Chapter 3

Feedforward Neural Networks

3.1 Model

A *feedforward neural network* (FFNN), or multilayer perceptron, is composed of alternating *linear layers* and nonlinear *activation functions.*

Definition 3.1. A *linear layer* is a function

$$\begin{aligned} lin\colon \mathbb{R}^{d} &\to \mathbb{R}^{d'} \\ \mathbf{x} &\mapsto \mathbf{W}\mathbf{x} + \mathbf{b} \end{aligned} \tag{3.1}$$

with attributes

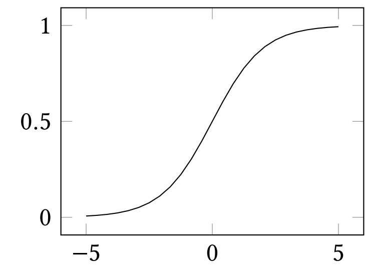
- $d \in \mathbb{N}$, called the *input size*
- $d' \in \mathbb{N}$, called the *output size*

and parameters

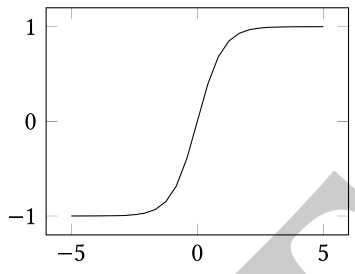
- $\mathbf{W} \in \mathbb{R}^{d' \times d}$, called the *weights*
- $\mathbf{b} \in \mathbb{R}^{d'}$, called the *bias.*

We'll use an idiosyncratic notation when defining and using neural networks (which should hopefully feel familiar, however, to anyone who has used PyTorch). If *lin* is a linear layer, we write *lin.d* for its input size, *lin.*$\mathbf{W}$ for its weights, and so on. In general, whenever we say that something is an "attribute" or "parameter" of a function, we use this dot notation to access it. A parameter of an attribute is also a parameter.

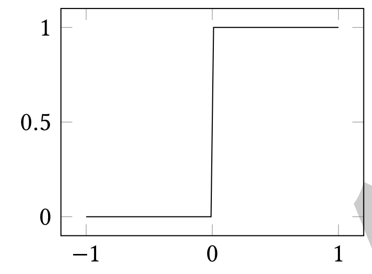
For the nonlinearities, several choices are common:



logistic sigmoid function
$\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$

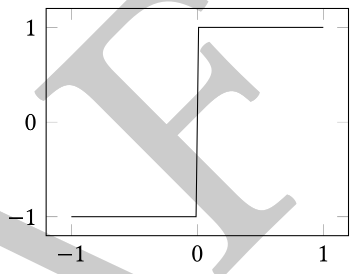


hyperbolic tangent
$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



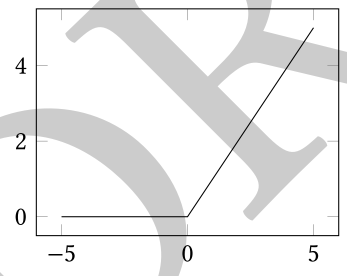
Heaviside step function

$$\mathrm{step}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

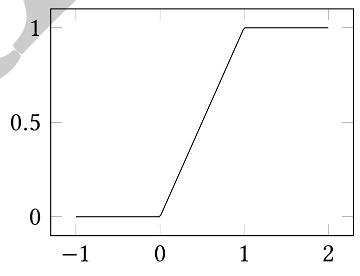


sign function

$$\mathrm{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ +1 & \text{if } x \geq 0 \end{cases}$$



rectified linear unit
$\mathrm{ReLU}(x) = \max\{0, x\}$



saturated linear unit
$\mathrm{SLU}(x) = \max\{0, \min\{1, x\}\}$

When applied to vectors, they are applied component-wise. That is, if σ is an activation function, then

$$\sigma\left(\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}\right) = \begin{bmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_d) \end{bmatrix}. \tag{3.2}$$

Although sigmoid is a traditional activation function, our preference will be for ReLUs. They are often easier to work with in proofs, and they are also very widely used in practice.

