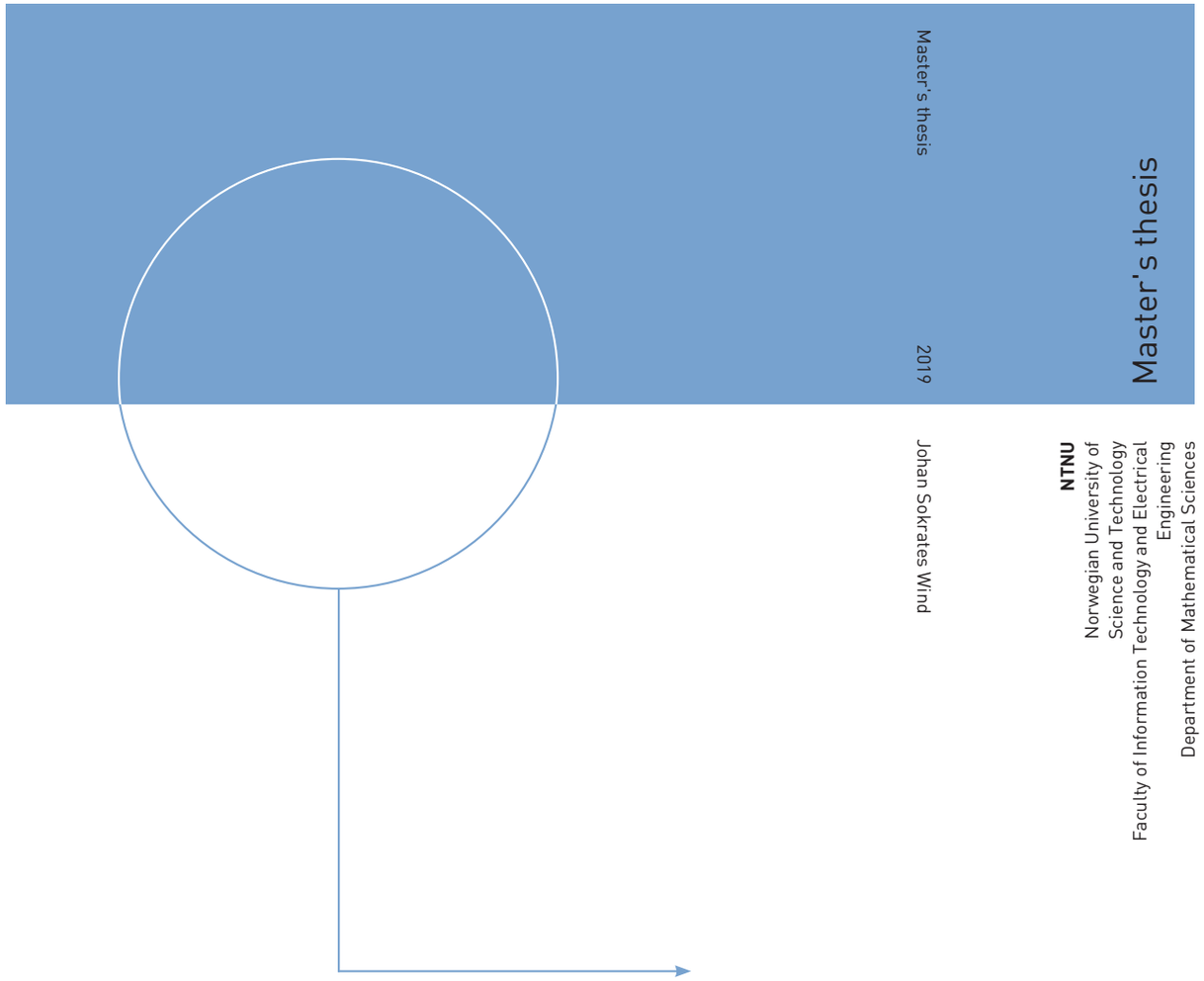

Master's thesis

2019

Johan Sokrates Wind

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Mathematical Sciences

Johan Sokrates Wind

Linearized Difference Networks as Approximate Kernel Methods

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

Linearized Difference Networks as Approximate Kernel Methods

Johan Sokrates Wind

Master of Science in Physics and Mathematics
Submission date: June 2019
Supervisor: Thiago Guerrera Martins

Norwegian University of Science and Technology
Department of Mathematical Sciences

Sammendrag

I denne artikkelen studerer vi nevrale nettverk som tilnærminger til en relatert kjerne-metode (kernel method). Vi analyserer forutsigelsene gjort av et enkelt nevralt nettverk med et skjult lag. To kilder til varians i forutsigelsene blir identifisert og eliminert gjennom introduksjonen av *Lineære Differeanse-Nettverk*. Det vises at for en uendelig ensemble av uendelig vide nevrale nettverk, vil forutsigelsene bli nøyaktig de samme som for en kjerne-metode. Vi verifiserer empirisk at endelige Lineære Differanse-Nettverk faktisk gir forutsigelser mer lignende denne ideelle kjerne-metoden enn et normalt nevralt nettverk.

Abstract

In this paper we study neural networks as approximations to a related kernel method. We analyze the predictions made by a vanilla neural network with one hidden layer. Two sources of variance in the predictions are identified and eliminated by introducing *Linearized Difference Networks*. It is shown that in the case of an infinite ensemble of infinitely wide neural networks (IEIN limit), the predictions will be exactly the same as those produced by a kernel method. We verify empirically that finite Linearized Difference Networks indeed produce predictions closer to this ideal kernel method than a vanilla neural network.

1 Introduction

Neural networks achieve impressive results in many domains such as speech recognition and computer vision. However, a complete understanding to why these methods work, is missing. There is a large discrepancy between practical use and the theory supporting it. For example, theory says that training even a neural network with a single neuron is NP-hard [11], while in practice neural networks are trained consistently and successfully with simple first-order methods.

Another challenge in understanding neural networks, is that they train highly flexible models with many times more parameters than training samples. Empirical results show that large, wide networks perform better than smaller ones. [10] shows that practical neural networks can perfectly fit random data. This makes neural networks a poor fit for many theoretical bounds and theory traditionally applied in machine learning.

Recent works (reviewed in the next section) focus on over-parameterized neural networks, and show that these networks are often *easier* to analyze than smaller ones. They also show that in the limit of infinite width, neural networks are often tractable to analyze analytically. In this paper we will analyze neural networks as approximations to an Infinitely large Ensemble of Infinitely wide Networks (IEIN limit). More precisely, we compare neural networks to the mean output (or equivalently expected output) of an infinite collection of infinitely wide, independently randomly initialized neural networks. Throughout the paper, we assume that the IEIN limit produces desirable predictions, and therefore benchmark neural network predictions against the IEIN limit's predictions. We believe the IEIN limit is a desirable goal, since empirically, wider networks typically perform better, and by using an infinite ensemble, we remove all randomness (with respect to initialization) from the predictions.

