

Package ‘nnR’

September 17, 2026

Type Package

Title Neural Networks Made Algebraic

Version 0.2.0

Maintainer Shakil Rafi <sarafi@uark.edu>

Description Build, compose, stack, sum, and realize feed-forward neural networks as algebraic objects. Implements the calculus and constructive approximations described by Rafi, Padgett, and Nakarmi (2024) <[doi:10.48550/arXiv.2402.01058](https://doi.org/10.48550/arXiv.2402.01058)>, Grohs, Hornung, Jentzen, et al. (2023) <[doi:10.1007/s10444-022-09970-2](https://doi.org/10.1007/s10444-022-09970-2)>, and Jentzen, Kuckuck, and von Wursterberger (2023) <[doi:10.48550/arXiv.2310.20360](https://doi.org/10.48550/arXiv.2310.20360)>. Includes neural network polynomials, transcendental-function approximations, multidimensional maximum convolution, and vectorized batch realization.

License GPL-3

Encoding UTF-8

Depends R (>= 4.1.0)

Suggests knitr, rmarkdown, testthat (>= 3.2.0)

Config/testthat/edition 3

URL <https://github.com/2shakilrafi/nnR/>

BugReports <https://github.com/2shakilrafi/nnR/issues?q=is%3Aissue+is%3Aopen+sort%3Aupdated-desc>

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Shakil Rafi [aut, cre] (ORCID: <<https://orcid.org/0000-0003-3791-9697>>),
Joshua Lee Padgett [aut] (ORCID:
<<https://orcid.org/0000-0001-9369-351X>>),
Ukash Nakarmi [ctb] (ORCID: <<https://orcid.org/0000-0002-5351-3956>>)

Repository CRAN

Date/Publication 2026-09-17 20:50:02 UTC

Contents

Aff	3
comp	4
Cpy	5
create_block_diagonal	5
create_nn	6
Csn	7
dep	7
draw_nn	8
Etr	9
generate_random_matrix	10
hid	10
i	11
Id	11
inn	12
inst	13
is_nn	14
lay	15
MC	16
Mxm	17
nn_sum	17
nn_sum_many	18
Nrm	19
out	20
param	20
Phi	21
Phi_k	22
Pnm	22
Prd	23
Pwr	24
realize_nn	25
ReLU	25
Sigmoid	26
slm	27
Sne	28
Sqr	28
srm	29
stk	30
stk_many	31
Sum	32
Tanh	32
Tay	33
Trp	34
Tun	34
view_nn	35
Xpn	36

Aff	Aff
-----	-----

Description

The function that returns Aff neural networks.

Usage

Aff(W, b)

Arguments

- W An $m \times n$ matrix representing the weight of the affine neural network
- b An $m \times 1$ vector representing the bias of the affine neural network

Value

Returns the network $((W, b))$ representing an affine neural network. Also denoted as $\text{Aff}_{W,b}$. See also [Cpy](#) and [Sum](#).

References

Definition 2.3.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>

And especially:

Definition 2.8. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Aff(4, 5)
c(5, 6, 7, 8, 9, 10) |>
  matrix(2, 3) |>
  Aff(c(5, 6))
```

 comp

comp

Description

The function that takes the composition of two neural networks assuming they are compatible, i.e., given $\nu_1, \nu_2 \in \text{NN}$, it must be the case that $l(\nu)_1 = O(\nu_2)$.

Usage

```
comp(phi_1, phi_2)
```

```
phi_1 %comp% phi_2
```

Arguments

phi_1 first neural network to be composed, goes on the left

phi_2 second neural network to be composed, goes on right

Value

The composed neural network. See also [dep](#).

Our definition derive specifically from:

References

Definition 2.1.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>

Remark: We have two versions of this function, an infix version for close resemblance to mathematical notation and prefix version.

Examples

