IFT 6085 - Lecture 10
Expressivity and Universal Approximation Theorems

This version of the notes has not yet been thoroughly checked. Please report any bugs to the scribes or instructor.

Scribes
Winter 2021: Martin Weiss
Winter 2020: Téo Orthlieb
Winter 2019: Moustafa Elarabi, Kun Ni

Instructor: Ioannis Mitliagkas

1 Summary

In this lecture we will discuss the expressivity of neural networks and introduce the universal approximation theorem. Informally, the *expressivity* of a model describes the class of functions a model can approximate. We will introduce the perceptron and show that it cannot model an exclusive or (XOR) of two binary variables. We will then introduce the multi-layer perceptron (the famous MLP), and prove in Section 3 that it is capable of approximating any continuous function arbitrarily well. In Section 4 we will conclude with two additional results: first, that a width-bounded neural network with ReLU non-linearities can approximate a continuous function $f : \mathbb{R}^n \to \mathbb{R}$ arbitrarily well, and second that there is a class of function for which networks with 2 layers and an exponential number of weights (2^k) will have significant error and that a network with $2k$ layers can fit exactly.

2 Introduction

2.1 Expressivity of the perceptron

How expressive is a neural network? Let's start with the simplest neural network, the perceptron. It is a neural network with a single unit, having an activation function σ. Often, we refer to the perceptron as an artificial "neuron". Here, both the input **x** and the weights **w** are a vector of dimension $1 \times n$.

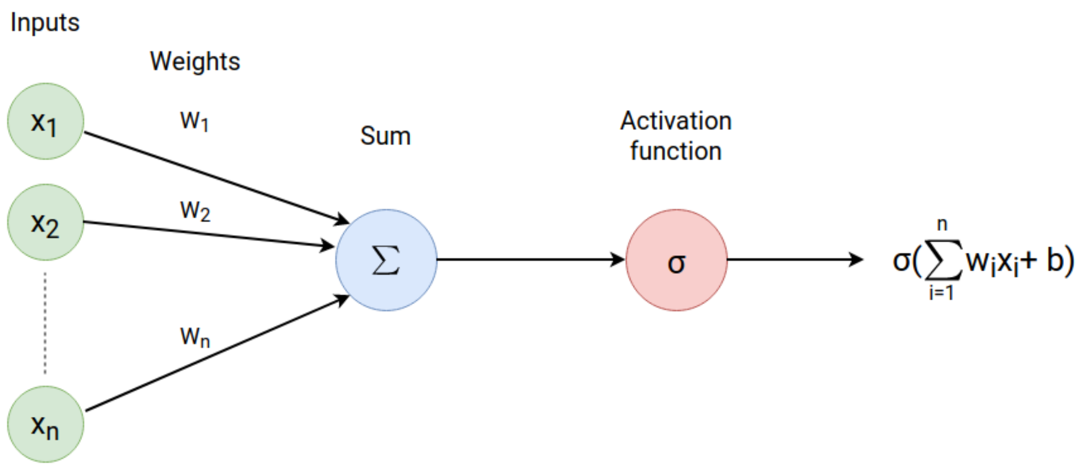

Figure 1: **The Perceptron** is a fundamental component of many modern neural network architectures. We often refer to these as neurons. It has a single hidden-unit which is connected to some inputs **x**. The perceptron computes the dot product of a weight vector **w** with the input **x**, optionally adds a bias term, then passes the result through a non-linearity.

The perceptron can model a function of this form: $\sigma(\sum_i^n \mathbf{w_i} x_i + b) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$. Often, we select the non-linear function σ to be one of the following:

$$\text{sigmoid} = \frac{1}{1+e^{-x}} \qquad \tanh = \frac{e^{2x}-1}{e^{2x}+1} \qquad \text{ReLU} = \max(0, x)$$

Notably, the perceptron is a linear classifier, and as such it famously cannot accurately classify XOR [9]. More precisely, given two binary variables x_1 and x_2, the XOR function returns 1 when exactly one of these binary variables is equal to 1, otherwise it returns 0. After selecting a reasonable loss function such as Mean-Squared Error, and minimizing the loss of our model with respect to the weights w_1 and w_2 and bias b in our perceptron, we obtain a model which incorrectly classifies at least half of the points. It cannot fit the training data! We show the in Figure 2 and refer interested readers to the Chapter 6 of the Deep Learning book for a more detailed analysis [4].

