# Failures of Deep Learning

Shai Shalev-Shwartz[1], Ohad Shamir[2], and Shaked Shammah[1]

[1]School of Computer Science and Engineering, The Hebrew University
[2]Weizmann Institute

**Abstract**

In recent years, Deep Learning has become the go-to solution for a broad range of applications, often outperforming state-of-the-art. However, it is important, for both theoreticians and practitioners, to gain a deeper understanding of the difficulties and limitations associated with common approaches and algorithms. We describe four families of problems for which some of the commonly used existing algorithms fail or suffer significant difficulty. We illustrate the failures through practical experiments, and provide theoretical insights explaining their source, and how they might be remedied.

## 1 Introduction

The success stories of deep learning form an ever lengthening list of practical breakthroughs and state-of-the-art performances, ranging the fields of computer vision [20, 11, 22, 30], audio and natural language processing and generation [4, 12, 9, 31], as well as robotics [21, 23], to name just a few. The list of success stories can be matched and surpassed by a list of practical "tips and tricks", from different optimization algorithms, parameter tuning methods [27, 19], initialization schemes [8], architecture designs [28], loss functions, data augmentation [20] and so on.

The current theoretical understanding of deep learning is far from being sufficient for a rigorous analysis of the difficulties faced by practitioners. Progress must be made from both parties: from a practitioner's perspective, emphasizing the difficulties provides practical insights to the theoretician, which in turn, supplies theoretical insights and guarantees, further strengthening and sharpening practical intuitions and wisdom. In particular, understanding *failures* of existing algorithms is as important as understanding where they succeed.

Our goal in this paper is to present and discuss families of simple problems for which commonly used methods do not show as exceptional a performance as one might expect. We use empirical results and insights as a ground on which to build a theoretical analysis, characterising the sources of failure. Those understandings are aligned, and sometimes lead to, different approaches, either for an architecture, loss function, or an optimization scheme, and explain their superiority when applied to members of those families. The code for running the experiments is found in `https://github.com/shakedshammah/failures_of_DL`. See command lines at Appendix C.

We start off in Section 2 by discussing a class of simple learning problems for which the gradient information, central to deep learning algorithms, provably carries negligible information on the target function which we attempt to learn. This result is a property of the learning problems themselves, and holds for any specific network architecture one may choose for tackling the learning problem, implying that no gradient-based method is likely to succeed. Our analysis builds upon tools from the Statistical Query literature, and underscores one of the main deficiencies of Deep Learning: its reliance on local properties of the loss function, with the objective being of a global nature.

Next, in Section 3, we tackle the ongoing dispute between two common approaches to learning. Most, if not all, learning and optimization problems can be viewed as some structured set of sub-problems. The first approach, which we refer to as the "End-to-end" approach, will tend to solve all of the sub-problems together in one shot, by optimizing a single primary objective. The second approach, which we refer to as the "Decomposition" one, will tend to handle these sub-problems separately, solving each one by defining and optimizing additional objectives, and not rely solely on the primary objective. The benefits of the End-to-end approach, both in terms of requiring a smaller amount of labeling and prior knowledge, and perhaps enabling more expressive architectures, cannot be ignored. On the other hand, intuitively and empirically, the extra supervision injected through decomposition is helpful in the optimization process. We experiment with two setups in which application of the two approaches is possible, and the distinction between them is clear and intuitive. We observe, in both experiments, that an "End-to-end" approach can be much slower than a "Decomposition" method, and sometimes even does not show any progress. We analyze this gap by comparing, theoretically and empirically, the signal-to-noise ratio of the stochastic gradient estimate used by Stochastic Gradient Descent (SGD) for the two approaches.

In Section 4, we demonstrate the importance of both the network's architecture and the optimization algorithm on the training time. While the choice of architecture is usually studied in the context of its expressive power, we show that even when two architectures have the same expressive power for a given task, there may be a tremendous difference in the ability to optimize them. We analyze the required runtime of gradient descent for the two architectures through the lens of the condition number of the problem. We further show that conditioning techniques can yield additional orders of magnitude speedups. The experimental setup in this section is around a seemingly simple problem — encoding a piece-wise linear one-dimensional curve. Despite the simplicity of this problem, we show that following the common rule of "perhaps I should use a deeper/wider network[1]" does not significantly help here.

Finally, in Section 5, we question deep learning's reliance on "vanilla" gradient information for the optimization process. We previously discussed the deficiency of using a local property of the objective in directing global optimization. Here, we focus on a simple case in which it is possible to solve the optimization problem based on local information, but not in the form of a gradient. We experiment with architectures that contain activation functions with flat regions, which leads to the well known vanishing gradient problem. Practitioners take great care when working with such activation functions, and many heuristic tricks are applied in order to initialize the network's weights in non-flat areas of its activations. Here, we show that by using a different update rule, we manage to solve the learning problem efficiently. Convergence guarantees exist for a family of such functions. This shows the power of non-gradient-based optimization schemes in overcoming the limitations of gradient-based learning.

## 2 Failure due to Non-Informative Gradients

Most existing deep learning algorithms are gradient-based methods; namely, algorithms which optimize an objective through access to its gradient w.r.t. some weight vector $\mathbf{w}$, or estimates of the gradient. We consider a setting where the goal of this optimization process is to learn some underlying hypothesis class $\mathcal{H}$, of which one member, $h \in \mathcal{H}$, is responsible for labelling the data. This yields an optimization problem of the form

$$\min_{\mathbf{w}} F_h(\mathbf{w}).$$

The underlying assumption is that the gradient of the objective w.r.t. $\mathbf{w}$, $\nabla F_h(\mathbf{w})$, contains useful information regarding the target function $h$, and will help us make progress.

Below, we discuss a family of problems for which with high probability, at any fixed point, the gradient, $\nabla F_h(\mathbf{w})$, will be essentially the same regardless of the underlying target function $h$. Furthermore, we prove that this holds independently of the choice of architecture or parametrization, and using a deeper/wider network will not help. The family we study is that of compositions of linear and periodic functions, and we experiment with the classical problem of learning parities. Our empirical and theoretical study shows that indeed, if there's little information in the gradient, using it for learning cannot succeed.

### 2.1 Experiment

We begin with the simple problem of learning random parities: After choosing some $\mathbf{v}^* \in \{0,1\}^d$ uniformly at random, our goal is to train a predictor mapping $\mathbf{x} \in \{0,1\}^d$ to $y = (-1)^{\langle \mathbf{x}, \mathbf{v}^* \rangle}$, where $\mathbf{x}$ is uniformly distributed. In words, $y$ indicates whether the number of 1's in a certain subset of coordinates of $\mathbf{x}$ (indicated by $\mathbf{v}^*$) is odd or even.

[1]See `http://joelgrus.com/2016/05/23/fizz-buzz-in-tensorflow/` for the inspiration behind this humoristic quote.

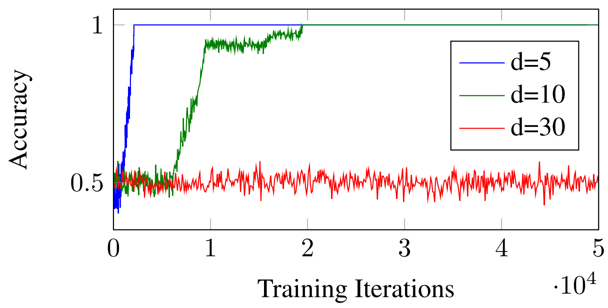


Figure 1: Parity Experiment: Accuracy as a function of the number of training iterations, for various input dimensions.

For our experiments, we use the hinge loss, and a simple network architecture of one fully connected layer of width $10d > \frac{3d}{2}$ with ReLU activations, and a fully connected output layer with linear activation and a single unit. Note that this class realizes the parity function corresponding to any $\mathbf{v}^*$ (see Lemma 3 in the appendix).

Empirically, as the dimension $d$ increases, so does the difficulty of learning, which can be measured in the accuracy we arrive at after a fixed number of training iterations, to the point where around $d = 30$, no advance beyond random performance is observed after reasonable time. Figure 1 illustrates the results.

## 2.2 Analysis

To formally explain the failure from a geometric perspective, consider the stochastic optimization problem associated with learning a target function $h$,

$$\min_{\mathbf{w}} F_h(\mathbf{w}) := \mathop{\mathbb{E}}_{\mathbf{x}} \left[\ell(p_{\mathbf{w}}(\mathbf{x}), h(\mathbf{x}))\right], \tag{1}$$

where $\ell$ is a loss function, $\mathbf{x}$ are the stochastic inputs (assumed to be vectors in Euclidean space), and $p_{\mathbf{w}}$ is some predictor parametrized by a parameter vector $\mathbf{w}$ (e.g. a neural network of a certain architecture). We will assume that $F$ is differentiable. A key quantity we will be interested in studying is the *variance* of the gradient of $F$ with respect to $h$, when $h$ is drawn uniformly at random from a collection of candidate target functions $\mathcal{H}$:

$$\mathrm{Var}(\mathcal{H}, F, \mathbf{w}) = \mathop{\mathbb{E}}_{h} \left\| \nabla F_h(\mathbf{w}) - \mathop{\mathbb{E}}_{h'} \nabla F_{h'}(\mathbf{w}) \right\|^2$$

Intuitively, this measures the expected amount of "signal" about the underlying target function contained in the gradient. As we will see later, this variance correlates with the difficulty of solving (1) using gradient-based methods[2].

The following theorem bounds this variance term.

**Theorem 1** *Suppose that*

- *$\mathcal{H}$ consists of real-valued functions $h$ satisfying $\mathbb{E}_{\mathbf{x}}[h^2(\mathbf{x})] \leq 1$, such that for any two distinct $h, h' \in \mathcal{H}$, $\mathbb{E}_{\mathbf{x}}[h(\mathbf{x})h'(\mathbf{x})] = 0$.*
- *$p_{\mathbf{w}}(\mathbf{x})$ is differentiable w.r.t. $\mathbf{w}$, and satisfies $\mathbb{E}_{\mathbf{x}}\left[\|\frac{\partial}{\partial \mathbf{w}} p_{\mathbf{w}}(\mathbf{x})\|^2\right] \leq G(\mathbf{w})^2$ for some scalar $G(\mathbf{w})$.*
- *The loss function $\ell$ in (1) is either the square loss $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ or a classification loss of the form $\ell(\hat{y}, y) = r(\hat{y} \cdot y)$ for some 1-Lipschitz function $r$, and the target function $h$ takes values in $\{\pm 1\}$.*

*Then*

$$Var(\mathcal{H}, F, \mathbf{w}) \leq \frac{G(\mathbf{w})^2}{|\mathcal{H}|}.$$

[2]This should not be confused with the variance of gradient estimates used by SGD, which we discuss in Section 3.

The proof is given in Appendix A.1. The theorem implies that if we try to learn an unknown target function, possibly coming from a large collection of uncorrelated functions, then the sensitivity of the gradient to the target function at any point decreases linearly with $|\mathcal{H}|$.

