt{\frac{m 2^{n+1}}{\pi \sigma^2}}$$

*Proof.* Clearly,  $Q_s^{(0)} = Q_\star^{(0)}$  for all  $s$ . Now, for each  $s \subseteq [n]$  and  $t > 0$ , let  $Q_{\star s}^{(t)}$  be the probability distribution of the output of  $\text{BoundedNoisyGradientDescentStep}(f, G'_t, \rho_s, L, \gamma, \mathcal{N}(0, \sigma^2 I), B)$ , where  $G'_t \sim Q_\star^{(t-1)}$ . Next, observe that  $Q_\star^{(t)}$  is the probability distribution of the output of  $\text{BoundedNoisyGradientDescentStep}(f, G'_t, \star, L, \gamma, \mathcal{N}(0, \sigma^2 I), B)$ . So, by the previous lemma,

$$\sum_{s \subseteq [n]} \|Q_\star^{(t)} - Q_{\star s}^{(t)}\|_1 \leq \gamma B \sqrt{\frac{m 2^{n+1}}{\pi \sigma^2}}$$

Also, for each  $s$  and  $t$ ,  $Q_s^{(t)}$  is the probability distribution of the output of  $\text{BoundedNoisyGradientDescentStep}(f, G''_t, \rho_s, L, \gamma, \mathcal{N}(0, \sigma^2 I), B)$ , where  $G''_t \sim Q_s^{(t-1)}$ . So,  $\|Q_{\star s}^{(t)} - Q_s^{(t)}\|_1 \leq \|Q_\star^{(t-1)} - Q_s^{(t-1)}\|_1$ . Thus,

$$\sum_{s \subseteq [n]} \|Q_\star^{(t)} - Q_s^{(t)}\|_1 \leq \gamma B \sqrt{\frac{m 2^{n+1}}{\pi \sigma^2}} + \sum_{s \subseteq [n]} \|Q_\star^{(t-1)} - Q_s^{(t-1)}\|_1$$

by the triangle inequality. The desired result follows by induction.  $\square$

This in turn implies the following elaboration of Theorem 2.

**Theorem 8.** Let  $(f, g)$  be a neural net with  $m$  edges,  $B, \gamma, \sigma, T > 0$ , and  $L : \mathbb{R} \rightarrow \mathbb{R}$  be a differentiable function. Also, let  $\star$  be the uniform distribution on  $\mathbb{B}^{n+1}$ , and for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen randomly from  $\mathbb{B}^n$ . Then for a random  $S \subseteq [n]$  and  $X \in \mathbb{B}^n$  the probability that the net output by  $\text{BoundedNoisyGradientDescentAlgorithm}(f, g, \rho_S, L, \gamma, \mathcal{N}(0, \sigma^2 I), B, T)$  computes  $p_S(X)$  correctly is at most

$$1/2 + \gamma B T \sqrt{\frac{m 2^{-n}}{2 \pi \sigma^2}}$$

*Proof.* Let  $Q_\star$  be the probability distribution of the output of  $\text{BoundedNoisyGradientDescentAlgorithm}(f, g, \star, L, \gamma, \mathcal{N}(0, \sigma^2 I), B, T)$  and  $Q_s$  be the probability distribution of the output of  $\text{BoundedNoisyGradientDescentAlgorithm}(f, g, \rho_s, L, \gamma, \mathcal{N}(0, \sigma^2 I), B, T)$  for each  $s \subseteq [n]$ . By the previous corollary, we know that

$$\sum_{s \subseteq [n]} \|Q_\star - Q_s\|_1 \leq \gamma B T \sqrt{\frac{m 2^{n+1}}{\pi \sigma^2}}$$

Now, let  $G^\star \sim Q_\star$  and  $G^s \sim Q_s$  for each  $s$ . Also, for each  $(x, y) \in \mathbb{B}^{n+1}$ , let  $R_{(x,y)}$  be the set of all neural nets that output  $y$  when  $x$  is input to them. For any  $s \subseteq [n]$  and  $x \in \mathbb{B}^n$ , we have that

$$\begin{aligned} P[\text{eval}_{(f,G^s)}(x) = p_s(x)] &= P[G^s \in R_{(x,p_s(x))}] \\ &\leq P[G^\star \in R_{(x,p_s(x))}] + \|Q_\star - Q_s\|_1/2 \end{aligned}$$

That means that

$$\begin{aligned} &\sum_{s \subseteq [n], x \in \mathbb{B}^n} P[\text{eval}_{(f,G^s)}(x) = p_s(x)] \\ &\leq \sum_{s \subseteq [n], x \in \mathbb{B}^n} P[G^\star \in R_{(x,p_s(x))}] + \|Q_\star - Q_s\|_1/2 \\ &= \sum_{x \in \mathbb{B}^n} 2^{n-1} + 2^n \sum_{s \subseteq [n]} \|Q_\star - Q_s\|_1/2 \\ &= 2^{2n-1} + 2^n \gamma BT \sqrt{\frac{m2^{n-1}}{\pi\sigma^2}} \end{aligned}$$

Dividing both sides by  $2^{2n}$  yields the desired conclusion.  $\square$

## 6.4 Proof of Theorem 3

Our next goal is to make a similar argument for stochastic gradient descent. We argue that if we use noisy SGD to train a neural net on a random parity function, the probability distribution of the resulting net is similar to the probability distribution of the net we would get if we trained it on random values in  $\mathbb{B}^{n+1}$ . This will be significantly harder to prove than in the case of noisy gradient descent, because while the difference in the expected gradients is exponentially small, the gradient at a given sample may not be. As such, drowning out the signal will require much more noise. However, before we get into the details, we will need to formally define a noisy version of SGD, which is as follows.

*NoisySampleGradientDescentStep*( $f, G, Y, X, L, \gamma, B, \delta$ ):

1. For each  $(v, v') \in E(G)$ :

(a) Set

$$w'_{v,v'} = w_{v,v'} - \gamma \frac{\partial L(\text{eval}_{(f,G)}(X) - Y)}{\partial w_{v,v'}} + \delta_{v,v'}$$

(b) If  $w'_{v,v'} < -B$ , set  $w'_{v,v'} = -B$ .

(c) If  $w'_{v,v'} > B$ , set  $w'_{v,v'} = B$ .

2. Return the graph that is identical to  $G$  except that its edge weight are given by the  $w'$ .

*NoisyStochasticGradientDescentAlgorithm*( $f, G, P_Z, L, \gamma, B, \Delta, t$ ):

1. Set  $G_0 = G$ .

2. If any of the edge weights in  $G_0$  are less than  $-B$ , set all such weights to  $-B$ .
3. If any of the edge weights in  $G_0$  are greater than  $B$ , set all such weights to  $B$ .
4. For each  $0 \leq i < t$ :
  - (a) Draw  $(X_i, Y_i) \sim P_Z$ , independently of all previous values.
  - (b) Generate  $\delta^{(i)}$  by independently drawing  $\delta_{v,v'}^{(i)}$  from  $\Delta$  for each  $(v, v') \in E(G)$ .
  - (c) Set  $G_{i+1} = \text{NoisySampleGradientDescentStep}(f, G_i, Y_i, X_i, L, \gamma, B, \delta)$
5. Return  $G_t$ .

### 6.4.1 Uniform noise and SLAs

The simplest way to add noise in order to impede learning a parity function would be to add noise drawn from a uniform distribution in order to drown out the information provided by the changes in edge weights. More precisely, consider setting  $\Delta$  equal to the uniform distribution on  $[-C, C]$ . If the change in each edge weight prior to including the noise always has an absolute value less than  $D$  for some  $D < C$ , then with probability  $\frac{C-D}{C}$ , the change in a given edge weight including noise will be in  $[-(C-D), C-D]$ . Furthermore, any value in this range is equally likely to occur regardless of what the change in weight was prior to the noise term, which means that the edge's new weight provides no information on the sample used in that step. If  $D/C = o(nE(G)/\ln(n))$  then this will result in there being  $o(n/\log(n))$  changes in weight that provide any relevant information in each timestep. So, the resulting algorithm will not be able to learn the parity function by an extension of corollary 5. This leads to the following result:

**Theorem 9.** *Let  $n > 0$ ,  $\gamma > 0$ ,  $D > 0$ ,  $t = 2^{o(n)}$ ,  $(f, G)$  be a normal neural net of size polynomial in  $n$ , and  $L : \mathbb{R} \rightarrow \mathbb{R}$  be a smooth, convex, symmetric function with  $L(0) = 0$ . Also, let  $\Delta$  be the uniform probability distribution on  $[-D|E(G)|, D|E(G)|]$ . Now, let  $S$  be a random subset of  $[n]$  and  $P_Z$  be the probability distribution  $(X, p_S(X))$  when  $X$  is drawn randomly from  $\mathbb{B}^n$ . Then when  $\text{NoisyStochasticGradientDescentAlgorithm}(f, G, P_Z, L, \gamma, \infty, \Delta, t)$  is run on a computer that uses  $O(\log(n))$  bits to store each edge's weight, with probability  $1 - o(1)$  either there is at least one step when the adjustment to one of the weights prior to the noise term has absolute value greater than  $D$  or the resulting neural net fails to compute  $p_S$  with nontrivial accuracy.*

This is a side result and we provide a concise proof.