2 Related work

[3] shows that sufficiently overparameterized neural networks with one hidden layer are guaranteed to converge to zero training error in the regression setting. Their setup is the same as the one we are working with, and their chosen neural network is our vanilla network. The required overparameterization is huge ($> 10^{30}$ parameters for MNIST [6]) compared to realistic implementations. They show

that the training dynamics depend on a Gram matrix H which approaches a constant matrix H^∞ in the limit of infinite number of hidden neurons, H^∞ is only a function of the training data (not the initialization). They prove that H^∞ is strictly positive definite under weak assumptions, which they show is sufficient for guaranteed exponential convergence to zero training error. [2] extends the work of [3] to deep neural networks in the case of smooth activation functions with bounded derivatives. They also improve the required amount of overparameterization, but it is still very far from practical sizes ($> 10^{20}$ parameters for MNIST).

[8] is studying a neural network with a single hidden layer. Their setup is identical to ours, except they rescale the network output according to the scale of the training targets. The required overparameterization is improved significantly compared to previous works. They need only about $\mathcal{O}(n^2)$ parameters to guarantee exponential convergence to zero training error, when the input data is drawn uniformly at random from the unit ball. Additionally, they give some insight into bounding the least eigenvalue of H^∞, and introduce many useful analytical tools to analyze neural network training.

[1] introduces a dual view on neural networks. This includes the dual activation and dual kernel. They use their dual view to give design principles, support empirical results and new ideas. In this paper, we make use of dual activation functions to calculate and analyze matrices such as H^∞.

[4] investigates the connection between neural networks and kernel methods. They show that during initialization, neural networks of any depth are equivalent to Gaussian processes in the limit of infinite width. They also show that in the limit of infinite width, the training dynamics are governed by a kernel they name the Neural Tangent Kernel. Finally, they investigate empirically how finitely overparameterized networks compare to the infinite width limit. They find that the behavior of finitely wide neural networks is close to the theoretical limit of infinite width.

[7] shows that in the limit of infinitely wide neural networks, the networks behave exactly as if they were linearized around their initialization. They also find excellent empirical agreement between finitely overparameterized networks and their linearization.

2.1 Contribution

- We analyze the predictions made by a fully trained, infinitely wide, shallow neural network. During this analysis, we identify a zero-mean noise term in the predictions, which can be removed to achieve deterministic predictions equal to the IEIN limit. We propose a simple alteration, applicable to many neural networks, which removes this noise term, and call the resulting models *difference networks*. This change is also applicable to finitely sized neural networks.

- We further employ recent research, which shows that in the infinite limit, neural networks make the same predictions as if they were linearized around their initialization. This motivates us to introduce *linearized difference networks*, also for finitely sized networks, which we show empirically produce predictions closer to the IEIN limit than the networks they were linearized from. These networks have desirable theoretical properties, and share

the same computational complexity as the network they were based upon (up to constant factors).

- We supplement recent works on identifying the IEIN limit as a kernel method, by providing explicit formulas and plots of the dual activation functions, and their derivatives, of many common activation functions. These let us explicitly compute the IEIN limit predictions for our shallow network model, which we use as reference in the empirical experiments.

- We use the connection with kernel methods to calculate specific numerical values for $\lambda_{min}(H^{\infty})$ (the least eigenvalue of H^{∞}) and $\|X\|_2$, for CIFAR10 [5] and MNIST [6] datasets, comparing some simple alternatives for dataset preprocessing. These quantities are essential to several recent analytical bounds on the training dynamics [3][2][8]. The quantities are dependent on neural network architecture, here we pick a simple neural network with one hidden layer and ReLU activations. However, to the author's knowledge, this is the first time any numerical values for these quantities have been presented for real world datasets.

3 Neural network model

We consider a neural network with one hidden layer in the regression setting, practically identical to the networks considered in [3] and [8]. We call this network the vanilla network, in contrast to the Linearized Difference Network introduced in the following sections.
The network parameters are $W \in \mathbb{R}^{m \times d}$, the weights of the first layer, and $a \in \mathbb{R}^m$, the output weights. Here m is the number of hidden neurons, n is the number of training samples and d is the input dimension.
The neural network output u can be computed as follows.

$$u(x) = \sigma(xW^T)a \tag{1}$$

Here σ is the activation function of the neural network.
W and a are initialized element-wise independently as follows:

$$W_{ij} \sim N(0,1) \tag{2}$$

$$a_i \sim \frac{1}{\sqrt{m}}\text{unif}[\{-1,1\}] \tag{3}$$

The initialization of a ensures that we have zero-mean expected output at initialization, and the scaling factor $\frac{1}{\sqrt{m}}$ ensures that the network behaves similarly when changing the number of hidden neurons m. Specifically, it will allow us to define H^{∞} in later sections. We note that initializing $a_i \sim N(0, \frac{1}{m})$ would have the same effects, but we choose to follow [3] and [8].
We train the network on a training dataset with inputs $X \in \mathbb{R}^{n \times d}$ and targets $y \in \mathbb{R}^n$. We denote the i'th training sample X_i and assume that the features are normalized so $\|X_i\| = 1$ for all training samples.

The loss function used is the square loss:

$$L = \frac{1}{2} \|y - u(X)\|^2 \tag{4}$$

We train the first layer of the network (W) using gradient flow, which can be interpreted as gradient descent with infinitesimal step length. $W(0)$ denotes the weights at initialization, and $W(t)$ may be computed by the following.

$$\frac{dW}{dt} = -\frac{\partial L}{\partial W}^T \tag{5}$$

A note on notation: in this paper, we abuse notation by implicitly applying functions element-wise when applied to vectors and matrices. For example $\sigma(X)_{ij} = \sigma(X_{ij})$.

4 Motivation

4.1 Limiting dynamics of predictions

We may study the dynamics of the predictions $u(t)$ during training:

$$\frac{du}{dt} = \frac{\partial u}{\partial \theta}\frac{d\theta}{dt} = -\frac{\partial u}{\partial \theta}\frac{\partial L}{\partial \theta}^T = \tag{6}$$

$$\frac{\partial u}{\partial \theta}\frac{\partial u}{\partial \theta}^T (y - u) = H(t)(y - u) \tag{7}$$

Here we defined $H(t) := \frac{\partial u}{\partial \theta}\frac{\partial u}{\partial \theta}^T$. θ is a vector containing all the parameters we are optimizing. In our case, we train only W, so θ contains all the elements of W.

[3] note that in the limit of infinite number of hidden neurons (i.e $m \to \infty$), $H \to H^\infty$, where H^∞ is independent of t and initialization. It is only dependent on the input training data X. When training only the first layer (W), we obtain the following equality.

$$H^\infty_{ij} = X_i \cdot X_j \ \mathbb{E}_{w \sim N(0,I)}(\sigma'(X_i \cdot w)\sigma'(X_j \cdot w)) \tag{8}$$

In the case of normalized input (i.e $\|X_i\| = 1$), H^∞_{ij} is a function of only the dot product $X_i \cdot X_j$. Then, $H^\infty_{ij} = f_\sigma(X_i \cdot X_j)$ for some function f_σ. We also note that if we define $k(x, y) := f_\sigma(x \cdot y)$, we get the kernel induced by the neural network training dynamics. We can calculate the function $f_\sigma(x)$, and hence H^∞, efficiently using tools from section 9. Properties and plots of f_σ are also deferred to that section.