Before we make a more general statement, let us return to the case of parities, and study it through the lens of this framework. Suppose that our target function is some parity function chosen uniformly at random, i.e. a random element from the set of $2^d$ functions $\mathcal{H} = \{\mathbf{x} \mapsto (-1)^{\langle \mathbf{x}, \mathbf{v}^* \rangle} : \mathbf{v}^* \in \{0,1\}^d\}$. These are binary functions, which are easily seen to be mutually orthogonal: Indeed, for any $\mathbf{v}, \mathbf{v}'$,

$$\mathbb{E}_{\mathbf{x}}\left[(-1)^{\langle \mathbf{x}, \mathbf{v} \rangle}(-1)^{\langle \mathbf{x}, \mathbf{v}' \rangle}\right] = \mathbb{E}_{\mathbf{x}}\left[(-1)^{\langle \mathbf{x}, \mathbf{v}+\mathbf{v}' \rangle}\right]$$
$$= \prod_{i=1}^{d} \mathbb{E}\left[(-1)^{x_i(v_i+v_i')}\right] = \prod_{i=1}^{d} \frac{(-1)^{v_i+v_i'} + (-1)^{-(v_i+v_i')}}{2}$$

which is non-zero if and only if $\mathbf{v} = \mathbf{v}'$. Therefore, by Theorem 1, we get that $\mathrm{Var}(\mathcal{H}, F, \mathbf{w}) \leq G(\mathbf{w})^2/2^d$ – that is, exponentially small in the dimension $d$. By Chebyshev's inequality, this implies that the gradient at any point $\mathbf{w}$ will be extremely concentrated around a fixed point independent of $h$.

This phenomenon of exponentially-small variance can also be observed for other distributions, and learning problems other than parities. Indeed, in [26], it was shown that this also holds in a more general setup, when the output $y$ corresponds to a linear function composed with a periodic one, and the input $\mathbf{x}$ is sampled from a smooth distribution:

**Theorem 2 (Shamir 2016)** *Let $\psi$ be a fixed periodic function, and let $\mathcal{H} = \{\mathbf{x} \mapsto \psi(\mathbf{v}^{*\top}\mathbf{x}) : \|\mathbf{v}^*\| = r\}$ for some $r > 0$. Suppose $\mathbf{x} \in \mathbb{R}^d$ is sampled from an arbitrary mixture of distributions with the following property: The square root of the density function $\varphi$ has a Fourier transform $\hat{\varphi}$ satisfying $\frac{\int_{\mathbf{x}:\|\mathbf{x}\|>r} \hat{\varphi}^2(\mathbf{x})d\mathbf{x}}{\int_{\mathbf{x}} \hat{\varphi}^2(\mathbf{x})d\mathbf{x}} \leq \exp(-\Omega(r))$. Then if $F$ denotes the objective function with respect to the squared loss,*

$$\mathit{Var}(\mathcal{H}, F, \mathbf{w}) \leq O\left(\exp(-\Omega(d)) + \exp(-\Omega(r))\right).$$

The condition on the Fourier transform of the density is generally satisfied for smooth distributions (e.g. arbitrary Gaussians whose covariance matrices are positive definite, with all eigenvalues at least $\Omega(1/r)$). Thus, the bound is extremely small as long as the norm $r$ and the dimension $d$ are moderately large, and indicates that the gradients contains little signal on the underlying target function.

Based on these bounds, one can also formally prove that a gradient-based method will fail in returning a reasonable predictor, unless the number of iterations is exponentially large in $r, d$: By Chebyshev's inequality, these bounds imply that with overwhelming probability, at any point $\mathbf{w}$ (say, the point at which our gradient-based method is initialized at), the gradient $\nabla F(\mathbf{w})$ will be fixed independent of the underlying target function (up to a deviation which for moderate $r, d$ is much smaller than machine precision, hence can be assumed non-existent on any realistic computing platform). As a result, even if we perform exact gradient descent, the update in the first iteration will be independent of the underlying target function[3] . Repeating the argument and using a union bound, it follows that as long as the number of iterations is sub-exponential, the trajectory of the gradient descent procedure will be independent of the underlying target function, so we cannot expect it to achieve a reasonable result. Overall, this gives strong evidence that gradient-based methods indeed cannot learn random parities and linear-periodic functions.

We emphasize that these results hold *regardless of which class of predictors we use* (e.g. they can be arbitrarily complex neural networks) – the problem lies in using a gradient-based method to train them. Also, we note that the difficulty lies in the random choice of $\mathbf{v}^*$, and the problem is not difficult if $\mathbf{v}^*$ is known and fixed in advance (for example, for a full parity $\mathbf{v}^* = (1, \ldots, 1)$, this problem is known to be solvable with an appropriate LSTM network [14]).

Finally, we remark that the connection between parities, difficulty of learning and orthogonal functions is not new, and has already been made in the context of statistical query learning [18, 1]. This refers to algorithms which

[3]Formally, this requires an oracle-based model, where given a point $\mathbf{w}$, the algorithm receives the gradient at $\mathbf{w}$ up to some arbitrary error much smaller than machine precision. See [26, Theorem 4] for details.

are constrained to interact with data by receiving estimates of the expected value of some query over the underlying distribution (e.g. the expected value of the first coordinate), and it is well-known that parities cannot be learned with such algorithms. Recently, [6] have shown that gradient-based methods with an approximate gradient oracle can be implemented as a statistical query algorithm, which implies that gradient-based methods are indeed unlikely to solve learning problems which are known to be hard in the statistical queries framework, in particular parities. In the discussion on random parities above, we have simply made the connection between gradient-based methods and parities more explicit, by direct examination of gradients' variance w.r.t. the target function.

## 3 Decomposition vs. End-to-end

Many practical learning problems, and more generally, algorithmic problems, can be viewed as a structured composition of sub-problems. Applicable approaches for a solution can either be tackling the problem in an End-to-end manner, or by decomposition. Whereas for a traditional algorithmic solution, the "Divide-and-conquer" strategy is an obvious choice, the ability of deep learning to utilize big data and expressive architectures have made "End-to-end training" an attractive alternative. Prior results of End-to-end [21, 9] and decomposition and added feedback [13, 29, 2] approaches show success in both directions. We try to answer the following questions: what is the price of the rather appealing "End-to-end" approach? Is letting a network "learn by itself" such a bad idea? When is it necessary, or worth the effort, to "help" it?

There are various aspects which can be considered in this context. For example, [25] analyzed the difference between the approaches from the sample complexity point of view. Here, we focus on the optimization aspect, showing that training SGD with the "End-to-end" approach might suffer from a low signal-to-noise ratio of the gradient estimates, which may significantly affect the training time. Helping the SGD process by decomposing the problem leads to much faster training. We present two experiments, motivated by questions each and every practitioner must answer when facing a new, non trivial problem; what exactly is the needed data, what architecture is planned to be used, and what is the right distribution of development efforts, are all correlated questions with no clear answer. Our experiments and analysis show that taking the wrong choice may be expensive.

### 3.1 Experiment I

Our first experiment compares the two approaches in a computer vision setting, where Convolutional Neural Networks (CNN) had become the most widely used and successful algorithmic architectures. We define a family of problems, parameterized by $k \in \mathbb{N}$, and show that as grows $k$, grows the gap between an "End-to-end" approach and a "Decomposition" one.

Let $X_1$ denote the space of $28 \times 28$ binary images, with a distribution $D$ defined by the following sampling procedure:

- Sample $\theta \sim U([0, \pi])$, $l \sim U([5, 28-5])$, $(x, y) \sim U([0, 27])^2$.
- The image $\mathbf{x}_{\theta,l,(x,y)}$ associated with the above sample is set to $0$ everywhere, except for a straight line of length $l$, centered at $(x, y)$, and rotated at an angle $\theta$. Note that as the images space is discrete, we round the values corresponding to the points on the lines to the closest integer coordinate.

Let $y : X_1 \to \{\pm 1\}$ be defined by

$$y(\mathbf{x}_{\theta,l,(x,y)}) = \begin{cases} 1 & \text{if } \theta < \pi/2 \\ -1 & \text{otherwise} \end{cases} .$$

Figure 2 shows a few examples of instances. We can now define the problem for each $k$. The sample space for our problem is $X := X_1^k$, namely, the $k$-tuples of elements from $X_1$, where each of the tuple's entries is sampled i.i.d. by the above procedure. We denote such tuples by $\mathbf{v} := (\mathbf{x}^{(1)}, ..., \mathbf{x}^{(k)})$. Each such tuple is associated with a vector in $\{\pm 1\}^k$, defined by the corresponding $y$ values for the tuple's entries, and denoted $y(\mathbf{v}) := (y(\mathbf{x}^{(1)}), ..., y(\mathbf{x}^{(k)}))$. The labeling function, $\tilde{y} : X \to \{\pm 1\}$, is defined to be the parity function over $y(\mathbf{v})$, namely, $\tilde{y}(\mathbf{v}) = \prod_{j=1...k} y(\mathbf{x}^{(j)})$.

Many architectures of DNN can be used for predicting $\tilde{y}(\mathbf{v})$ given $\mathbf{v}$. A natural "high-level" choice can be:

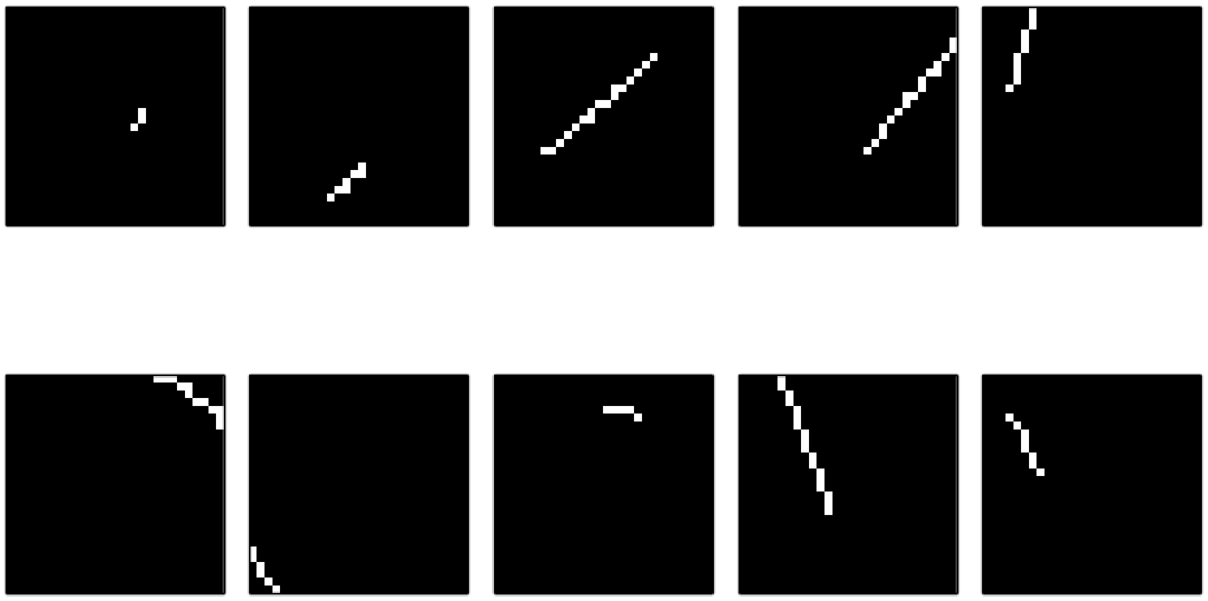


Figure 2: Section 3.1's experiment - examples of samples from $X_1$. The $y$ values of the top and bottom rows are $1$ and $-1$, respectively.