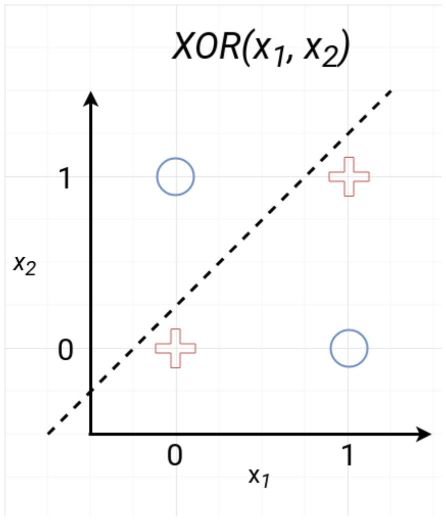

Figure 2: **The XOR** function is perhaps the simplest non-linear function. Here, we show that the transformation it applies to x_1 and x_2 cannot be modelled by a perceptron. In other words, we cannot train a perceptron to classify this data because it is not linearly separable and the perceptron is only able to express linear functions.

2.2 The Multi-Layer Perceptron

The reading in Understanding Machine Learning (UML) [10] introduces the Multi-Layer Perceptron (MLP) and discusses its expressivity. Here, we will provide a brief and lossy summary. Figure 3 shows a very small Multi-Layer Perceptron (MLP) which can correctly classify XOR. As you can see, it is composed of 3 perceptrons, 2 in the hidden layer and 1 in the output layer.

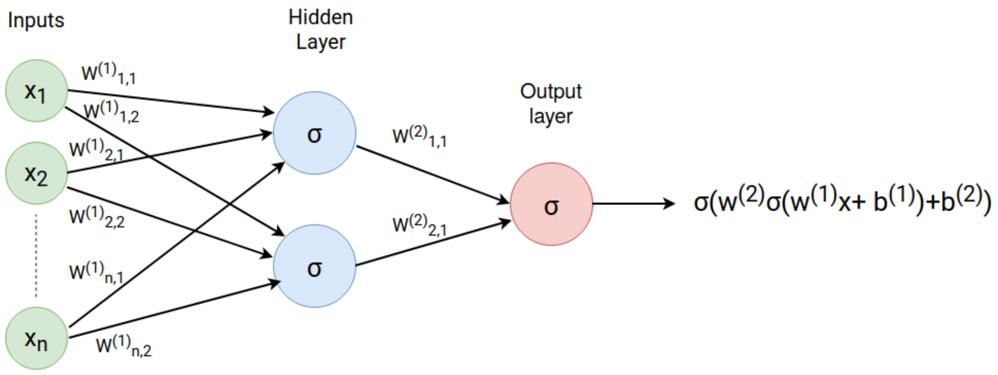

Figure 3: **A Multi-Layer Perceptron** with 2 hidden units and an output layer. This model can learn to classify XOR.

The MLP is a *feedforward neural network*. Typically, the architecture of a neural network is fixed (i.e., we select the number of neurons, how they are connected, their non-linear activation functions, and their weight initialization) before training. The feedforward network architecture is formalized in [10] as a directed acyclic graph $G = (V, E)$, with a weight function over the edges, $w : E \to \mathbb{R}$. Nodes on the graph correspond to neurons and σ is some non-linearity (e.g., the sign function, threshold function, some sigmoidal function, ReLU).

The MLP architecture (when the number of units in the hidden layer is permitted to grow) is a **universal approximator**. In Section 3 we will discuss the classic result from Cybenko in '89 [2] that any neural network with sigmoidal activation (i.e., 1 as $x \to +\infty$ and 0 as $x \to -\infty$ can approximate any continuous function arbitrarily well. While this was the first proof of the universal approximation theorem, many more have followed. Some of these do not rely on this assumption about the non-linearity, in particular Leshno-Lin-Pinkus-Schocken '93 shows the theorem holds iff σ is not a polynomial [7]. Next, we will show a visual proof of the Universal Approximation Theorem.

3 Universal Approximation Theorem

The universal approximation theorem states that any continuous function $f : [0,1]^n \rightarrow [0,1]$ can be approximated arbitrarily well by a Multi-Layer Perceptron with at least 1 hidden layer and a finite number of hidden units.