```
create_nn(c(5, 4, 6, 7)) |> comp(create_nn(c(4, 1, 5)))
```

Cpy	Cpy
-----	-----

Description

The function that returns Cpy neural networks. These are neural networks defined as such

$$\text{Aff}_{[\mathbb{I}_k \ \mathbb{I}_k \ \cdots \ \mathbb{I}_k]^T, 0_k}$$

Usage

Cpy(n, k)

Arguments

- n
- number of copies to make.
- k
- the size of the input vector.

Value

Returns an affine network that makes a concatenated vector that is n copies of the input vector of size k . See [Aff](#) and [Sum](#).

References

Definition 2.9. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

create_block_diagonal	<i>Function for creating a block diagonal given two matrices.</i>
-----------------------	---

Description

Function for creating a block diagonal given two matrices.

Usage

create_block_diagonal(matrix1, matrix2)

Arguments

- matrix1
- A matrix.
- matrix2
- A matrix

Value

A block diagonal matrix with matrix1 on top left and matrix2 on bottom right.

create_nn	<i>create_nn</i>
-----------	------------------

Description

Function to create a list of lists for neural network layers

Usage

create_nn(layer_architecture)

Arguments

layer_architecture
A numeric vector specifying the width of each layer, including input and output layers.

Value

An ordered list of ordered pairs of (W, b) . Where W is the matrix representing the weight matrix at that layer and b the bias vector. Entries on the matrix come from a standard normal distribution.

References

Definition 2.1 in Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Which in turn is a modified version of the one found in:

Definition 2.3. Grohs, P., Hornung, F., Jentzen, A. et al. Space-time error estimates for deep neural network approximations for differential equations. (2019). <https://arxiv.org/abs/1908.03833>.

Examples

```
create_nn(c(8, 7, 8))
create_nn(c(4,4))
```

Csn	Csn
-----	-----

Description

Construct a neural network cosine approximation.

Usage

Csn(n, q, eps)

Arguments

- n A non-negative Taylor truncation index.
- q A finite real number greater than 2.
- eps A finite positive approximation parameter. Extreme combinations of q and eps that cannot be represented by R doubles are rejected.

Value

A neural network that approximates cos under instantiation with ReLU activation. See also [Sne](#).

References

Definition 2.29 in Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Csn(2, 2.5, 0.5)
Csn(2, 2.5, 0.5) |> inst(ReLU, 1.50)
```

dep	dep
-----	-----

Description

The function that returns the depth of a neural network. Denoted D.

Usage

dep(nu)

Arguments

`nn` a neural network of the type generated by `create_nn()`. Very straightforwardly it is the length of the list where neural networks are defined as an ordered list of lists.

Value

Integer representing the depth of the neural network.

References

Definition 1.3.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Examples

```
create_nn(c(4, 5, 6, 2)) |> dep()
```

draw_nn

Draw a neural network architecture

Description

Draw a simple fully connected architecture diagram from either a neural network object or a numeric layer-width vector.

Usage

```
draw_nn(x, ...)
```

Arguments

`x` A neural network or a numeric vector of positive layer widths.

`...` Additional named arguments passed to `graphics::plot.default()`. These override the diagram defaults, so arguments such as `main`, `xlim`, and `bty` can be customized.

Value

The plotted node coordinates, invisibly.

Examples

```
draw_nn(c(2, 3, 1))
draw_nn(create_nn(c(2, 4, 1)))
```

Etr	Etr
-----	-----

Description

The function that returns the Etr networks.

Usage

Etr(*n*, *h*)

Arguments

- n* number of trapezoids to make. Note this will result in a set of trapezoids. A natural number.
- h* width of trapezoids. A positive real number.
Note: Upon instantiation with any continuous function this neural network must be fed with $n + 1$ real numbers representing the values of the function being approximated at the $n + 1$ meshpoints which are the legs of the n trapezoids as stipulated in the input parameter n ..