4.2 Limiting predictions

We look at the prediction $u_z(t)$ our infinitely wide neural network would make for a new feature vector $z \in \mathbb{R}^d$. First we look at the training dynamics of $u_z(t)$.

$$\frac{du_z}{dt} = \frac{\partial u_z}{\partial \theta}\frac{d\theta}{dt} = \frac{\partial u_z}{\partial \theta}\frac{\partial u}{\partial \theta}^T \big(y - u(t)\big) = H_z(t)\big(y - u(t)\big) \tag{9}$$

Here we defined $H_z(t) := \frac{\partial u_z}{\partial \theta}\frac{\partial u}{\partial \theta}^T$.
In the limiting case of infinite width $H_z(t) \to H_z^\infty = f_\sigma(Xz)^T$. In the limiting case, we also have the exact dynamics of the predictions on the training data

$$\frac{du}{dt} = H^\infty(y - u(t)) \implies \tag{10}$$

$$\frac{d(y-u)}{dt} = -H^\infty(y - u(t)) \implies \tag{11}$$

$$y - u(t) = \exp(-H^\infty t)(y - u(0)) \tag{12}$$

Now we are ready to calculate the predictions of the fully trained network.

$$u_z(\infty) = u_z(0) + \int_0^\infty H_z(t)(y - u(t))\, dt = \tag{13}$$

$$u_z(0) + \int_0^\infty H_z^\infty \exp(-H^\infty t)(y - u(0))\, dt = \tag{14}$$

$$u_z(0) + H_z^\infty (H^\infty)^{-1}(y - u(0)) \tag{15}$$

We finally arrive at the following important equation for the predictions of the fully trained network:

$$\boxed{u_z(\infty) = H_z^\infty (H^\infty)^{-1} y \; + u_z(0) - H_z^\infty (H^\infty)^{-1} u(0)} \tag{16}$$

4.2.1 Connection to kernel method

The first term in the predictions, $H_z^\infty (H^\infty)^{-1} y$, is what a kernel method with kernel $k(x, y) = f_\sigma(x \cdot y)$ would predict. To see this, we may rewrite the predictions in a notation more common to kernel methods.

$$u_z = \sum_{i=1}^{n} k(X_i, z)\alpha_i \tag{17}$$

If we optimize the weights $\alpha \in \mathbb{R}^n$ for zero error on the training data X, we require

$$\sum_{i=1}^{n} k(X_i, X_j)\alpha_i = y_j, \text{ for all } j \in \{1..n\} \tag{18}$$

Replacing $k(X_i, z) = (H_z^\infty)_i$ and $k(X_i, X_j) = H_{ij}^\infty$ gives:

$$u_z = H_z^\infty \alpha \tag{19}$$

$$H^\infty \alpha = y \tag{20}$$

From this we conclude:

$$u_z = H_z^\infty (H^\infty)^{-1} y \tag{21}$$

4.2.2 Remaining terms

We now want to know how the remaining term, $u_z(0) - H_z^\infty (H^\infty)^{-1} u(0)$, contribute to the predictions. First we look at the expected initial output of the network on any fixed input feature $x \in \mathbb{R}^d$, $u_x(0)$.

$$\mathbb{E}(u_x(0)) = \mathbb{E}_{w\sim N(0,I), a\sim \text{unif}[\{-1,1\}]} \left(\sigma(w \cdot x)a\right) = \tag{22}$$

$$\mathbb{E}_{w\sim N(0,I)} \left(\sigma(w \cdot x)\right) \mathbb{E}_{a\sim \text{unif}[\{-1,1\}]}(a) = 0 \tag{23}$$

Here we utilized the fact that a and w are initialized independently, and that a has expectation zero. Since $u_z(0) - H_z^\infty (H^\infty)^{-1} u(0)$ is a weighted sum (with weights independent of initialization) of zero-mean variables, it has expectation zero. This indicates that this is an unwanted noise term, as it would not be existent in mean predictions of an infinite ensemble of independently initialized networks (IEIN limit).

We now want to study the amplitude, to see if this error is negligible (e.g goes to zero for infinitely large networks) or not.

$$\text{Var}(u(0)) = \mathbb{E}(u(0)u(0)^T) = \mathbb{E}_{w\sim N(0,I), a\sim \text{unif}[\{-1,1\}]}(\sigma(Xw^T)a^2\sigma(Xw^T)^T) = \tag{24}$$

$$\mathbb{E}_{w\sim N(0,I)}(\sigma(Xw^T)\sigma(Xw^T)^T) = g_\sigma(XX^T) \tag{25}$$

Here we defined the element-wise function

$$g_\sigma(x \cdot y) := \mathbb{E}_{w\sim N(0,I)}(\sigma(x \cdot w)\sigma(y \cdot w)), \text{ for all } x, y \text{ s.t } \|x\| = \|y\| = 1 \tag{26}$$

This function shares many similarities with f_σ. We can calculate it efficiently using tools from section 9, there are also plots and analytical expressions for g_σ in that section. g_σ typically has magnitude around 1, and is therefore *not* generally negligible.

As will become apparent in the following section, $u_z(0) - H_z^\infty (H^\infty)^{-1} u(0)$ comes from undoing the non-zero neural network output at initialization. Since the output at initialization $u(0)$ is independent of the magnitude of the targets y, the term will mostly be seen when the targets have small magnitude, while for large magnitudes $u(0)$ will be negligible compared to y. This will also be seen in the numerical experiments in section 8.

5 Difference networks

We may simply remove the noise term, $u_z(0) - H_z^\infty (H^\infty)^{-1} u(0)$, discussed in the previous section, completely. One way to do this is to subtract the network output at initialization from the output of the trained network. Formally, we define the a vanilla neural network with parameters θ as a function $u_\theta(x)$ from an input feature vector x to a prediction. We denote the vanilla network at initialization $u_{\theta(0)}$. In our case, θ denotes W and a. We introduce the *difference network*, described by the following equation.

$$\Delta u = u_\theta - u_{\theta(0)} \tag{27}$$

Note that $\Delta u_{\theta(0)}(x) = 0$ for any input feature vector x, i.e at initialization the difference network has zero output for all inputs.

We may train the first layer (W) of the difference network by gradient flow, the same way as the vanilla network.

$$\frac{dW}{dt} = -\frac{\partial L}{\partial W}^T \tag{28}$$

The loss of the difference network becomes:

$$L = \frac{1}{2} \left\| y - \Delta u(X) \right\|^2 = \frac{1}{2} \left\| y + u_{\theta(0)}(X) \; - \; u_\theta(X) \right\|^2 \tag{29}$$

We note that the loss is exactly the same as training a vanilla neural network with shifted targets $y' = y + u_{\theta(0)}(X)$. This allows us to reuse the results derived in the previous sections. Specifically, in the infinite width limit, equations (9) and (12) become

$$\frac{\Delta u(z)}{dt} = \frac{du(z)}{dt} = H_z^\infty (y' - u_{\theta(t)}(X)) \tag{30}$$

$$y' - u_{\theta(t)}(X) = \exp(-H^\infty t)(y' - u_{\theta(0)}(X)) = \exp(-H^\infty t) y \tag{31}$$

Together, these give us the dynamics of the predictions.

$$\frac{\Delta u(z)}{dt} = H_z^\infty \exp(-H^\infty t) y \tag{32}$$

Integrating from initialization, we can calculate the predictions of the trained difference network:

$$\Delta u_z(\infty) = \Delta u_z(0) + \int_0^\infty H_z^\infty \exp(-H^\infty t) y \, dt = H_z^\infty (H^\infty)^{-1} y \tag{33}$$

The predictions of the infinitely wide difference network are hence exactly the predictions of the IEIN limit, with no variance.