3.1 Visual proof of Universal Approximation

In this section we will present a good intuition for the universal approximation theorem by making a summary of this page http://neuralnetworksanddeeplearning.com/chap4.html. (All credit is due to Michael A. Nielsen for all the pictures in this subsection)

Say we want to approximate a function with 1 input and 1 output like so:

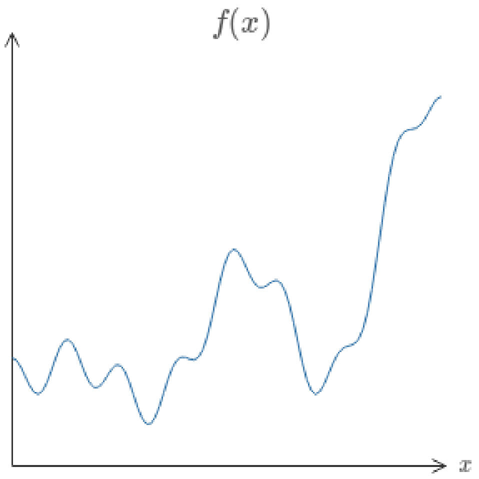

Figure 4: A continuous function

We will first consider a simple MLP with 2 hidden neurons that have a sigmoid activation function, and for now the output neuron will just be linear.

Step 1 Make a step function with 1 of the neuron.

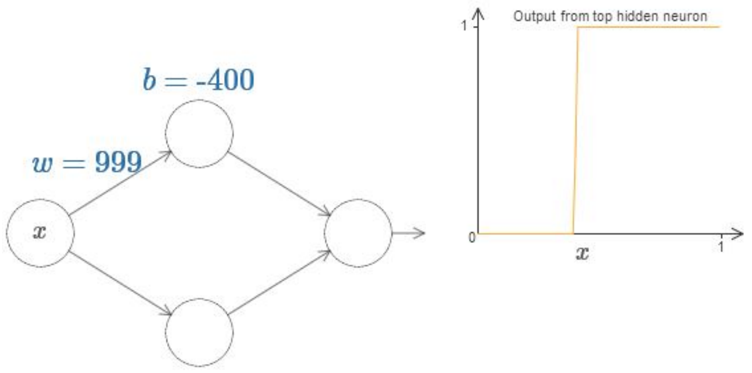

Figure 5: For the top neuron, by selecting a large weight, and a bias term as some proportion of that weight, we can construct and position a step function arbitrarily well.

Let's focus on the top hidden neuron first, by using a big weight on the top neuron we can approximate the step function with a sigmoid arbitrarily well, and by adjusting the bias we can place it anywhere. The same argument could be made for the tanh activation, but not for ReLU. In this toy example, we won't be interested in changing the weights of the first layer, they just have to be high enough that we may consider them to be constant. To simplify the plots, we will report only the position of the step, s, instead of the weight and bias values. s is easily computed as $s = -\frac{b}{w}$. With these changes, the plot becomes:

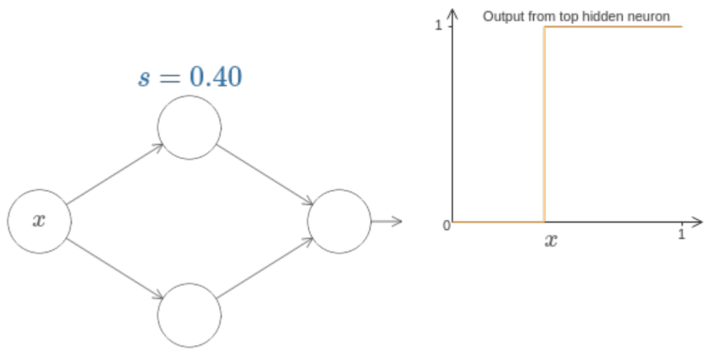

Figure 6: Having selected the top neuron's weight to be large, we can re-write the weight and bias of the top neuron as a single variable that describes the position of the step function, $s = -\frac{b}{w}$

Step 2 We can construct a "bin" by setting the bottom neuron to "step down" at some later x value.