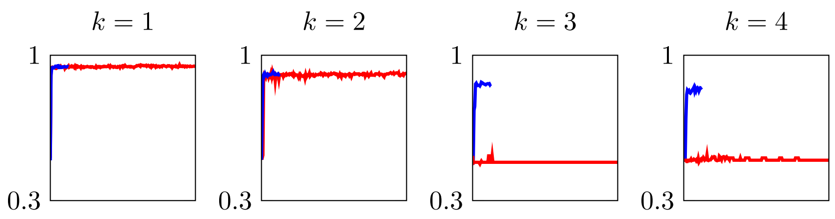


Figure 3: Performance comparison, Section 3.1's experiment. The red and blue curves correspond to the "End-to-end" and "Decomposition" approaches, respectively. The plots show the zero-one accuracy with respect to the "primary" objective, over a held out test set, as a function of training iterations. We have trained "End-to-end" and "Decomposition" for 20000 and 2500 SGD iterations, respectively.

- Feed each of the images, separately, to a single CNN (of some standard specific architecture, for example, LeNet-like), denoted $N^{(1)}_{\mathbf{w}_1}$ and parameterized by its weights vector $\mathbf{w}_1$, outputting a single scalar, which can be regarded as a "score".
- Concatenate the "scores" of a tuple's entries, transform them to the range $[0, 1]$ using a sigmoid function, and feed the resulting vector into another network, $N^{(2)}_{\mathbf{w}_2}$, of a similar architecture to the one defined in Section 2, outputting a single "tuple-score", which can then be thresholded for obtaining the binary prediction.

Let the whole architecture be denoted $N_{\mathbf{w}}$. Assuming that $N^{(1)}$ is expressive enough to provide, at least, a weak learner for $y$ (a reasonable assumption), and using Lemma 3 to obtain the sufficiency of $N^{(2)}$ in terms of expressibility for the parity function, we obtain that this architecture has the potential for good performance.

The final piece of the experimental setting is the choice of a loss function. Clearly, the "primary" loss which we'd like to minimize is the expected zero-one loss over the prediction, $N_{\mathbf{w}}(\mathbf{v})$, and the label, $\tilde{y}(\mathbf{v})$, namely:

$$\tilde{L}_{0-1}(\mathbf{w}) := \mathop{\mathbb{E}}_{\mathbf{v}}\left[N_{\mathbf{w}}(\mathbf{v}) \neq \tilde{y}(\mathbf{v})\right]$$

A "secondary" loss which may be considerable, is the zero-one loss over the prediction of $N^{(1)}_{\mathbf{w}_1}(\mathbf{x})$ and the respective $y(\mathbf{x})$ value:

$$L_{0-1}(\mathbf{w}_1) := \mathop{\mathbb{E}}_{\mathbf{x}}\left[N^{(1)}_{\mathbf{w}_1}(\mathbf{x}) \neq y(\mathbf{x})\right]$$

Let $\tilde{L}, L$ be some differentiable surrogates for $\tilde{L}_{0-1}, L_{0-1}$. A classical "End-to-end" approach will be to minimize $\tilde{L}$, and only it; this is our "primary" objective. We have no explicit desire for $N^{(1)}$ to output any specific value, and hence $L$ is, a priori, irrelevant. A "Decomposition" approach would be to minimize both losses, under the assumption that $L$ can "direct" $\mathbf{w}_1$ towards an "area" in which we know that the resulting outputs of $N^{(1)}$ can be separated by $N^{(2)}$. Note that using $L$ is only possible when the $y$ values are known to us.

Empirically, when comparing based on the "primary" objective, we see significant inferiority of "End-to-end" with respect to "Decomposition". Using "Decomposition", we quickly arrive at a good solution, regardless of the tuple's length, $k$ (as long as $k$ is in the range where perfect input to $N^{(2)}$ is solvable by SGD, as described in Section 2). However, using "End-to-end" works only for very small $k$ values, and completely fails for $k \geq 3$. This may be somewhat surprising, as the "End-to-end" approach optimizes exactly the comparison criterion, with no additional irrelevant objectives. Figure 3 illustrates the results.

## 3.2 Experiment II

We now consider a more synthetic setup, which will later allow us crisp analysis and explanation for the empirical results. Consider the problem of training a predictor, which given a "positive media reference" $\mathbf{x}$ to a certain stock option, will distribute our assets between the $k = 500$ stocks in the S&P500 index in some manner. One can, again, come up with two rather different strategies for solving the problem.

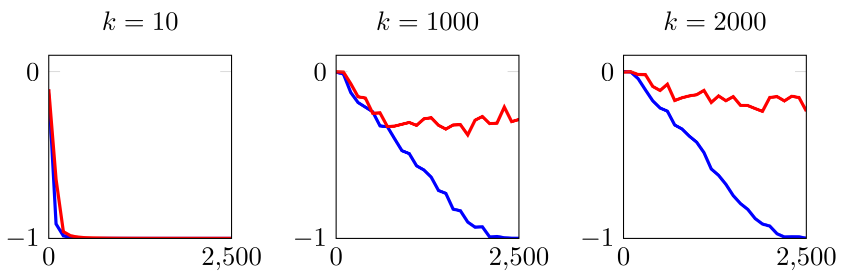


Figure 4: Decomposition vs. End-to-end Experiment: Loss as a function of the number of training iterations, for input dimension $d = 1000$ and for various $k$ values. The red and blue curves correspond to the losses of the End-to-end and Decomposition estimators, respectively.

- An "End-to-end" approach: train a deep network $N_{\mathbf{w}}$ that given $\mathbf{x}$ outputs a distribution over the $k$ stocks. The objective for training is maximizing the gain obtained by allocating our money according to this distribution.
- A "Decomposition" approach: train a deep network $N_{\mathbf{w}}$ that given $\mathbf{x}$ outputs a single stock, $y \in [k]$, whose future gains are the most positively correlated to $\mathbf{x}$. Of course, we may need to gather extra labeling for training $N_{\mathbf{w}}$ based on this criterion.

To make a crisp analysis and experiment, we make the (non-realistic) assumption that every instance of media reference is strongly and positively correlated to a single stock $y \in [k]$, and it has no correlation with future performance of other stocks. This obviously makes our problem rather toyish; the stock exchange and media worlds have highly complicated correlations. However, it indeed rises from, and is motivated by, practical problems.

To examine the problem in a simple and theoretically clean manner, we design a synthetic experiment defined by the following optimization problem: Let $X \times Z \subset \mathbb{R}^d \times \{\pm 1\}^k$ be the sample space, and let $y : X \to [k]$ be some labelling function. We would like to learn a mapping $N_{\mathbf{w}} : X \to S^{k-1}$, with the objective being:

$$\min_{\mathbf{w}} L(\mathbf{w}) := \mathop{\mathbb{E}}_{\mathbf{x},\mathbf{z} \sim X \times Z} \left[ -\mathbf{z}^\top N_{\mathbf{w}}(\mathbf{x}) \right].$$

To connect this to our story, $N_{\mathbf{w}}(\mathbf{x})$ is our asset distribution, $\mathbf{z}$ indicates the future performance of the stocks, and thus, we are seeking minimization of our expected future negative gains, or in other words, maximization of expected profit. We further assume that given $\mathbf{x}$, the coordinate $\mathbf{z}_{y(\mathbf{x})}$ equals 1, and the rest of the coordinates are sampled i.i.d from the uniform distribution over $\{\pm 1\}$.

Whereas in the previous experiment, the difference between the "End-to-end" and "Decomposition" approaches could be summarized by a different loss function choice, in this experiment, the difference will boil down to the different gradient estimators we would use, where we are again taking as a given fact that exact gradient computations are expensive for large-scale problems, implying the method of choice to be SGD. For the purpose of the experimental discussion, let us write the two estimators explicitly as two unconnected update rules. We will later analyze their (equal) expectation.

For an "End-to-end" approach, we sample a pair $(\mathbf{x}, \mathbf{z})$, and use $\nabla_{\mathbf{w}}(-\mathbf{z}^\top N_{\mathbf{w}}(\mathbf{x}))$ as a gradient estimate. It is clear that this is an unbiased estimator of the gradient.

For a "Decomposition" approach, we sample a pair $(\mathbf{x}, \mathbf{z})$, *completely ignore* $\mathbf{z}$, and instead, pay the extra costs and gather the required labelling to get $y(\mathbf{x})$. We will then use $\nabla_{\mathbf{w}}(-e_{y(\mathbf{x})}^\top N_{\mathbf{w}}(\mathbf{x}))$ as a gradient estimate. It will be shown later that this too is an unbiased estimator of the gradient.

Figure 4 clearly shows that optimizing using the "End-to-end" estimator is inferior to working with the "Decomposition" one, in terms of training time and final accuracy, to the extent that for large $k$, the "End-to-end" estimator cannot close the gap in performance in reasonable time.

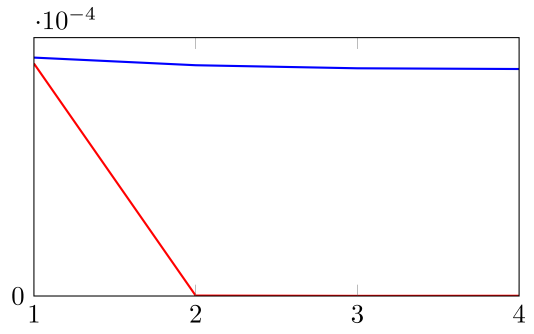


Figure 5: Section 3.1's experiment: comparing the SNR for the "End-to-end" approach (red) and the "Decomposition" approach (blue), as a function of $k$. The SNR of the "End-to-end" approach, for $k \geq 3$, is below the precision obtained by a 32-bit floating point representation.

## 3.3 Analysis

We approach the analysis through examination of the Signal to Noise Ratio (SNR) of the two stochastic gradient estimators, showing that the "End-to-end" one is significantly inferior in that regard.

Let us first gain intuition with regard to the first experiment, by empirically measuring:

- The signal's strength, namely, the squared norm of the gradients' mean (w.r.t. the losses $\tilde{L}, L$), which we estimate by SGD, denoted $\|\nabla \tilde{L}\|^2$, $\|\nabla L\|^2$.
- The signal's variance, namely, $\mathbb{E}_{\mathbf{v}} \|\hat{\nabla} L - \nabla L\|^2$, where $\hat{\nabla} L$ is the estimate of $\nabla L$ obtained by differentiating w.r.t. to a single sample $\mathbf{v}$. We define this similarly for $\tilde{L}$.

Figure 5 shows an estimation of the SNR for the last layer of $N^{(1)}$, for the two approaches, as a function of $k$, where our estimation is over the random initialization of the network (clearly, there are weights $\mathbf{w}$ for which all gradients are 0, no matter what approach is used). It is easy to see that the "End-to-end" approach suffers from significantly lower SNR, and more importantly, it shows dependence on $k$, quickly falling below machine precision, whereas the "Decomposition" approach's SNR is constant. This gives strong empirical evidence for a connection between slow convergence and low SNR.