6 Finite width

In practice we are not working with infinitely wide neural networks. It is therefore of interest to look at how close realistically sized neural networks are to the infinite limit.

6.1 H vs H^∞ at initialization

Previous papers[3][2][8] have focused on bounding the minimum eigenvalue of $H(t)$, which is necessary for guaranteeing exponential convergence of the training error. We instead look at the matrix $(H^\infty)^{-\frac{1}{2}} H (H^\infty)^{-\frac{1}{2}}$, and prove that it is close to identity. This implies that H is close to H^∞, which means the predictions of the finitely sized neural network become close to the ones of the IEIN limit.

We will apply the matrix Chernoff bound [9]. $\lambda_{min}(A)$ and $\lambda_{max}(A)$ denote the minimum and maximum eigenvalues of the symmetric matrix A, respectively.

Theorem (Matrix Chernoff)**.** *Consider a finite sequence $\{\mathbf{X}_k\}$ of independent, random, self-adjoint matrices with dimension n. Assume that each random matrix satisfies*

$$\mathbf{X}_k \succeq \mathbf{0} \quad \textit{and} \quad \lambda_{max}(\mathbf{X}_k) \leq R \quad \textit{almost surely.}$$

Define

$$\mu_{min} := \lambda_{min}\big(\sum_k \mathbb{E}\,\mathbf{X}_k\big) \quad \textit{and} \quad \mu_{max} := \lambda_{max}\big(\sum_k \mathbb{E}\,\mathbf{X}_k\big).$$

Then

$$\mathbb{P}\Big\{\lambda_{min}\big(\sum_k \mathbf{X}_k\big) \leq (1-\delta)\mu_{min}\Big\} \leq n \cdot \Big[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\Big]^{\mu_{min}/R}$$

$$\mathbb{P}\Big\{\lambda_{max}\big(\sum_k \mathbf{X}_k\big) \geq (1+\delta)\mu_{max}\Big\} \leq n \cdot \Big[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\Big]^{\mu_{max}/R}$$

We split $H(0)$ into m independent positive semi-definite matrices. Here $\text{diag}(x)$ is the square matrix with the vector x along its diagonal.

$$H_k = \frac{1}{m}\text{diag}\big[\sigma'\big(XW_k(0)\big)\big] X X^T \text{diag}\big[\sigma'\big(XW_k(0)\big)\big] \tag{34}$$

$$H(0) = \sum_k H_k \tag{35}$$

We assume that $|\sigma'(x)| \leq B$, so $\sigma(x)$ is Lipschitz continuous with constant B. This is the case for most common activation functions, for example all activation function discussed in section 9. Most

remaining activation functions, like $\sigma(x) = \frac{1}{2}x^2$, have bounded derivative in a small section around $x = 0$ (say $x \in [-10, 10]$). We may treat these as Lipschitz by noting that with high probability, only a small section (say 10 standard deviations) around $x = 0$ of the function will ever be encountered due to initialization by the normal distribution.
When $|\sigma'(x)| \leq B$, we have

$$\|H_k\| \leq \frac{1}{m} B^2 \|X\|^2 \tag{36}$$

$$\left\| (H^\infty)^{-\frac{1}{2}} H_k (H^\infty)^{-\frac{1}{2}} \right\| \leq \frac{B^2 \|X\|^2}{m \lambda_{min}(H^\infty)} \tag{37}$$

Because $\mathbb{E}(H_k) = \frac{1}{m} H^\infty$, we also have $\mu_{min} = \mu_{max} = 1$.

Applying the Matrix Chernoff bound now gives:

$$\mathbb{P}\Big\{ \lambda_{min}\big((H^\infty)^{-\frac{1}{2}} H(0) (H^\infty)^{-\frac{1}{2}} \big) \leq 1 - \delta \Big\} \leq n \cdot \Big[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}} \Big]^{m/\kappa} \tag{38}$$

$$\mathbb{P}\Big\{ \lambda_{max}\big((H^\infty)^{-\frac{1}{2}} H(0) (H^\infty)^{-\frac{1}{2}} \big) \geq 1 + \delta \Big\} \leq n \cdot \Big[\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \Big]^{m/\kappa} \tag{39}$$

$$\kappa = \frac{B^2 \|X\|^2}{\lambda_{min}(H^\infty)} \tag{40}$$

Note that bounding this matrix also bounds the minimum eigenvalue of $H(0)$, giving an equally powerful bound to the one obtained in [8].

$$\lambda_{min}\big((H^\infty)^{-\frac{1}{2}} H(0) (H^\infty)^{-\frac{1}{2}} \big) \geq 1 - \delta \tag{41}$$

$$\implies \lambda_{min} H(0) \geq (1 - \delta) \lambda_{min}(H^\infty) \tag{42}$$

This means if we have $m \geq c \log(n) \frac{B^2 \|X\|^2}{\lambda_{min}(H^\infty)}$ for some sufficiently large constant c, we will with high probability have a well-conditioned optimization problem at initialization. I.e at initialization, $\|y - u(0)\|$ is shrinking at an exponential rate. The same condition on m also tells us when the predictions of the neural network will be close to the predictions of the IEIN limit.
Note that when $\lambda_{min}(H^\infty)$, B and $\log(n)$ are roughly constants, and $\|X\|^2$ scales as $\frac{n}{d}$, this gives the requirement $md \geq c'n$. I.e that the number of parameters exceeds the number of training samples by a (large) constant. This is the common assumption for overparameterized neural networks, as it often holds true for neural networks in practice [10].

7 Linearization

In the previous section we bounded H to H^∞ at initialization. However, H may change during the optimization process, as it is dependent on W, which is optimized. Previous works [8][3][2] have tried to bound the change in H by exploiting properties of σ and bounding the distance traveled by $W(t)$ from initialization. We may however eliminate the problem of the changing H by linearizing the neural network around its initialization.

[7] showed empirically that wide neural networks of any depth behave similarly to their linearization. They also show that infinitely wide networks behave exactly as their linearization. This can be understood intuitively by the fact that since there are an infinite number of weights, each weight will change infinitesimally. Hence the activation function will only be evaluated very close to its initialization, and in this neighborhood the function behaves like its linearization (assuming it is differentiable in the linearization point).
By linearization we mean taking the first order Taylor expansion of the neural network around its initialization, with respect to the parameters:

$$u_\theta^1 = u_{\theta(0)} + \frac{\partial u_\theta}{\partial \theta}\Big|_{\theta=\theta(0)} (\theta - \theta(0)) \tag{43}$$

It is hard to prove whether the non-linearity in the optimization process will bring $H(t)$ closer or further away from H^∞ than its initialization $H(0)$. However, we argue informally that it will most likely move $H(t)$ away from H^∞. Put shortly, the dynamics of $H(t)$ at initialization are highly dependent on the random initialization of a. This gives the impression that it moves in a random direction depending on the initialization, and this makes it unlikely for $H(t)$ to move towards H^∞. Also, empirical experiments indicate that the linearized model gives predictions closer to the IEIN limit than its corresponding non-linear vanilla network.
Another appealing aspect of linearization is that it is easier to find optimization algorithms with good convergence and guarantees for linear models compared to non-linear models.

