Function Approximation Using Rectified Linear Units

Nicholas Johnson, Teodor-Andrei Andronache, Paula Gradu

1 Universal Approximation Theorem and Limitations

Many applications of neural networks consist of giving the network a set of input and their function values of a certain (usually smooth) function and asking the network to approximate the function. One could ask which conditions on the architecture of a network are necessary to accomplish this task, at least on a theoretical level.

We call a bounded function $\sigma : \mathbb{R} \to \mathbb{R}$ **sigmoidal** if $\lim_{x\to-\infty} \sigma(x)$ and $\lim_{x\to\infty} \sigma(x)$ exist and are finite. The Universal Approximation Theorem states that this task can be done (with arbitrarily small error) by a feed-forward network with a single hidden layer containing a finite number of neurons, under mild assumptions on the activation function:

Theorem 1.1. *(Universal Approximation Theorem)*[1] *Let σ be a continuous sigmoidal function. Then for any continuous function f defined on a compact set $K \subset \mathbb{R}^n$ and for any $\epsilon > 0$, there are some $N \in \mathbb{N}, a_i \in \mathbb{R}^n$ and $b_i, c_i \in \mathbb{R}$ for any $i = \overline{1, N}$ such that*

$$\sup_{x\in K} \left| f(x) - \sum_{i=1}^{N} c_i \sigma(a_i^T x + b_i) \right| < \epsilon.$$

Although the theorem is very powerful in describing sufficient conditions on the activation function and on the architecture by stating that every continuous function can be approximated by some network with one hidden layer of N neurons and continuous sigmoidal activation function, it does not say anything about the computational complexity of such a network with respect to the error ϵ. We are interested in bounding the wideness N by a function of ϵ (such as a polynomial of $1/\epsilon$). The following theorem states a similar result and also addresses this problem:

Theorem 1.2. *(Debao Chen's Theorem)*[2] *For any continuous function $f : [0, 1] \to \mathbb{R}$, any sigmoidal function σ and $n \in \mathbb{N}$, there are some $a_i, b_i, c_i \in \mathbb{R}$ for any $i = \overline{1, n}$ such that*

$$\sup_{x\in[0,1]} \left| f(x) - \sum_{i=1}^{n} c_i \sigma(a_i x + b_i) \right| < ||\sigma||_\infty \omega\left(f, \frac{1}{n}\right), \tag{1}$$

where $||\sigma||_\infty := \sup_{x\in[0,1]} \sigma(x)$ and $\omega(f, \delta) = \sup_{\substack{x,\, y \in [0,1] \\ |x-y|\le\delta}} |f(x) - f(y)|$, for any $\delta > 0$.

Note that $\omega\left(f, \frac{1}{n}\right)$ measures the local variation of the function f (when $n \to \infty$). We expect that in general, for a smooth σ, small variation of f yields a better approximation

[1] For the proof, see [1].

[2] For the proof, see Theorem 6, Chapter 22 of [3].

because any linear combination $\sum_{i=1}^{n} c_i\sigma(a_ix + b_i)$ is smooth. However, if $\omega\left(f, \frac{1}{n}\right)$ is large, then it is more difficult for any such linear combination to capture the large variation of f.

One limitations of Theorem 1.2. is that it does not say anything for the case when the activation function is unbounded, which is the case of ReLU activation function. Intuitively, we can concatenate two translations of ReLU and get a sigmoidal function, and then we expect Theorem 1.2. to give us an approximation of f. However, the problem with this approach is that the left side (1) involves $||\sigma||_\infty$, and we have little control over the approximation bound.

In this paper, we are going to address this issue by proving a similar result to Theorem 1.2., but for ReLU instead of a sigmoidal activation function:

Theorem 1.3. *For any $f : [0, 1] \to \mathbb{R}$, we can find $a_i, b_i \in \mathbb{R}$ and $c_i \in \{\pm 1\}$, $i = \overline{1, 2n + 1}$, such that:*

$$\sup_{x\in[0,1]} \left| f(x) - \sum_{i=1}^{2n+1} c_i \cdot ReLU(a_ix + b_i) \right| \leq \frac{1}{2}\,\omega\left(f, \frac{1}{n}\right) + \omega\left(f, \frac{1}{2n}\right).$$

Note that the right side coincides with the previous error if f is a Lipschitz function and σ is normalized. The unboundness of ReLU doubles the number of necessary neurons for a similar error. Also, note that we can assume that the factors c_i are ± 1 because any positive factor can be introduced inside the ReLU function.

The rest of this paper is dedicated to first provide some relevant background in neural networks, proving Theorem 1.3., and then presenting experiments to study how feasible approximation using ReLU activation function is.

2 Background Information

The building block of a Multilayer Perceptron (MLP) is the single neuron. Given an input vector $x \in \mathbb{R}^k$, the single neuron outputs $a = \sigma(w^T x + b) \in \mathbb{R}$ where $w \in \mathbb{R}^k$ is an arbitrary weight vector, $b \in \mathbb{R}$ is an arbitrary bias value and $\sigma : \mathbb{R} \to \mathbb{R}$ is a (typically non-linear) activation function. A hidden layer l of a MLP consists of $n_h^{[l]} \in \mathbb{N}$ individual neurons acting on the same input vector. The output of layer l is the vector $a^{[l]} \in \mathbb{R}^{n_h^{[l]}}$ where $a^{[l]}(j) = \sigma_j^{[l]}(w_j^{[l]T} x + b_j^{[l]})$ and $\sigma_j^{[l]}$, $w_j^{[l]}$ and $b_j^{[l]}$ denote the activation function, weight vector and bias of the j^{th} neuron in layer l respectively. $a^{[l]}$ is then given as the input to layer $l + 1$. Let $L \in \mathbb{N}$ denote the total number of layers in a MLP. The architecture of a MLP is characterized by specifying L, $n_h^{[l]}$ for $l = \overline{1, L}$, and $\sigma_j^{[l]}$ for $l = \overline{1, L}$ and $j = \overline{1, n_h^{[l]}}$. Typically, on any given layer l, we have $\sigma_j^{[l]} = \sigma^{[l]}$ for $j = \overline{1, n_h^{[l]}}$ for some choice of activation function $\sigma^{[l]}$. A MLP is a function $f(\cdot|\theta)$ that maps $\mathbb{R}^p \to \mathbb{R}^{n_h^{[L]}}$. θ denotes the set of parameters $\{W^{[l]}, b^{[l]}\}_{l=1}^{L}$ where $W^{[l]} = [w_j^{[l]}(i)]_{ij} \in \mathbb{R}^{n_h^{[l-1]} \times n_h^{[l]}}$ and $b^{[l]} = [b_j^{[l]}]_j \in \mathbb{R}^{n_h^{[l]}}$. Given some input $x \in \mathbb{R}^p$, the MLP feeds x as the input into its first layer and returns as output the vector $a^{[L]}$ output by layer L of the MLP. Figure 1 shows a schematic of a MLP where $L = 2$, $p = 5$, $n_h^{[1]} = 3$, and $n_h^{[2]} = 2$ [2].