Equipped with the above empirical observation, we turn on to analytically examine the second experiment. First, let us show that indeed, both estimators are unbiased estimators of the true gradient. As stated above, it is clear, by definition of $L$, that the "End-to-end" estimator is an unbiased estimator of $\nabla_{\mathbf{w}} L(\mathbf{w})$. To observe this is also the case for the "Decomposition" estimator, we write:

$$\begin{aligned}\nabla_{\mathbf{w}} L(\mathbf{w}) &= \nabla_{\mathbf{w}} \mathop{\mathbb{E}}_{\mathbf{x},\mathbf{z}}[-\mathbf{z}^\top N_{\mathbf{w}}(\mathbf{x})] \\ &= \mathop{\mathbb{E}}_{\mathbf{x}}[\mathop{\mathbb{E}}_{\mathbf{z}|\mathbf{x}}[\nabla_{\mathbf{w}}(-\mathbf{z}^\top N_{\mathbf{w}}(\mathbf{x}))]] \\ &\overset{(1)}{=} \mathop{\mathbb{E}}_{\mathbf{x}}[\mathop{\mathbb{E}}_{\mathbf{z}|\mathbf{x}}[-\mathbf{z}^\top \nabla_{\mathbf{w}}(N_{\mathbf{w}}(\mathbf{x}))]] \overset{(2)}{=} \mathop{\mathbb{E}}_{\mathbf{x}}[-e_{y(\mathbf{x})}^\top \nabla_{\mathbf{w}}(N_{\mathbf{w}}(\mathbf{x}))]\end{aligned}$$

where $(1)$ follows from the chain rule, and $(2)$ from the assumption on the distribution of $\mathbf{z}$ given $\mathbf{x}$. It is now easy to see that the "Decomposition" estimator is indeed a (different) unbiased estimator of the gradient.

Intuition says that when a choice between two unbiased estimators is presented, we should choose the one with the lower variance. In our context, [7] showed that when running SGD (even on non-convex objectives), arriving at a point where $\|\nabla_{\mathbf{w}} L(\mathbf{w})\|^2 \leq \epsilon$ requires order of $\bar{\nu}^2/\epsilon^2$ iterations, where

$$\bar{\nu}^2 = \max_t \mathop{\mathbb{E}}_{\mathbf{x},q} \|\nabla_{\mathbf{w}}^t(\mathbf{x}, q)\|^2 - \|\nabla_{\mathbf{w}} L(\mathbf{w}^{(t)})\|^2,$$

$\mathbf{w}^t$ is the weight vector at time $t$, $q$ is sampled along with $\mathbf{x}$ (where it can be replaced by $\mathbf{z}$ or $y(\mathbf{x})$, in our experiment), and $\nabla^t_{\mathbf{w}}$ is the unbiased estimator for the gradient. This serves as a motivation for analyzing the problem through this lens.

We examine the quantity $\mathbb{E}_{\mathbf{x},q} \|\nabla^t_{\mathbf{w}}(\mathbf{x}, q)\|^2$ explicitly. For the "End-to-end" estimator, this quantity equals

$$\mathbb{E}_{\mathbf{x},\mathbf{z}} \| - \mathbf{z}^\top \nabla_{\mathbf{w}} N_{\mathbf{w}}(\mathbf{x})\|^2 = \mathbb{E}_{\mathbf{x},\mathbf{z}} \| - \sum_{i=1}^{k} \mathbf{z}_i \nabla_{\mathbf{w}} N_{\mathbf{w}}(\mathbf{x})_i\|^2$$

Denoting by $G_i := \nabla_{\mathbf{w}} N_{\mathbf{w}}(\mathbf{x})_i$, we get:

$$= \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{z}|\mathbf{x}} \| - \sum_{i=1}^{k} \mathbf{z}_i G_i\|^2 = \mathbb{E}_{\mathbf{x}} \sum_{i=1}^{k} \|G_i\|^2 \tag{2}$$

where the last equality follows from expanding the squared sum, and taking expectation over $\mathbf{z}$, while noting that mixed terms cancel out (from independence of $\mathbf{z}$'s coordinates), and that $\mathbf{z}_i^2 = 1$ for all $i$.

As for the "Decomposition" estimator, it is easy to see that

$$\mathbb{E}_{\mathbf{x}} \| - e_{y(\mathbf{x})}^\top \nabla_{\mathbf{w}} N_{\mathbf{w}}(\mathbf{x})\|^2 = \mathbb{E}_{\mathbf{x}} \|G_{y(\mathbf{x})}\|^2. \tag{3}$$

Observe that in 2 we are summing up, per $\mathbf{x}$, $k$ summands, compared to the single element in 3. When randomly initializing a network it is likely that the values of $\|G_i\|^2$ are similar, hence we obtain that at the beginning of training, the variance of the "End-to-end" estimator is roughly $k$ times larger than that of the "Decomposition" estimator. To summarize, we get further evidence for the benefits of direct supervision, whenever it is applicable to a problem.

## 4 Architecture and Conditioning

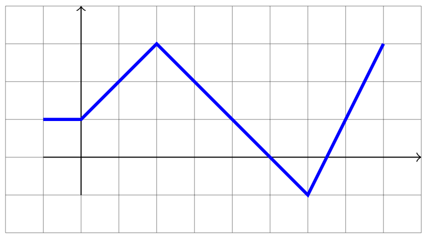
Network architecture choice is a crucial element in the success of deep learning. New variants and development of novel architectures are one of the main tools for achieving practical breakthroughs [11, 29]. When choosing an architecture, one consideration is how to inject prior knowledge on the problem at hand, improving the network's expressiveness for that problem, while not dramatically increasing sample complexity. Another aspect involves improving the computational complexity of training. In this section we formally show how the choice of architecture affects the training time through the lens of the condition number of the problem.

The study and practice of conditioning techniques, for convex and non-convex problems, gained much attention recently (e.g., [15, 19, 5, 24]). Here we show how architectural choice may have a dramatic effect on the applicability of better conditioning techniques.

The learning problem we consider in this section is that of encoding one-dimensional, piecewise linear curves. We show how different architectures, all of them of sufficient expressive power for solving the problem, have orders-of-magnitude difference in their condition numbers. In particular, this becomes apparent when considering convolutional vs. fully connected layers. This sheds a new light over the success of convolutional neural networks, which is generally attributed to their sample complexity benefits. Moreover, we show how conditioning, applied in conjunction with a better architecture choice, can further decrease the condition number by orders of magnitude. The direct effect on the convergence rate is analyzed, and is aligned with the significant performance gaps observed empirically. We also demonstrate how performance may not significantly improve by employing deeper and more powerful architectures, as well as the price that comes with choosing a sub-optimal architecture.

### 4.1 Experiments and Analysis

We experiment with various deep learning solutions for encoding the structure of one-dimensional, continuous, piecewise linear (PWL) curves. Any PWL curve with $k$ pieces can be written as: $f(x) = b + \sum_{i=1}^{k} a_i[x - \theta_i]_+$, where $a_i$ is the difference between the slope at the $i$'th segment and the $(i-1)$'th segment. For example, the curve below can be parametrized by $b = 1, a = (1, -2, 3), \theta = (0, 2, 6)$.



The problem we consider is that of receiving a vector of the values of $f$ at $x \in \{0, 1, \ldots, n-1\}$, namely $\mathbf{f} := (f(0), f(1), \ldots, f(n-1))$, and outputting the values of $b, \{a_i, \theta_i\}_{i=1}^k$. We can think of this problem as an encoding problem, since we would like to be able to rebuild $f$ from the values of $b, \{a_i, \theta_i\}_{i=1}^k$. Observe that $b = f(0)$, so from now on, let us assume without loss of generality that $b = 0$.

Throughout our experiments, we use $n = 100$, $k = 3$. We sample $\{\theta_i\}_{i \in [k]}$ uniformly without replacement from $\{0, 1, \ldots, n-1\}$, and sample each $a_i$ i.i.d. uniformly from $[-1, 1]$.

### 4.1.1 Convex Problem, Large Condition Number

As we assume that each $\theta_i$ is an integer in $\{0, 1, \ldots, n-1\}$, we can represent $\{a_i, \theta_i\}_{i=1}^k$ as a vector $\mathbf{p} \in \mathbb{R}^n$ such that $p_j = 0$ unless there is some $i$ such that $\theta_i = j$, and in this case we set $p_j = a_i$. That is, $p_j = \sum_{i=1}^k a_i \, \mathbf{1}_{[\theta_i = j-1]}$.

This allows us to formalize the problem as a convex optimization problem. Define a matrix $W \in \mathbb{R}^{n,n}$ such that $W_{i,j} = [i - j + 1]_+$. It is not difficult to show that $\mathbf{f} = W\mathbf{p}$. Moreover, $W$ can be shown to be invertible, so we can extract $\mathbf{p}$ from $\mathbf{f}$ by $\mathbf{p} = W^{-1}\mathbf{f}$.

We hence start by attempting to learn this linear transformation directly, using a connected architecture of one layer, with $n$ output channels. Let the weights of this layer be denoted $\hat{U}$. We therefore minimize the objective:

$$\min_{\hat{U}} \mathbb{E}_{\mathbf{f}} \left[ \frac{1}{2} (W^{-1}\mathbf{f} - \hat{U}\mathbf{f})^2 \right] \tag{4}$$

where $\mathbf{f}$ is sampled according to some distribution. As a convex, realizable (by $\hat{U} = W^{-1}$) problem, convergence is guaranteed, and we can explicitly analyze its rate. However, perhaps unexpectedly, we observe a very slow rate of convergence to a satisfactory solution, where significant inaccuracies are present at the non-smoothness points. Figure 6a illustrates the results.

To analyze the convergence rate of this approach, and to benchmark the performance of the next set of experiments, we start off by giving an explicit expression for $W^{-1}$:

**Lemma 1** *The inverse of $W$ is the matrix $U$ s.t. $U_{i,i} = U_{i+2,i} = 1, U_{i+1,i} = -2$, and the rest of the coordinates of $U$ are zero.*

The proof is given in Appendix A.2. Next, we analyze the iteration complexity of SGD for learning the matrix $U$. To that end, we give an explicit expression for the expected value of the learned weight matrix at each iteration $t$, denoted as $\hat{U}^t$:

**Lemma 2** *Assume $\hat{U}^0 = 0$, and that $\mathbb{E}_{\mathbf{f}}[U\mathbf{f}\mathbf{f}^\top U^\top] = \lambda I$ for some $\lambda$. Then, running SGD with learning rate $\eta$ over objective 4 for $t$ iterations yields:*

$$\mathbb{E}\, \hat{U}_t = \eta\lambda W^\top \sum_{i=0}^{t-1} (I - \eta\lambda W W^\top)^i$$

The proof is given at Appendix A.3. Note that the assumption that $\mathbb{E}_{\mathbf{f}}[U\mathbf{f}\mathbf{f}^\top U^\top] = \lambda I$ holds under the distributional assumption over the curves, as changes of direction in the curve are independent, and are sampled each time from the same distribution. The following theorem establishes a lower bound on $\|\, \mathbb{E}\, \hat{U}_{t+1} - U\|$, which by Jensen's inequality, implies a lower bound on $\mathbb{E}\, \|\hat{U}_{t+1} - U\|$, the expected distance of $\hat{U}_{t+1}$ from $U$. Note that the lower bound holds even if we use all the data for updating (that is, gradient descent and not stochastic gradient descent).

**Theorem 3** *Let $W = QSV^\top$ be the singular value decomposition of $W$. If $\eta\,\lambda\,S_{1,1}^2 \geq 1$ then $\mathbb{E}\,\hat{U}_{t+1}$ diverges. Otherwise, we have*

$$t+1 \leq \frac{S_{1,1}^2}{2\,S_{n,n}^2} \quad \Rightarrow \quad \|\,\mathbb{E}\,\hat{U}_{t+1} - U\| \geq 0.5\ ,$$

*where the norm is the spectral norm. Furthermore, the condition number $\frac{S_{1,1}^2}{S_{n,n}^2}$ (where $S_{1,1}, S_{n,n}$ are the top and bottom singular values of $W$) is $\Omega(n^{3.5})$.*

The proof is given at Appendix A.4. The theorem implies that the condition number of W, and hence, the number of GD iterations required for convergence, scales quite poorly with $n$. In the next subsection, we will try to decrease the condition number of the problem.