Value

An approximation for value of the integral of a function. Must be instantiated with a list of $n + 1$ reals

References

Definition 2.33. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Etr(5, 0.1)
seq(0, pi, length.out = 1000) |> sin() -> samples
Etr(1000 - 1, pi / (1000 - 1)) |> inst(ReLU, samples)

seq(0, 2, length.out = 1000)^2 -> samples
Etr(1000 - 1, 2 / (1000 - 1)) |> inst(Tanh, samples)
```

generate_random_matrix	
Function to generate a random matrix with specified dimensions.	

Description	
Function to generate a random matrix with specified dimensions.	
Usage	
generate_random_matrix(rows, cols)	
Arguments	
rows	number of rows.
cols	number of columns.
Value	
a random matrix of dimension rows times columns with elements from a standard normal distribution	

hid	hid
-----	-----

Description

The function that returns the number of hidden layers of a neural network. Denoted H

Usage

hid(nu)

Arguments

nu a neural network of the type generated by create_nn()
By definition $H(\nu) = D(\nu) - 1$

Value

Integer representing the number of hidden layers.

References

Definition 1.3.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Examples

```
create_nn(c(4, 5, 6, 2)) |> hid()
```

<i>i</i>	<i>i</i>
----------	----------

Description

The function that returns the $\sqsupset$ network.

Usage

```
i(d)
```

Arguments

d the size of the *i* network

Value

returns the *i_d* network

References

Definition 2.2.6. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>

Examples

```
i(5)
i(10)
```

<i>Id</i>	<i>: Id</i>
-----------	-------------

Description

The function that returns the *Id*₁ networks.

Usage

```
Id(d = 1)
```

Arguments

`d` the dimension of the *Id* network, by default it is 1.

Value

Returns the Id_1 network.

References

Definition 2.17. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

`Id()`
`Id(3)`

<code>inn</code>	<i>inn</i>
------------------	------------

Description

The function that returns the input layer size of a neural network. Denoted l

Usage

`inn(nu)`

Arguments

`nu` A neural network of the type generated by `create_nn()`.

Value

An integer representing the input width of the neural network.

References

Definition 1.3.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Examples

`create_nn(c(4, 5, 6, 2)) |> inn()`

<i>inst</i>	<i>inst</i>
-------------	-------------

Description

The function that instantiates a neural network as created by `create_nn()`.

Usage

```
inst(neural_network, activation_function, x)
```

Arguments

neural_network An ordered list of lists, of the type generated by `create_nn()` where each element in the list of lists is a pair (W, b) representing the weights and biases of that layer.

NOTE: We call instantiation what Grohs et al. call "realization".

activation_function

A continuous function applied to the output of each layer. For now we only have ReLU, Sigmoid, and Tanh. Note, all proofs are only valid for ReLU activation.

x

Input to the instantiated function. A numeric vector represents one point in $\mathbb{R}^d$. A numeric matrix represents a batch with one point per column and must have `inn(neural_network)` rows.

Value

The output of the continuous function that is the instantiation of the given neural network with the given activation function at the given x . Where x is of vector size equal to the input layer of the neural network.

References

Grohs, P., Hornung, F., Jentzen, A. et al. Space-time error estimates for deep neural network approximations for differential equations. (2019). <https://arxiv.org/abs/1908.03833>.

Definition 1.3.4. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>

Very precisely we will use the definition in:

Definition 2.3 in Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
create_nn(c(1, 3, 5, 6)) |> inst(ReLU, 5)
create_nn(c(3, 3, 5, 6)) |> inst(ReLU, c(4, 4, 4))
Aff(matrix(c(1, 2), nrow = 1), 0) |>
  inst(ReLU, matrix(c(1, 2, 3, 4), nrow = 2))
```

is_nn

Test and validate neural network objects

Description

is_nn() provides a fast, non-throwing predicate. validate_nn() reports the first structural problem it finds, which is useful when constructing networks by hand.

Usage

```
is_nn(nn)
```

```
validate_nn(nn)
```

Arguments

nn	A neural network. Neural networks are defined to be an ordered list of ordered pairs of (W, b) . Where W is the matrix representing the weight matrix W at that layer and b the bias vector.
----	--

Value

is_nn() returns TRUE or FALSE. validate_nn() returns TRUE invisibly when nn is valid and otherwise throws an informative error.