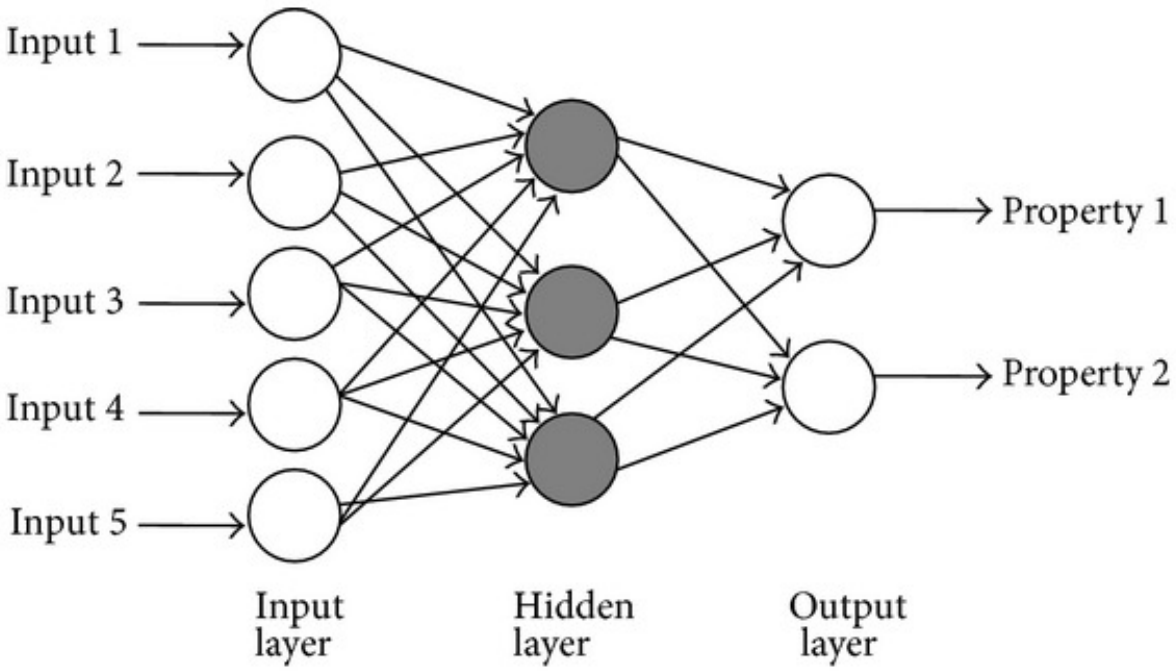

Figure 1: MLP Schematic

In this paper, we focus on MLPs $f(\cdot|\theta)$ with a single hidden layer using ReLU activation functions, an output layer using the linear activation function and no bias, a real valued input and a real valued output. The ReLU activation function is defined as $ReLU(x) := max(x, 0)$. Note that the ReLU function is defined element-wise for vector valued inputs. Formally, we set $L = 2$, $p = 1$, $n_h^{[2]} = 1$ and $b^{[2]} = 0$. Note that we do not specify $n_h^{[1]}$ here. Thus, we can write our MLP as

$$f(x|\theta) = \sum_{j=1}^{n_h^{[1]}} w_1^{[2]}(j) \cdot ReLU(w_j^{[1]} \cdot x + b_j^{[1]}) = w^{[2]T} ReLU(w^{[1]T} x + b^{[1]})$$

The collection of parameters θ is learned from training data $\{x_i, y_i\}_{i=1}^m$ by defining some loss function $\mathcal{L}(y, f(x|\theta)$ and minimizing $\mathcal{L}$ through an iterative optimization procedure, an example of which is Stochastic Gradient Descent as seen in lecture.

3 Proof of Theorem 1.3.

Lemma 3.1. *Any constant function $f(x) = c$ can be written as $sign(c) \cdot ReLU(|c|)$.*

Proof. $|c| \geq 0 \implies sign(c) \cdot ReLU(|c|) = sign(c) \cdot |c| = c$ as desired. □

Lemma 3.2. *Any function $h : [0, 1] \to \mathbb{R}$ of the form:*

$$h(x) = \begin{cases} 0 & x < \alpha \\ mx - m\alpha & x \in [\alpha, \beta] \\ m\beta - m\alpha & x > \beta \end{cases}$$

for $0 \leq \alpha \leq \beta \leq 1$ *and* $m \in \mathbb{R}$ *can be written as* $ReLU(mx - m\alpha) - ReLU(mx - m\beta)$.

Proof. (i) $x < \alpha \leq \beta \implies m(x-\alpha) < 0$ and $m(x-\beta) < 0 \implies$

$$ReLU(mx - m\alpha) - ReLU(mx - m\beta) = 0$$

(ii) $\alpha \leq x \leq \beta \implies m(x-\alpha) \geq 0$ and $m(x-\beta) \leq 0 \implies$

$$ReLU(mx - m\alpha) - ReLU(mx - m\beta) = mx - m\alpha$$

(iii) $\alpha \leq \beta < x \implies m(x-\alpha) > 0$ and $m(x-\beta) > 0 \implies$

$$\begin{aligned} ReLU(mx - m\alpha) - ReLU(mx - m\beta) &= mx - m\alpha - (mx - m\beta) \\ &= m\beta - m\alpha \end{aligned}$$

(i), (ii) and (iii) $\implies h(x) = ReLU(mx - m\alpha) - ReLU(mx - m\beta)$ as desired. □

Proof of Theorem 1.3. First consider the piecewise linear function $g : [0,1] \to \mathbb{R}$, defined by:

$$g(x) = n\left(f\left(\frac{i+1}{n}\right) - f\left(\frac{i}{n}\right)\right) \cdot x + \left((i+1)f\left(\frac{i}{n}\right) - if\left(\frac{i+1}{n}\right)\right)$$

if $x \in \left[\frac{i}{n}, \frac{i+1}{n}\right]$, for $i = \overline{1, n-1}$.