Definition 3.2. A *feed-forward neural network* (FFNN) with σ activations (where σ is any activation function) is a function

$$\begin{aligned} \mathit{ffn}\colon \mathbb{R}^d &\to \mathbb{R}^{d'} \\ \mathbf{x} &\mapsto \mathbf{h}^{(L)} \text{ where} \\ \mathbf{h}^{(1)} &= \sigma(\mathit{lin}_1(\mathbf{x})) \\ &\vdots \\ \mathbf{h}^{(L-1)} &= \sigma(\mathit{lin}_L(\mathbf{h}^{(L-2)})) \\ \mathbf{h}^{(L)} &= \mathit{lin}_L(\mathbf{h}^{(L-1)}) \end{aligned} \tag{3.3}$$

with attributes

- $d > 0$, $d' > 0$, called the *input* and *output size*
- $L > 0$, called the *depth*
- linear layers lin_ℓ for $\ell = 1, \ldots, L$, such that

$$\begin{aligned} \mathit{lin}_1.d &= d \\ \mathit{lin}_i.d &= \mathit{lin}_{\ell-1}.d' && 1 < i \leq L \\ \mathit{lin}_L.d' &= d'. \end{aligned}$$

  We call lin_L the *output layer* and all other layers *hidden layers*. (Thus a FFNN with depth L has $(L-1)$ hidden layers.)

The maximum input/output size of any layer of *ffn* is called the *width* of *ffn*.

Note that the output layer does not have a nonlinearity; I'm not sure how standard or nonstandard this is, but it's convenient for us.

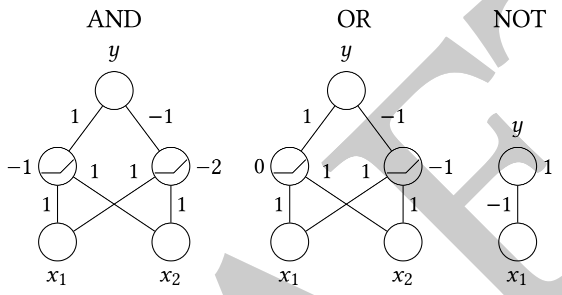
We call the components of $\mathbf{h}^{(\ell)}$ *activation values*. We also call the whole vector $\mathbf{h}^{(\ell)}$ an "activation value," and sometimes an "activation vector" if we need to emphasize that we're talking about the whole vector. We sometimes call other values computed by the network "activation values" as well.

3.2 Expressivity

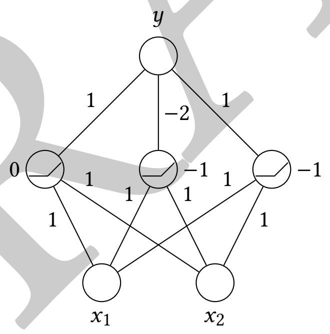
3.2.1 Boolean functions

In Section 2.2, we showed that perceptrons can compute AND, OR, and NOT. Unsurprisingly, FFNNs can as well, but we redo these examples using ReLUs and using 0 for false and 1 for true.

Example 3.3. The Boolean operations AND, OR, and NOT can be computed by FFNNs with ReLU activations. In the pictures below, nodes are units, edges are connections with their weights, and numbers written next to nodes are biases.



Example 3.4. Unlike perceptrons, FFNNs with ReLU activations can compute XOR:



Note that $\text{XOR}(x_1, x_2) = \text{OR}(x_1, x_2) - \text{AND}(x_1, x_2)$.

Theorem 3.5. *Any Boolean function can be computed by a FFNN with* ReLU *activations.*

Proof. Just use the above constructions for AND, OR, and NOT. In fact, a two-layer network suffices. Any Boolean function f can be converted to *disjunctive* *normal form (DNF)*, that is,

$$f(x_1, \dots x_n) = \phi_1 \lor \cdots \lor \phi_l \tag{3.4}$$

$$\phi_i = \phi_{i1} \land \cdots \land \phi_{im_i} \qquad i \in [l] \tag{3.5}$$

where each ϕ_{ij} is one of the x_k or its negation. Furthermore, any f can be converted into *canonical* DNF, where for all values of $x_1, \dots, x_n$, at most one of the clauses ϕ_i is true. Then we can construct a 2-layer FFNN *ffn* with ReLU activations:

$$h_j = \text{ReLU}\left(\sum_k l_{jk} - m_i + 1\right) \tag{3.6}$$

$$l_{jk} = \begin{cases} x_k & \phi_{jk} = x_k \\ 1 - x_k & \phi_{jk} = \neg x_k \end{cases} \tag{3.7}$$