We will use the definition of neural networks as found in:

References

Definition 2.1 in Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Which in turn is a modified version of the one found in:

Definition 2.3. Grohs, P., Hornung, F., Jentzen, A. et al. Space-time error estimates for deep neural network approximations for differential equations. (2019). <https://arxiv.org/abs/1908.03833>.

Examples

```
create_nn(c(5, 6, 7)) |> is_nn()
Sqr(2.1, 0.1) |> is_nn()

network <- create_nn(c(2, 3, 1))
is_nn(network)
validate_nn(network)
is_nn(list(list(W = matrix(1), b = matrix(c(0, 1))))))
```

lay

*lay***Description**

The function that returns the layer architecture of a neural network.

Usage

```
lay(nu)
```

Arguments

nu A neural network of the type generated by create_nn(). Denoted L.

Value

A tuple representing the layer architecture of our neural network.

References

Definition 1.3.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Examples

```
create_nn(c(4, 5, 6, 2)) |> lay()
```

MC

*The MC neural network***Description**

This function implements the maximum-convolution approximation scheme outlined in the references for samples in any finite dimension.

Usage

```
MC(X, y, L)
```

Arguments

<code>X</code>	A numeric matrix with one sample per column and one coordinate per row (shape $d \times N$). A vector is treated as one-dimensional samples.
<code>y</code>	A numeric vector of the N sampled function values.
<code>L</code>	the Lipschitz constant for the function. Not necessarily global, but could be an absolute upper limit of slope, over the domain.

Value

A neural network that gives the maximum convolution approximation of a function whose outputs are y at N sample points given by the columns of X , when instantiated with ReLU.

References

Lemma 4.2.9. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Examples

```
seq(0, 3.1416, length.out = 200) -> X
sin(X) -> y
MC(X, y, 1) |> inst(ReLU, 0.25) # compare to sin(0.25)
```

Mxm	<i>Mxm</i>
-----	------------

Description

The function that returns the Mxm neural networks.
The construction reduces pairs in a balanced hierarchy and supports every positive input dimension.

Usage

Mxm(d)

Arguments

d The dimension of the input vector on instantiation.

Value

The neural network that will output the maximum of a vector of size d when activated with the ReLU function.
For a specific definition, see:

References

Lemma 4.2.4. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>

Examples

```
Mxm(1) |> inst(ReLU, -5)
Mxm(3) |> inst(ReLU, c(4, 5, 1))
Mxm(5) |> inst(ReLU, c(5, 3, -1, 6, 6))
```

nn_sum	<i>nn_sum</i>
--------	---------------

Description

A function that performs the neural network sum for two neural networks of the type generated by create_nn().
If the inputs have unequal depths, tunnel padding makes the realization identity ReLU-specific.
For a specific definition, see:

Usage

```
nn_sum(nu_1, nu_2)
```

```
nu_1 %nn_sum% nu_2
```

Arguments

nu_1 A neural network.

nu_2 A neural network.

Value

A neural network that is the neural network sum of ν_1 and ν_2 i.e. $\nu_1 \oplus \nu_2$.

Note: We have two versions, an infix version and a prefix version.

References

Proposition 2.25. Grohs, P., Hornung, F., Jentzen, A. et al. Space-time error estimates for deep neural network approximations for differential equations. (2019). <https://arxiv.org/abs/1908.03833>.

Examples

```
Prd(2.1, 0.1) |> nn_sum(Prd(2.1, 0.1))
```

nn_sum_many	<i>Sum any number of neural networks</i>
-------------	--

Description

This is the n-ary counterpart of `nn_sum()`. All networks must have the same input and output widths; unequal depths are padded once before the networks are stacked. Since padding uses `Tun()`, the realization identity for unequal-depth inputs is guaranteed under ReLU activation.