*Proof.* Consider the following attempt to simulate  $\text{NoisyStochasticGradientDescentAlgorithm}(f, G, P_Z, L, \gamma, \infty, \Delta, t)$  with a sequential learning algorithm. First, independently draw  $b_{v,v'}^{t'}$  from the uniform probability distribution on  $[-D|E(G)| + D, D|E(G)| - D]$  for each  $(v, v') \in E(G)$  and  $t' \leq t$ . Next, simulate  $\text{NoisyStochasticGradientDescentAlgorithm}(f, G, P_Z, L, \gamma, \infty, \Delta, t)$  with the following modifications. If there is ever a step where one of the adjustments to the weights before the noise term is added in is greater than  $D$ , record "failure" and give up. If there is ever a step where more than  $n/\ln^2(n)$  of the weights change by more than  $D|E(G)| - D$  after including the noise record "failure" and give up. Otherwise, record a list of which weights changed by more than  $D|E(G)| - D$  and exactly what they changed by. In all subsequent steps, assume that  $W_{v,v'}$  increased by  $b_{v,v'}^{t'}$  in step  $t'$  unless the amount it changed by in that step is recorded.

First, note that if the values of  $b$  are computed in advance, the rest of this algorithm is a sequential learning algorithm that records  $O(n/\log(n))$  bits of information per step and runs for a subexponential number of steps. As such, any attempt to compute  $p_S(X)$  based on the information

provided by its records will have accuracy  $1/2 + o(1)$  with probability  $1 - o(1)$ . Next, observe that in a given step in which all of the adjustments to weights before the noise is added in are at most  $D$ , each weight has a probability of changing by more than  $D|E(G)| - D$  of at most  $1/|E(G)|$  and these probabilities are independent. As such, with probability  $1 - o(1)$ , the algorithm will not record "failure" as a result of more than  $n/\ln^2(n)$  of the weights changing by more than  $D|E(G)| - D$ . Furthermore, the probability distribution of the change in the weight of a given vertex conditioned on the assumption that said change is at most  $D|E(G)| - D$  and a fixed value of said change prior to the inclusion of the noise term that has an absolute value of at most  $D$  is the uniform probability distribution on  $[-D|E(G)| + D, D|E(G)| - D]$ . As such, substituting the values of  $b_{v,v'}'$  for the actual changes in weights that change by less than  $D|E(G)| - D$  has no effect on the probability distribution of the resulting graph. As such, the probability distribution of the network resulting from `NoisyStochasticGradientDescentAlgorithm(f, G,  $P_Z$ , L,  $\gamma$ ,  $\infty$ ,  $\Delta$ , t)` if none of the weights change by more than  $D$  before noise is factored in differs from the probability distribution of the network generated by this algorithm if it succeeds by  $o(1)$ . Thus, the fact that the SLA cannot generate a network that computed  $p_S$  with nontrivial accuracy implies that `NoisyStochasticGradientDescentAlgorithm(f, G,  $P_Z$ , L,  $\gamma$ ,  $\infty$ ,  $\Delta$ , t)` also fails to generate a network that computes  $p_S$  with nontrivial accuracy.  $\square$

**Remark 12.** *At first glance, the amount of noise required by this theorem is ridiculously large, as it will almost always be the dominant contribution to the change in any weight in any given step. However, since the noise is random it will tend to largely cancel out over a longer period of time. As such, the result of this noisy version of stochastic gradient descent will tend to be similar to the result of regular stochastic gradient descent if the learning rate is small enough. In particular, this form of noisy gradient descent will be able to learn to compute most reasonable functions with nontrivial accuracy for most sets of starting weights, and it will be able to learn to compute some functions with nearly optimal accuracy. Admittedly, it still requires a learning rate that is smaller than anything people are likely to use in practice.*

We next move to handling lower levels of noise.

#### 6.4.2 Gaussian noise, noise accumulation, and blurring

While the previous result works, it requires more noise than we would really like. The biggest problem with it is that it ultimately argues that even given a complete list of the changes in all edge weights at each time step, there is no way to determine the parity function with nontrivial accuracy, and this requires a lot of noise. However, in order to prove that a neural net optimized by NSGD cannot learn to compute the parity function, it suffices to prove that one cannot determine the parity function from the edge weights at a single time step. Furthermore, in order to prove this, we can use the fact that noise accumulates over multiple time steps and argue that the amount of accumulated noise is large enough to drown out the information on the function provided by each input.

More formally, we plan to do the following. First of all, we will be running NSGD with a small amount of Gaussian noise added to each weight in each time step, and a larger amount of Gaussian noise added to the initial weights. Under these circumstances, the probability distribution of the edge weights resulting from running NSGD on truly random input for a given number of steps will be approximately equal to the convolution of a multivariable Gaussian distribution with something else. As such, it would be possible to construct an oracle approximating the edge weights such that the probability distribution of the edge weights given the oracle's output is essentially a multivariable Gaussian distribution. Next, we show that given any function on  $\mathbb{B}^{n+1}$ , the expected

value of the function on an input generated by a random parity function is approximately equal to its expected value on a true random input. Then, we use that to show that given a slight perturbation of a Gaussian distribution for each  $z \in \mathbb{B}^{n+1}$ , the distribution resulting from averaging together the perturbed distributions generated by a random parity function is approximately the same as the distribution resulting from averaging together all of the perturbed distributions. Finally, we conclude that the probability distribution of the edge weights after this time step is essentially the same when the input is generated by a random parity function as it is when the input is truly random.

Our first order of business is to establish that the probability distribution of the weights will be approximately equal to the convolution of a multivariable Gaussian distribution with something else, and to do that we will need the following definition.

**Definition 14.** For  $\sigma, \epsilon \geq 0$  and a probability distribution  $\hat{P}$ , a probability distribution  $P$  over  $\mathbb{R}^m$  is a  $(\sigma, \epsilon)$ -blurring of  $\hat{P}$  if

$$\|P - \hat{P} * \mathcal{N}(0, \sigma I)\|_1 \leq 2\epsilon$$

In this situation we also say that  $P$  is a  $(\sigma, \epsilon)$ -blurring. If  $\sigma \leq 0$  we consider every probability distribution as being a  $(\sigma, \epsilon)$ -blurring for all  $\epsilon$ .

The following are obvious consequences of this definition:

**Lemma 4.** Let  $\mathcal{P}$  be a collection of  $(\sigma, \epsilon)$ -blurings for some given  $\sigma$  and  $\epsilon$ . Now, select  $P \sim \mathcal{P}$  according to some probability distribution, and then randomly select  $x \sim P$ . The probability distribution of  $x$  is also a  $(\sigma, \epsilon)$ -blurring.

**Lemma 5.** Let  $P$  be a  $(\sigma, \epsilon)$ -blurring and  $\sigma' > 0$ . Then  $P * \mathcal{N}(0, \sigma' I)$  is a  $(\sigma + \sigma', \epsilon)$ -blurring

We want to prove that if the probability distribution of the weights at one time step is a blurring, then the probability distribution of the weights at the next time step is also a blurring. In order to do that, we need to prove that a slight distortion of a blurring is still a blurring. The first step towards that proof is the following lemma:

**Lemma 6.** Let  $\sigma, B > 0$ ,  $m$  be a positive integer,  $m\sqrt{2\sigma/\pi} < r \leq 1/(mB)$ , and  $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$  such that  $f(0) = 0$ ,  $|\frac{\partial f_i}{\partial x_j}(0)| = 0$  for all  $i$  and  $j$ , and  $|\frac{\partial^2 f_i}{\partial x_j \partial x_{j'}}(x)| \leq B$  for all  $i, j, j'$ , and all  $x$  with  $\|x\|_1 < r$ . Next, let  $P$  be the probability distribution of  $X + f(X)$  when  $X \sim \mathcal{N}(0, \sigma I)$ . Then  $P$  is a  $(\sigma, \epsilon)$ -blurring for  $\epsilon = \frac{4(m+2)m^2 B \sqrt{2\sigma/\pi} + 3m^5 B^2 \sigma}{8} + (1 - Bmr)e^{-(r/2\sqrt{\sigma} - m/\sqrt{2\pi})^2/m}$ .

*Proof.* First, note that for any  $x$  with  $\|x\|_1 < r$  and any  $i$  and  $j$ , it must be the case that  $|\frac{\partial f_i}{\partial x_j}(x)| \leq B\|x\|_1 < Br$ . That in turn means that for any  $x, x'$  with  $\|x\|_1, \|x'\|_1 < r$  and any  $i$ , it must be the case that  $|f(x)_i - f(x')_i| \leq Br\|x - x'\|_1$  with equality only if  $x = x'$ . In particular, this means that for any such  $x, x'$ , it must be the case that  $\|f(x) - f(x')\|_1 \leq mBr\|x - x'\|_1 \leq \|x - x'\|_1$  with equality only if  $x = x'$ . Thus,  $x + f(x) \neq x' + f(x')$  unless  $x = x'$ . Also, note that the bound on the second derivatives of  $f$  implies that  $|f_i(x)| \leq B\|x\|_1^2/2$  for all  $\|x\|_1 < r$  and all  $i$ . This means that