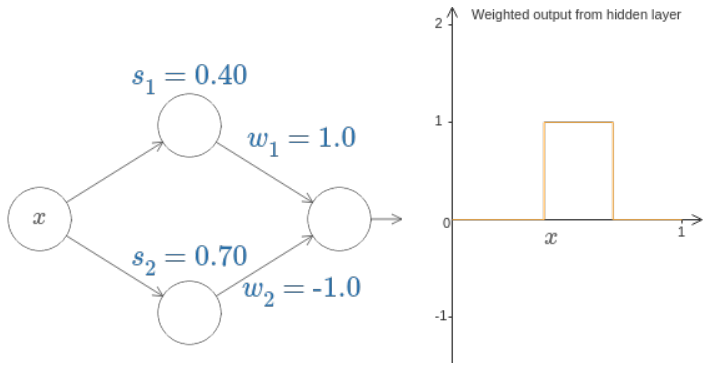

Figure 7: Making a bin with 2 opposite step functions

As illustrated above, by using the other neuron to make a step function, and setting opposing weights in the second layer, we can effectively approximate a bin and control its position, size and height. Now you can probably see where this is going, to make things even clearer, we will just use 1 value for both w_1 and $-w_2$, called h, representing the height of the "bin".

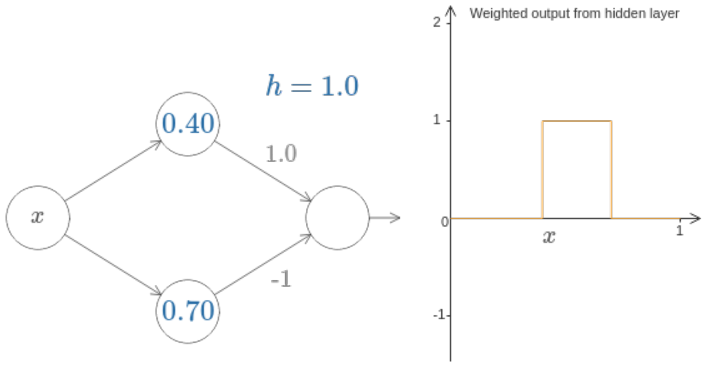

Figure 8: Making a bin with 2 opposite step functions

Step 3 In this final step, we show that we can approximate a discretized $f(x)$ by combining several "bins" to make a histogram approximating the function. Illustrated below is a very rough approximation using only 5 bins (10 hidden units), but we can obviously make it as sharp as we like by adding more bins. Recall that the neuron in the output layer is linear. If we instead added a sigmoid activation function on the output unit we just have to approximate $\sigma^{-1} \circ f$ instead of f, which we can do with the same method.

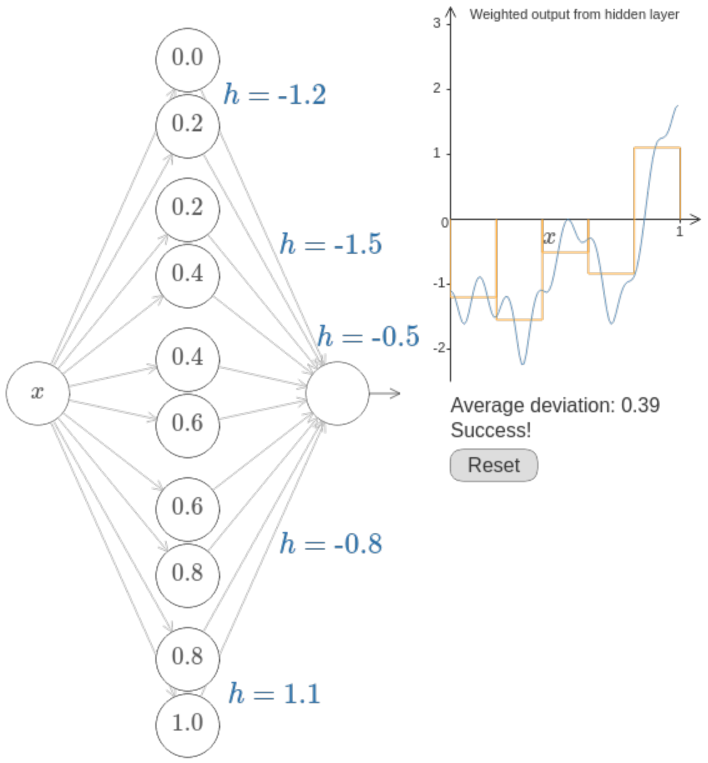

Figure 9: Approximating f with an histogram

Remarks: It is important to remember that although an MLP is a universal approximator, that does not mean that it can *learn* any function through gradient descent. Indeed, many important problems are not learnable in practice through gradient descent [11], even in the context of i.i.d. examples. There are even fewer guarantees on the performance of neural networks when applied to data outside the domain seen during training; this is an active area of research [13]. *Exercise for the reader*: if we wanted to use this technique to approximate an *L-Lipschitz* function $f : \mathbb{R} \to \mathbb{R}$ on the interval $[0, 1]$ with an error at most ϵ at any point, how many bins would we need?