Usage

```
nn_sum_many(networks)
```

Arguments

networks A non-empty list of neural networks with matching end widths.

Value

A neural network whose ReLU realization is the component-wise sum of all supplied network realizations.

References

Definitions 2.19 and 2.20 in Rafi S., Padgett, J.L., Nakarmi, U. (2024), *Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials*. <https://arxiv.org/abs/2402.01058>

Examples

```
total <- nn_sum_many(list(Aff(1, 1), Aff(2, 0), Aff(-1, 3)))
inst(total, ReLU, 4)
```

Nrm	<i>Nrm</i>
-----	------------

Description

A function that creates the Nrm neural networks.that take the 1- norm of a d -dimensional vector when instantiated with ReLU activation.

Usage

```
Nrm(d)
```

Arguments

d the dimensions of the vector or list being normed.

Value

a neural network that takes the 1-norm of a vector of size d.under ReLU activation.

Note: This function is split into two cases much like the definition itself.

Note: If you choose to specify a d other than 0 you must instantiate with a vector or list of that length.

For a specific definition, see:

References

Lemma 4.2.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>

Examples

```
Nrm(2) |> inst(ReLU, c(5,6))
Nrm(5) |> inst(ReLU,c(0,-9,3,4,-11))
```

<code>out</code>	<i>out</i>
------------------	------------

Description

The function that returns the output layer size of a neural network. Denoted O .

Usage

`out(nu)`

Arguments

`nu` A neural network of the type generated by `create_nn()`.

Value

An integer representing the output width of the neural network.

References

Definition 1.3.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Examples

```
create_nn(c(4, 5, 6, 2)) |> out()
```

<code>param</code>	<i>param</i>
--------------------	--------------

Description

The function that returns the number of parameters of a neural network.

Usage

`param(nu)`

Arguments

`nu` A neural network of the type generated by `create_nn()`. Denoted P .

Value

An integer representing the parameter count of our neural network.

References

Definition 1.3.1. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Examples

```
create_nn(c(4, 5, 6, 2)) |> param()
```

Phi	<i>The Phi function</i>
-----	-------------------------

Description

The Phi function

Usage

```
Phi(eps)
```

Arguments

eps parameter for Phi in $(0, \infty)$

Value

neural network Phi that approximately squares a number between 0 and 1.

References

Definition 2.23. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Phi(0.5) |> view_nn()
Phi(0.1) |> view_nn()
```

Phi_k	The Phi_k function
Description	
The Phi_k function	
Usage	
Phi_k(k)	
Arguments	
k	an integer $k \in (2, \infty)$
Value	
The Phi_k neural network	
References	
Definition 2.22. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials https://arxiv.org/abs/2402.01058	
Examples	
Phi_k(4) > view_nn() Phi_k(5) > view_nn()	

Pnm	Neural network polynomial
-----	---------------------------

Description

Construct a neural network analogue of $c_0+c_1x+\cdots+c_nx^n$. Power networks are built once, padded to equal depth, scaled by their coefficients, and summed in one n-ary operation. Zero-coefficient terms are omitted without changing the realization.

Usage

Pnm(coefficients, q, eps)

Arguments

- coefficients Finite numeric coefficients in ascending power order: $c(c_0, c_1, \dots, c_n)$.
- q A finite real number greater than 2.
- eps A finite positive approximation parameter.

Value

A one-input, one-output neural network polynomial.

References

Definition 2.27 in Rafi S., Padgett, J.L., Nakarmi, U. (2024), *Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials*. <https://arxiv.org/abs/2402.01058>

Examples

```
quadratic <- Pnm(c(1, -2, 0.5), q = 3, eps = 0.5)
inst(quadratic, ReLU, 2)
```

Prd	<i>Prd</i>
-----	------------

Description

A function that returns the Prd neural networks that approximates the product of two real numbers when given an appropriate q, ε , a real number x and instantiation with ReLU. activation.

Usage