$$\begin{aligned}
& \|P - \mathcal{N}(0, \sigma I)\|_1 \\
& \leq 2 - 2 \int_{x: \|x\|_1 < r} \min \left( (2\pi\sigma)^{-m/2} e^{-\|x\|_2^2/2\sigma}, (2\pi\sigma)^{-m/2} e^{-\|x+f(x)\|_2^2/2\sigma} |I + [\nabla f^T](x)| \right) dx \\
& \leq 2 - 2 \int_{x: \|x\|_1 < r} (2\pi\sigma)^{-m/2} e^{-(\|x\|_2^2 + B\|x\|_1^3/2 + mB^2\|x\|_1^4/4)/2\sigma} (1 - Bm\|x\|_1) dx \\
& = 2(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 < r} e^{-\|x\|_2^2/2\sigma} - e^{-(\|x\|_2^2 + B\|x\|_1^3/2 + mB^2\|x\|_1^4/4)/2\sigma} (1 - Bm\|x\|_1) dx \\
& \quad + 2(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx \\
& = 2(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 < r} e^{-\|x\|_2^2/2\sigma} - e^{-(\|x\|_2^2 + B\|x\|_1^3/2 + mB^2\|x\|_1^4/4)/2\sigma} dx \\
& \quad + 2(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 < r} Bm\|x\|_1 e^{-(\|x\|_2^2 + B\|x\|_1^3/2 + mB^2\|x\|_1^4/4)/2\sigma} dx \\
& \quad + 2(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx \\
& \leq 2(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 < r} \frac{2B\|x\|_1^3 + mB^2\|x\|_1^4}{8\sigma} e^{-\|x\|_2^2/2\sigma} dx \\
& \quad + 2(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 < r} Bm\|x\|_1 e^{-\|x\|_2^2/2\sigma} dx + 2(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx \\
& \leq 2(2\pi\sigma)^{-m/2} \int_{x \in \mathbb{R}^m} \frac{2B\|x\|_1^3 + mB^2\|x\|_1^4}{8\sigma} e^{-\|x\|_2^2/2\sigma} dx \\
& \quad + 2(2\pi\sigma)^{-m/2} \int_{x \in \mathbb{R}^m} Bm\|x\|_1 e^{-\|x\|_2^2/2\sigma} dx \\
& \quad + 2(2\pi\sigma)^{-m/2} (1 - Bmr) \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx \\
& \leq \frac{m^3 B}{2\sigma} \sqrt{8\sigma^3/\pi} + \frac{m^5 B^2}{4\sigma} \cdot 3\sigma^2 + 2m^2 B \sqrt{2\sigma/\pi} + 2(2\pi\sigma)^{-m/2} (1 - Bmr) \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx \\
& = m^3 B \sqrt{2\sigma/\pi} + \frac{3m^5 B^2 \sigma}{4} + 2m^2 B \sqrt{2\sigma/\pi} + 2(2\pi\sigma)^{-m/2} (1 - Bmr) \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx \\
& = \frac{4(m+2)m^2 B \sqrt{2\sigma/\pi} + 3m^5 B^2 \sigma}{4} + 2(2\pi\sigma)^{-m/2} (1 - Bmr) \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx
\end{aligned}$$

Next, observe that for any  $\lambda \geq 0$ , it must be the case that

$$\begin{aligned}
& (2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx \\
& \leq (2\pi\sigma)^{-m/2} e^{-\lambda r/\sigma} \int_{x: \|x\|_1 \geq r} e^{\lambda \|x\|_1/\sigma} e^{-\|x\|_2^2/2\sigma} dx \\
& \leq (2\pi\sigma)^{-m/2} e^{-\lambda r/\sigma} \int_{x \in \mathbb{R}^m} e^{\lambda \|x\|_1/\sigma} e^{-\|x\|_2^2/2\sigma} dx \\
& = e^{-\lambda r/\sigma} \left[ (2\pi\sigma)^{-1/2} \int_{x_1 \in \mathbb{R}} e^{\lambda |x_1|/\sigma} e^{-x_1^2/2\sigma} dx_1 \right]^m \\
& = e^{-\lambda r/\sigma} \left[ 2(2\pi\sigma)^{-1/2} \int_0^\infty e^{\lambda x_1/\sigma} e^{-x_1^2/2\sigma} dx_1 \right]^m \\
& = e^{-\lambda r/\sigma} \left[ 2(2\pi\sigma)^{-1/2} \int_0^\infty e^{\lambda^2/2\sigma} e^{-(x_1-\lambda)^2/2\sigma} dx_1 \right]^m \\
& = e^{-\lambda r/\sigma} \left[ 2e^{\lambda^2/2\sigma} (2\pi\sigma)^{-1/2} \int_{-\lambda}^\infty e^{-x_1^2/2\sigma} dx_1 \right]^m \\
& \leq e^{-\lambda r/\sigma} \left[ e^{\lambda^2/2\sigma} (1 + 2\lambda/\sqrt{2\pi\sigma}) \right]^m \\
& \leq e^{-\lambda r/\sigma + m\lambda^2/2\sigma + 2m\lambda/\sqrt{2\pi\sigma}}
\end{aligned}$$

In particular, if we set  $\lambda = r/m - \sqrt{2\sigma/\pi}$ , this shows that  $(2\pi\sigma)^{-m/2} \int_{x: \|x\|_1 \geq r} e^{-\|x\|_2^2/2\sigma} dx \leq e^{-(r/2\sqrt{\sigma} - m/\sqrt{2\pi})^2/m}$ . The desired conclusion follows.  $\square$

**Lemma 7.** Let  $\sigma, B_1, B_2 > 0$ ,  $m$  be a positive integer with  $m < 1/B_1$ ,  $m\sqrt{2\sigma/\pi} < r \leq (1 - mB_1)/(mB_2)$ , and  $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$  such that  $|\frac{\partial f_i}{\partial x_j}(0)| \leq B_1$  for all  $i$  and  $j$ , and  $|\frac{\partial^2 f_i}{\partial x_j \partial x_{j'}}(x)| \leq B_2$  for all  $i, j, j'$ , and all  $x$  with  $\|x\|_1 < r$ . Next, let  $P$  be the probability distribution of  $X + f(X)$  when  $X \sim \mathcal{N}(0, \sigma I)$ . Then  $P$  is a  $((1 - mB_1)^2 \sigma, \epsilon)$ -blurring for  $\epsilon = \frac{4(m+2)m^2 B_2 \sqrt{2\sigma/\pi} / (1 - mB_1) + 3m^5 B_2^2 \sigma / (1 - mB_1)^2}{8} + (1 - (1 + mB_1)B_2 m r) e^{-(r/2\sqrt{\sigma} - m/\sqrt{2\pi})^2/m}$ .

*Proof.* First, define  $h : \mathbb{R}^m \rightarrow \mathbb{R}^m$  such that  $h(x) = f(0) + x + [\nabla f^T]^T(0)x$  for all  $x$ . Every eigenvalue of  $[\nabla f](0)$  has a magnitude of at most  $mB_1$ , so  $h$  is invertible. Next, define  $f^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$  such that  $f^*(x) = h^{-1}(x + f(x)) - x$  for all  $x$ . Clearly,  $f^*(0) = 0$ , and  $\frac{\partial f_i^*}{\partial x_j}(0) = 0$  for all  $i$  and  $j$ . Furthermore, for any given  $x$  it must be the case that  $\max_{i,j,j'} |\frac{\partial^2 f_i}{\partial x_j \partial x_{j'}}| \geq (1 - mB_1) \max_{i,j,j'} |\frac{\partial^2 f_i^*}{\partial x_j \partial x_{j'}}|$ . So,  $|\frac{\partial^2 f_i}{\partial x_j \partial x_{j'}}| \leq B_2/(1 - mB_1)$  for all  $i, j, j'$ , and all  $x$  with  $\|x\|_1 < r$ . Now, let  $P^*$  be the probability distribution of  $x + f^*(x)$  when  $x \sim \mathcal{N}(0, \sigma I)$ . By the previous lemma,  $P^*$  is a  $(\sigma, \epsilon)$ -blurring for  $\epsilon = \frac{4(m+2)m^2 B_2 \sqrt{2\sigma/\pi} / (1 - mB_1) + 3m^5 B_2^2 \sigma / (1 - mB_1)^2}{8} + (1 - (1 + mB_1)B_2 m r) e^{-(r/2\sqrt{\sigma} - m/\sqrt{2\pi})^2/m}$ .

Now, let  $\widehat{P}^*$  be a probability distribution such that  $P^*$  is a  $(\sigma, \epsilon)$ -blurring of  $\widehat{P}^*$ . Next, let  $\widehat{P}$  be the probability distribution of  $h(x)$  when  $x$  is drawn from  $\widehat{P}^*$ . Also, let  $M = (I + [\nabla f^T]^T(0))(I + [\nabla f^T](0))$ . The fact that  $\|P^* - \widehat{P}^* * \mathcal{N}(0, \sigma I)\|_1 \leq 2\epsilon$  implies that

$$\|P - \widehat{P} * \mathcal{N}(0, \sigma M)\|_1 \leq 2\epsilon$$

For any  $x \in \mathbb{R}^m$ , it must be the case that

$$\begin{aligned} x \cdot Mx &\geq \|x\|_2^2 - 2B_1\|x\|_1^2 - mB_1^2\|x\|_1^2 \\ &\geq \|x\|_2^2 - 2mB_1\|x\|_2^2 - m^2B_1^2\|x\|_2^2 = (1 - mB_1)^2\|x\|_2^2 \end{aligned}$$

That in turn means that  $\sigma M - \sigma(1 - mB_1)^2I$  is positive semidefinite. So,  $\hat{P} * \mathcal{N}(0, \sigma M) = \hat{P} * \mathcal{N}(0, \sigma M - \sigma(1 - mB_1)^2I) * \mathcal{N}(0, \sigma(1 - mB_1)^2I)$ , which proves that  $P$  is a  $((1 - mB_1)^2\sigma, \epsilon)$ -blurring of  $\hat{P} * \mathcal{N}(0, \sigma M - \sigma(1 - mB_1)^2I)$ .  $\square$

Any blurring is approximately equal to a linear combination of Gaussian distributions, so this should imply a similar result for  $X$  drawn from a  $(\sigma, \epsilon)$  blurring. However, we are likely to use functions that have derivatives that are large in some places. Not all of the Gaussian distributions that the blurring combines will necessarily have centers that are far enough from the high derivative regions. As such, we need to add an assumption that the centers of the distributions are in regions where the derivatives are small. We formalize the concept of being in a region where the derivatives are small as follows.