3.2 Cybenko's Universal Approximation Result

In this subsection we will present Cybenko's [2] result for Universal Approximation of MLPs with a single hidden layer and any sigmoidal non-linearity. This was the first result achieved of this kind, and is based on the following intuition. Artificial neural networks are formed by compositions and superpositions of a single, simple nonlinear activation function. The output of the network is the value of the function that results from that particular composition and superposition of non-linearities as activated by the input. First, we define a *sigmoidal function* σ as:

$$\sigma(x) \to \begin{cases} 1 & \text{as } x \to +\infty \\ 0 & \text{as } x \to -\infty \end{cases}$$

***N.B.** sigmoidal functions are usually assumed to be monotonic increasing, but this is not necessary for this result.*

Theorem 1. *Let $C([0,1]^n)$ denote the set of all continous function $[0,1]^n \to \mathbb{R}$, let σ be any sigmoidal activation function then the finite sum of the form $f(x) = \sum_{i=1}^{N} \alpha_i \sigma(\mathbf{w}_i^\top x + b_i)$ is dense in $C([0,1]^n)$*

Informally, this theorem is saying that for any $g \subset C([0,1]^n)$ and any $\epsilon > 0$, there exists $f : x \to \sum_{i=1}^{N} \alpha_i \sigma(\mathbf{w_i}^\top x + b)$ such that $|f(x) - g(x)| < \epsilon$ for all $x \subset [0,1]^n$. So, for any ϵ you choose, there is a function in this parametric family that is arbitrarily close to what your desired function. The ϵ is "hiding" in the statement of density.

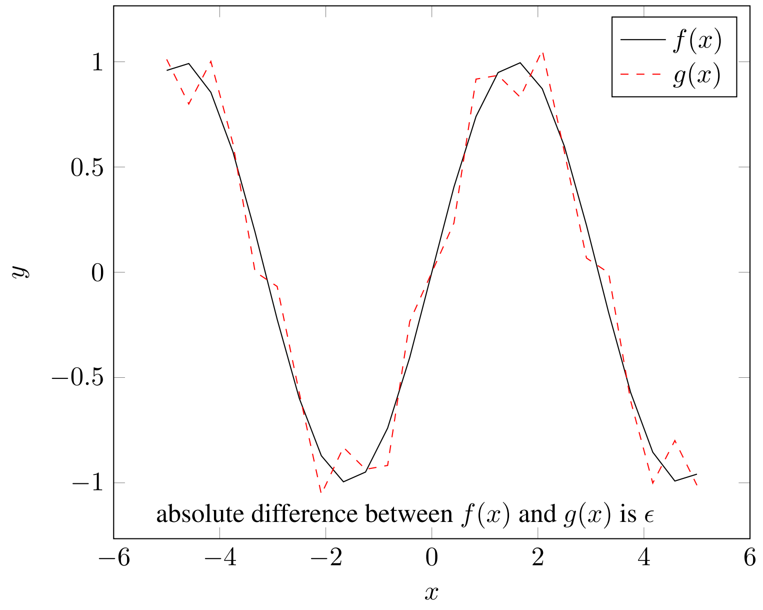

Figure 10: We show the fitting of $f(x)$ to $g(x)$ within a tolerance of ϵ to illustrate Cybenko's theorem.

Cybenko's result relied upon the Kolmogorov-Arnold Representation theorem described in the next section. A similar result was independently obtained by Hornik[5] and also by Funahashi[3] using different tools. Hornik's proof relies on the Stone-Weierstrass Theorem which states that every continuous function defined on a closed interval $[a, b]$ can be uniformly approximated as closely as desired by a polynomial function.

3.3 Kolmogorov-Arnold Representation theorem

The Kolmogorov-Arnold representation theorem (or superposition theorem) [6] states that every multivariate continuous function can be represented as a superposition of continuous functions of one variable.
It solved a more general form of Hilbert's thirteenth problem [1] which was questioning whether a solution to 7^{th} degree equations could be expressed by a finite sum of two-variable functions.