7.1 Linearized difference networks

The linearized difference network can be written as

$$\Delta u_\theta^1 = u_\theta^1 - u_{\theta(0)} = \frac{\partial u_\theta}{\partial \theta}\Big|_{\theta=\theta(0)} (\theta - \theta(0)) = \frac{\partial u_\theta}{\partial \theta}\Big|_{\theta=\theta(0)} \Delta\theta \tag{44}$$

Here we defined $\Delta\theta = \theta - \theta(0)$.
A simple, approximate method for linearizing a difference network is by scaling down the targets y during training and then scaling up predictions during testing. An informal proof sketch of the validity of this approach is given in the following. Let ϵ be a small constant denoting the scaling of the targets. Here we let θ denote only the parameters we are optimizing (i.e W in our case). The loss and dynamics are as follows:

$$L = \frac{1}{2} \|\epsilon y - \Delta u\|^2 \tag{45}$$

$$\frac{d\theta}{dt} = -\frac{\partial L}{\partial \theta}^T = -\frac{\partial \Delta u}{\partial \theta}^T (\epsilon y - \Delta u) = -\epsilon \frac{\partial \Delta u}{\partial \theta}^T (y - \frac{1}{\epsilon}\Delta u) \tag{46}$$

$$\frac{d\frac{1}{\epsilon}\Delta u}{dt} = \frac{1}{\epsilon}\frac{\partial \Delta u}{\partial \theta}\frac{d\theta}{dt} = \frac{\partial \Delta u}{\partial \theta}\frac{\partial \Delta u}{\partial \theta}^T (y - \frac{1}{\epsilon}\Delta u) \tag{47}$$

Note that since $\frac{d\theta}{dt}$ is proportional to ϵ, when $\epsilon \to 0$, the perturbation of θ from its initialization becomes infinitesimal. This in turn gives

$$\frac{d\frac{1}{\epsilon}\Delta u}{dt} = \frac{\partial \Delta u}{\partial \theta}\frac{\partial \Delta u}{\partial \theta}^T (y - \frac{1}{\epsilon}\Delta u) \approx \frac{\partial \Delta u}{\partial \theta}\Big|_{\theta=\theta(0)} \frac{\partial \Delta u}{\partial \theta}^T\Big|_{\theta=\theta(0)} (y - \frac{1}{\epsilon}\Delta u) \tag{48}$$

We recognize (48) as the dynamics of the linearized model, with $\frac{1}{\epsilon}\Delta u \approx \Delta u^1_\theta$. This method, although simple, is numerically unstable, i.e it might suffer from rounding errors if scaling down y too much. For a better implementation we should parameterize the network by the difference from initialization, as done in (44). This gives a more involved implementation that gives the exact linearization and no numerical problems.

A third possible implementation, which may be implemented efficiently by automatic differentiation software is as follows. We define the model $u_{\theta(0)+\delta\Delta\theta}$, where δ is a scalar parameter which automatic differentiation is used on. This gives us the simple formula

$$\Delta u^1_\theta = \frac{du_{\theta(0)+\delta\Delta\theta}}{d\delta}\Big|_{\delta=0} \tag{49}$$

The advantage of this method is that it may be used to produce predictions efficiently for neural network architectures with multiple outputs, useful for different loss functions than the sum squared loss. However, using this expression for training (i.e differentiating again on $\Delta\theta$) requires control over the automatic differentiation which may be hard to get in some deep learning frameworks.

7.2 Interpretation as data-independent transform

We note that $\Delta u^1_\theta = \frac{\partial u_\theta}{\partial \theta}\Big|_{\theta=\theta(0)} \Delta\theta$ may be viewed as a linear combination of transformed input features. The surprising fact is that the transform, $\frac{\partial u_\theta}{\partial \theta}\Big|_{\theta=\theta(0)}$, is independent of the training data! This means that the linearized difference network may be seen as a way to randomly generate a data-independent transformation of the input features to a high-dimensional space, and then running a linear algorithm like linear regression or logistic regression on these transformed features. The optimization algorithm is usually stochastic gradient descent, and the feature weights ($\Delta\theta$) are initialized to zero. This interpretation enables us to apply the vast array of methods developed for linear methods (for example linear regression), and their theoretical guarantees. One simple idea is to center and normalize each input feature dimension, after the non-linear transformation. Another idea is to apply lasso regularization to find the most important feature dimensions. We leave this as an interesting direction of future work.

8 Numerical verification

To numerically verify the reduced noise of the linearized difference network compared to vanilla networks, we train both on a subset of MNIST and measure which predictions are closer to the IEIN limit. The IEIN limit is computed directly using the connection to a kernel method, using closed-form expressions (58) and (62) from section 9.2.

Our training data consists of the first 50 zeros and 50 ones from the MNIST dataset. The test set consists of the next 50 zeros and 50 ones. All these input features are normalized to have unit length $\|X_i\| = 1$. The targets are set to 0 for zeros and 1 for ones. The targets are then scaled, this scaling factor is given on the x-axes of figures 1, 2 and 3. We train both a vanilla network and a linearized difference network (implemented using equation (44)). These networks share the same initial parameter weights before training. ReLU and erf(2x) were used as activation functions. We train each network for 10000 batch gradient descent iterations with learning rate 0.01, these hyper-parameters were manually selected to be sufficient for convergence. The networks were then evaluated on the unseen test data. If a network gave output u, it's prediction error was given as $\frac{\|u-u_{IEIN}\|}{\|y\|}$, where u_{IEIN} was computed analytically by applying the limiting kernel method (see section 4.2.1) on the training and test data.

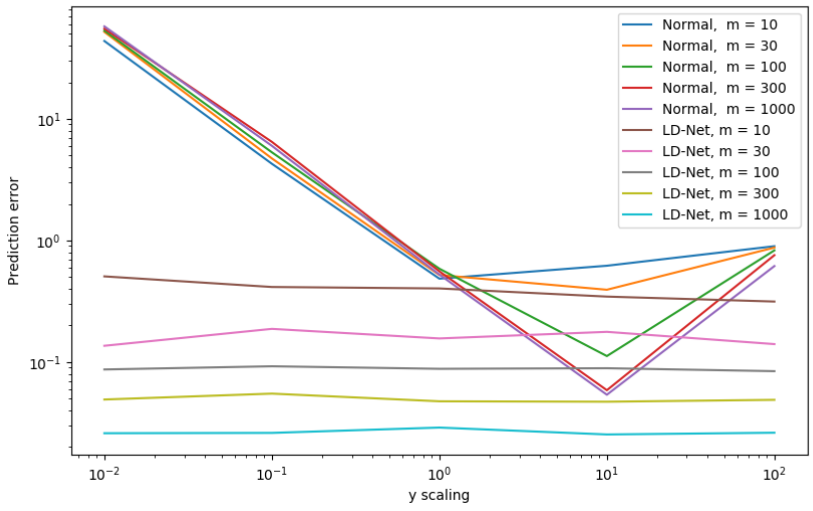

Figure 1: Comparison between linearized difference network (LD-Net) and vanilla network (Normal), with the erf activation function, $\sigma(x) = \text{erf}(2x)$. The prediction error along the y-axis is $\frac{\|u-u_{IEIN}\|}{\|y\|}$.