$$y = \sum_j h_j. \tag{3.8}$$

□

3.2.2 Continuous functions

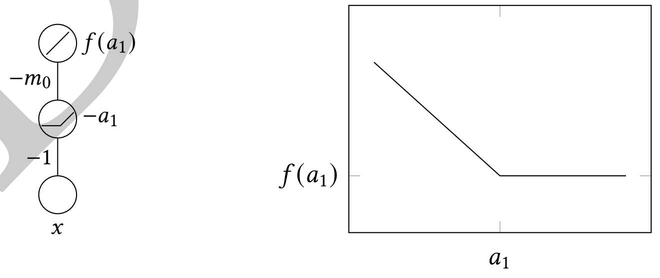
Theorem 3.6 (Arora et al., 2018). *FFNNs with ReLU activations compute exactly the set of continuous piecewise linear functions with a finite number of pieces.*

Proof. We only prove the univariate ($d = 1$) case, which is easy and will come in handy later. Assume that f has the form:

$$f(x) = \begin{cases} m_0 x + b_0 & x < a_1 \\ m_1 x + b_1 & a_1 \le x < a_2 \\ \vdots & \\ m_{k-1} x + b_{k-1} & a_{k-1} \le x < a_k \\ m_k x + b_k & a_k \le x. \end{cases} \tag{3.9}$$

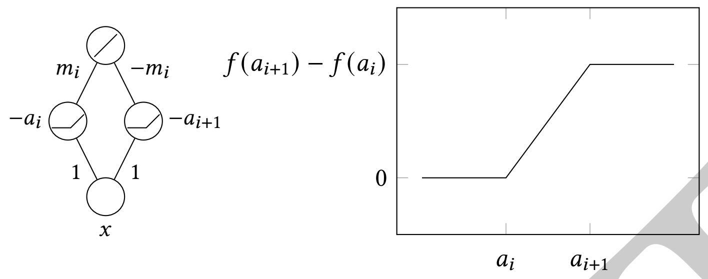
where $m_{i-1} a_i + b_{i-1} = m_i a_i + b_i$ for all i.

The first piece is made of a ReLU flipped left-to-right:



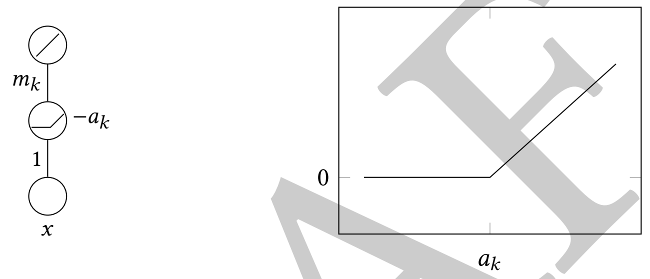
(3.10)

For each middle piece ($i = 1, \ldots, k-1$), we add in a "wedge" (also known as a *saturated linear unit*, or SLU) like this:



(3.11)

Finally, we add in one more ReLU for the last piece:



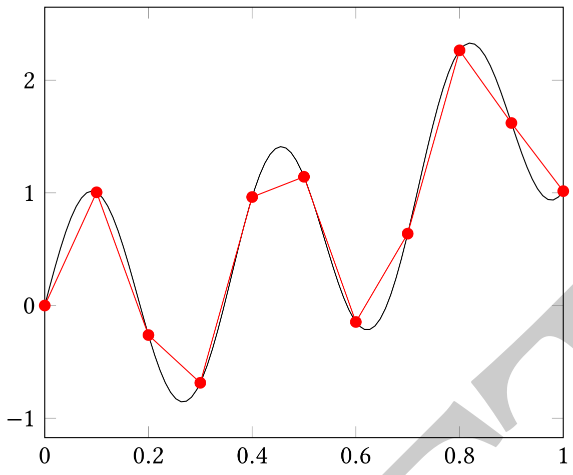
(3.12)

□

Definition 3.7. A function $f\colon \mathbb{R}^d \to \mathbb{R}$ is *ρ-Lipschitz-continuous* if there is a ρ such that $|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x})| \leq \rho\|\mathbf{h}\|$ for all $\mathbf{x}$ and $\mathbf{h}$.