Theorem 2. *Any continuous function $f : [0,1]^n \to \mathbb{R}$ can be written as*

$$f(x) = f(x_1, .., .., x_n) = \sum_{q=1}^{Z_n=2n+1} \phi_q \left(\sum_{p=1}^{n} \Psi_p q(x_p) \right)$$

This implies, among other things, that if we could chose a "bespoke" non-linearity for each unit we can represent any continuous function **exactly** with a NN with 1 hidden layer. In practice we don't care about the ϵ in the approximation of Cybenko's result, so the Kolmogorov-Arnold Representation theorem is more powerful than we need.

4 The expressive power of Deep neural networks

This section contains several results that relate certain classes of problem, neural network architecture choices, and the number of parameters required to achieve an error on the aforementioned problem.

4.1 Expressivity of MLPs (Understanding Machine Learning Sec. 20)

Recall from Subsection 2.2 that the MLP may be defined as a directed acyclic graph $G = (V, E)$ equipped with a non-linearity σ. In order to study the expressivity of the feedforward network, Understanding Machine Learning (UML) [10] discusses which Boolean functions (functions from $\{\pm 1\}^n \to \{\pm 1\}$ can be implemented by the hypothesis class of feedforward networks $\mathcal{H}_{V,E,\text{sign}}$. The non-linearity selected here is the sign function (1 if input is positive, -1 if input is negative), but the results that follow in this subsection can also be shown for other non-linearities.

Claim 1. *$\forall n$, there exists a graph (V, E) of depth 2, s.t. $\mathcal{H}_{V,E,sign}$ contains all functions from $\{0,1\}^n \to \{0,1\}$.*

In order to prove this claim, UML [10] constructs a network which can compute any Boolean function. However, this network might be exponentially large, and in fact they show in Theorem 3 that it is impossible to express all Boolean functions using a network of polynomial size.

Proof. We construct a graph with $|V_0| = n+1$, $|V_1| = 2^n + 1$, and $|V_2| = 1$. Let E be all possible edges between adjacent layers. Now, let $f : \{\pm 1\}^n \to \{\pm 1\}$ be some Boolean function. We need to show that we can adjust the weights so that the network will implement f. Let $\mathbf{u}_1, \ldots, \mathbf{u}_k$ be all the vectors in $\{\pm 1\}^n$ on which f outputs 1. Observe that for every i and every $\mathbf{x} \in \{\pm 1\}^n$, if $\mathbf{x} \neq \mathbf{u}_i$, then $\langle \mathbf{x}, \mathbf{u}_i \rangle \leq n-2$ and if $\mathbf{x} = \mathbf{u}_i$ then $\langle \mathbf{x}, \mathbf{u}_i \rangle = n$.

It follows that the function $g_i(\mathbf{x}) = \text{sign}(\langle \mathbf{x}, \mathbf{u}_i \rangle - n + 1)$ equals 1 if and only if $\mathbf{x} = \mathbf{u}_i$. We can adapt the weights between V_0 and V_1 so that for every $i \in [k]$, the neuron $v_{1,i}$ implements the function $g_i(\mathbf{x})$. Finally, we observe that $f(\mathbf{x})$ is the disjunction of the functions $g_i(\mathbf{x})$ and can therefore be written as

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{k} g_i(\mathbf{x}) + k - 1 \right)$$

This defines a weight setting for a depth-2 network to compute an arbitrary Boolean function, concluding our proof. □

Theorem 3 (Shallow Networks Require Exponential Parameters to Implement Boolean Functions)**.** *For every n, let $s(n)$ be the minimal integer such that there exists a graph (V, E) with $|V| = s(n)$ such that the hypothesis class $\mathcal{H}_{V,E,sign}$ contains all the functions from $0, 1^n \to 0, 1$. Then, $s(n)$ is exponential in n. Similar results hold for $\mathcal{H}_{V,E,\sigma}$ where σ is the sigmoid function.*

Proof. Suppose that for some (V, E) we have that $\mathcal{H}_{V,E,\text{sign}}$ contains all functions from $\{0,1\}^n \to \{0,1\}$. It follows that it can shatter the set of $m = 2^n$ vectors in $\{0,1\}^n$ and hence the VC dimension of $\mathcal{H}_{V,E,\text{sign}}$ is 2^n. On the other hand, the VC dimension of $\mathcal{H}_{V,E,\text{sign}}$ is bounded by $O(|E|log(|E|)) \leq O(|V|^3)$ (shown in Section 20.4 of [10]). This implies that $|V| \geq \Omega(2^{\frac{n}{3}})$ which concludes our proof for the case of networks with the sign activation function. The proof for the sigmoid case is analogous. □

So, it is impossible to express all Boolean functions of n bits using a depth-2 network of size polynomial in n. And it's possible to fit any continuous function arbitrarily well. What about other classes of problem? Or with other constraints on the network? We will see this next.