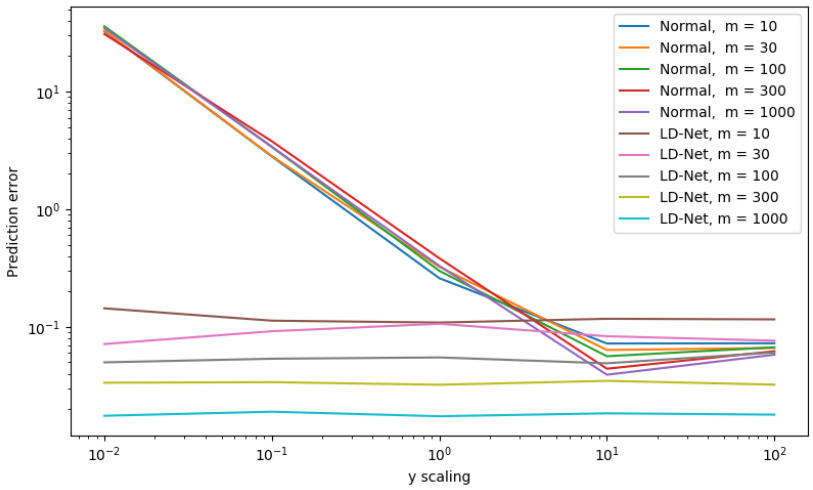

Figure 2: Comparison between linearized difference network (LD-Net) and vanilla network (Normal), with the ReLU activation function, $\sigma(x) = \max\{x, 0\}$. The prediction error along the y-axis is $\frac{\|u - u_{IEIN}\|}{\|y\|}$.

From figures 1 and 2 we see that linearized difference networks generally produce predictions closer to the IEIN limit than the corresponding vanilla networks. Note also that the linearized difference network is independent of target scaling (any fluctuations are due to variance in the sampling and possibly numerical rounding errors, see figure 3).

We can clearly see that the initialization is causing a lot of variance in the vanilla network predictions for low target scales. This noise is independent of target scale, and therefore makes up a large portion of the total noise when the targets are small. Even at no target scaling (y scale $= 10^0$) the initialization causes the bulk of the variance in the vanilla networks. On the other hand, when the target scales are large, non-linearity from moving parameters causes most of the noise. Because the ReLU activation function is mostly linear, it suffers less noise from non-linearity than the more non-linear erf(2x) activation. Vanilla networks with bounded activation functions, like erf, will have even more problems when fitting large targets, as they may never produce output larger than $O(\sqrt{m})$. This is not a problem for linearized difference networks, as they may produce predictions of any magnitude, giving the same predictions (up to scale) independent of target scale.

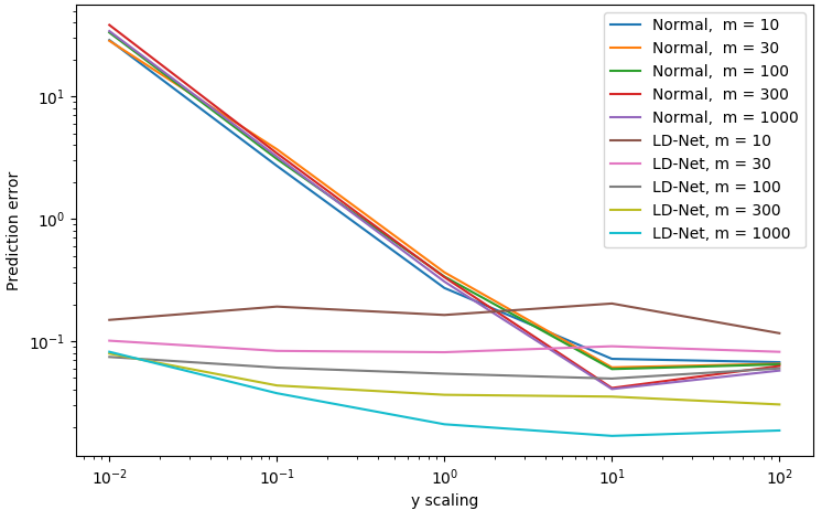

Figure 3: Comparison between linearized difference network (implemented as a difference network with targets downscaled 1000 times) and vanilla network (Normal), with the ReLU activation function, $\sigma(x) = \max\{x, 0\}$. The prediction error along the y-axis is $\frac{\|u-u_{IEIN}\|}{\|y\|}$.

We include an example of results obtained by implementing the linearized difference network as a difference network with downscaled targets in Figure 3. We see the numerical rounding errors causing problems for large networks with already low target scales, in the lower left part of the figure. We need to be careful to avoid this combination of small targets, large networks and high target downscaling in the implementation. This simple and efficient implementation can otherwise be seen to produce results very close to the implementation using equation (44), which was used in Figure 2.

9 Dual activation

Within our analysis we often come across expressions involving f_σ and g_σ. The function $f_\sigma(c)$ can be used for calculating $H^\infty = f_\sigma(XX^T)$, and $g_\sigma(c)$ can be used for calculating the covariance of the initial output of the neural network $g_\sigma(XX^T)$. $g_\sigma(c)$ also describes the kernel we get if we train the output layer instead of the input layer $H^\infty = g_\sigma(XX^T)$.

It is therefore of interest to calculate f_σ and g_σ for common activation functions, and find closed form expressions when possible. [1] introduces the dual of an activation function. In their terms, g_σ is the dual activation of the activation function σ. They give many more or less practically useful properties of the dual activation function and its relationship with the original activation function. In this section, we list some properties allowing for practical calculation of f_σ and g_σ. Additionally, we compute and plot f_σ and g_σ numerically for a variety of activation functions.