Clearly, $g\left(\frac{i}{n}\right) = f\left(\frac{i}{n}\right), \forall i = \overline{0, n-1}$. Therefore, if $x \in \left[\frac{i}{n}, \frac{i}{n} + \frac{1}{2n}\right]$,

$$\begin{aligned} \left|g(x) - g\left(\frac{i}{n}\right)\right| &= n\left|f\left(\frac{i+1}{n}\right) - f\left(\frac{i}{n}\right)\right| \cdot \left(x - \frac{i}{n}\right) \\ &\leq \frac{1}{2}\,\omega\left(f, \frac{1}{n}\right) \\ \implies |g(x) - f(x)| &\leq \left|g(x) - g\left(\frac{i}{n}\right)\right| + \left|f\left(\frac{i}{n}\right) - f(x)\right| \\ &\leq \frac{1}{2}\,\omega\left(f, \frac{1}{n}\right) + \omega\left(f, \frac{1}{2n}\right) \end{aligned} \tag{2}$$

Similarly, if $x \in \left[\frac{i}{n} + \frac{1}{2n}, \frac{i+1}{n}\right]$,

$$\begin{aligned} \left|g(x) - g\left(\frac{i+1}{n}\right)\right| &= n\left|f\left(\frac{i+1}{n}\right) - f\left(\frac{i}{n}\right)\right| \cdot \left(\frac{i+1}{n} - x\right) \\ &\leq \frac{1}{2}\,\omega\left(f, \frac{1}{n}\right) \\ \implies |g(x) - f(x)| &\leq \left|g(x) - g\left(\frac{i+1}{n}\right)\right| + \left|f\left(\frac{i+1}{n}\right) - f(x)\right| \\ &\leq \frac{1}{2}\,\omega\left(f, \frac{1}{n}\right) + \omega\left(f, \frac{1}{2n}\right). \end{aligned} \tag{3}$$

$$(2) \text{ and } (3) \implies |g(x) - f(x)| \leq \frac{1}{2}\,\omega\left(f, \frac{1}{n}\right) + \omega\left(f, \frac{1}{2n}\right), \forall x \in [0,1].$$

We now prove that g can be written as $\sum_{i=1}^{2n+1} c_i \cdot ReLU(a_i \cdot x + b_i)$: $\forall i = \overline{0, n-1}$, if we take $\alpha_i = \frac{i}{n}$, $\beta_i = \frac{i+1}{n}$ and $m_i = n\left(f\left(\frac{i+1}{n}\right) - f\left(\frac{i}{n}\right)\right)$ in Lemma 1.2., we have that:

$$h_i(x) = \begin{cases} 0 & x < \frac{i}{n} \\ n\left(f\left(\frac{i+1}{n}\right) - f\left(\frac{i}{n}\right)\right) \cdot \left(x - \frac{i}{n}\right) & x \in \left[\frac{i}{n}, \frac{i+1}{n}\right] \\ f\left(\frac{i+1}{n}\right) - f\left(\frac{i}{n}\right) & x > \frac{i+1}{n} \end{cases}$$

can be written as $ReLU(m_i x + m_i \alpha_i) - ReLU(m_i x + m_i \beta_i)$.

Consider $H(x) = \sum_{i=0}^{n-1} h_i(x)$ and look at some arbitrary $x_0 \in \left[\frac{i_0}{n}, \frac{i_0+1}{n}\right]$:

$$\begin{aligned} \sum_{i=0}^{n-1} h_i(x_0) &= \sum_{i=0}^{i_0-1} h_i(x_0) + h_{i_0}(x_0) + \sum_{i=i_0+1}^{n-1} h_i(x_0) \\ &= \sum_{i=0}^{i_0-1} \left(f\left(\frac{i+1}{n}\right) - f\left(\frac{i}{n}\right)\right) + h_{i_0}(x_0) + \sum_{i=i_0+1}^{n-1} 0 \\ &= f\left(\frac{i_0}{n}\right) - f\left(\frac{1}{n}\right) + n\left(f\left(\frac{i_0+1}{n}\right) - f\left(\frac{i_0}{n}\right)\right) \cdot \left(x_0 - \frac{i_0}{n}\right) \\ &= g(x_0) - f\left(\frac{1}{n}\right). \end{aligned}$$

Since x_0 was arbitrary, we have that $g(x) = H(x) + f\left(\frac{1}{n}\right)$, $\forall x \in [0,1]$. (3)

By letting $a_{2i+1} = a_{2i+2} = m_i$, $b_{2i+1} = m_i \alpha_i$, $b_{2i+2} = m_i \beta_i$, $c_{2i+1} = 1$ and $c_{2i+2} = -1$, we can write $h_i(x) = c_{2i+1} \cdot ReLU(a_{2i+1} \cdot x + b_{2i+1}) + c_{2i+2} \cdot ReLU(a_{2i+2} \cdot x + b_{2i+2}), \forall i = \overline{0, n-1}$, we have that $H(x) = \sum_{i=0}^{n-1} h_i(x) = \sum_{i=1}^{2n} c_i \cdot ReLU(a_i \cdot x + b_i)$. (4)

By Lemma 1.1., since $f\left(\frac{1}{n}\right)$ is a constant, if we take $a_{2n+1} = 0$, $b_{2n+1} = f\left(\frac{1}{n}\right)$ and $c_{2n+1} = sign\left(f\left(\frac{1}{n}\right)\right)$, then $c_{2n+1} \cdot ReLU(a_{2n+1} \cdot x + b_{2n+1}) = f\left(\frac{1}{n}\right)$, $\forall x \in [0,1]$. (5)

(3), (4) and (5) $\implies g(x) = \sum_{i=1}^{2n+1} c_i \cdot ReLU(a_i \cdot x + b_i)$ as desired.