Theorem 3.8. *For any ρ-Lipschitz-continuous function $f\colon [0,1] \to \mathbb{R}$ and any $\epsilon > 0$, there is a FFNN ffn with two layers and ReLU activations, such that for all $x \in [0,1]$, $|f(x) - \mathit{ffn}(x)| \leq \epsilon$.*

Proof. This proof is adapted from Telgarsky (2021), but uses ReLU instead of sgn activations. The idea is to choose a δ depending on ϵ and construct a piecewise linear function that goes through all the points $(i\delta, f(i\delta))$ for all i such that $0 \leq i\delta \leq 1$.

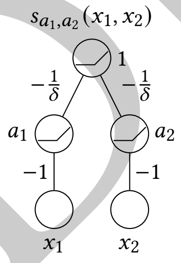


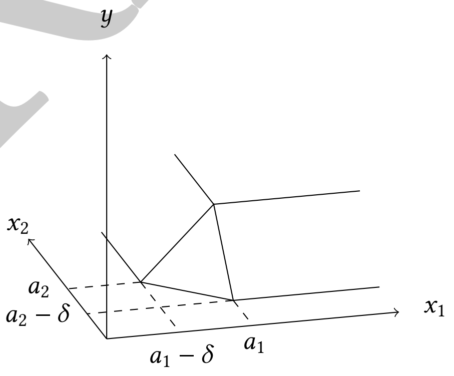
Choose $\delta = \epsilon/(2\rho)$, and use Theorem 3.6 to construct the FFNN *ffn* to go through all points $(i, \delta, f(i\delta))$ for all i such that $0 \le i\delta \le 1$; the first and last pieces can just be horizontal. If $\mathit{ffn}(i\delta) = f(i\delta)$, then for any $x \in [i\delta, (i+1)\delta]$, we have $|\mathit{ffn}(x) - f(x)| \le |\mathit{ffn}(x) - f(i\delta)| + |f(i\delta) - f(x)| \le \epsilon/2 + \epsilon/2 = \epsilon$. □

Theorem 3.9. *For any ρ-Lipschitz-continuous function $f\colon [0,1]^d \to \mathbb{R}$ and any $\epsilon > 0$, there is a FFNN ffn with three layers and ReLU activations, such that for all $x \in [0,1]^d$, $|f(x) - \mathit{ffn}(x)| \le \epsilon$.*

Proof. This proof is similar to the previous one, but requires an additional hidden layer. Choose $\delta = \epsilon/(2\rho\sqrt{d})$.

We just show how to construct *ffn* for $d = 2$. Generalizing to $d > 2$ is not difficult. Define a two-dimensional wedge, s_{a_1,a_2}, with height 1:





We can linearly combine the wedges like this:

$$\mathit{ffn}(x_1, x_2) = \sum_i \sum_j c_{ij} s_{i\delta, j\delta}(x_1, x_2). \tag{3.13}$$

To make *ffn* to equal f at all points $(i\delta, j\delta)$ where i and j are integers, we must have

$$c_{ij} = -\sum_{\substack{i'=0,\ldots,i \\ j'=0,\ldots,j \\ (i',j')\neq(i,j)}} c_{i'j'} + f(i\delta, j\delta). \tag{3.14}$$

□

3.2.3 The universal approximation theorem

Theorem 3.10 (Stone–Weierstrass)**.** *Let $\mathcal{F}$ be a family of functions $\mathbb{R}^d \to \mathbb{R}$ such that*

1. *Each $f \in \mathcal{F}$ is continuous.*
2. *For all $x \in \mathbb{R}^d$, there is an $f \in \mathcal{F}$ such that $f(x) \neq 0$.*
3. *For all $x \neq x' \in \mathbb{R}^d$, there is an $f \in \mathcal{F}$ such that $f(x) \neq f(x')$.*
4. *If $f, g \in \mathcal{F}$, then $f + g \in \mathcal{F}$, $fg \in \mathcal{F}$, and $cf \in \mathcal{F}$ for $c \in \mathbb{R}$.*

Then $\mathcal{F}$ is universal in the following sense: For any continuous function $g\colon [0,1]^d \to \mathbb{R}$ and $\epsilon > 0$, there is an $f \in \mathcal{F}$ such that for all $\mathbf{x} \in [0,1]^d$, $|f(x) - g(x)| \leq \epsilon$.

Hornik et al. (1989) originally proved that FFNNs with cos activations are universal, but exp is easier (doesn't require you to remember your trigonometry identities) and also more plausible as an activation function.