**Definition 15.** Let  $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$ ,  $x \in \mathbb{R}^m$ , and  $r, B_1, B_2 > 0$ . Then  $f$  is  $(r, B_1, B_2)$ -stable at  $x$  if  $|\frac{\partial f_i}{\partial x_j}(0)| \leq B_1$  for all  $i$  and  $j$  and all  $x'$  with  $\|x' - x\|_1 < r$ , and  $|\frac{\partial^2 f_i}{\partial x_j \partial x_{j'}}| \leq B_2$  for all  $i, j, j'$ , and all  $x'$  with  $\|x' - x\|_1 < 2r$ . Otherwise,  $f$  is  $(r, B_1, B_2)$ -unstable at  $x$ .

This allows us to state the following variant of the previous lemma.

**Lemma 8.** Let  $\sigma, B_1, B_2 > 0$ ,  $m$  be a positive integer with  $m < 1/B_1$ ,  $m\sqrt{2\sigma/\pi} < r \leq (1 - mB_1)/(mB_2)$ , and  $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$  such that there exists  $x$  with  $\|w\|_1 < r$  such that  $f$  is  $(r, B_1, B_2)$ -stable at  $x$ . Next, let  $P$  be the probability distribution of  $X + f(X)$  when  $X \sim \mathcal{N}(0, \sigma I)$ . Then  $P$  is a  $((1 - mB_1)^2\sigma, \epsilon)$ -blurring for  $\epsilon = \frac{4(m+2)m^2B_2\sqrt{2\sigma/\pi}/(1-mB_1)+3m^5B_2^2\sigma/(1-mB_1)^2}{8} + (1 - (1 + mB_1)B_2mr)e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m}$ .

*Proof.*  $|\frac{\partial f_i}{\partial x_j}(0)| \leq B_1$  for all  $i$  and  $j$ , and  $|\frac{\partial^2 f_i}{\partial x_j \partial x_{j'}}(x')| \leq B_2$  for all  $i, j, j'$ , and all  $x'$  with  $\|x'\|_1 < r$ . Then, the desired conclusion follows by the previous lemma.  $\square$

This lemma could be relatively easily used to prove that if we draw  $X$  from a  $(\sigma, \epsilon)$ -blurring instead of drawing it from  $\mathcal{N}(0, \sigma I)$  and  $f$  is stable at  $X$  with high probability then the probability distribution of  $X + f(X)$  will be a  $(\sigma', \epsilon')$ -blurring for  $\sigma' \approx \sigma$  and  $\epsilon' \approx \epsilon$ . However, that is not quite what we will need. The issue is that we are going to repeatedly apply a transformation along these lines to a variable. If all we know is that its probability distribution is a  $(\sigma^{(t)}, \epsilon^{(t)})$ -blurring in each step, then we potentially have a probability of  $\epsilon^{(t)}$  each time step that it behaves badly in that step. That is consistent with there being a probability of  $\sum \epsilon^{(t)}$  that it behaves badly eventually, which is too high.

In order to avoid this, we will think of these blurrings as approximations of a  $(\sigma, 0)$  blurring. Then, we will need to show that if  $X$  is good in the sense of being present in the idealized form of the blurring then  $X + f(X)$  will also be good. In order to do that, we will need the following definition.

**Definition 16.** Let  $P$  be a  $(\sigma, \epsilon)$ -blurring of  $\hat{P}$ , and  $X \sim P$ . A  $\sigma$ -revision of  $X$  to  $\hat{P}$  is a random pair  $(X', M)$  such that the probability distribution of  $M$  is  $\hat{P}$ , the probability distribution of  $X'$  given that  $M = \mu$  is  $\mathcal{N}(\mu, \sigma I)$ , and  $P[X' \neq X] = \|P - \mathcal{N}(0, \sigma I) * \hat{P}\|_1/2$ . Note that a  $\sigma$ -revision of  $X$  to  $\hat{P}$  will always exist.



### 6.4.3 Means, SLAs, and Gaussian distributions

Our plan now is to consider a version of NoisyStochasticGradientDescent in which the edge weights get revised after each step and then to show that under suitable assumptions when this algorithm is executed none of the revisions actually change the values of any of the edge weights. Then, we will show that whether the samples are generated randomly or by a parity function has minimal effect on the probability distribution of the edge weights after each step, allowing us to revise the edge weights in both cases to the same probability distribution. That will allow us to prove that the probability distribution of the final edge weights is nearly independent of which probability distribution the samples are drawn from.

The next step towards doing that is to show that if we run NoisySampleGradientDescentStep on a neural network with edge weights drawn from a linear combination of Gaussian distributions, the probability distribution of the resulting graph is essentially independent of what parity function we used to generate the sample. In order to do that, we are going to need some more results on the difficulty of distinguishing an unknown parity function from a random function. First of all, recall that corollary 6 says that

**Corollary 8.** *Let  $n > 0$  and  $f : \mathbb{B}^{n+1} \rightarrow \mathbb{R}$ . Also, let  $X$  be a random element of  $\mathbb{B}^n$  and  $Y$  be a random element of  $\mathbb{B}$ . Then*

$$\sum_{s \subseteq [n]} |E[f((X, Y))] - E[f((X, p_s(X)))]| \leq 2^{n/2} \sqrt{E[f^2((X, Y))]}$$

We can apply this to probability distributions to get the following.

**Theorem 10.** *Let  $m > 0$ , and for each  $z \in \mathbb{B}^{n+1}$ , let  $P_z$  be a probability distribution on  $\mathbb{R}^m$  with probability density function  $f_z$ . Now, randomly select  $Z \in \mathbb{B}^{n+1}$  and  $X \in \mathbb{B}^n$  uniformly and independently. Next, draw  $W$  from  $P_Z$  and  $W'_s$  from  $P_{(X, p_s(X))}$  for each  $s \subseteq [n]$ . Let  $P^\star$  be the probability distribution of  $W$  and  $P_s^\star$  be the probability distribution of  $W'_s$  for each  $s$ . Then*

$$2^{-n} \sum_{s \subseteq [n]} \|P^\star - P_s^\star\|_1 \leq 2^{-n/2} \int_{\mathbb{R}^m} \max_{z \in \mathbb{B}^{n+1}} f_z(w) dw$$

*Proof.* Let  $f^\star = 2^{-n-1} \sum_{z \in \mathbb{B}^{n+1}} f_z$  be the probability density function of  $P^\star$ , and for each  $s \subseteq [n]$ , let  $f_s^\star = 2^{-n} \sum_{x \in \mathbb{B}^n} f_{(x, p_s(x))}$  be the probability density function of  $P_s^\star$ .

For any  $w \in \mathbb{R}^m$ , we have that

$$\begin{aligned} & \sum_{s \subseteq [n]} |f^\star(w) - f_s^\star(w)| \\ &= \sum_{s \subseteq [n]} |E[f_Z(w)] - E[f_{(X, p_s(X))}(w)]| \\ &\leq 2^{n/2} \sqrt{E[f_Z^2(w)]} \\ &\leq 2^{n/2} \max_{z \in \mathbb{B}^{n+1}} f_z(w) \end{aligned}$$

That means that

$$\begin{aligned}
& \sum_{s \subseteq [n]} \|P^\star - P_s^\star\|_1 \\
&= \sum_{s \subseteq [n]} \int_{\mathbb{R}^m} |f^\star(w) - f_s^\star(w)| dw \\
&\leq \int_{\mathbb{R}^m} \sum_{s \subseteq [n]} |f^\star(w) - f_s^\star(w)| dw \\
&\leq 2^{n/2} \int_{\mathbb{R}^m} \max_{z \in \mathbb{B}^{n+1}} f_z(w) dw
\end{aligned}$$

□

In particular, if these probability distributions are the result of applying a well-behaved distortion function to a Gaussian distribution, we have the following.

**Theorem 11.** *Let  $\sigma, B_0, B_1 > 0$ , and  $n$  and  $m$  be positive integers with  $m < 1/B_1$ . Also, for every  $z \in \mathbb{B}^{n+1}$ , let  $f^{(z)} : \mathbb{R}^m \rightarrow \mathbb{R}^m$  be a function such that  $|f_i^{(z)}(w)| \leq B_0$  for all  $i$  and  $w$  and  $|\frac{\partial f_i^{(z)}}{\partial w_j}(w)| \leq B_1$  for all  $i, j$ , and  $w$ . Now, randomly select  $Z \in \mathbb{B}^{n+1}$  and  $X \in \mathbb{B}^n$  uniformly and independently. Next, draw  $W_0$  from  $\mathcal{N}(0, \sigma I)$ , set  $W = W_0 + f^{(Z)}(W_0)$  and  $W'_s = W_0 + f^{(X, p_s(X))}(W_0)$  for each  $s \subseteq [n]$ . Let  $P^\star$  be the probability distribution of  $W$  and  $P_s^\star$  be the probability distribution of  $W'_s$  for each  $s$ . Then*

$$2^{-n} \sum_{s \subseteq [n]} \|P^\star - P_s^\star\|_1 \leq 2^{-n/2} \cdot e^{2mB_0/\sqrt{2\pi}\sigma} / (1 - mB_1)$$

*Proof.* First, note that the bound on  $|\frac{\partial f_i^{(z)}}{\partial w_j}(w)|$  ensures that if  $w + f^{(z)}(w) = w' + f^{(z)}(w')$  then  $w = w'$ . So, for any  $z$  and  $w$ , the probability density function of  $W_0 + f^{(z)}(W_0)$  at  $w$  is less than or equal to

$$(2\pi\sigma)^{-m/2} e^{-\sum_{i=1}^m \max^2(|w_i| - B_0, 0)/2\sigma} / |I + [\nabla f^{(z)}]^T(w)|$$

which is less than or equal to

$$(2\pi\sigma)^{-m/2} e^{-\sum_{i=1}^m \max^2(|w_i| - B_0, 0)/2\sigma} / (1 - mB_1)$$