```
Prd(q, eps)
```

Arguments

- q A finite real number greater than 2.
- eps A finite positive approximation parameter. Extreme combinations of q and eps that cannot be represented by R doubles are rejected.

Value

A neural network that takes in x and y and approximately returns xy when instantiated with ReLU activation at $c(x, y)$.
Note that this must be instantiated with a tuple $c(x,y)$

References

Proposition 3.5. Grohs, P., Hornung, F., Jentzen, A. et al. Space-time error estimates for deep neural network approximations for differential equations. (2019). <https://arxiv.org/abs/1908.03833>

Definition 2.25. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Prd(2.1, 0.1) |> inst(ReLU, c(4, 5))
```

Pwr	Pwr
-----	-----

Description

A function that returns the Pwr neural networks.

Usage

```
Pwr(q, eps, exponent)
```

Arguments

- q A finite real number greater than 2.
- eps A finite positive approximation parameter. Extreme combinations of q and eps that cannot be represented by R doubles are rejected.
- exponent A non-negative integer power.

Value

A neural network that approximates raising a number to exponent, when given appropriate q, ϵ and exponent when instantiated under ReLU activation at x .

Examples

```
Pwr(2.1, 0.1, 2) |> inst(ReLU, 3)
```

realize_nn	Create a reusable neural network realization
------------	--

Description

realize_nn() validates a network once and returns a function that can be called repeatedly. This is convenient and avoids repeated structural checks in simulation or plotting loops.

Usage

```
realize_nn(neural_network, activation_function = ReLU)
```

Arguments

- neural_network A valid neural network.
- activation_function A component-wise activation function.

Value

A function of one argument, x, with the same vector and column-batch input rules as `inst()`.

References

Definition 2.3 in Rafi S., Padgett, J.L., Nakarmi, U. (2024), *Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials*. <https://arxiv.org/abs/2402.01058>

Examples

```
identity_map <- realize_nn(Id(2), ReLU)
identity_map(c(-1, 2))
```

ReLU	: <i>ReLU</i>
------	---------------

Description

The ReLU activation function

Usage

```
ReLU(x)
```

Arguments

x A real number that is the input to our ReLU function.

Value

The output of the standard ReLU function, i.e. $\max\{0, x\}$. See also [Sigmoid](#). and [Tanh](#).

Examples

ReLU(5)
ReLU(-5)

Sigmoid	: <i>Sigmoid</i>
---------	------------------

Description

The Sigmoid activation function.

Usage

Sigmoid(x)

Arguments

x a real number that is the input to our Sigmoid function.

Value

The output of a standard Sigmoid function, i.e. $\frac{1}{1+\exp(-x)}$. See also [Tanh](#).and [ReLU](#).

Examples

Sigmoid(0)
Sigmoid(-1)

slm

slm

Description

The function that returns the left scalar multiplication neural network

Usage

```
slm(a, nu)
```

```
a %slm% nu
```

Arguments

a	A real number.
nu	A neural network of the type generated by create_nn().

Value

Returns a neural network that is $a \triangleright \nu$. This instantiates as $a \cdot f(x)$ under continuous function activation. More specifically we define operation as:

Let $\lambda \in \mathbb{R}$. We will denote by $(\cdot) \triangleright (\cdot) : \mathbb{R} \times \text{NN} \rightarrow \text{NN}$ the function satisfying for all $\nu \in \text{NN}$ and $\lambda \in \mathbb{R}$ that $\lambda \triangleright \nu = \text{Aff}_{\lambda \mathbb{I}_{(\nu)}, 0} \bullet \nu$.

References

Definition 2.3.4. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Note: We will have two versions of this operation, a prefix and an infix version.

Examples

```
5 |> slm(Prd(2.1, 0.1))
Prd(2.1, 0.1) |> srm(5)
```

Sne	<i>Sne</i>
-----	------------

Description

Construct a neural network sine approximation.