4.2 A view from the width (Lu et al. 2017)

In the previous sections, we focused on the setting of depth-bounded (e.g. depth-2) neural networks. With [8] we're going to see some interesting results for **width-bounded** neural networks instead !

Theorem 4. *(Universal Approximation Theorem for Width-Bounded ReLU Networks). For any Lebesgue-integrable function $f : \mathbb{R}^n \to \mathbb{R}$ and any $\epsilon > 0$, there exists a fully-connected ReLU network A with width $d_m \leq n+4$, such that the function F_A represented by this network satisfies*

$$\int_{\mathbb{R}^n} |f(x) - F_A(x)| dx < \epsilon$$

This theorem states that any continuous function $f : \mathbb{R}^n \to \mathbb{R}$ can be approximated by a deep ReLU network with width $\leq n + 4$.

Theorem 5. *Let n be the input dimension. For any integer $k \geq n + 4$ there exists $F_\alpha : \mathbf{R^n} \to \mathbf{R}$ represented by a relu neural network α with width $d_m = 2k^2$ and depth $h = 3$ such that for any constant $b > 0$, there exists an $\epsilon > 0$ and for any function $F_\beta : \mathbf{R^n} \to \mathbf{R}$ represented by a ReLu neural network β whose parameters are bounded in $[-b, b]$ with width $d_m \leq k^{\frac{3}{2}}$ and depth $h \leq k + 2$ the following inequality holds*

$$\int_{\mathbf{R}} |F_\alpha - F_\beta| dx \geq \epsilon$$

This theorem states that there are networks such that reducing width requires increasing in the size to compensate, which is similar to that of depth qualitatively.

4.3 Representation benefits of deep NN (Telgarsky 2015)

In this section, we want to show interesting results from [12], that will allow us to compare the expressivity of wide networks against deep and recurrent networks, on a specific classification problem. Let n-ap (*n-alternating-points*) be the set of $n := 2^k$ points uniformly spaced within $[0, 1 - 2^{-k}]$ with alternating labels. We will use different types of neural network to fit this dataset, and see the number of parameters required to achieve a certain classification error.

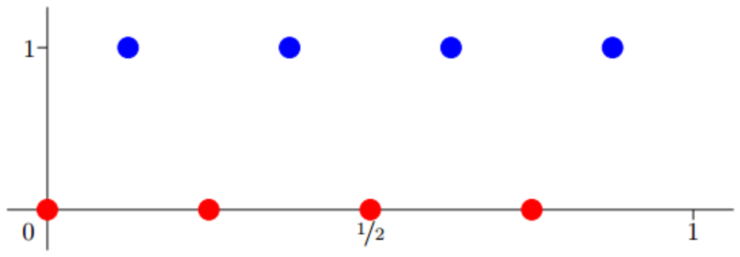

Figure 11: The 2^3-ap [12]

In the following theorems we will note $\mathscr{N}(\sigma, m, l)$ the set of function given by a feedforward neural network with activation σ and l layers with at most m nodes. The ReLU activation will be noted σ_R and the classification error $\mathcal{R}_z$.

Theorem 6. *With $m \leq 2^{(k-3)/(l-1)}$, for any positive integer k, $\exists$ a collection of $n := 2^k$ points $S = (x_i, y_i)_{i=1}^n$ where $x_i \in [0,1], y \in \{0,1\}$ such that*

$$\min_{f \in \mathscr{N}(\sigma_R, m, l)} \mathcal{R}_z(f) = \frac{1}{6} \qquad \textit{and} \qquad \min_{g \in \mathscr{N}(\sigma_R, 2, 2k)} \mathcal{R}_z(g) = 0$$

Notably, if we look at the case of 2 layered networks, this tells us that even with 2^{k-3} units in the hidden layer, the wide network is going to misclassify $\frac{1}{6}$ of the points, whereas a deep network with $2k$ hidden layers with 2 units each can achieve 0 classification error. What about a recurrent network?