By the previous theorem, that implies that

$$\begin{aligned}
& 2^{-n} \sum_{s \subseteq [n]} \|P^\star - P_s^\star\|_1 \\
&\leq 2^{-n/2} \int_{\mathbb{R}^m} (2\pi\sigma)^{-m/2} e^{-\sum_{i=1}^m \max^2(|w_i| - B_0, 0)/2\sigma} / (1 - mB_1) dw \\
&= 2^{-n/2} \left[ \int_{\mathbb{R}} (2\pi\sigma)^{-1/2} e^{-\max^2(|w'| - B_0, 0)/2\sigma} dw' \right]^m / (1 - mB_1) \\
&= 2^{-n/2} [1 + 2B_0/\sqrt{2\pi}\sigma]^m / (1 - mB_1) \\
&\leq 2^{-n/2} \cdot e^{2mB_0/\sqrt{2\pi}\sigma} / (1 - mB_1)
\end{aligned}$$

□

The problem with this result is that it requires  $f$  to have values and derivatives that are bounded everywhere, and the functions that we will encounter in practice will not necessarily have that property. We can reasonably require that our functions have bounded values and derivatives in the regions we are likely to evaluate them on, but not in the entire space. Our solution to this will be to replace the functions with new functions that have the same value as them in small regions that we are likely to evaluate them on, and that obey the desired bounds. The fact that we can do so is established by the following theorem.

**Theorem 12.** *Let  $B_0, B_1, B_2, r, \sigma > 0$ ,  $\mu \in \mathbb{R}^m$ , and  $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$  such that there exists  $x$  with  $\|x - \mu\|_1 < r$  such that  $f$  is  $(r, B_1, B_2)$ -stable at  $x$  and  $|f_i(x)| \leq B_0$  for all  $i$ . Then there exists a function  $f^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$  such that  $f^*(x) = f(x)$  for all  $x$  with  $\|x - \mu\|_1 < r$ , and  $|f_i(x)| \leq B_0 + 2rB_1 + 2r^2B_2$  and  $|\frac{\partial f_i}{\partial x_j}(x)| \leq 2B_1 + 2rB_2$  for all  $x \in \mathbb{R}^m$  and  $i, j \in [m]$ .*

*Proof.* First, observe that the  $(r, B_1, B_2)$ -stability of  $f$  at  $x$  implies that for every  $x'$  with  $\|x - x'\| \leq 2r$ , we have that  $|\frac{\partial f_i}{\partial x_j}(x')| \leq B_1 + rB_2$  and  $|f_i(x')| \leq B_0 + 2r(B_1 + rB_2)$ . In particular, this holds for all  $x'$  with  $\|x' - \mu\|_1 \leq 2r - \|x - \mu\|_1 < r$ .

That means that there exists  $r' > r$  such that the values and derivatives of  $f$  satisfy the desired bounds for all  $x'$  with  $\|x' - \mu\|_1 \leq r'$ . Now, define the function  $\bar{f} : \mathbb{R}^m \rightarrow \mathbb{R}^m$  such that  $\bar{f}(x') = f(\mu + (x' - \mu) \cdot r' / \|x' - \mu\|_1)$ . This function satisfies the bounds for all  $x'$  with  $\|x'\|_1 > r'$ , except that it may not be differentiable when  $x'_j = \mu_j$  for some  $j$ . Consider defining  $f^*(x')$  to be equal to  $f(x')$  when  $\|x' - \mu\|_1 \leq r'$  and  $\bar{f}(x')$  otherwise. This would almost work, except that it may not be differentiable when  $\|x' - \mu\|_1 = r'$ , or  $\|x' - \mu\|_1 > r'$  and  $x'_j = \mu_j$  for some  $j$ .

In order to fix this, we define a smooth function  $h$  of bounded derivative such that  $h(x') = 0$  whenever  $\|x' - \mu\|_1 \leq r$ , and  $h(x') \geq 1$  whenever  $\|x' - \mu\|_1 \geq r'$ . Then, for all sufficiently small positive constants  $\delta$ ,  $f^* * \mathcal{N}(0, \delta \cdot h^2(x')I)$  has the desired properties.  $\square$

Combining this with the previous theorem yields the following.

**Corollary 9.** *Let  $\sigma, B_0, B_1, B_2, r > 0$ ,  $\mu \in \mathbb{R}^m$ , and  $n$  and  $m$  be positive integers with  $m < 1/(2B_1 + 2rB_2)$ . Then, for every  $z \in \mathbb{B}^{n+1}$ , let  $f^{(z)} : \mathbb{R}^m \rightarrow \mathbb{R}^m$  be a function such that there exists  $x$  with  $\|x - \mu\|_1 < r$  such that  $f$  is  $(r, B_1, B_2)$ -stable at  $x$  and  $|f_i(x)| \leq B_0$  for all  $i$ . Next, draw  $W_0$  from  $\mathcal{N}(\mu, \sigma I)$ . Now, randomly select  $Z \in \mathbb{B}^{n+1}$  and  $X \in \mathbb{B}^n$  uniformly and independently. Then, set  $W = W_0 + f^{(Z)}(W_0)$  and  $W'_s = W_0 + f^{(X, p_s(X))}(W_0)$  for each  $s \subseteq [n]$ . Let  $P^*$  be the probability distribution of  $W$  and  $P_s^*$  be the probability distribution of  $W'_s$  for each  $s$ . Then*

$$2^{-n} \sum_{s \subseteq [n]} \|P^* - P_s^*\|_1 \leq 2^{-n/2} \cdot e^{2m(B_0 + 2rB_1 + 2r^2B_2)/\sqrt{2\pi}\sigma} / (1 - 2mB_1 - 2rmB_2) + 2e^{-(r/2\sqrt{\sigma} - m/\sqrt{2\pi})^2/m}$$

*Proof.* For each  $z$ , we can define  $f^{(z)*}$  as an approximation of  $f^{(z)}$  as explained in the previous theorem.  $\|W_0 - \mu\|_1 \leq r$  with a probability of at least  $1 - e^{-(r/2\sqrt{\sigma} - m/\sqrt{2\pi})^2/m}$ , in which case  $f^{(z)*}(W_0) = f^{(z)}(W_0)$  for all  $z$ . For a random  $s$ , the probability distributions of  $W_0 + f^{(Z)*}(W_0)$  and  $W_0 + f^{(X, p_s(X))*}(W_0)$  have an  $L_1$  difference of at most  $2^{-n/2} \cdot e^{2m(B_0 + 2rB_1 + 2r^2B_2)/\sqrt{2\pi}\sigma} / (1 - 2mB_1 - 2rmB_2)$  on average by 11. Combining these yields the desired result.  $\square$

That finally gives us the components needed to prove the following.

**Theorem 13.** *Let  $m, n > 0$  and define  $f^{[z]} : \mathbb{R}^m \rightarrow \mathbb{R}^m$  to be a smooth function for all  $z \in \mathbb{B}^{n+1}$ . Also, let  $\sigma, B_0, B_1, B_2 > 0$  such that  $B_1 < 1/2m$ ,  $m\sqrt{2\sigma/\pi} < r \leq (1 - 2mB_1)/(2mB_2)$ ,  $T$  be a positive integer, and  $\mu_0 \in \mathbb{R}^m$ . Then, let  $\star$  be the uniform distribution on  $\mathbb{B}^{n+1}$ , and for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen randomly from  $\mathbb{B}^n$ . Next, let*

$P_Z$  be a probability distribution on  $\mathbb{B}^{n+1}$  that is chosen by means of the following procedure. First, with probability  $1/2$ , set  $P_Z = \star$ . Otherwise, select a random  $S \subseteq [n]$  and set  $P_Z = \rho_S$ .

Now, draw  $W^{(0)}$  from  $\mathcal{N}(\mu_0, \sigma I)$ , independently draw  $Z_i \sim P_Z$  and  $\Delta^{(i)} \sim \mathcal{N}(0, [2mB_1 - m^2B_1^2]\sigma I)$  for all  $0 < i \leq T$ . Then, set  $W^{(i)} = W^{(i-1)} + f^{[Z_i]}(W^{(i-1)}) + \Delta^{(i)}$  for each  $0 < i \leq T$ , and let  $p$  be the probability that there exists  $0 \leq i \leq T$  such that  $f^{[Z_i]}$  is  $(r, B_1, B_2)$ -unstable at  $W^{(i)}$  or  $\|f^{[Z_i]}(W^{(i)})\|_\infty > B_0$ . Finally, let  $Q$  and  $Q'_s$  be the probability distribution of  $W^{(T)}$  given that  $P_Z = \star$  and the probability distribution of  $W^{(T)}$  given that  $P_Z = \rho_s$ . Then

$$2^{-n} \sum_{s \subseteq [n]} \|Q - Q'_s\|_1 \leq 4p + T(4\epsilon + \epsilon' + 4\epsilon'')$$

where

$$\begin{aligned} \epsilon = & \frac{4(m+2)m^2B_2\sqrt{2\sigma/\pi}/(1-mB_1) + 3m^5B_2^2\sigma/(1-mB_1)^2}{8} \\ & + (1 - (1+mB_1)B_2mr)e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m} \end{aligned}$$

$$\epsilon' = 2^{-n/2} \cdot e^{2m(B_0+2rB_1+2r^2B_2)/\sqrt{2\pi\sigma}}/(1-2mB_1-2rmB_2) + 2e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m}$$

$$\epsilon'' = e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m}$$

*Proof.* In order to prove this, we plan to define new variables  $\widetilde{W}^{(i) \prime}$  such that  $\widetilde{W}^{(i) \prime} = W^{(i)}$  with high probability for each  $i$  and the probability distribution of  $\widetilde{W}^{(i) \prime}$  is independent of  $P_Z$ . More precisely, we define the variables  $\widetilde{W}^{(i)}$ ,  $\widetilde{W}^{(i) \prime}$ , and  $\widetilde{M}^{(i)}$  for each  $i$  as follows. First, set  $\widetilde{M}^{(0)} = \mu_0$  and  $\widetilde{W}^{(0) \prime} = \widetilde{W}^{(0)} = W^{(0)}$ .