Usage

Sne(*n*, *q*, *eps*)

Arguments

- n* A non-negative Taylor truncation index.
- q* A finite real number greater than 2.
- eps* A finite positive approximation parameter. Extreme combinations of *q* and *eps* that cannot be represented by R doubles are rejected.

Value

A neural network that approximates sin when given an appropriate *n, q, ε* and instantiated with ReLU activation and given value *x*.

References

Definition 2.30. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

Sne(2, 2.3, 0.3)
Sne(2, 2.3, 0.3) |> inst(ReLU, 1.57)

Sqr	<i>Sqr</i>
-----	------------

Description

A function that returns the Sqr neural networks.

Usage

Sqr(*q*, *eps*)

Arguments

q	A finite real number greater than 2.
eps	A finite positive approximation parameter. Extreme combinations of q and eps that cannot be represented by R doubles are rejected.

Value

A neural network that approximates the square function when instantiated with ReLU.

References

Proposition 3.4. Grohs, P., Hornung, F., Jentzen, A. et al. Space-time error estimates for deep neural network approximations for differential equations. (2019). <https://arxiv.org/abs/1908.03833>

Definition 2.24. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Sqr(2.5, 0.1)
Sqr(2.5, 0.1) |> inst(ReLU, 4)
```

srm	<i>srm</i>
-----	------------

Description

The function that returns the right scalar multiplication neural network

Usage

```
srm(nu, a)

nu %srm% a
```

Arguments

nu	A neural network
a	A real number.

Value

Returns a neural network that is $\nu \triangleleft a$. This instantiates as $f(a \cdot x)$.under continuous function activation. More specifically we will define this operation as:

Let $\lambda \in \mathbb{R}$. We will denote by $(\cdot) \triangleleft (\cdot) : \text{NN} \times \mathbb{R} \rightarrow \text{NN}$ the function satisfying for all $\nu \in \text{NN}$ and $\lambda \in \mathbb{R}$ that $\nu \triangleleft \lambda = \nu \bullet \text{Aff}_{\lambda \mathbb{I}_{(\nu)}, 0}$.

References

Definition 2.3.4. Jentzen, A., Kuckuck, B., and von Wurstemberger, P. (2023). Mathematical introduction to deep learning: Methods, implementations, and theory. <https://arxiv.org/abs/2310.20360>.

Note: We will have two versions of this operation, a prefix and an infix version.

stk	stk
-----	-----

Description

A function that stacks neural networks.

Usage

```
stk(nu, mu)
```

```
nu %stk% mu
```

Arguments

nu	neural network.
mu	neural network.

Value

A stacked neural network of ν and μ , i.e. $\nu \boxplus \mu$

NOTE: This is different than the one given in Grohs, et. al. 2023. While we use padding to equalize neural networks being parallelized our padding is via the Tun network whereas Grohs et. al. uses repetitive composition of the i network. We use repetitive composition of the Id_1 network. See [Id comp](#)

NOTE: The terminology is also different from Grohs et. al. 2023. We call stacking what they call parallelization. This terminology change was inspired by the fact that parallelization implies commutativity but this operation is not quite commutative. When depths differ, padding with [Tun\(\)](#) preserves the stated realization semantics under ReLU activation.

Also the word parallelization has a lot of baggage when it comes to artificial neural networks in that it often means many different CPUs working together.

Remark: We will use only one symbol for stacking equal and unequal depth neural networks, namely "stk". This is for usability but also that for all practical purposes only the general stacking of neural networks of different sizes is what is needed.

Remark: We have two versions, a prefix and an infix version.

This operation on neural networks, called "parallelization" is found in:

A stacked neural network of nu and mu.

References

Grohs, P., Hornung, F., Jentzen, A. et al. Space-time error estimates for deep neural network approximations for differential equations. (2023). <https://arxiv.org/abs/1908.03833>

And especially in:

' Definition 2.14 in Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
create_nn(c(4,5,6)) |> stk(create_nn(c(6,7)))
create_nn(c(9,1,67)) |> stk(create_nn(c(4,4,4,4,4)))
```

stk_many	<i>Stack any number of neural networks</i>
----------	--

Description

stk_many() implements the n-ary stacking (parallelization) operation in a single pass. Networks of unequal depth are padded with [Tun\(\)](#) networks before their layers are combined. Because tunnel networks realize the identity under ReLU, the concatenation guarantee for unequal depths is ReLU-specific. Equal-depth stacking does not need padding.