This completes the proof since we have shown that:

$$|g(x) - f(x)| \leq \frac{1}{2}\,\omega\left(f, \frac{1}{n}\right) + \omega\left(f, \frac{1}{2n}\right), \forall x \in [0,1].$$

4 Empirical Results and Analysis

In this section, we will explore the performance of a single hidden layer neural network with ReLU activations as a function of the number of neurons used (i.e. the width of the hidden layer).

Before we proceed, it is important to underline a limitation of the scope of our experiments. The theoretical result derived above guarantees the existence of a particular weight configuration for which the error falls below our bound $\frac{1}{2}\omega\left(f, \frac{1}{n}\right) + \omega\left(f, \frac{1}{2n}\right)$, but it does not tells us anything about how to find these weights nor about how likely we are to converge to a weight setup which satisfies this bound. In practice, one of two things will happen: we converge to a local minimum or we get stuck in a saddle point.

Thankfully, empirically, almost all local minima produce functions that are very close to the function corresponding to the globally optimal weight setup. As a result, it is very likely that the errors we get in our experiments are indeed representative of the error we would get using an optimal configuration. Therefore, from a practical standpoint, our experiments are actually highly illustrative of the general relationship between a single layer network's ability to fit a given function and its width.

The more problematic scenario is getting stuck in a saddle point. If this happens, we will observe a large error, possibly exceeding our bound. In this case, drawing any conclusions is impossible.

Another limitation of our experiments comes from the fact that Theorem 1.3. assumes that all the values of the function f are known, while a neural network is only given finite set of training points. It is possible that our approximation only overfits the function f for the known training sets. However, most of the functions that appear in practice are smooth and can be approximated well using a sample of their values, so this is less concerning than our previous issue.

4.1 Method and Notations

In the theoretical section above, we showed that a single hidden layer neural network with $2n + 1$ neurons can achieve error less than $\frac{1}{2}\omega\left(f, \frac{1}{n}\right) + \omega\left(f, \frac{1}{2n}\right)$. Because our plots will be in terms of the number of neurons N in the hidden layer, we will rewrite our bound $omega(f, N)$ in terms of N:

$$N = 2n + 1 \implies n = \frac{N-1}{2}$$

$$omega(f, N) = \frac{1}{2}\omega\left(f, \frac{2}{N-1}\right) + \omega\left(f, \frac{1}{N-1}\right).$$

To estimate the supremum of the absolute difference between our function and the function produced by the network, will compute the maximal absolute error over a validation set. We will compare this value to our bound $omega(f, N)$.

Finally, we will use Mean Absolute Error as our loss function during training. We will therefore also plot the Mean Absolute Error for the validation set to get additional insight into general performance (and not only the worst-case illustrated by max error).

4.2 Fitting $f(x) = \sqrt{x}$

By Lemma 1 in the Appendix, $\omega(f, \delta) = \sqrt{\delta} \implies$

$$omega(f, N) = \frac{1}{2}\omega\left(f, \frac{2}{N-1}\right) + \omega\left(f, \frac{1}{N-1}\right) = \frac{1+\sqrt{2}}{\sqrt{2(N-1)}}.$$

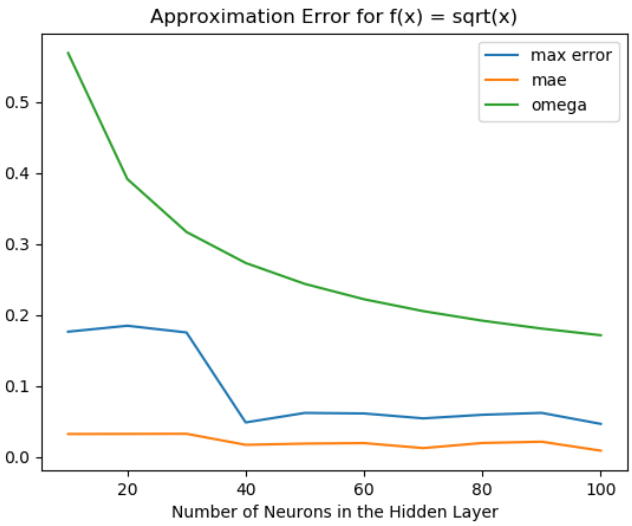

Figure 2: Plot of maximum error, mean absolute error and our bound omega as a function the number of neurons in the hidden layer for $f(x) = \sqrt{x}$.

From the plot in figure 2, we see that our bound is not tight for $\sqrt{x}$. This suggests that the learnability of $\sqrt{x}$ is not determined by its ω coefficient, but by some other intrinsic properties.

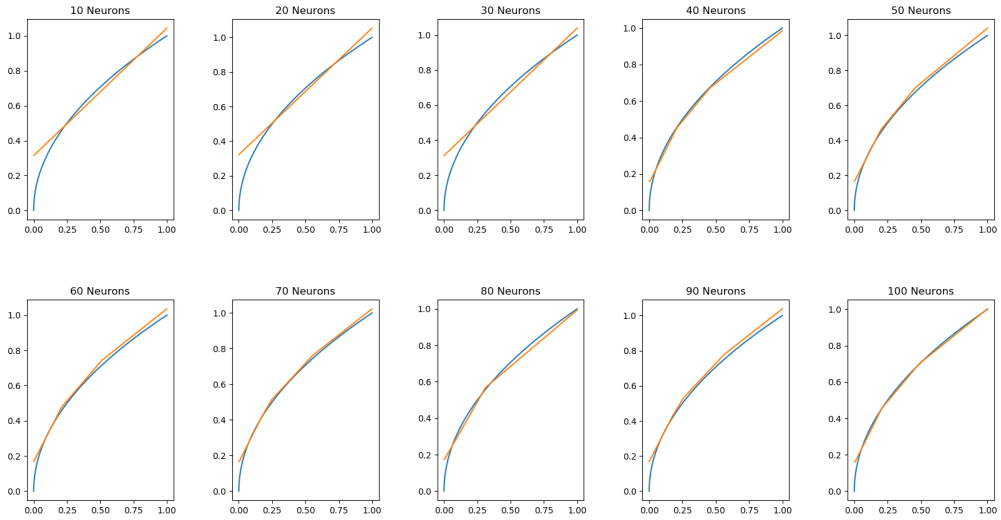

Figure 3: Plot of function learnt by network (orange) for increasing number of neurons against the true function $f(x) = \sqrt{x}$ (blue).