### 4.1.2 Improved Condition Number through Convolutional Architecture

Examining the explicit expression for $U$ given in Lemma 1, we see that $U\mathbf{f}$ can be written as a one-dimensional convolution of $\mathbf{f}$ with the kernel $[1, -2, 1]$. Therefore, the mapping from $\mathbf{f}$ to $\mathbf{p}$ is realizable using a convolutional layer.

Empirically, convergence to an accurate solution is faster using this architecture. Figure 6b illustrates a few examples. To theoretically understand the benefit of using a convolution, from the perspective of the required number of iterations for training, we will consider the new problem's condition number, providing understanding of the gap in training time. In the previous section we saw that GD requires $\Omega(n^{3.5})$ iterations to learn the full matrix $U$. In the appendix (sections A.5 and A.6) we show that under some mild assumptions, the condition number is only $\Theta(n^3)$, and GD requires only that order of iterations to learn the optimal filter $[1, -2, 1]$.

### 4.1.3 Additional Improvement through Explicit Conditioning

In Section 4.1.2, despite observing an improvement from the fully connected architecture, we saw that GD still requires $\Omega(n^3)$ iterations even for the simple problem of learning the filter $[1, -2, 1]$. This motivates an application of additional conditioning techniques, in the hope for extra performance gains.

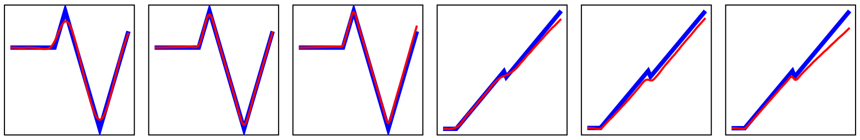
First, let us explicitly represent the convolutional architecture as a linear regression problem. We perform Vec2Row operation on $\mathbf{f}$ as follows: given a sample $\mathbf{f}$, construct a matrix, $F$, of size $n \times 3$, such that the $t$'th row of $F$ is $[\mathbf{f}_{t-1}, \mathbf{f}_t, \mathbf{f}_{t+1}]$. Then, we obtain a vanilla linear regression problem in $\mathbb{R}^3$, with the filter $[1, -2, 1]$ as its solution. Given a sample $\mathbf{f}$, we can now approximate the correlation matrix of $F$, denoted $C \in \mathbb{R}^{3,3}$, by setting $C_{i,j} = \mathbb{E}_{\mathbf{f},t}[\mathbf{f}_{t-2+i}\mathbf{f}_{t-2+j}]$. We then calculate the matrix $C^{-1/2}$ and replace every instance (namely, a row of $F$) $[\mathbf{f}_{t-1}, \mathbf{f}_t, \mathbf{f}_{t+1}]$ by the instance $[\mathbf{f}_{t-1}, \mathbf{f}_t, \mathbf{f}_{t+1}]C^{-1/2}$. By construction, the correlation matrix of the resulting instances is approximately the identity matrix, hence the condition number is approximately 1. It follows (see again Appendix A.5) that SGD converges using order of $\log(1/\epsilon)$ iterations, independently of $n$. Empirically, we quickly converge to extremely accurate results, illustrated in Figure 6c.

We note that the use of a convolution architecture is crucial for the efficiency of the conditioning; had the dimension of the problem not been reduced so dramatically, the difficulty of estimating a large $n \times n$ correlation matrix scales strongly with $n$, and furthermore, its inversion becomes a costly operation. The combined use of a better architecture and of conditioning is what allows us to gain this dramatic improvement.

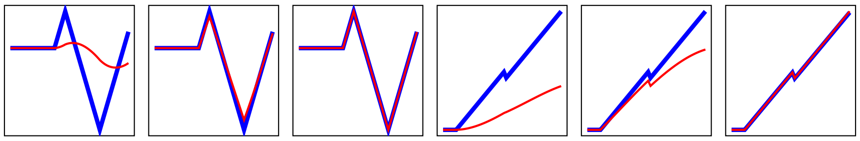
### 4.1.4 Perhaps I should use a deeper network?

The solution arrived at in Section 4.1.3 indicates that a suitable architecture choice and conditioning scheme can provide training time speedups of multiple orders of magnitude. Moreover, the benefit of reducing the number of parameters, in the transition from a fully connected architecture to a convolutional one, is shown to be helpful in terms of convergence time. However, we should not rule out the possibility that a deeper, wider network will not suffer from the deficiencies analyzed above for the convex case.

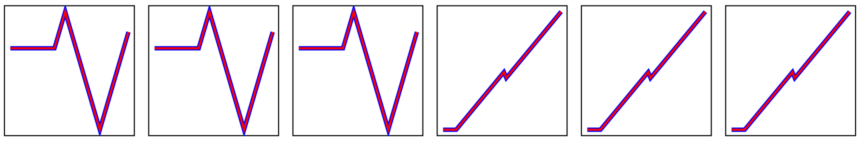
Motivated by the success of deep auto-encoders, we experiment with a deeper architecture for encoding $\mathbf{f}$. Namely, we minimize $\min_{\mathbf{v}_1,\mathbf{v}_2} \mathbb{E}_{\mathbf{f}}[(\mathbf{f} - M_{\mathbf{v}_2}(N_{\mathbf{v}_1}(\mathbf{f})))^2]$, Where $N_{\mathbf{v}_1}, M_{\mathbf{v}_2}$ are deep networks parametrized by their weight



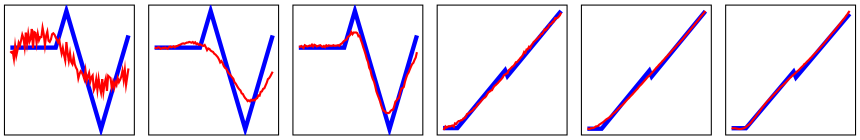
(a) Section 4.1.1's experiment - linear architecture.



(b) Section 4.1.2's experiment - convolutional architecture.



(c) Section 4.1.3's experiment - convolutional architecture with conditioning.



(d) 4.1.4's experiment - vanilla auto encoder.

Figure 6: Examples for decoded outputs of Section 4's experiments, learning to encode PWL curves. In blue are the original curves. In red are the decoded curves. The plot shows the outputs for two curves, after 500, 10000, and 50000 iterations, from left to right.

vectors $\mathbf{v}_1, \mathbf{v}_2$, with the output of $N$ being of dimension $2k$, enough for realization of the encoding problem. Each of the two networks has three layers with ReLU activations, except for the output layer of $M$ having a linear activation. The dimensions of the layers are, $500, 100, 2k$ for $N$, and $100, 100, n$ for $M$.

Aligned with the intuition gained through the previous experiments, we observe that additional expressive power, when unnecessary, does not solve inherent optimization problems, as this stronger Auto-Encoder fails to capture the fine details of $\mathbf{f}$ at its non-smooth points. See Figure 6d for examples.

## 5 Flat Activations

We now examine a different aspect of gradient-based learning which poses difficulties for optimization: namely, flatness of the loss surface due to saturation of the activation functions, leading to vanishing gradients and a slow-down of the training process. This problem is amplified in deeper architectures, since it is likely that the backpropagated message to lower layers in the architecture would vanish due to a saturated activation somewhere along the way. This is a major problem when using sigmoids as a gating mechanisms in Recurrent Neural Networks such as LSTMS and GRUs [10, 3].

While non-local search-based optimization for large scale problems seems to be beyond reach, variants on the gradient update, whether by adding momentum, higher order methods, or normalized gradients, are quite successful, leading to consideration of update schemes deviating from "vanilla" gradient updates.

In this section, we consider a family of activation functions which amplify the "vanishing gradient due to saturated activation" problem; they are piecewise flat. Using such activations in a neural network architecture will result in a gradient equal to 0, which will be completely useless. We consider different ways to implement, approximate or learn such activations, such that the error will effectively propagate through them. Using a different variant of a local search-based update, based on [17, 16] , we arrive at an efficient solution. Convergence guarantees exist for a one-layer architecture. We leave further study of deeper networks to future work.

## 5.1 Experimental Setup

Consider the following optimization setup. The sample space $X \subset \mathbb{R}^d$ is symmetrically distributed. The target function $y : \mathbb{R}^d \to \mathbb{R}$ is of the form $y(\mathbf{x}) = u(\mathbf{v}^{*\top}\mathbf{x} + b^*)$, where $\mathbf{v}^* \in \mathbb{R}^d$, $b^* \in \mathbb{R}$, and $u : \mathbb{R} \to \mathbb{R}$ is a monotonically non-decreasing function. The objective of the optimization problem is given by:

$$\min_{\mathbf{w}} \mathbb{E}_{x} \left[\ell\left(u(N_{\mathbf{w}}(\mathbf{x})), y(\mathbf{x})\right)\right]$$

where $N_{\mathbf{w}}$ is some neural network parametrized by $\mathbf{w}$, and $\ell$ is some loss function (for example, the squared or absolute difference).

For the experiments, we use $u$ of the form:

$$u(r) = z_0 + \sum_{i \in [55]} \mathbf{1}_{[r > z_i]} \cdot (z_i - z_{i-1}) \,,$$

where $z_0 < z_1 < \ldots < z_{55}$ are known. In words, given $r$, the function rounds down to the nearest $z_i$. We also experiment with normally distributed $X$. Our theoretical analysis is not restricted to $u$ of this specific form, nor to normal $X$. All figures are found in Figure 7.

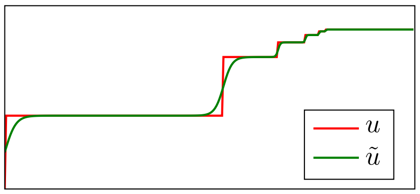
Of course, applying gradient-based methods to solve this problem directly, is doomed to fail as the derivative of $u$ is identically 0. Is there anything which can be done instead?

## 5.2 Non-Flat Approximation Experiment

We start off by trying to approximate $u$ using a non flat function $\tilde{u}$ defined by

$$\tilde{u}(r) = z_0 + \sum_{i \in [55]} (z_i - z_{i-1}) \cdot \sigma(c \cdot (r - z_i)),$$

where $c$ is some constant, and $\sigma$ is the sigmoid function $\sigma(z) = (1 + \exp(-z))^{-1}$. Intuitively, we approximate the "steps" in $u$ using a sum of sigmoids, each of amplitude corresponding to the step's height, and centered at the step's position. This is similar to the motivation for using sigmoids as activation functions and as gates in LSTM cells — a non-flat approximation of the step function. Below is an example for $u$, and its approximation $\tilde{u}$.



The objective is the expected squared loss, propagated through $\tilde{u}$, namely

$$\min_{\mathbf{v},b} \mathbb{E}_{\mathbf{x}} \left[\left(\tilde{u}(\mathbf{v}^\top\mathbf{x} + b) - y(\mathbf{x})\right)^2\right] .$$

Although the objective is not completely flat, and is continuous, it suffers from the flatness and non continuity deficiencies of the original $u$, and training using this objective is much slower, and sometimes completely failing. In particular, sensitivity to the initialization of bias term is observed, where the wrong initialization can cause the starting point to be in a very wide flat region of $u$, and hence a very flat region of $\tilde{u}$.