9.1 Practical tools for calculation

The tools in this section can be derived from results in [1]. This section is however arguably more specific towards computing f_σ and g_σ, than the results in [1].
If we have ways to calculate $g_\sigma(c)$, we can use the following identities to calculate $f_\sigma(c)$, and extend properties from $g_\sigma(c)$ to $f_\sigma(c)$. We therefore focus on properties of $g_\sigma(c)$.

$$f_\sigma(c) = c\, g_{\sigma'}(c) = c\, g'_\sigma(c) \tag{50}$$

We want to efficiently calculate expectations of the following form:

$$g_\sigma(x \cdot y) = \mathbb{E}_{w \sim N(0,I)}(\sigma(x \cdot w)\sigma(y \cdot w)), \text{ for } x, y \text{ s.t } \|x\| = \|y\| = 1 \tag{51}$$

One useful way to rewrite the function is as follows:

$$g_\sigma(c) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \sigma(x)\sigma(y)\frac{1}{2\pi\sqrt{1-c^2}} \exp(\frac{2xyc - x^2 - y^2}{2(1-c^2)})\, dx\, dy \tag{52}$$

The problem is also closely related to Hermite polynomials, as shown in the following form.

$$g_\sigma(c) = \sum_{k=0}^{\infty} c^k \Big(\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi k!}} \sigma(x) He_k(x) \exp(-\frac{x^2}{2})\, dx \Big)^2 \tag{53}$$

Here, $He_k(x)$ is the k'th probabilists' Hermite polynomial.
Noting that $\gamma_k := \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi k!}} \sigma(x) He_k(x) \exp(-\frac{x^2}{2})\, dx$ is the k'th coefficient of the Hermite series of σ, we can also write (53) as:

$$g_\sigma(c) = \sum_{k=0}^{\infty} c^k \gamma_k^2 \tag{54}$$

Differentiation of σ leads to differentiation of g_σ:

$$g_{\sigma'}(c) = g'_\sigma(c) \tag{55}$$

Rescaling σ rescales g_σ:

$$g_{a\sigma}(c) = a^2 g_\sigma(c) \tag{56}$$

Offsetting σ, offsets g_σ:

$$g_{\sigma+b}(c) = g_\sigma(c) + b^2 + 2b\sqrt{g_\sigma(0)} \tag{57}$$

9.2 Duals of common activation functions

Using the tools from the preceeding section, we compute the analytical forms of several activation functions (58-63). We note again that $f_\sigma(c) = c\, g'_\sigma(c)$.

$$f_{\text{ReLU}}(c) = \frac{c}{2}\big(1 - \frac{\cos^{-1}(c)}{\pi}\big) \tag{58}$$

$$g_{\text{ReLU}}(c) = \frac{\sqrt{1-c^2} + c\big(\pi - \cos^{-1}(c)\big)}{2\pi} \tag{59}$$

$$f_{\sin+\cos}(c) = ce^{c-1} \tag{60}$$

$$g_{\sin+\cos}(c) = e^{c-1} \tag{61}$$

$$f_{a\cdot\text{erf}(\gamma x)+b}(c) = \frac{2a^2 c}{\pi}\big((1+\frac{1}{2\gamma^2})^2 - c^2\big)^{-\frac{1}{2}} + b^2 \tag{62}$$

$$g_{a\cdot\text{erf}(\gamma x)+b}(c) = \frac{2a^2}{\pi}\tan^{-1}\Big(c\,\big((1+\frac{1}{2\gamma^2})^2 - c^2\big)^{-\frac{1}{2}}\Big) + b^2 \tag{63}$$

When interpreting (58-63) as kernels, it is important to remember the condition $\|x\| = \|y\| = 1$. This means $c = x \cdot y = 1 - \frac{\|x-y\|^2}{2}$. Then we can rewrite for example, $g_{\sin+\cos}(x \cdot y) = e^{-\frac{\|x-y\|^2}{2}}$ and recognize it as the RBF kernel.

We implemented equation (53) by numerical integration, and use this to plot f_σ and g_σ for a variety of standard activation functions used in neural networks. See figure 4 and figure 5. Each result in (58-63) was indistinguishable from it's numerical approximation when overlaying the numerical and exact solutions in plots.

We note that the sigmoid and tanh activation functions are very well approximated by rescaled versions of erf. We define

$$\text{erf}_a(x) := \frac{\text{erf}(x/2.4)+1}{2} \approx \text{sigmoid}(x) \tag{64}$$

$$\text{erf}_b(x) := \text{erf}(x/1.2) \approx \tanh(x) \tag{65}$$

We may use these approximations to get approximate closed form expressions for $f_{\text{sigmoid}}(c)$, $g_{\text{sigmoid}}(c)$, $f_{\tanh}(c)$ and $g_{\tanh}(c)$. We note that there are several other similar activation functions (like $\tan^{-1}$) which can also be approximated in this way.

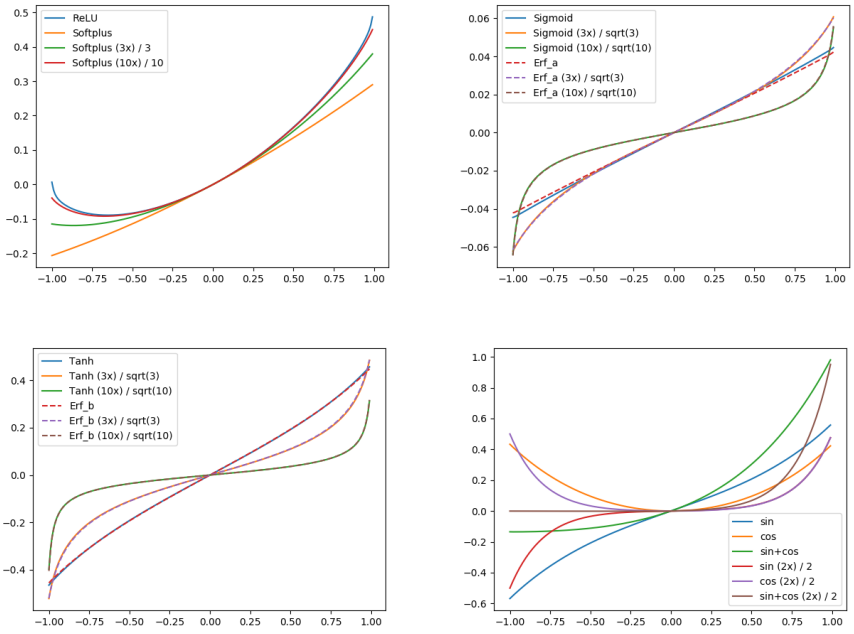

Figure 4: Numerical f_σ for different activation functions and different scalings of the activation functions. For example "Tanh (3x) / 3" means $\sigma(c) = \frac{1}{3}\tanh(3x)$

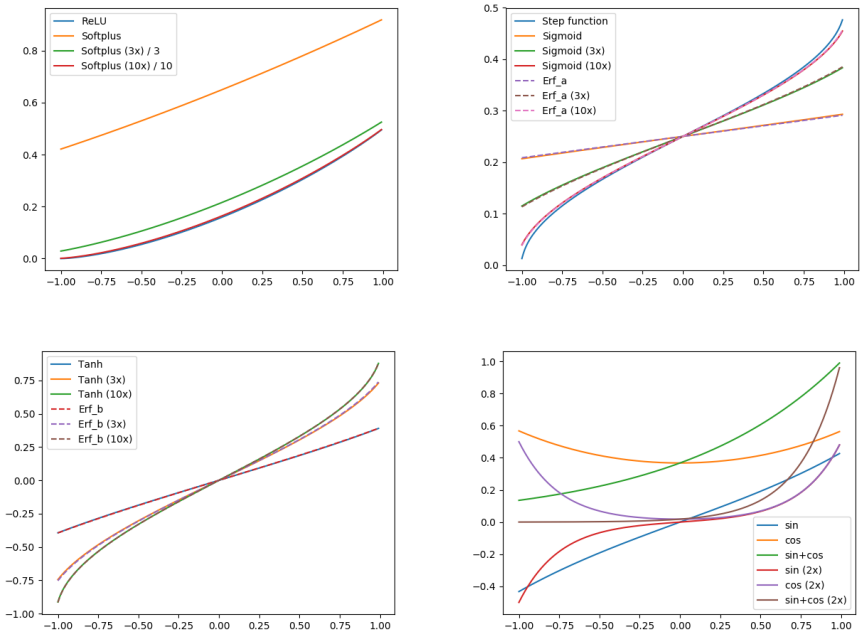

Figure 5: Numerical g_σ for different activation functions and different scalings of the activation functions. For example "Tanh (3x) / 3" means $\sigma(c) = \frac{1}{3}\tanh(3x)$