Next, for a function  $f$  and a point  $w$ , we say that  $f$  is quasistable at  $w$  if there exists  $w'$  such that  $\|w' - w\|_1 \leq r$ ,  $f^{[Z_i]}$  is  $(r, B_1, B_2)$ -stable at  $w'$ , and  $\|f^{[Z_i]}(w')\|_\infty \leq B_0$ , and that it is quasiunstable at  $w$  otherwise.

for each  $0 < i \leq T$ , if  $f^{[Z_i]}$  is quasistable at  $\widetilde{M}^{(i-1)}$ , set

$$\widetilde{W}^{(i)} = \widetilde{W}^{(i-1) \prime} + f^{[Z_i]}(\widetilde{W}^{(i-1) \prime}) + \Delta^{(i)}$$

Otherwise, set

$$\widetilde{W}^{(i)} = \widetilde{W}^{(i-1) \prime} + \Delta^{(i)}$$

Next, for each  $\rho$ , let  $P_\rho^{(i)}$  be the probability distribution of  $\widetilde{W}^{(i)}$  given that  $P_Z = \rho$ . Then, define  $\widehat{P}^{(i)}$  as a probability distribution such that  $P_\star^{(i)}$  is a  $(\sigma, \epsilon_0)$ -blurring of  $\widehat{P}^{(i)}$  with  $\epsilon_0$  as small as possible. Finally, for each  $\rho$ , if  $P_Z = \rho$ , let  $(\widetilde{W}^{(i) \prime}, \widetilde{M}^{(i)})$  be a  $\sigma$ -revision of  $\widetilde{W}^{(i)}$  to  $\widehat{P}^{(i)}$ .

In order to analyse the behavior of these variables, we will need to make a series of observations. First, note that for every  $i$ ,  $\rho$ , and  $\mu$  the probability distribution of  $\widetilde{W}^{(i-1) \prime}$  given that  $P_Z = \rho$  and  $M^{(i-1)} = \mu$  is  $\mathcal{N}(\mu, \sigma I)$ . Also, either  $f^{[Z_i]}$  is quasistable at  $\mu$  or  $0$  is quasistable at  $\mu$ . Either way, the probability distribution of  $\widetilde{W}^{(i)}$  under these circumstances must be a  $(\sigma, \epsilon)$ -blurring by Lemma 8 and Lemma 5. That in turn means that  $P_\rho^{(i)}$  is a  $(\sigma, \epsilon)$  blurring for all  $i$  and  $\rho$ , and thus that  $P_\star^{(i)}$  must be a  $(\sigma, \epsilon)$  blurring of  $\widehat{P}^{(i)}$ . Furthermore, by the previous corollary,

$$2^{-n} \sum_{s \subseteq [n]} \|P_\star^{(i)} - P_{\rho_s}^{(i)}\|_1 \leq \epsilon'$$

The combination of these implies that

$$2^{-n} \sum_{s \subseteq [n]} \|\mathcal{N}(0, \sigma I) * \widehat{P}^{(i)} - P_{\rho_s}^{(i)}\|_1 \leq 2\epsilon + \epsilon'$$

which in turn means that  $P[\widehat{W}^{(i)'} \neq \widetilde{W}^{(i)}] \leq \epsilon + \epsilon'/4$ . That in turn means that with probability at least  $1 - T(\epsilon + \epsilon'/4)$  it is the case that  $\widehat{W}^{(i)'} = \widetilde{W}^{(i)}$  for all  $i$ .

If  $\widehat{W}^{(i)'} = \widetilde{W}^{(i)}$  for all  $i$  and  $\widehat{W}^{(T)'} \neq W^{(T)}$  then there must exist some  $i$  such that  $\widehat{W}^{(i-1)'} = W^{(i-1)}$  but  $\widetilde{W}^{(i)} \neq W^{(i)}$ . That in turn means that

$$\begin{aligned} \widehat{W}^{(i)} &\neq W^{(i)} \\ &= W^{(i-1)} + f^{[Z_i]}(W^{(i-1)}) + \Delta^{(i)} \\ &= \widetilde{W}^{(i-1)'} + f^{[Z_i]}(\widetilde{W}^{(i-1)'}) + \Delta^{(i)} \end{aligned}$$

If  $F^{[Z_i]}$  were quasistable at  $M^{(i-1)}$ , that is exactly the formula that would be used to calculate  $\widetilde{W}^{(i)}$ , so  $F^{[Z_i]}$  must be quasiunstable at  $M^{(i-1)}$ . That in turn requires that either  $F^{[Z_i]}$  is  $(r, B_1, B_2)$ -unstable at  $\widetilde{W}^{(i-1)'} = W^{(i-1)}$ ,  $\|f^{[Z_i]}(W^{(i-1)})\|_\infty > B_0$ , or  $\|\widetilde{W}^{(i-1)'} - M^{(i-1)}\|_1 > r$ . With probability at least  $1 - p$ , neither of the first two scenarios occur for any  $i$ , while for any given  $i$  the later occurs with a probability of at most  $\epsilon''$ . Thus,

$$P[\widehat{W}^{(T)'} \neq W^{(T)}] \leq p + T(\epsilon + \epsilon'/4 + \epsilon'')$$

The probability distribution of  $\widehat{W}^{(T)'}$  is independent of  $P_Z$ , so it must be the case that

$$2^{-n} \sum_{s \subseteq [n]} \|Q - Q'\|_1 \leq 2P[\widehat{W}^{(T)'} \neq W^{(T)} | P_Z = \star] + 2P[\widehat{W}^{(T)'} \neq W^{(T)} | P_Z \neq \star] \leq 4p + T(4\epsilon + \epsilon' + 4\epsilon'')$$

□

In particular, if we let  $(h, G)$  be a neural net,  $G_W$  be  $G$  with its edge weights changed to the elements of  $W$ ,  $L$  be a loss function,

$$\bar{f}^{(x,y)}(W) = L(\text{eval}_{(h, G_W)}(x) - y)$$

, and  $f^{(x,y)} = -\gamma \nabla \bar{f}^{(x,y)}$  for each  $x, y$  then this translates to the following.

**Corollary 10.** *Let  $(h, G)$  be a neural net with  $n$  inputs and  $m$  edges,  $G_W$  be  $G$  with its edge weights changed to the elements of  $W$ , and  $L$  be a loss function. Also, let  $\gamma, \sigma, B_0, B_1, B_2 > 0$  such that  $B_1 < 1/2m$ ,  $m\sqrt{2\sigma/\pi} < r \leq (1 - 2mB_1)/(2mB_2)$ , and  $T$  be a positive integer. Then, let  $\star$  be the uniform distribution on  $\mathbb{B}^{n+1}$ , and for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen randomly from  $\mathbb{B}^n$ . Next, let  $P_Z$  be a probability distribution on  $\mathbb{B}^{n+1}$  that is chosen by means of the following procedure. First, with probability  $1/2$ , set  $P_Z = \star$ . Otherwise, select a random  $S \subseteq [n]$  and set  $P_Z = \rho_S$ .*

*Now, let  $G'$  be  $G$  with each of its edge weights perturbed by an independently generated variable drawn from  $\mathcal{N}(0, \sigma I)$  and run*

*NoisyStochasticGradientDescentAlgorithm( $h, G', P_Z, L, \gamma, \infty, \mathcal{N}(0, [2mB_1 - m^2B_1^2]\sigma I), T$ ). Then, let  $p$  be the probability that there exists  $0 \leq i < T$  such that at least one of the following holds:*

1. *One of the first derivatives of  $L(\text{eval}_{(h, G_i)}(X_i) - Y_i)$  with respect to the edge weights has magnitude greater than  $B_0/\gamma$ .*

2. There exists a perturbation  $G'_i$  of  $G_i$  with no edge weight changed by more than  $r$  such that one of the second derivatives of  $L(\text{eval}_{(h,G'_i)}(X_i) - Y_i)$  with respect to the edge weights has magnitude greater than  $B_1/\gamma$ .
3. There exists a perturbation  $G'_i$  of  $G_i$  with no edge weight changed by more than  $2r$  such that one of the third derivatives of  $L(\text{eval}_{(h,G'_i)}(X_i) - Y_i)$  with respect to the edge weights has magnitude greater than  $B_2/\gamma$ .