## 5.3 End-to-End Experiment

Next, we attempt to solve the problem using improper learning, with the objective now being:

$$\min_{\mathbf{w}} \mathbb{E}_{\mathbf{x}} \left[ (N_{\mathbf{w}}(x) - y(\mathbf{x}))^2 \right]$$

where $N_{\mathbf{w}}$ is a network parametrized by its weight vector $\mathbf{w}$. We use a simple architecture of four fully connected layers, the first three with ReLU activations and 100 output channels, and the last, with only one output channel and no activation function.

As covered in Section 4, difficulty arises when regressing to non smooth functions. In this case, with $u$ not even being continuous, the inaccuracies in capturing the non continuity points are brought to the forefront. Moreover, this solution has its extra price in terms of sample complexity, training time, and test time, due to the use of a much larger than necessary network. An advantage is of course the minimal prior knowledge about $u$ which is required. While this approach manages to find a reasonable solution, it is far from being perfect.

## 5.4 Multi-Class Experiment

In this experiment, we approach the problem as a general multi-class classification problem, with each value of the image of $u$ is treated as a separate class. We use a similar architecture to that of the End-to-end experiment, with one less hidden layer, and with the final layer outputting $55$ outputs, each corresponding to one of the steps defined by the $z_i$s. A problem here is the inaccuracies at the boundaries between classes, due to the lack of structure imposed over the predictor. The fact that the linear connection between $x$ and the input to $u$ is not imposed through the architecture results in "blurry" boundaries. In addition, the fact that we rely on an "improper" approach, in the sense that we ignore the ordering imposed by $u$, results in higher sample complexity.

## 5.5 The "Forward-Only" Update Rule

Let us go back to a direct formulation of the problem, in the form of the objective function

$$\min_{\mathbf{w}} F(\mathbf{w}) = \mathbb{E}_{x} \left[ (u(\mathbf{w}^\top \mathbf{x}) - y(\mathbf{x}))^2 \right]$$

where $y(\mathbf{x}) = u(\mathbf{v}^{*\top}\mathbf{x})$. The gradient update rule in this case is $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla F(\mathbf{w}^{(t)})$, where for our objective we have

$$\nabla F(\mathbf{w}) = \mathbb{E}_{\mathbf{x}} \left[ (u(\mathbf{w}^\top \mathbf{x}) - y(\mathbf{x})) \cdot u'(\mathbf{w}^\top \mathbf{x}) \cdot \mathbf{x} \right]$$

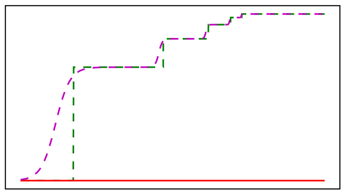
Since $u'$ is zero a.e., the gradient update is meaningless. [17, 16] proposed to replace the gradient with the following:

$$\tilde{\nabla} F(\mathbf{w}) = \mathbb{E}_{\mathbf{x}} \left[ (u(\mathbf{w}^\top \mathbf{x}) - y(\mathbf{x})) \cdot \mathbf{x} \right] \tag{5}$$

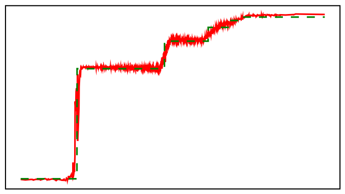
In terms of the backpropagation algorithm, this kind of update can be interpreted as replacing the backpropagation message for the activation function $u$ with an identity message. For notation simplicity, we omitted the bias terms $b, b^*$, but the same Forward-only concept is applied to them too.

This method empirically achieves the best results, both in terms of final accuracy, training time, and test time cost. As mentioned before, the method is due to [17, 16], where it is proven to converge to an $\epsilon$-optimal solution in $O(L^2/\epsilon^2)$, under the additional assumptions that the function $u$ is $L$-Lipschitz, and that $\mathbf{w}$ is constrained to have bounded norm. For completeness, we provide a short proof in Appendix A.7.

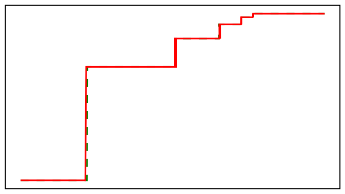
**Acknowledgements:** This research is supported by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).



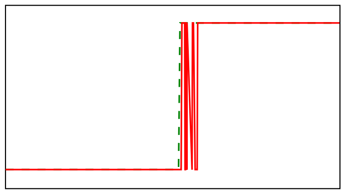
(a) Experiment 5.2, Non-flat approximation



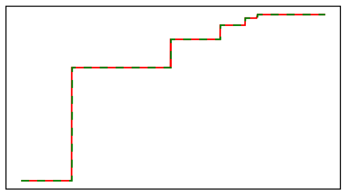
(b) Experiment 5.3, End-to-end



(c) Experiment 5.4, Multi Class



(d) Experiment 5.4, Multi Class. Zoomed-in boundary between classes.



(e) Experiment 5.5, Forward only

Figure 7: Section 5's Experiment: backpropagating through the steps function. The horizontal axis is the value of $r = v^{*\top}x + b^*$. The dashed green curves show the label, $u(r)$. The red curves show the outputs of the learnt hypothesis. The plots are zoomed around the mean of $r$. In the non-flat approximation plot, the dashed magenta curve shows the non-flat approximation, namely $\tilde{u}(r)$. Note the inaccuracies around the boundaries between classes in the Multi Class experiment. All plots show the results after 10000 training iterations, except for the forward only plot, showing results after 5000 iterations.

## References

[1] Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning dnf and characterizing statistical query learning using fourier analysis. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 253–262. ACM, 1994.

[2] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[6] Vitaly Feldman, Cristobal Guzman, and Santosh Vempala. Statistical query algorithms for stochastic convex optimization. *arXiv preprint arXiv:1512.09170*, 2015.

[7] Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[9] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[10] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

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

[13] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[16] Sham M Kakade, Varun Kanade, Ohad Shamir, and Adam Kalai. Efficient learning of generalized linear and single index models with isotonic regression. In *Advances in Neural Information Processing Systems*, pages 927–935, 2011.

[17] Adam Tauman Kalai and Ravi Sastry. The isotron algorithm: High-dimensional isotonic regression. In *COLT*, 2009.

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

[19] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[22] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[23] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[24] Shai Shalev-Shwartz, Alon Gonen, and Ohad Shamir. Large-scale convex minimization with a low-rank constraint. *arXiv preprint arXiv:1106.1622*, 2011.

[25] Shai Shalev-Shwartz and Amnon Shashua. On the sample complexity of end-to-end training vs. semantic abstraction training. *arXiv preprint arXiv:1604.06915*, 2016.

[26] Ohad Shamir. Distribution-specific hardness of learning neural networks. *arXiv preprint arXiv:1609.01037*, 2016.

[27] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28:1139–1147, 2013.

[28] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[30] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[31] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

## A Proofs

### A.1 Proof of Theorem 1

**Proof** Given two square-integrable functions $f, g$ on an Euclidean space $\mathbb{R}^n$, let $\langle f, g\rangle_{L_2} = \mathbb{E}_{\mathbf{x}}[f(\mathbf{x})g(\mathbf{x})]$ and $\|f\|_{L_2} = \sqrt{\mathbb{E}_{\mathbf{x}}[f^2(\mathbf{x})]}$ denote inner product and norm in the $L_2$ space of square-integrable functions (with respect to the relevant distribution). Also, define the vector-valued function

$$\mathbf{g}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{w}} p_{\mathbf{w}}(\mathbf{x}),$$

and let $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_n(\mathbf{x}))$ for real-valued functions $g_1, \ldots, g_n$. Finally, let $\mathbb{E}_h$ denote an expectation with respect to $h$ chosen uniformly at random from $\mathcal{H}$. Let $|\mathcal{H}| = d$.

We begin by proving the result for the squared loss. To prove the bound, it is enough to show that $\mathbb{E}_h \|\nabla F_h(\mathbf{w}) - \mathbf{a}\|^2 \le \frac{G^2}{|\mathcal{H}|}$ for any vector $\mathbf{a}$ independent of $h$. In particular, let us choose $\mathbf{a} = \mathbb{E}_{\mathbf{x}}[p_{\mathbf{w}}(\mathbf{x})\mathbf{g}(\mathbf{x})]$. We thus bound the following:

$$\begin{aligned}
\mathop{\mathbb{E}}_h \|\nabla F_h(\mathbf{w}) - \mathop{\mathbb{E}}_{\mathbf{x}}[p_{\mathbf{w}}(\mathbf{x})\mathbf{g}(\mathbf{x})]\|^2 &= \mathop{\mathbb{E}}_h \|\mathop{\mathbb{E}}_{\mathbf{x}}[(p_{\mathbf{w}}(\mathbf{x}) - h(\mathbf{x}))\mathbf{g}(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{x}}[p_{\mathbf{w}}(\mathbf{x})\mathbf{g}(\mathbf{x})]\|^2 \\
&= \mathop{\mathbb{E}}_h \|\mathop{\mathbb{E}}_{\mathbf{x}}[h(\mathbf{x})\mathbf{g}(\mathbf{x})]\|^2 \;=\; \mathop{\mathbb{E}}_h \sum_{j=1}^n \left(\mathop{\mathbb{E}}_{\mathbf{x}}[h(\mathbf{x})g_j(\mathbf{x})]\right)^2 \\
&= \mathop{\mathbb{E}}_h \sum_{j=1}^n \langle h, g_j\rangle_{L_2}^2 \;=\; \sum_{j=1}^n \left(\frac{1}{|\mathcal{H}|}\sum_{i=1}^d \langle h_i, g_j\rangle_{L_2}^2\right) \\
&\overset{(*)}{\le} \sum_{j=1}^n \left(\frac{1}{|\mathcal{H}|}\|g_j\|_{L_2}^2\right) \;=\; \frac{1}{|\mathcal{H}|}\sum_{j=1}^n \mathop{\mathbb{E}}_{\mathbf{x}}[g_j^2(\mathbf{x})] \\
&= \frac{1}{|\mathcal{H}|}\mathop{\mathbb{E}}_{\mathbf{x}}\left[\|\mathbf{g}(\mathbf{x})\|^2\right] \;\le\; \frac{G(\mathbf{w})^2}{|\mathcal{H}|},
\end{aligned}$$

where $(*)$ follows from the functions in $\mathcal{H}$ being mutually orthogonal, and satisfying $\|h\|_{L_2} \le 1$ for all $h \in \mathcal{H}$.