In the plots in figure 3, we see how increasing the width of the hidden layer allows the network to start learning the more variable section of the function, even in the neighborhood of 0. Based on these plots, we can also make a conjecture as to why $\sqrt{x}$ is easy to learn: far from 0, the graph of $\sqrt{x}$ can be approximated very well even by quite large linear segments.

Also, the fact that $\sqrt{x}$ can be approximated with error much smaller than *omega* is also a consequence of the fact that *omega* is proportional to $1/\sqrt{N}$. The function $\sqrt{x}$ has nice behavior far from 0 (being Lipschitz), but ω is not proportional to $1/N$ because of the behavior around 0. Thus, our task of approximating an almost Lipschitz function better than a quantity proportional to $1/\sqrt{N}$ becomes very easy.

Moreover, our network even ignores the behavior of $\sqrt{x}$ at 0. Since the error is proportional to the output size, we get that the values x in a neighborhood of 0 do not sufficiently impact mean absolute error (the loss) and therefore our network focuses on fitting the points far from 0 instead. This leads to the gap between maximum error and mean absolute error observed in figure 2. This also suggests that a different loss (one which correspondingly punishes deviations for small inputs) might push the network to fit $\sqrt{x}$ better.

4.3 Fitting $f(x) = \sin(2\pi x)$

By Lemma 2 in the Appendix, $\omega(f, \delta) \leq 2\pi\delta \implies$

$$omega(f, N) = \frac{1}{2}\omega\left(f, \frac{2}{N-1}\right) + \omega\left(f, \frac{1}{N-1}\right) \leq \frac{4\pi}{N-1}$$

Since $|f(x) - f(0)| = |f(x)| \approx 2\pi x$ for $x \approx 0$, we have that $\omega(f, \delta) \approx \delta$ for $\delta \approx 0$. Therefore, $omega(f, N) \approx \frac{4\pi}{N-1}$ for large enough number of neurons N so, in this case, our curve tightly captures the bound from the theoretical component.

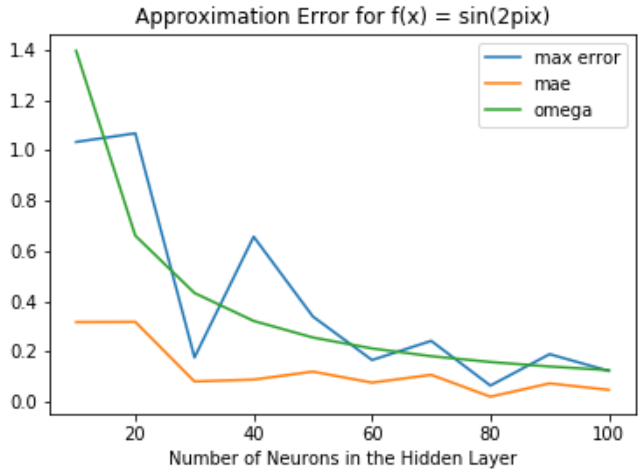

Figure 4: Plot of maximum error, mean absolute error and our bound omega as a function the number of neurons in the hidden layer for $f(x) = \sin(2\pi x)$.

Taking into account that the plot above corresponds to particular local minima and that the error across minima should generally be less variable, figure 4 suggests that our bound is quite tight for $\sin(2\pi x)$.

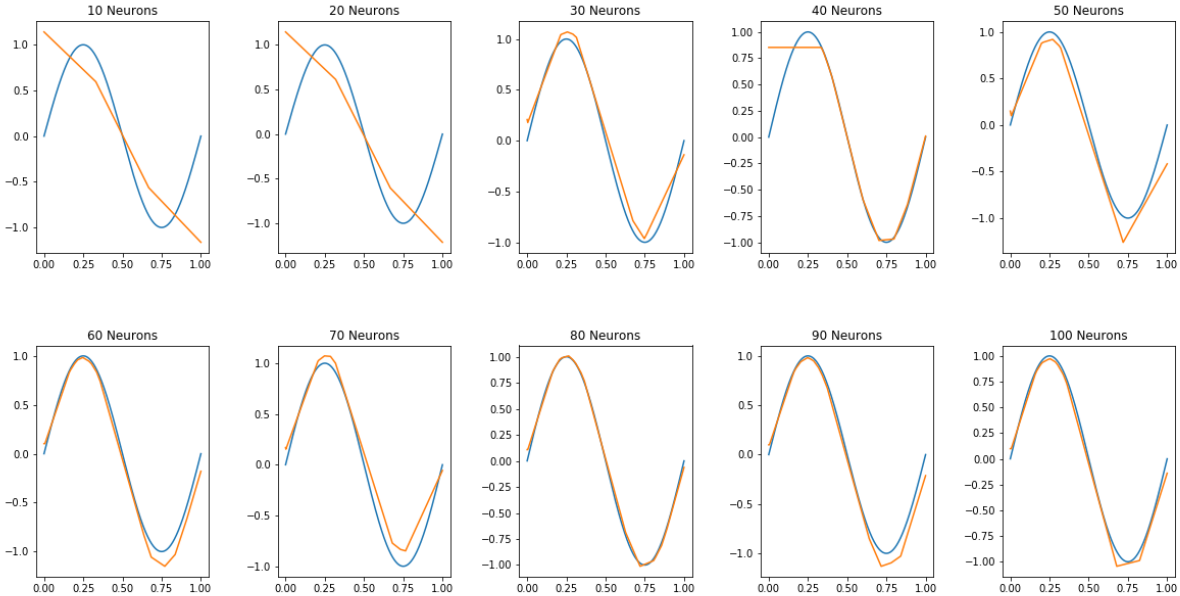

Figure 5: Plot of function learnt by network (orange) for increasing number of neurons against the true function $f(x) = \sin(2\pi x)$ (blue).

Figure 5 clearly illustrates why more neurons lead to better function approximation. Since $\sin(2\pi x)$, unlike $\sqrt{x}$, is smooth everywhere, the network treats mistakes over the entire

range $[0, 1]$ equally (as opposed to in the case of $\sqrt{x}$). Therefore, the rewards of increasing the width are more clearly observed in this case.