Finally, let  $Q$  be the probability distribution of the final edge weights given that  $P_Z = \star$  and  $Q'_s$  be the probability distribution of the final edge weights given that  $P_Z = \rho_s$ . Then

$$2^{-n} \sum_{s \subseteq [n]} \|Q - Q'_s\|_1 \leq 4p + T(4\epsilon + \epsilon' + 4\epsilon'')$$

where

$$\begin{aligned} \epsilon &= \frac{4(m+2)m^2B_2\sqrt{2\sigma/\pi}/(1-mB_1) + 3m^5B_2^2\sigma/(1-mB_1)^2}{8} \\ &\quad + (1 - (1+mB_1)B_2mr)e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m} \\ \epsilon' &= 2^{-n/2} \cdot e^{2m(B_0+2rB_1+2r^2B_2)/\sqrt{2\pi}\sigma}/(1-2mB_1-2rmB_2) + 2e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m} \\ \epsilon'' &= e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m} \end{aligned}$$

**Corollary 11.** Let  $(h, G)$  be a neural net with  $n$  inputs and  $m$  edges,  $G_W$  be  $G$  with its edge weights changed to the elements of  $W$ ,  $L$  be a loss function, and  $B > 0$ . Next, define  $\gamma$  such that  $0 < \gamma \leq \pi n/80m^2B$ , and let  $T$  be a positive integer. Then, let  $\star$  be the uniform distribution on  $\mathbb{B}^{n+1}$ , and for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen randomly from  $\mathbb{B}^n$ . Next, let  $P_Z$  be a probability distribution on  $\mathbb{B}^{n+1}$  that is chosen by means of the following procedure. First, with probability  $1/2$ , set  $P_Z = \star$ . Otherwise, select a random  $S \subseteq [n]$  and set  $P_Z = \rho_S$ .

Next, set  $\sigma = \left(\frac{40m\gamma B}{n}\right)^2/2\pi$ . Now, let  $G'$  be  $G$  with each of its edge weights perturbed by an independently generated variable drawn from  $\mathcal{N}(0, \sigma I)$  and run *NoisyStochasticGradientDescentAlgorithm* $(h, G', P_Z, L, \gamma, \infty, \mathcal{N}(0, [2mB\gamma - m^2B^2\gamma^2]\sigma I), T)$ . Let  $p$  be the probability that there exists  $0 \leq i < T$  such that there exists a perturbation  $G'_i$  of  $G_i$  with no edge weight changed by more than  $160m^2\gamma B/\pi n$  such that one of the first three derivatives of  $L(\text{eval}_{(h,G'_i)}(X_i) - Y_i)$  with respect to the edge weights has magnitude greater than  $B$ . Finally, let  $Q$  be the probability distribution of the final edge weights given that  $P_Z = \star$  and  $Q'_s$  be the probability distribution of the final edge weights given that  $P_Z = \rho_s$ . Then

$$2^{-n} \sum_{s \subseteq [n]} \|Q - Q'_s\|_1 \leq 4p + T(720m^4B^2\gamma^2/\pi n + 14[e/4]^{n/4})$$

*Proof.* First, set  $r = 80m^2\gamma B/\pi n$ . Also, set  $B_1 = B_2 = B_3 = \gamma B$ . By the previous corollary, we have that

$$2^{-n} \sum_{s \subseteq [n]} \|Q - Q'_s\|_1 \leq 4p + T(4\epsilon + \epsilon' + 4\epsilon'')$$

where

$$\epsilon = \frac{4(m+2)m^2B_2\sqrt{2\sigma/\pi}/(1-mB_1) + 3m^5B_2^2\sigma/(1-mB_1)^2}{8} \\ + (1 - (1 + mB_1)B_2mr)e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m}$$

$$\epsilon' = 2^{-n/2} \cdot e^{2m(B_0+2rB_1+2r^2B_2)/\sqrt{2\pi\sigma}}/(1-2mB_1-2rmB_2) + 2e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m}$$

$$\epsilon'' = e^{-(r/2\sqrt{\sigma}-m/\sqrt{2\pi})^2/m}$$

If  $720m^4B^2\gamma^2/\pi n \geq 2$ , then the conclusion of this corollary is uninterestingly true. Otherwise,  $\epsilon \leq 180m^4\gamma^2B^2/\pi n + \epsilon''$ . Either way,  $\epsilon' \leq 4[e/4]^{n/4} + 2\epsilon''$ , and  $\epsilon'' \leq e^{-m/2\pi}$ .  $m \geq n$  and  $e^{1/2\pi} \geq [4/e]^{1/4}$ , so  $\epsilon'' \leq [e/4]^{n/4}$ . The desired conclusion follows.  $\square$

That allows us to prove the following elaboration of theorem 3.

**Theorem 14.** *Let  $(h, G)$  be a neural net with  $n$  inputs and  $m$  edges,  $G_W$  be  $G$  with its edge weights changed to the elements of  $W$ ,  $L$  be a loss function, and  $B > 0$ . Next, define  $\gamma$  such that  $0 < \gamma \leq \pi n/80m^2B$ , and let  $T$  be a positive integer. Then, for each  $s \subseteq [n]$ , let  $\rho_s$  be the probability distribution of  $(X, p_s(X))$  when  $X$  is chosen randomly from  $\mathbb{B}^n$ . Now, select  $S \subseteq [n]$  at random. Next, set  $\sigma = \left(\frac{40m\gamma B}{n}\right)^2/2\pi$ . Now, let  $G'$  be  $G$  with each of its edge weights perturbed by an independently generated variable drawn from  $\mathcal{N}(0, \sigma I)$  and run  $\text{NoisyStochasticGradientDescentAlgorithm}(h, G', P_Z, L, \gamma, \infty, \mathcal{N}(0, [2mB\gamma - m^2B^2\gamma^2]\sigma I), T)$ . Let  $p$  be the probability that there exists  $0 \leq i < T$  such that there exists a perturbation  $G'_i$  of  $G_i$  with no edge weight changed by more than  $160m^2\gamma B/\pi n$  such that one of the first three derivatives of  $L(\text{eval}_{(h, G_i)}(X_i) - Y_i)$  with respect to the edge weights has magnitude greater than  $B$ . For a random  $X \in \mathbb{B}^n$ , the probability that the resulting net computes  $p_S(X)$  correctly is at most  $1/2 + 2p + T(360m^4B^2\gamma^2/\pi n + 7[e/4]^{n/4})$ .*

*Proof.* Let  $Q'_s$  be the probability distribution of the resulting neural net given that  $S = s$ , and let  $Q$  be the probability distribution of the net output by  $\text{NoisyStochasticGradientDescentAlgorithm}(h, G', \star, L, \gamma, \infty, \mathcal{N}(0, [2mB\gamma - m^2B^2\gamma^2]\sigma I), T)$ , where  $\star$  is the uniform distribution on  $\mathbb{B}^{n+1}$ . Also, for each  $(x, y) \in \mathbb{B}^{n+1}$ , let  $R(x, y)$  be the set of all neural nets that output  $y$  when given  $x$  as input. The probability that the neural net in question computes  $p_S(X)$  correctly is at most

$$2^{-2n} \sum_{s \subseteq [n], x \in \mathbb{B}^n} P_{G \sim Q'_s}[(f, G) \in R_{x, p_S(y)}] \\ \leq 2^{-2n} \sum_{s \subseteq [n], x \in \mathbb{B}^n} P_{G \sim Q}[(f, G) \in R_{x, p_S(x)}] + \|Q - Q'_s\|_1/2 \\ \leq 1/2 + 2p + T(360m^4B^2\gamma^2/\pi n + 7[e/4]^{n/4})$$

$\square$

## 7 Universality of deep learning

In previous sections, we were attempting to show that under some set of conditions a neural net trained by SGD is unable to learn a function that is reasonably learnable. However, there

are some fairly reasonable conditions under which we actually can use a neural net trained by SGD to learn any function that is reasonably learnable. More precisely, we claim that given any probability distribution of functions from  $\mathbb{B}^n \rightarrow \mathbb{B}$  such that there exists an algorithm that learns a random function drawn from this distribution with accuracy  $1/2 + \epsilon$  using a polynomial amount of time, memory, and samples, there exists a series of polynomial-sized neural networks that can be constructed in polynomial time and that can learn a random function drawn from this distribution with an accuracy of at least  $1/2 + \epsilon$  after being trained by SGD on a polynomial number of samples.

## 7.1 Emulation of Arbitrary Algorithms 1

It is easiest to show that we can emulate an arbitrary learning algorithm by running SGD on a neural net if we do not require the neural net to be normal or require a low learning rate. In that case, we use the following argument. Any learning algorithm that uses polynomial time and memory can be reexpressed as a circuit that computes a predicted output and new values for its memory given the actual output from the old values of its memory and an input.

Our neural net for emulating this algorithm will work as follows. First of all, for every bit of memory that the original algorithm uses,  $b_i$ , our net will have a corresponding vertex  $v_{b[i]}$  such that the weight of the edge from the constant vertex to  $v_{b[i]}$  will encode the current value of the bit. Secondly, the net will contain a section that computes the predicted value of the output, and the desired new values of each input contingent on each possible value of the actual output. This section will also work in such a way that when the net is evaluated the values of the section's output vertices will always be generated by computing the evaluation function on an input that it has a derivative of 0 on in order to ensure that nothing can backpropagate through this section, and thus that its edge weights will never change. Thirdly, the net will contain a section that in each step decides whether the net will actually try to get the output right, or whether it will try to learn more about the function. Generally, this section will attempt to learn for a fixed number of steps and then start seriously estimating the output. This section will also be backpropagation proofed.

Fourth, there will be vertices with edges from the vertex computing the desired output for the network and edges to the official output vertex, with edge weights chosen such that if it sets the output correctly the edge weights do not change, and if it sets it incorrectly, these edge weights change in ways that effectively cancel out. Fifth, for every bit of memory that the original algorithm uses, there will be two paths connecting  $v_{b[i]}$  to the output vertex. The middle vertices of these paths will also have control paths leading from the part of the network that determines what to change the values in memory to. It will be set up in such a way that these vertices will always evaluate to 0, but depending on the values of the vertices they are connected to by the control paths, their values might or might not have a nonzero derivative with respect to the input they receive from  $v_{b[i]}$ . This will allow the network to control whether or not backpropagation can change the value of  $b_i$ .