Let $\mathscr{R}(\sigma, m, l; k)$ denote k iterations of a recurrent neural network, every $f \in \mathscr{N}(\sigma, m, l, k)$ can be expressed as some fixed network $g \in \mathscr{N}(\sigma, m, l)$ applied k times:

$$f(x) = g^k(x) = \underbrace{(g \circ g \cdots \circ g)}_{\text{k times}}(x)$$

Consequently, $\mathscr{R}(\sigma, m, l; k) \subseteq \mathscr{N}(\sigma, m, lk)$ but the former has $\mathcal{O}(ml)$ parameters whereas the latter has $\mathcal{O}(mlk)$ parameters. We can view the latter ($\mathscr{N}(\sigma, m, lk)$) as an unrolled RNN. Lastly we define the following functions:

Definition 7 (Sawtooth)**.** *A function $f : \mathbb{R} \to \mathbb{R}$ is t-sawtooth if it is piece-wise linear with t-pieces*

We can say for example that σ_R (ReLU) is a 2-sawtooth function, decision stumps used in boosting are also 2-sawtooth, and decision trees with t-1 nodes are t-sawtooth.

Theorem 8. *Let positive integer k, number of layers l, and number of nodes per layer m be given.*
Given a t-sawtooth $\sigma : \mathbb{R} \to \mathbb{R}$ and $n := 2k$ points as specified by the n-ap, then

$$\min_{f \in \mathscr{N}(\sigma, m, l)} \mathcal{R}_z(f) \geq \frac{n - 4(tm)^l}{3n} \qquad \textit{and} \qquad \min_{g \in \mathscr{R}(\sigma_R, 2, 2, k)} \mathcal{R}_z(g) = 0$$

In summary, this means that on the 2^k-ap, one needs exponentially (in k) many parameters with a wide network, linearly many parameters with a deep network and constantly many parameters with a recurrent network.

4.3.1 Analysis

The required reading [12] provides a proof of the lower bound via a counting argument which tracks the number of times a function within $\mathbb{R}(\sigma; m, l)$ can cross $\frac{1}{2}$. The upper bound is proved via the construction of a network $\mathbb{R}(\sigma_R; 2, 2)$ which can be composed with itself k times to exactly fit the n-ap. These bounds together prove Theorems 6 and 8.

While these notes focus on the *parameters* in the network, the parametric requirement of each architecture is itself produced by a more fundamental idea: how adding and composing sawtooth functions grows their complexity. In Lemma 2.3 of [12], Telgarsky claims:

Lemma 1. *Let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be respectively $k-$ and $l-$sawtooth. Then $f + g$ is $(k + l)-$sawtooth, and $f \circ g$ is kl-sawtooth.*

This lemma says that the addition of sawtooth functions results in a function which is "as bumpy as both of the functions together", while the *composition* of them is multiplicatively as bumpy. This multiplication of non-linear complexity is fundamental to the increased expressive power of deep neural networks. Put another way, depth is exponentially more expressive than width **because** composition yields more complex functions than summation.

References

[1] S. S. ABHYANKAR. Hilbert's thirteenth problem. In *Proc. of the Franco-Belgian Conference in Reims, Société Mathématique de France, Séminaires et Congres*, volume 2, page 1, 1997.

[2] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[3] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[6] A. N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Doklady Akademii Nauk*, volume 114, pages 953–956. Russian Academy of Sciences, 1957.

[7] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. URL `http://dblp.uni-trier.de/db/journals/nn/nn6.html#LeshnoLPS93`.

[8] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, pages 6231–6239, 2017.

[9] M. Minsky and S. Papert. Perceptrons: an introduction to computational geometry. 1969.

[10] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

[11] S. Shalev-Shwartz, O. Shamir, and S. Shammah. Failures of deep learning. *CoRR*, abs/1703.07950, 2017. URL `http://arxiv.org/abs/1703.07950`.

[12] M. Telgarsky. Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*, 2015.

[13] K. Xu, M. Zhang, J. Li, S. S. Du, K. ichi Kawarabayashi, and S. Jegelka. How neural networks extrapolate: From feedforward to graph neural networks, 2021.