To handle a classification loss, note that by its definition and the fact that $h(\mathbf{x}) \in \{-1, +1\}$,

$$\begin{aligned}
\nabla F_h(\mathbf{w}) &= \mathop{\mathbb{E}}_{\mathbf{x}}\left[r'(h(\mathbf{x})p_{\mathbf{w}}(\mathbf{x})) \cdot \frac{\partial}{\partial \mathbf{w}} p_{\mathbf{w}}(\mathbf{x})\right] \\
&= \mathop{\mathbb{E}}_{\mathbf{x}}\left[\left(\frac{r'(p_{\mathbf{w}}(\mathbf{x})) + r'(-p_{\mathbf{w}}(\mathbf{x}))}{2} + h(\mathbf{x}) \cdot \frac{r'(p_{\mathbf{w}}(\mathbf{x})) - r'(-p_{\mathbf{w}}(\mathbf{x}))}{2}\right) \cdot \frac{\partial}{\partial \mathbf{w}} p_{\mathbf{w}}(\mathbf{x})\right] \\
&= \mathop{\mathbb{E}}_{\mathbf{x}}\left[\frac{r'(p_{\mathbf{w}}(\mathbf{x})) + r'(-p_{\mathbf{w}}(\mathbf{x}))}{2} \cdot \frac{\partial}{\partial \mathbf{w}} p_{\mathbf{w}}(\mathbf{x})\right] + \mathop{\mathbb{E}}_{\mathbf{x}}\left[h(\mathbf{x}) \cdot \left(\frac{r'(p_{\mathbf{w}}(\mathbf{x})) - r'(-p_{\mathbf{w}}(\mathbf{x}))}{2}\right) \cdot \frac{\partial}{\partial \mathbf{w}} p_{\mathbf{w}}(\mathbf{x})\right].
\end{aligned}$$

Letting $\mathbf{g}(\mathbf{x}) = \left(\frac{r'(p_{\mathbf{w}}(\mathbf{x})) - r'(-p_{\mathbf{w}}(\mathbf{x}))}{2}\right) \cdot \frac{\partial}{\partial \mathbf{w}} p_{\mathbf{w}}(\mathbf{x})$ (which satisfies $\mathbb{E}_{\mathbf{x}}[\|\mathbf{g}(\mathbf{x})\|^2] \le G^2$ since $r$ is 1-Lipschitz) and $\mathbf{a} = \mathbb{E}_{\mathbf{x}}\left[\frac{r'(p_{\mathbf{w}}(\mathbf{x})) + r'(-p_{\mathbf{w}}(\mathbf{x}))}{2} \cdot \frac{\partial}{\partial \mathbf{w}} p_{\mathbf{w}}(\mathbf{x})\right]$ (which does not depend on $h$), we get that

$$\mathop{\mathbb{E}}_h \|\nabla F_h(\mathbf{w}) - \mathbf{a}\|^2 = \mathop{\mathbb{E}}_h \|\mathop{\mathbb{E}}_{\mathbf{x}}[h(\mathbf{x})\mathbf{g}(\mathbf{x})]\|^2.$$

Proceeding now exactly in the same manner as the squared loss case, the result follows. ■

## A.2 Proof of lemma 1

**Proof**

$$(UW)_{i,j} = \sum_t U_{i,t}W_{t,j} = W_{i,j} - 2W_{i-1,j} + W_{i-2,j}$$

If $i \ge j+1$ then $\frac{W_{i,j} + W_{i-2,j}}{2} = W_{i-1,j}$ and therefore the above is clearly zero. If $i < j$ then all the values of $W$ are zeros. Finally, if $i = j$ we obtain 1. This concludes our proof. ■

## A.3 Proof of lemma 2

**Proof** Given a sample $\mathbf{f}$, and that our current weight matrix is $\hat{U}$, let $\mathbf{p} = W^{-1}\mathbf{f}$. The loss function on $\mathbf{f}$ is given by

$$\frac{1}{2}\|\hat{U}\mathbf{f} - \mathbf{p}\|^2$$

The gradient w.r.t. $\hat{U}$ is

$$\nabla = (\hat{U}\mathbf{f} - \mathbf{p})\mathbf{f}^\top = \hat{U}\mathbf{f}\mathbf{f}^\top - \mathbf{p}\mathbf{f}^\top$$

We obtain that the update rule is

$$\hat{U}_{t+1} = \hat{U}_t - \eta\left(\hat{U}_t\mathbf{f}\mathbf{f}^\top - \mathbf{p}\mathbf{f}^\top\right) = \hat{U}_t(I - \eta\mathbf{f}\mathbf{f}^\top) + \eta\mathbf{p}\mathbf{f}^\top$$

Taking expectation with respect to the random choice of the pair, using again $\mathbf{f} = W\mathbf{p}$, and assuming $\mathbb{E}\,\mathbf{p}\mathbf{p}^\top = \lambda I$, we obtain that the stochastic gradient update rule satisfies

$$\mathbb{E}\,\hat{U}_{t+1} = \hat{U}_t(I - \eta\lambda WW^\top) + \eta\lambda W^\top$$

Continuing recursively, we obtain

$$\begin{aligned}
\mathbb{E}\,\hat{U}_{t+1} &= \mathbb{E}\,\hat{U}_t(I - \eta\lambda WW^\top) + \eta\lambda W^\top \\
&= \left[\mathbb{E}\,\hat{U}_{t-1}(I - \eta\lambda WW^\top) + \eta\lambda W^\top\right](I - \eta\lambda WW^\top) + \eta\lambda W^\top \\
&= \mathbb{E}\,\hat{U}_{t-1}(I - \eta\lambda WW^\top)^2 + \eta\lambda W^\top(I - \eta\lambda WW^\top) + \eta\lambda W^\top \\
&= \hat{U}_0(I - \eta\lambda WW^\top)^t + \eta\lambda W^\top\sum_{i=0}^{t}(I - \eta\lambda WW^\top)^i
\end{aligned}$$

We assume that $\hat{U}_0 = \mathbf{0}$, and thus

$$\mathbb{E}\,\hat{U}_{t+1} = \eta\lambda W^\top\sum_{i=0}^{t}(I - \eta\lambda WW^\top)^i$$

■

## A.4 Proof of Theorem 3

**Proof** Fix some $i$, we have that

$$(I - \eta\,\lambda WW^\top)^i = (QIQ^\top - \eta\,\lambda QSV^\top VSQ^\top)^i = Q(I - \eta\,\lambda SS)^i Q^\top = Q\Lambda^i Q^\top$$

where $\Lambda^i$ is diagonal with $\Lambda^i_{j,j} = (1 - \eta\lambda S_j^2)^i$. Therefore, by the properties of geometric series, $\mathbb{E}\,\hat{U}_{t+1}$ converges if and only if $\eta\,\lambda\,S_{1,1}^2 < 1$. When this condition holds we have that

$$\begin{aligned}
\hat{U}_\infty &= \eta\,\lambda\,W^\top\sum_{i=0}^{\infty}(I - \eta\,\lambda WW^\top)^i \\
&= \eta\,\lambda\,W^\top(\eta\,\lambda\,WW^\top)^{-1} = W^\top(WW^\top)^{-1} \\
&= VSQ^\top(QSV^\top VSQ^\top)^{-1} = VSQ^\top QS^{-2}Q^\top = VS^{-1}Q^\top = U\,.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbb{E}\,\hat{U}_{t+1} - U &= \eta\,\lambda W^\top\sum_{i=t+1}^{\infty}(I - \eta\,\lambda WW^\top)^i \\
&= \eta\,\lambda VSQ^\top\sum_{i=t+1}^{\infty}Q\Lambda^i Q^\top \\
&= V\left[\sum_{i=t+1}^{\infty}(\eta\,\lambda S)\Lambda^i\right]Q^\top\,.
\end{aligned}$$

The matrix in the parentheses is diagonal, where the $j$'th diagonal element is

$$\eta \lambda S_{j,j} \cdot \frac{(1-\eta\lambda S_{j,j}^2)^{t+1}}{\eta\lambda S_{j,j}^2} = S_{j,j}^{-1}(1-\eta\lambda S_{j,j}^2)^{t+1}$$

It follows that

$$\| \mathbb{E}\, \hat{U}_{t+1} - U\| = \max_j S_{j,j}^{-1}(1-\eta\lambda S_{j,j}^2)^{t+1}$$

Using the inequality $(1-a)^{t+1} \geq 1-(t+1)a$, that holds for every $a \in (-1,1)$, we obtain that

$$\| \mathbb{E}\, \hat{U}_{t+1} - U\| \geq S_{n,n}^{-1}(1-(t+1)\eta\lambda S_{n,n}^2) \geq S_{n,n}^{-1}(1-(t+1)\frac{S_{n,n}^2}{S_{1,1}^2}) \; .$$

It follows that whenever,

$$t+1 \leq \frac{S_{1,1}^2}{2\,S_{n,n}^2} \; ,$$

we have that $\| \mathbb{E}\, \hat{U}_{t+1} - U\| \geq 0.5\, S_{n,n}^{-1}$. Finally, observe that

$$S_{n,n}^2 = \min_{x:\|x\|=1} x^\top W W^\top x \leq e_1^\top W W^\top e_1 = 1 \; ,$$

hence $S_{n,n}^{-1} \geq 1$.

We now prove the second part of the theorem, regarding the condition number of $W^\top W$, namely, $\frac{S_{1,1}^2}{2\,S_{n,n}^2} \geq \Omega(n^{3.5})$. We note that the condition number of $W^\top W$ can be calculated through its inverse matrix's, namely, $U^\top U$'s condition number, as those are equal.

It is easy to verify that $Ue_n = e_n$. Therefore, the maximal eigenvalue of $U^\top U$ is at least 1. To construct an upper bound on the minimal eigenvalue of $U^\top U$, consider $v \in \mathbb{R}^n$ s.t. for $i \leq \sqrt{n}$ we have $v_i = -\frac{1}{2}(i/n)^2$ and for $i > \sqrt{n}$ we have $v_i = \frac{1}{2n} - \frac{i/n}{\sqrt{n}}$. We have that $|v_i| = O(1/n)$ for $i < \sqrt{n}+2$ and $v$ is linear for $i \geq \sqrt{n}$. This implies that $(Uv)_i = 0$ for $i \geq \sqrt{n}+2$. We also have $(Uv)_1 = v_1 = -0.5/n^2$, $(Uv)_2 = -2v_1 + v_2 \approx -1/n^2$, and for $i \in \{3, \ldots, \sqrt{n}\}$ we have

$$(Uv)_i = v_{i-2} - 2v_{i-1} + v_i = -3/n^2$$

Finally, for $i = \sqrt{n}+1$ we have

$$\begin{aligned}(Uv)_i &= v_{i-2} - 2v_{i-1} + v_i = v_i - v_{i-1} - (v_{i-1} - v_{i-2}) \\ &= \frac{1}{n\sqrt{n}} - \frac{1}{2n^2}\left((i-1)^2 - (i-2)^2\right) = \frac{1}{n\sqrt{n}} - \frac{1}{2n^2}(2i-3) = \frac{1}{2n^2} \; .\end{aligned}$$

This yields

$$\|Uv\|^2 \approx \Theta(\sqrt{n}/n^4) = \Theta(n^{-3.5})$$

In addition,

$$\|v\|^2 \geq \sum_{i=n/2}^{n} v_i^2 \geq \frac{n}{2}\, v_{n/2}^2 = \frac{n}{2}\left(\frac{1}{2n} - \frac{1}{2\sqrt{n}}\right)^2 = \Omega(1) \; .$$

Therefore,

$$\frac{\|Uv\|^2}{\|v\|^2} = O(n^{-3.5}) \; ,$$

which implies that the minimal eigenvalue of $U^\top U$ is at most $O(n^{-3.5})$. All in all, we have shown that the condition number of $U^\top U$ is $\Omega(n^{-3.5})$, implying the same over $W^\top W$. ■

## A.5 Gradient Descent for Linear Regression

The loss function is

$$\mathop{\mathbb{E}}_{x,y} \frac{1}{2}(x^\top w - y)^2$$

The gradient at $w$ is

$$\nabla = \mathop{\mathbb{E}}_{x,y} x(x^\top w - y) = \left(\mathop{\mathbb{E}}_{x} xx^\top\right) w - \mathop{\mathbb{E}}_{x,y} xy := Cw - z$$