In steps where the network tries to learn more about the function, the network will randomly guess an output, set the value of the output vertex to the opposite of the value it guessed, and ensure that the output has a nonzero derivative with respect to any bits it wants to change the value of if it guessed correctly. That way, if it guesses wrong about the output, the loss function and its derivative are 0 so nothing changes. If it guesses right, then the values in memory change in exactly the desired manner. No matter what the true output is, it has a  $1/2$  chance of guessing right, so it is still updating based on samples drawn from the correct probability distribution. So, running SGD on the neural network described emulates the desired algorithm.



## 7.2 Emulation of Arbitrary Algorithms 2

Of course, the previous result uses choices of a neural net and SGD parameters that are in many ways unreasonable. The activation function is badly behaved, many of the vertices do not have edges from the constant vertex, and the learning rate is deliberately chosen to be so high that it keeps overshooting the minima. If one wanted to do something normal with a neural net trained by SGD one is unlikely to do it that way, and using it to emulate an algorithm is much less efficient than just running the algorithm directly, so this is unlikely to come up.

In order to emulate a learning algorithm with a more reasonable neural net and choice of parameters, we will need to use the following ideas in addition to the ideas from the previous result. First of all, we can control which edges tend to have their weights change significantly by giving edges that we want to change a very low starting weight and then putting high weight edges after them to increase the derivative of the output with respect to them. Secondly, rather than viewing the algorithm we are trying to emulate as a fixed circuit, we will view it as a series of circuits that each compute a new output and new memory values from the previous memory values and the current inputs. Thirdly, a lower learning rate and tighter restrictions on how quickly the network can change prevent us from setting memory values in one step. Instead, we initialize the memory values to a local maximum so that once we perturb them, even slightly, they will continue to move in that direction until they take on the final value. Fourth, in most steps the network will not try to learn anything, so that with high probability all memory values that were set in one step will have enough time to stabilize before the algorithm tries to adjust anything else. Finally, once we have gotten to the point that the algorithm is ready to approximate the function, its estimates will be connected to the output vertex, and the output will gradually become more influenced by it over time as a basic consequence of SGD.

## 8 Acknowledgements

This research was partly supported by the NSF CAREER Award CCF-1552131 and the Google Faculty Research Award. We thank Sebastien Bubeck for discussions on coordinate descent, Jean-Baptiste Cordonnier for helping with the experiment of Section 1.2, and Sanjeev Arora for feedback on the first manuscript.

## References

- [AB18] Emmanuel Abbe and Enric Boix, *An Information-Percolation Bound for Spin Synchronization on General Graphs*, arXiv e-prints (2018), arXiv:1806.03227. 17
- [All96] Eric Allender, *Circuit complexity before the dawn of the new millennium*, Foundations of Software Technology and Theoretical Computer Science (Berlin, Heidelberg) (V. Chandru and V. Vinay, eds.), Springer Berlin Heidelberg, 1996, pp. 1–18. 6
- [BFJ<sup>+</sup>94] Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich, *Weakly learning dnf and characterizing statistical query learning using fourier analysis*, Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '94, ACM, 1994, pp. 253–262. 6
- [BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman, *Noise-tolerant learning, the parity problem, and the statistical query model*, J. ACM **50** (2003), no. 4, 506–519. 4, 10

- [BOY17] Paul Beame, Shayan Oveis Gharan, and Xin Yang, *Time-Space Tradeoffs for Learning from Small Test Spaces: Learning Low Degree Polynomial Functions*, arXiv e-prints (2017), arXiv:1708.02640. 7
- [CG88] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, SIAM Journal on Computing **17** (1988), no. 2, 230–261. 7
- [CLB17] Zhengdao Chen, Xiang Li, and Joan Bruna, *Supervised Community Detection with Line Graph Neural Networks*, arXiv e-prints (2017), arXiv:1705.08415. 14
- [DSS16] Amit Daniely and Shai Shalev-Shwartz, *Complexity theoretic limitations on learning dnf’s*, 29th Annual Conference on Learning Theory (Columbia University, New York, New York, USA) (Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, eds.), Proceedings of Machine Learning Research, vol. 49, PMLR, 23–26 Jun 2016, pp. 815–830. 3
- [FGR<sup>+</sup>17] Vitaly Feldman, Elena Grigorescu, Lev Reyzin, Santosh S. Vempala, and Ying Xiao, *Statistical algorithms and a lower bound for detecting planted cliques*, J. ACM **64** (2017), no. 2, 8:1–8:37. 11
- [FGV17] Vitaly Feldman, Cristóbal Guzmán, and Santosh Vempala, *Statistical query algorithms for mean vector estimation and stochastic convex optimization*, Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (Philadelphia, PA, USA), SODA ’17, Society for Industrial and Applied Mathematics, 2017, pp. 1265–1277. 6
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, 2016, <http://www.deeplearningbook.org>. 3
- [GHJY15] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan, *Escaping from saddle points — online stochastic gradient for tensor decomposition*, Proceedings of The 28th Conference on Learning Theory (Paris, France) (Peter Grnwald, Elad Hazan, and Satyen Kale, eds.), Proceedings of Machine Learning Research, vol. 40, PMLR, 03–06 Jul 2015, pp. 797–842. 4, 9
- [GRT18] Sumegha Garg, Ran Raz, and Avishay Tal, *Extractor-based time-space lower bounds for learning*, Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (New York, NY, USA), STOC 2018, ACM, 2018, pp. 990–1002. 7
- [Hås87] Johan Håstad, *Computational limitations of small-depth circuits*, MIT Press, Cambridge, MA, USA, 1987. 6
- [HDY<sup>+</sup>12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups*, IEEE Signal Processing Magazine **29** (2012), no. 6, 82–97. 3
- [HRS16] Moritz Hardt, Benjamin Recht, and Yoram Singer, *Train faster, generalize better: Stability of stochastic gradient descent*, Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML’16, JMLR.org, 2016, pp. 1225–1234. 9

- [HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015, 2015, pp. 1026–1034. 4, 9
- [Kea98] Michael Kearns, *Efficient noise-tolerant learning from statistical queries*, J. ACM **45** (1998), no. 6, 983–1006. 6, 11
- [KLY18] R. Kleinberg, Y. Li, and Y. Yuan, *An Alternative View: When Does SGD Escape Local Minima?*, ArXiv e-prints (2018). 9
- [KRT17] Gillat Kol, Ran Raz, and Avishay Tal, *Time-space hardness of learning sparse parities*, Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (New York, NY, USA), STOC 2017, ACM, 2017, pp. 1067–1080. 7
- [KS09] Adam R. Klivans and Alexander A. Sherstov, *Cryptographic hardness for learning intersections of halfspaces*, J. Comput. Syst. Sci. **75** (2009), no. 1, 2–12. 3
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems 25 (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2012, pp. 1097–1105. 3
- [LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324. 3
- [LBH15] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton, *Deep learning*, Nature **521** (2015), no. 7553, 436–444 (English (US)). 3
- [LeC16] Y. LeCun, Personal communication, 2016. 3
- [LSJR16] Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Recht, *Gradient descent only converges to minimizers*, 29th Annual Conference on Learning Theory (Columbia University, New York, New York, USA) (Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, eds.), Proceedings of Machine Learning Research, vol. 49, PMLR, 23–26 Jun 2016, pp. 1246–1257. 9
- [MP87] Marvin Minsky and Seymour Papert, *Perceptrons - an introduction to computational geometry*, MIT Press, 1987. 6
- [MRT12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, *Foundations of machine learning*, The MIT Press, 2012. 12
- [Par94] Ian Parberry, *Circuit complexity and neural networks*, MIT Press, Cambridge, MA, USA, 1994. 3
- [PGC<sup>+</sup>17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, *Automatic differentiation in pytorch*, NIPS-W, 2017. 4
- [PP17] Ioannis Panageas and Georgios Piliouras, *Gradient descent only converges to minimizers: Non-isolated critical points and invariant regions*, ITCS, 2017. 9

- [Raz16] R. Raz, *Fast Learning Requires Good Memory: A Time-Space Lower Bound for Parity Learning*, ArXiv e-prints (2016). 3, 7
- [Raz17] Ran Raz, *A time-space lower bound for a large class of learning problems*, 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS) (2017), 732–742. 7
- [Raz18] R. Raz, Private communications, December 2018. 7
- [Reg05] Oded Regev, *On lattices, learning with errors, random linear codes, and cryptography*, Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '05, ACM, 2005, pp. 84–93. 4, 10
- [RRT17] M. Raginsky, A. Rakhlin, and M. Telgarsky, *Non-convex learning via Stochastic Gradient Langevin Dynamics: a nonasymptotic analysis*, ArXiv e-prints (2017). 4
- [Sha18] Ohad Shamir, *Distribution-specific hardness of learning neural networks*, Journal of Machine Learning Research **19** (2018), no. 32, 1–29. 3, 7
- [SSBD14] Shai Shalev-Shwartz and Shai Ben-David, *Understanding machine learning: From theory to algorithms*, Cambridge University Press, New York, NY, USA, 2014. 3, 4, 9
- [SSS17] S. Shalev-Shwartz, O. Shamir, and S. Shammah, *Failures of Gradient-Based Deep Learning*, ArXiv e-prints (2017). 7, 15, 31
- [SVW15] Jacob Steinhardt, Gregory Valiant, and Stefan Wager, *Memory, communication, and statistical queries*, Electronic Colloquium on Computational Complexity, 2015. 7
- [WT11] Max Welling and Yee Whye Teh, *Bayesian learning via stochastic gradient langevin dynamics*, Proceedings of the 28th International Conference on International Conference on Machine Learning (USA), ICML'11, Omnipress, 2011, pp. 681–688. 4
- [ZBH<sup>+</sup>16] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals, *Understanding deep learning requires rethinking generalization*, CoRR **abs/1611.03530** (2016). 9