4.4 Fitting a Combination of ReLU Functions

Let $f(x) = 10ReLU(x - 0.8) - 6ReLU(x - 0.3) + 4ReLU(x - 0.5) - 2ReLU(0.2 - 2 * x)$. By Lemma 3 in the Appendix, $\omega(f, \delta) \leq 8\delta \implies$

$$omega(f, N) = \frac{1}{2}\omega\left(f, \frac{2}{N-1}\right) + \omega\left(f, \frac{1}{N-1}\right) \leq \frac{16}{N-1}$$

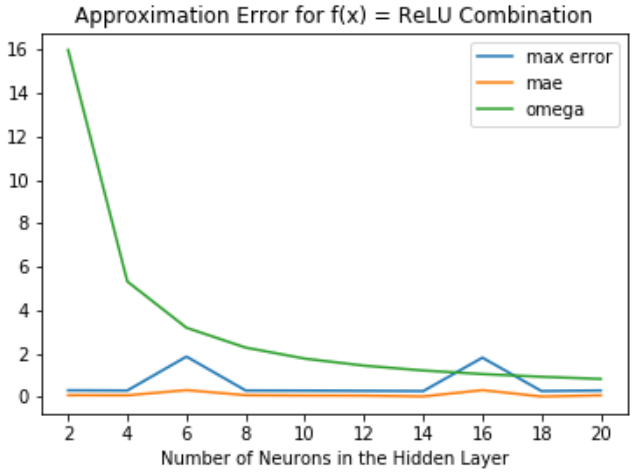

Figure 6: Plot of maximum error, mean absolute error and our bound omega as a function the number of neurons in the hidden layer.

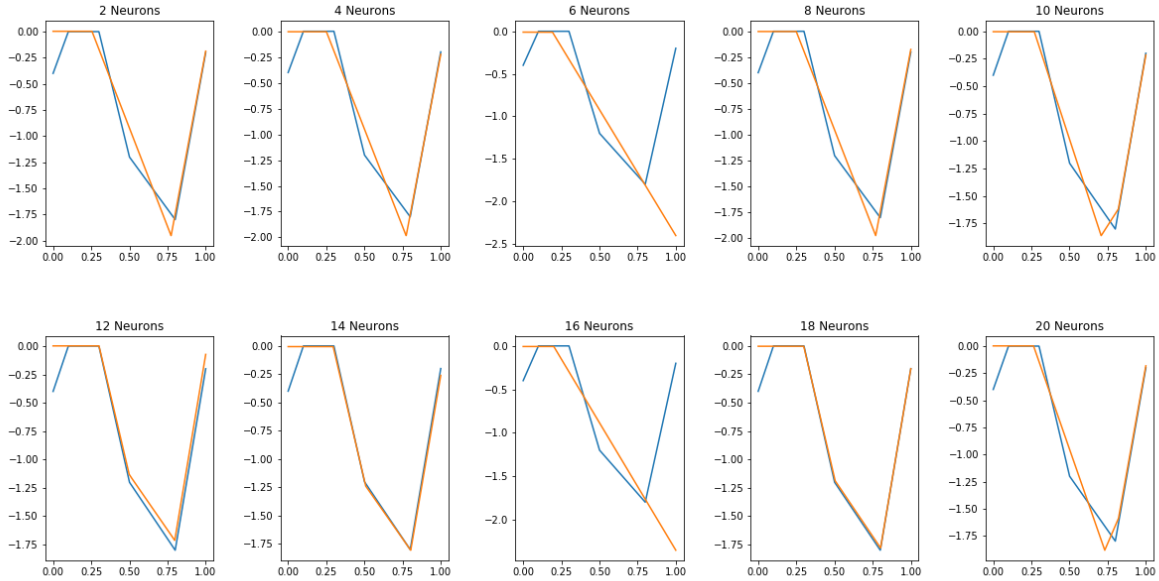

Figure 7: Plot of function learnt by network (orange) for increasing number of neurons against the true function (blue).

Although figure 6 would suggest that increasing the width is not essential, we see from the plots in figure 7 that the network is more capable of properly fitting f as we increase the number of neurons. Because the loss is so small, the network stops learning once it does reasonably well which leads to lousier approximations (e.g. the plots for 6, 16 and 20 neurons). However, we see that it is still very likely to land in a very good local optimum (e.g. the plots for 12, 14 and 18 neurons).

This is an example where knowing all the values of f would give a perfect approximation with only 4 neurons. In practice, it is possible, by chance, to get an almost perfect approximation with low number of neurons, since f is a piecewise linear function with 4 pieces. However, this is very unlikely and it still seems that we benefit from increasing the width a lot beyond 4. This is likely caused by the tendency for coadaptation of weights. So, overparametrization helps learning even in very simplistic setups.

4.5 Fitting $f(x) = \sin(3\pi x) + \cos(5\pi x)$

Let $f(x) = \sin(3\pi x) + \cos(5\pi x)$. By Lemma 4 in the Appendix, $\omega(f, \delta) \leq 8\pi\delta \implies$

$$omega(f, N) = \frac{1}{2}\omega\left(f, \frac{2}{N-1}\right) + \omega\left(f, \frac{1}{N-1}\right) \leq \frac{16\pi}{N-1}.$$

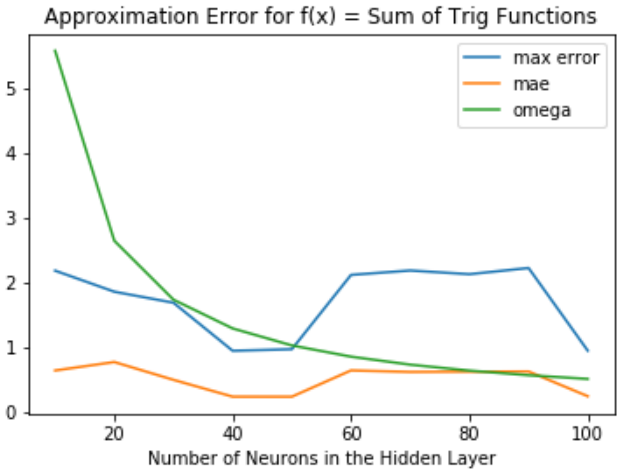

Figure 8: Plot of maximum error, mean absolute error and our bound omega as a function the number of neurons in the hidden layer for $f(x) = \sin(3\pi x) + \cos(5\pi x)$.