Theorem 3.11 (Telgarsky, 2021)**.** *FFNNs with* exp *activations are universal.*

Proof. Without loss of generality, we can ignore biases (**b**). Let

$$f(\mathbf{x}) = \mathbf{u} \cdot \exp \mathbf{U}\mathbf{x} \tag{3.15}$$
$$g(\mathbf{x}) = \mathbf{v} \cdot \exp \mathbf{V}\mathbf{x}. \tag{3.16}$$

Closure under scalar multiplication and addition are easy, and would work for any activation function:

$$\begin{aligned}
(cf)(\mathbf{x}) &= cf(\mathbf{x}) && (3.17)\\
&= c\mathbf{u} \cdot \exp \mathbf{U}\mathbf{x} && (3.18)\\
(f+g)(\mathbf{x}) &= f(\mathbf{x}) + g(\mathbf{x}) && (3.19)\\
&= \mathbf{u} \cdot \exp \mathbf{U}\mathbf{x} + \mathbf{v} \cdot \exp \mathbf{V}\mathbf{x} && (3.20)\\
&= \begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix} \cdot \exp \begin{bmatrix}\mathbf{U}\\ \mathbf{V}\end{bmatrix} \mathbf{x}. && (3.21)
\end{aligned}$$

Closure under multiplication relies on the fact that $\exp a \exp b = \exp(a+b)$ for any a, b:

$$\begin{aligned}
(fg)(\mathbf{x}) &= f(\mathbf{x})\, g(\mathbf{x}) && (3.22)\\
&= (\mathbf{u} \cdot \exp \mathbf{U}\mathbf{x})\ (\mathbf{v} \cdot \exp \mathbf{V}\mathbf{x}) && (3.23)\\
&= \left(\sum_i \mathbf{u}[i] \exp(\mathbf{U}\mathbf{x})[i]\right)\left(\sum_i \mathbf{v}[i] \exp(\mathbf{V}\mathbf{x})[i]\right) && (3.24)\\
&= \sum_i \sum_{i'} \mathbf{u}[i]\mathbf{v}[i'] \exp(\mathbf{U}\mathbf{x})[i] \exp(\mathbf{V}\mathbf{x})[i'] && (3.25)\\
&= \sum_i \sum_{i'} \mathbf{u}[i]\mathbf{v}[i'] \exp((\mathbf{U}\mathbf{x})[i] + (\mathbf{V}\mathbf{x})[i']) && (3.26)\\
&= \sum_i \sum_{i'} \mathbf{u}[i]\mathbf{v}[i'] \exp(\mathbf{U}[i,:] + \mathbf{V}[i',:])\mathbf{x} && (3.27)
\end{aligned}$$

which does belong to $\mathcal{F}$. □

Theorem 3.12 (Hornik et al., 1989)**.** *Let σ be any continuous, nondecreasing function such that*

$$\begin{aligned}
\lim_{z\to-\infty} \sigma(z) &= 0 && (3.28)\\
\lim_{z\to+\infty} \sigma(z) &= 1. && (3.29)
\end{aligned}$$

Then FFNNs with two layers and σ activations are universal.

Cybenko (1989) proved a similar theorem, but it requires some more advanced math.

Proof. The proof uses three nested approximations. First, by Theorem 3.11, any continuous function can be approximated by a linear combination of exp units. Second, by Theorem 3.6, exp can be approximated by a linear combination of wedges (Eq. (3.11)). Third, a wedge can be approximated by an appropriately transformed σ.

Let $f\colon \mathbb{R}^d \to \mathbb{R}$ be the function we are trying to approximate. For the first approximation, by Theorem 3.11, there is a FFNN *ffn* with 2 layers and exp activations, such that, for all $\mathbf{x} \in [0,1]^d$,