9.3 $\lambda_{min}(H^\infty)$ and $\|X\|$ for real datasets

In the case of finite width neural networks, as shown in section 6.1, we depend on the two quantities $\|X\|$ and $\lambda_{min}(H^\infty)$ (the minimum eigenvalue of H^∞) to bound the approximation of H^∞ by H. These two quantities were also contained in the bounds of [8], [2] and [3], both for bounding the number of required hidden neurons m, and for bounding the convergence rate of gradient descent training.

[8] studied these quantities under the assumption that each training sample was chosen uniformly from the unit ball in dimension d. They claim that $\lambda_{min}(H^\infty)$ is roughly constant, and $\|X\|$ scales roughly as $\sqrt{\frac{n}{d}}$ "in many cases". Verification of these claims for real datasets has however, to the author's knowledge, not been done. We calculating the quantities for two common datasets: MNIST [6] and CIFAR10 [5].

We consider three different ways to preprocess the input features:

- **Normalization**
  Each training image is normalized so that $\|X_i\| = 1$.
- **Zero-mean**
  We subtract the average image intensity of each image from it $X_i \to X_i(\mathrm{I} - \frac{1}{d}\mathbf{1}\mathbf{1}^T)$. Here $\mathbf{1}$ is a column vector of ones, and I is the identity matrix. The image is then normalized.
- **Whitening**
  Each training image is first made to have zero mean. The dataset is then whitened by singular value decomposition, so that each of the d feature directions has unit variance (PCA whitening). Each image is then normalized.

We calculated $\|X\|^2$ by finding the largest eigenvalue of the symmetric matrix $X^T X$. H^∞ for the ReLU activation function, was obtained by using the closed form expression $f_{\text{ReLU}}(XX^T)$ given in equation (58). We note that the results were calculated using 32-bit floating point numbers, so they might contain slight rounding errors.

	$\|X\|^2$	$\lambda_{min} H^\infty$
MNIST / normalization	24454.8	0.01515
MNIST / zero-mean	18488.5	0.01574
MNIST / whitening	314.871	0.02003
CIFAR10 / normalization	40885.2	0.00647
CIFAR10 / zero-mean	8154.87	0.01311
CIFAR10 / whitening	33.7366	0.09878

For MNIST, $n = 60000$, $d = 784$. While for CIFAR10, $n = 50000$, $d = 3072$. We see that preprocessing can have a dramatic effect on how well the dataset is conditioned. With whitening we come close to the bounds on random samples from the unit ball predicted by [8] for $\|X\|$. For MNIST: $314.871/4.1 < \frac{n}{d} \approx 77$, while for CIFAR10: $33.7366/2.1 < \frac{n}{d} \approx 16$.

10 Discussion

10.1 Extension to other over-parameterized models and loss functions

Most of the central derivations in this paper make few assumptions on the neural network architecture. The main requirements for applying linearized difference networks to a model are: optimization by (stochastic) gradient descent and training an over-parameterized model. The technical requirement of H^∞ existing in the limit of infinite width, is often automatically satisfied, because the models are usually already scaled so they don't change output scaling when increasing the number of parameters. Supported models includes convolutional networks, recurrent networks, some attention models, different loss functions, etc. In these cases, the linearized difference networks should produce predictions closer to the IEIN limit than their vanilla counterparts for the same reasons as our shallow network. These linearized difference networks would also enjoy the theoretical properties explained in section 7.2.

To apply linearized difference networks in these settings we may use the same definition as in (44), but generalized to vector outputs.

$$\Delta u_\theta^1 = \frac{\partial u_\theta}{\partial \theta}\Big|_{\theta=\theta(0)} \Delta\theta \tag{66}$$

The network output Δu_θ^1 may then be trained by (stochastic) gradient descent on any loss function. For practical implementations, we may use the tricks and reformulations described in section 7.1. In this paper we only verified the validity of using linearized difference networks on a simple, shallow neural network with accessible IEIN limit. Evaluating the performance of these more complicated linearized difference models on real world datasets remains an exciting avenue for future research.

10.2 Using a neural network vs directly using the limiting kernel method

Since we can compute the exact IEIN limit using kernel methods, it might seem unnecessary to train neural networks at all. However, there are advantages to using neural networks. The most important advantage might be that the computational cost of evaluating a neural network does not depend on the number of training samples. Even if the number of parameters of the neural network is much larger than the number of training samples, the number of hidden nodes is typically smaller. This may lead to better computational complexity compared to kernel methods (which often require the complexity of about one hidden node per training sample). Additionally, several techniques, such as early stopping, dropout, etc., were developed to improve neural networks, and some of them might be hard to apply successfully to kernel methods.

11 Conclusion

In this paper, we have introduced linearized difference networks, an extension to over-parameterized neural networks. This model was analytically shown to remove noise in the predictions made by vanilla neural networks, in the case of infinitely wide networks, when compared to an infinite

ensemble of infinitely wide neural networks (IEIN limit). We also showed empirically that even for finitely sized linearized difference networks, they produce predictions closer to the IEIN limit than their vanilla counterparts. This was made possible through analytical analysis allowing us to compute the exact predictions of the IEIN limit, by exploiting the connection to a kernel method. Finally, we discussed how linearized difference networks may be extended to deep networks and different loss functions, and view this as an exciting direction of future research.

References

[1] Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. *CoRR*, abs/1602.05897, 2016.

[2] Simon S. Du, Jason D. Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *CoRR*, abs/1811.03804, 2018.

[3] Simon S. Du, Xiyu Zhai, Barnabás Póczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *CoRR*, abs/1810.02054, 2018.

[4] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *CoRR*, abs/1806.07572, 2018.

[5] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[6] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[7] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. *arXiv e-prints*, page arXiv:1902.06720, Feb 2019.

[8] Samet Oymak and Mahdi Soltanolkotabi. Towards moderate overparameterization: global convergence guarantees for training shallow neural networks. *CoRR*, abs/1902.04674, 2019.

[9] Joel A. Tropp. User-friendly tail bounds for sums of random matrices. *arXiv e-prints*, page arXiv:1004.4389, Apr 2010.

[10] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv e-prints*, page arXiv:1611.03530, Nov 2016.

[11] J. Šíma. Training a single sigmoidal neuron is hard. *Neural Computation*, 14(11):2709–2728, Nov 2002.