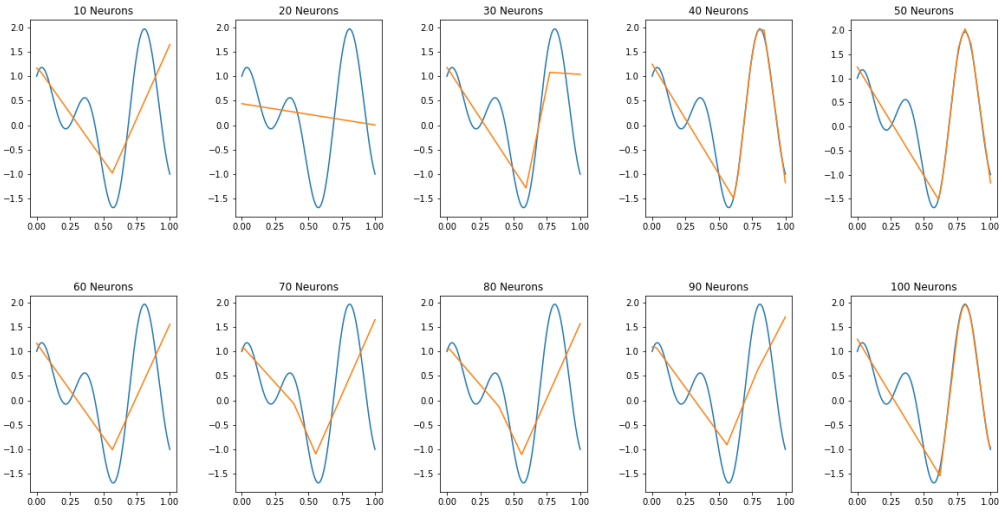

Figure 9: Plot of function learnt by network (orange) for increasing number of neurons against the true function $f(x) = \sin(3\pi x) + \cos(5\pi x)$ (blue).

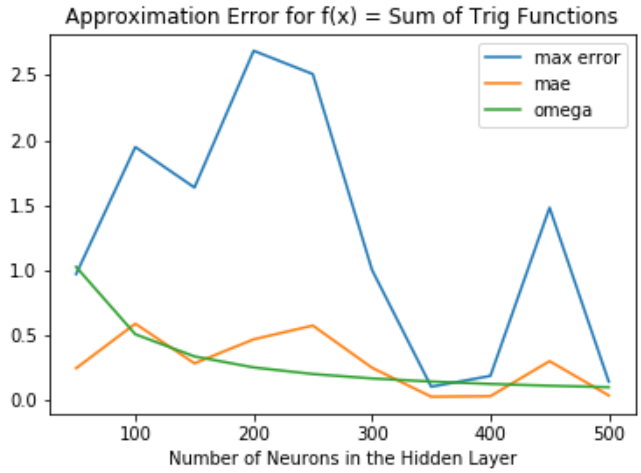

Figure 10: Plot of maximum error, mean absolute error and our bound omega as a function the number of neurons in the hidden layer for $f(x) = \sin(3\pi x) + \cos(5\pi x)$.

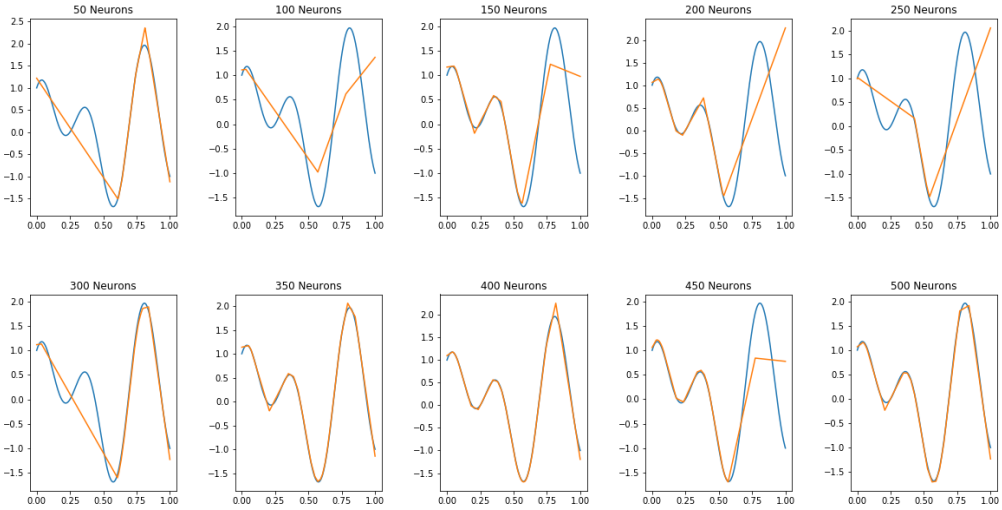

Figure 11: Plot of function learnt by network (orange) for increasing number of neurons against the true function $f(x) = \sin(3\pi x) + \cos(5\pi x)$ (blue).

The behavior in figure 8 suggests that, for this particular function, our network is prone to falling into saddle points which stops us from reaching a desirable weight configuration. As we can see from figures 9 and 11, we need to increase the number of neurons to at least 350 to get a desirable approximation. However, in that case, there is a chance that our network still falls into a saddle point (see the figure for 450 neurons). Moreover, the problem for a large number of neurons is that the max error is above *omega* (see figure 10).