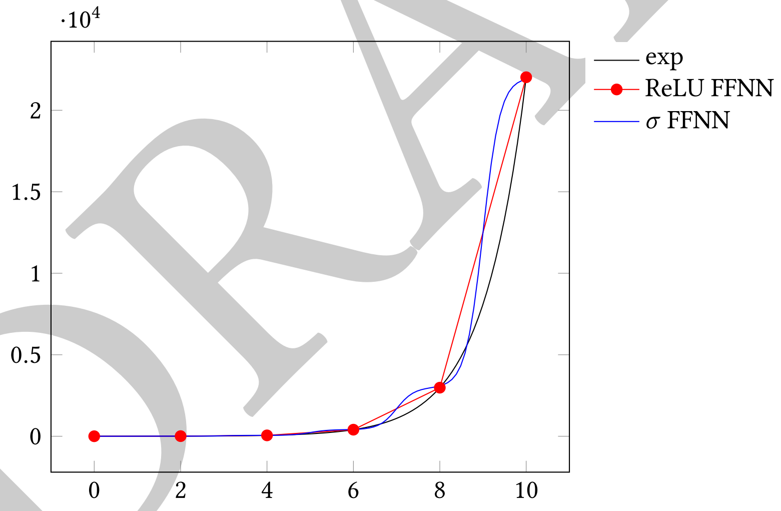
$$|\mathit{ffn}(\mathbf{x}) - f(\mathbf{x})| \le \frac{\epsilon}{3}. \tag{3.30}$$

The next step will be to approximate the exps, but we need two pieces of information from the first approximation. What interval will we need to approximate exp on? Let

$$r = \max_i \left(\sum_j |\mathit{ffn}.\mathit{lin}_1.\mathbf{W}[i,j]| + |\mathit{ffn}.\mathit{lin}_1.\mathbf{b}[i]| \right) \tag{3.31}$$

so that the range of the first affine transformation is at most $[-r, r]$. How good does the approximation need to be? Let $m = \sum_i |\mathit{ffn}.\mathit{lin}_2.\mathbf{W}[1,i]|$, the total amplitude of all the exps. We want the total error due to approximating the exps to be at most $\epsilon/3$, so we need exp to be approximated with error at most $\epsilon/(3m)$.

The remaining approximations are pictured below.

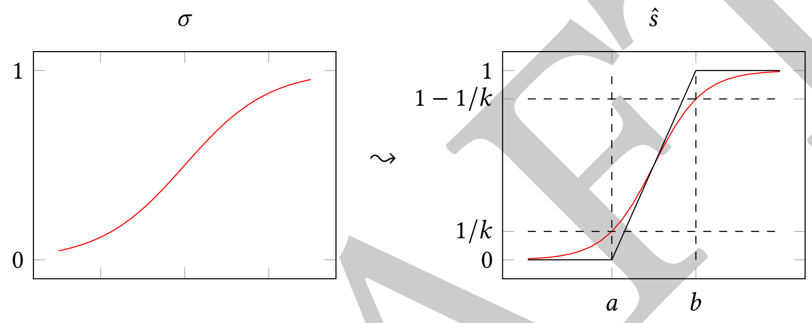


For the second approximation, by Theorem 3.8, there is a linear combination of k wedges that approximates $\exp(z)$ for $z \in [-r, r]$ with error $\epsilon/(3m)$. Observe that to get this approximation, the only assumption we made about the shape of a wedge $s_{a,b}$ is that $s_{a,b}(a) = 0$, $s_{a,b}(b) = 1$, and for $z \in [a,b]$, $s_{a,b}(z) \in [0,1]$. Now σ satisfies the last condition, so in the third approximation, we will only have to worry about the error for $z \le a$ and $z \ge b$. Since each wedge is scaled to have height at most $\epsilon/(3m)$, and we want to limit the remaining error to $\epsilon/(3m)$, we need the height-1 wedge to be approximated with error at most $1/k$.

For the third approximation, by the definition of σ, there exists an A such that $\sigma(A) \le 1/k$ and B such that $\sigma(B) \ge 1 - 1/k$. Then for any a, b, let

$$\hat{s}_{a,b}(z) = \sigma\left(\frac{z-a}{b-a}(B-A) + A\right) \tag{3.32}$$

so that if $z \le a$ or $z \ge b$, then $|\hat{s}_{a,b}(z) - s_{a,b}(z)|$. In other words, σ can always be squashed (horizontally) so that it has small enough error outside the interval (a, b).



$\square$

Bibliography

Arora, Raman, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee (2018). Understanding deep neural networks with rectified linear units. In: *Proceedings of ICLR*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. In: *Math. Control Signal Systems* 2, pp. 303–314.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). Multilayer feedforward networks are universal approximators. In: *Neural Networks* 2.5, pp. 359–366.

Telgarsky, Matus (2021). Deep learning theory lecture notes. Unpublished lecture notes.