For the optimal solution we have $Cw^* - z = \mathbf{0}$, hence $z = Cw^*$. The update is therefore

$$w_{t+1} = w_t - \eta(Cw_t - z) = (I - \eta C)w_t + \eta z = \ldots = \sum_{i=0}^{t} (I - \eta C)^i \eta z = \sum_{i=0}^{t} (I - \eta C)^i \eta C w^*$$

Let $C = VDV^\top$ be the eigenvalue decomposition of $C$. Observe that

$$w_{t+1} = V \sum_{i=0}^{t} \eta (I - \eta D)^i D V^\top w^*$$

Hence

$$\begin{aligned}
\|w^* - w_{t+1}\| &= \|(VV^\top - V \sum_{i=0}^{t} \eta(I - \eta D)^i D V^\top) w^*\| \\
&= \|(I - \sum_{i=0}^{t} \eta(I - \eta D)^i D) V^\top w^*\| \\
&= \|(I - \eta D)^{t+1} V^\top w^*\| ,
\end{aligned}$$

where the last equality is because for every $j$ we have

$$(I - \sum_{i=0}^{t} \eta(I - \eta D)^i D)_{j,j} = 1 - \sum_{i=0}^{t} \eta (1 - \eta D_{j,j})^i D_{j,j} = (1 - \eta D_{j,j})^{t+1} .$$

Denote $v^* = V^\top w^*$. We therefore obtain that

$$\|w^* - w_{t+1}\|^2 = \sum_{j=1}^{n} \left((1 - \eta D_{j,j})^{t+1} v_j^*\right)^2$$

To obtain an upper bound, choose $\eta = 1/D_{1,1}$ and $t + 1 \geq \frac{D_{1,1}}{D_{n,n}} \log(\|w^*\|/\epsilon)$, and then, using $1 - a \leq e^{-a}$, we get that

$$\|w^* - w_{t+1}\|^2 \leq \sum_{j=1}^{n} \left(\exp(-\eta D_{j,j}(t+1)) v_j^*\right)^2 \leq \frac{\epsilon^2}{\|w^*\|^2} \sum_{j=1}^{n} \left(v_j^*\right)^2 = \epsilon^2 .$$

To obtain a lower bound, observe that if $v_1^*$ is non-negligible then $\eta$ must be at most $1/D_{1,1}$ (otherwise the process will diverge). If in addition $v_n^*$ is a constant (for simplicity, say $v_n^* = 1$), then

$$\|w^* - w_{t+1}\| \geq (1 - \eta D_{n,n})^{t+1} \geq (1 - D_{n,n}/D_{1,1})^{t+1} \geq 1 - (t+1) D_{n,n}/D_{1,1} ,$$

where we used $(1 - a)^{t+1} \geq 1 - (t+1)a$ for $a \in [-1, 1]$. It follows that if $t + 1 < 0.5\, D_{n,n}/D_{1,1}$ then we have that $\|w^* - w_{t+1}\| \geq 0.5$.

## A.6 The Covariance Matrix of Section 4.1.2

Denote by $C \in \mathbb{R}^{3,3}$ the covariance matrix, and let $\lambda_i(C)$ denote the $i$'th eigenvalue of $C$ (in a decreasing order). The condition number of $C$ is $\lambda_1(C)/\lambda_3(C)$. Below we derive lower and upper bounds on the condition number under some assumptions.

**Lower bound** We assume that $\mathbb{E}_{f,t} f_t^2 = \Omega(n^2)$ (this would be the case in many typical cases, for example when the allowed slopes are in $\{\pm 1\}$), that $k$ (the number of pieces of the curve) is constant, and that the changes of slope are in $[-1, 1]$.

Now, take $v = [1, 1, 1]^\top$, then

$$v^\top C v = \mathbb{E}_{f,t} (f_{t-1} + f_t + f_{t+1})^2 = \Omega(n^2)$$

This yields $\lambda_1(C) \geq \Omega(n^2)$. Next, take $v = [1, -2, 1]^\top$ we obtain

$$v^\top C v = \mathbb{E}_{f,t} (f_{t-1} - f_t + f_{t+1})^2 = O(k/n)$$

This yields $\lambda_3(C) \leq O(k/n)$. All in all, we obtain that the condition number of $C$ is $\Omega(n^3)$.

**Upper bound** We consider distribution over $\boldsymbol{f}$ s.t. for every $\boldsymbol{f}$, at exactly $k$ indices $\boldsymbol{f}$ changes slope from $1$ to $-1$ or from $-1$ to $1$ (with equal probability over the indices), and at the rest of the indices we have that $\boldsymbol{f}$ is linear with a slope of $1$ or $-1$. Denote $p = k/n$. Take any unit vector $v$, and denote $\bar{v} = v_1 + v_2 + v_3$. Then

$$\begin{aligned} v^\top C v &= \mathbb{E}_{f,t} (v_1 f_{t-1} + v_2 f_t + v_3 f_{t+1})^2 \\ &= \mathbb{E}_{f} \left[0.5(1-p)\left((\bar{v} f_t + v_3 - v_1)^2 + (\bar{v} f_t + v_1 - v_3)^2\right) + 0.5p\left((\bar{v} f_t + v_3 + v_1)^2 + (\bar{v} f_t - v_3 - v_1)^2\right)\right] \\ &= \bar{v}^2 \mathbb{E}_{f} f_t^2 + (1-p)(v_1 - v_3)^2 + p(v_1 + v_3)^2 \ . \end{aligned}$$

Since $\mathbb{E}_{f,t} f_t^2 = \Theta(n^2)$, it is clear that $v^\top C v = O(n^2)$. We next establish a lower bound of $\Omega(1/n)$. Observe that if $\bar{v}^2 \geq \Omega(1/n^3)$, we are done. If this is not the case, then $-v_2 \approx v_1 + v_3$. If $|v_1 + v_3| > 0.1$, we are done. Otherwise, $0.9 \leq 1 - v_2^2 = v_1^2 + v_3^2$, so we must have that $v_1$ and $v_3$ has opposite signs, and one of them is large, hence $(v_1 - v_3)^2$ is larger than a constant. This concludes our proof.

## A.7 Proof of Update Rule 5 Convergence in the Lipschitz Case

**Proof** Let $B$ be an upper bound over $\|\mathbf{w}^{(t)}\|$ for all time step $t$, and over $\|\mathbf{v}^*\|$. Moreover, assume $|u| \leq c$ for some constant $c$. We denote the update rule by $\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} + \eta\tilde{\nabla}$, and bound from below:

$$\begin{aligned} \|\mathbf{w}^{(t)} - \mathbf{v}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{v}^*\|^2 &= 2\langle \mathbf{w}^{(t)} - \mathbf{v}^*, \eta\tilde{\nabla}\rangle - \eta^2\|\tilde{\nabla}\|^2 \\ &= 2\eta\,\mathbb{E}\left[(u((\mathbf{w}^{(t)})^\top x) - u((\mathbf{v}^*)^\top x))(\mathbf{w}^{(t)} - \mathbf{v}^*)\mathbf{x}\right] - \eta^2\|\tilde{\nabla}\|^2 \\ &\overset{(1)}{\geq} \frac{2\eta}{L}\mathbb{E}\left[(u((\mathbf{w}^{(t)})^\top x) - u((\mathbf{v}^*)^\top x))^2\right] - \eta^2\|\tilde{\nabla}\|^2 \\ &\overset{(2)}{\geq} \frac{2\eta}{L}\mathbb{E}\left[(u((\mathbf{w}^{(t)})^\top x) - u((\mathbf{v}^*)^\top x))^2\right] - \eta^2 B^2 c^2 \end{aligned}$$

where (1) follows from $L$-Lipschitzness and monotonicity of $u$, (2) follows from bounding $\|\mathbf{w}\|$, $\|\mathbf{x}\|$ and $u$. Let the expected error of the regressor parametrized by $\mathbf{w}^t$ be denoted $e^t$. We separate into cases:

- If $\|\mathbf{w}^t - \mathbf{v}^*\|^2 - \|\mathbf{w}^{t+1} - \mathbf{v}^*\|^2 \geq \eta^2 B^2 c^2$, we can rewrite $\|\mathbf{w}^t - \mathbf{v}^*\|^2 - \eta^2 B^2 c^2 \geq \|\mathbf{w}^{t+1} - \mathbf{v}^*\|^2$, and note that since $\|\mathbf{w}^{t+1} - \mathbf{v}^*\|^2 \geq 0$, and $\|\mathbf{w}^0 - \mathbf{v}^*\|^2 \leq B^2$, there can be at most $\frac{B^2}{\eta^2 B^2 c^2} = \frac{1}{\eta^2 c^2}$ iterations where this condition will hold.
- Otherwise, we get that $e^t \leq \eta B^2 c^2 L$.

Therefore, for a given $\epsilon$, by taking $T = \frac{c^4 B^4 L^2}{\epsilon^2}$, and setting $\eta = \sqrt{\frac{1}{Tc^2}}$, we obtain that after $T$ iterations, the first case is not holding anymore, and the second case implies $e^T \leq \epsilon$. ■

## B Technical Lemmas

**Lemma 3** *Any parity function over $d$ variables is realizable by a network with one fully connected layer of width $\tilde{d} > \frac{3d}{2}$ with ReLU activations, and a fully connected output layer with linear activation and a single unit.*

**Proof** Let the weights entering each of the first $\frac{3d}{2}$ hidden units be set to $\mathbf{v}^*$, and the rest to $0$. Further assume that for $i \in [d/2]$, the biases of the first $3i + \{1, 2, 3\}$ units are set to $-(2i - \frac{1}{2})$, $-2i$, $-(2i + \frac{1}{2})$ respectively, and that their weights in the output layer are $1$, $-2$, and $1$. It is not hard to see that the weighted sum of those triads of neurons is $\frac{1}{2}$ if $\langle \mathbf{x}, \mathbf{v}^* \rangle = 2i$, and $0$ otherwise. Observe that there's such a triad defined for each even number in the range $[d]$. Therefore, the output of this net is $0$ if $y = -1$, and $\frac{1}{2}$ otherwise. It is easy to see that scaling of the output layer's weights by $4$, and introduction of a $-1$ bias value to it, results in a perfect predictor. ■

## C Command Lines for Experiments

Our experiments are implemented in a simple manner in python. We use the tensorflow package for optimization. The following command lines can be used for viewing all optional arguments:

To run experiment 2.1, use:

```
python ./parity.py --help
```

To run experiment 3.1, use:

```
python ./tuple_rect.py --help
```

For SNR estimations, use:

```
python ./tuple_rect_SNR.py --help
```

To run experiment 3.2, use:

```
python ./dec_vs_e2e_stocks.py --help
```

For Section 4's experiments, given below are the command lines used to generate the plots. Additional arguments can be viewed by running:

```
python PWL_fail1.py --help
```

To run experiment 4.1.1, use:

```
python PWL_fail1.py --FtoK
```

To run experiment 4.1.2, use:

```
python PWL_fail1.py --FtoKConv
```

To run experiment 4.1.3, use:

```
python PWL_fail1.py --FtoKConvCond --batch_size 10
--number_of_iterations 500 --learning_rate 0.99
```

To run experiment 4.1.4, use:

```
python PWL_fail1.py --FAutoEncoder
```

To run Section 5's experiments, run:

```
python step_learn.py --help
```