Interestingly, it seems that adding another layer improves the approximation drastically:

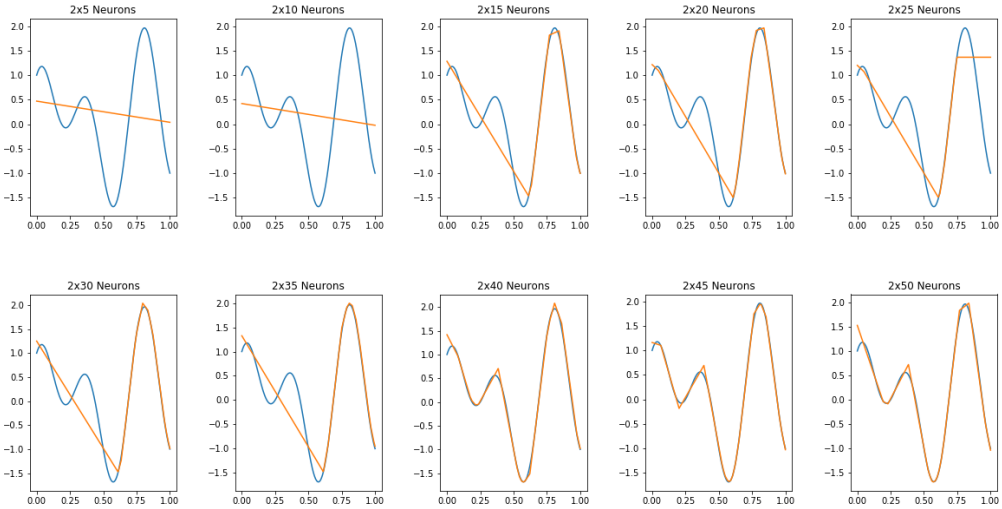

Figure 12: Plot of function learnt by 2-layer network (orange) for increasing number of neurons against the true function $f(x) = \sin(3\pi x) + \cos(5\pi x)$ (blue).

5 Concluding Remarks

Besides being an extension of a particular case of the Universal Approximation Theorem, Theorem 1.3. gives us a hint to the potential power of a neural network with only one hidden layer and using ReLU activation function. In practice, we expect that even a network with a small number of neurons can approximate well a function with no inflections, such as the one presented in Section 4.2. However, we also expect the result to have some limitations, especially for functions with many inflections.

First, the limited amount of information might make the network overfit the function for the training set. Section 4.4 shows the contrast between theory and practice: while there is a network with only 4 neurons in the hidden layer which approximates the presented function perfectly, we need many more in practice.

Secondly, our result does not have any control over the possibility of the network landing in a saddle point, as we can see in Section 4.3. For a wilder function, such as the one in Section 4.5, increasing the number of neurons (to get a better approximation of the function) decreases the value of *omega* under the max error.

For further work, it would be interesting to explore how the approximation bound given by Theorem 1.3. improves after adding one layer to the network. As Section 4.5 suggests, adding one more layer might improve the approximation of a function with many inflection points, even when the number of neurons is small.

6 Appendix

Lemma 6.1. *If $f(x) = \sqrt{x}$, then $\omega(f, \delta) = \sqrt{\delta}$.*

Proof. By the Mean Value Theorem,

$$\forall x, y \in [0,1] \; \exists \, z \in (x,y) \text{ s.t. } \frac{f(y) - f(x)}{y - x} = f'(z) = \frac{1}{2\sqrt{z}} \implies$$

$$|f(x) - f(y)| \leq |\sqrt{x} - \sqrt{y}| \iff \frac{|x - y|}{2\sqrt{z}} \leq |\sqrt{x} - \sqrt{y}| \iff$$

$$\frac{\sqrt{x} + \sqrt{y}}{2} \leq \sqrt{z} \text{ which is true by the concavity of } f \implies$$

$$\omega(f, \delta) \leq \sqrt{\delta}$$

But $|f(\delta) - f(0)| = \sqrt{\delta} \implies \omega(f, \delta) \geq \sqrt{\delta} \implies \omega(f, \delta) = \sqrt{\delta}$ as desired. □

Lemma 6.2. *If $f(x) = \sin(2\pi x)$, then $\omega(f, \delta) \leq 2\pi\delta$.*

Proof. By the Mean Value Theorem,

$$\forall x, y \in [0,1] \; \exists \, z \in (x,y) \text{ s.t. } \frac{f(y) - f(x)}{y - x} = f'(z) = 2\pi \cos(2\pi z) \implies$$

$$|f(y) - f(x)| = 2\pi |y - x| |\cos(2\pi z)| \leq 2\pi |y - x| \implies$$

$$\omega(f, \delta) \leq 2\pi\delta \text{ as desired.}$$

□

Lemma 6.3. *If $f(x) = 10ReLU(x-0.8) - 6ReLU(x-0.3) + 4ReLU(x-0.5) - 2ReLU(0.2 - 2x)$, then $\omega(f, \delta) \leq 8\delta$.*

Proof. By the Mean Value Theorem,

$$\forall x, y \in [0,1] \; \exists \, z \in (x,y) \text{ s.t. } \frac{f(x) - f(y)}{x - y} = f'(z) \implies$$

$$|f(x) - f(y)| \leq \max(8, 2, 6, 4) = 8|x - y| \implies$$

$$\omega(f, \delta) \leq 8\delta \text{ as desired.}$$

□

Lemma 6.4. *If $f(x) = \sin(k\pi x) + \cos(l\pi x)$, then $\omega(f, \delta) \leq (k + l)\pi\delta$.*

Proof.

$$\begin{aligned} |f(x) - f(y)| &= |\sin(k\pi x) + \cos(l\pi x) - \sin(k\pi y) - \cos(l\pi y)| \\ &\leq |\sin(k\pi x) - \sin(k\pi y)| + |\cos(l\pi x) - \cos(l\pi y)| \end{aligned}$$

As in the lemma above, by the Mean Value Theorem, $\exists \, z_1, z_2 \in (x, y)$ s.t.

$$|\sin(k\pi x) - \sin(k\pi y)| = k\pi |x - y| |\cos(k\pi z_1)| \leq k\pi |x - y|$$

$$|\cos(l\pi x) - \cos(l\pi y)| = l\pi |x - y| |\sin(l\pi z_2)| \leq l\pi |x - y|$$

Therefore, $|f(x) - f(y)| \leq (k + l)\pi \; |x - y| \implies \omega(f, \delta) \leq (k + l)\pi\delta$ as desired. □

References

[1] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[2] Jelena Djuris et al. "Design Space Approach in Optimization of Fluid Bed Granulation and Tablets Compression Process". In: *TheScientificWorldJournal* 2012 (July 2012), p. 185085. DOI: 10.1100/2012/185085.

[3] Arno Kuijlaars. *Ward Cheney and Will Light, A Course in Approximation Theory*. 2001.