Usage

```
stk_many(networks)
```

Arguments

networks A non-empty list of neural networks.

Value

One neural network whose ReLU realization concatenates the supplied realizations in list order.

References

Definitions 2.14 and 2.18 in Rafi S., Padgett, J.L., Nakarmi, U. (2024), *Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials*. <https://arxiv.org/abs/2402.01058>

Examples

```
stacked <- stk_many(list(Aff(2, 1), Aff(-1, 3), Aff(0.5, 0)))
inst(stacked, ReLU, c(4, 4, 4))
```

Sum	<i>Sum</i>
-----	------------

Description

The function that returns Sum neural networks.
These are neural networks defined as such

$$\text{Aff}_{[\mathbb{I}_k \ \mathbb{I}_k \ \cdots \ \mathbb{I}_k], 0_k}$$

Usage

Sum(n, k)

Arguments

- n number of copies of a certain vector to be summed.
- k the size of the summation vector.

Value

An affine neural network that will take a vector of size $n \times k$ and return the summation vector that is of length k . See also [Aff](#) and [Cpy](#).

References

Definition 2.10. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Tanh	<i>Tanh</i>
------	-------------

Description

The tanh activation function

Usage

Tanh(x)

Arguments

- x a real number

Value

the *tanh* of x. See also [Sigmoid](#) and [ReLU](#).

Examples

```
Tanh(0)
Tanh(0.1)
```

Tay	<i>The Tay function</i>
-----	-------------------------

Description

The Tay function

Usage

```
Tay(f, n, q, eps)
```

Arguments

- f One of "exp", "sin", or "cos".
- n A non-negative Taylor truncation index.
- q A finite real number greater than 2.
- eps A finite positive approximation parameter. Extreme combinations of q and eps that cannot be represented by R doubles are rejected.

Value

a neural network that approximates the function f. For now only *sin*, *cos*, and e^x are available.

Examples

```
Tay("sin", 2, 2.3, 0.3) |> inst(ReLU, 1.5)
Tay("cos", 2, 2.3, 0.3) |> inst(ReLU, 1)
Tay("exp", 4, 2.3, 0.3) |> inst(ReLU, 1.5)
```

Trp	<i>Trp</i>
-----	------------

Description

The function that returns the Trp networks.

Usage

Trp(h)

Arguments

h the horizontal distance between two mesh points

Value

The Trp network that gives the area when activated with ReLU or any continuous function and two meshpoint values x_1 and x_2 .

References

Definition 2.31. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Trp(0.1)
Trp(0.5) |> inst(ReLU, c(9, 7))
Trp(0.1) |> inst(Sigmoid, c(9, 8))
```

Tun	<i>Tun: The function that returns tunneling neural networks</i>
-----	---

Description

Tun: The function that returns tunneling neural networks

Usage

Tun(n, d = 1)

Arguments

- n The depth of the tunnel network where $n \in \mathbb{N} \cap [1, \infty)$.
- d The dimension of the tunneling network. By default it is assumed to be 1.

Value

A tunnel neural network of depth n. A tunneling neural network is defined as the neural network $\text{Aff}_{1,0}$ for $n = 1$, the neural network Id_1 for $n = 2$ and the neural network $\bullet^{n-2}\text{Id}_1$ for $n > 2$. For this to work we must provide an appropriate n and instantiate with ReLU at some real number x .

References

Definition 2.17. Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Tun(4)
Tun(4, 3) |> view_nn()

Tun(5)
Tun(5, 3)
```

view_nn	view_nn
---------	---------

Description

Takes a neural network shown in vectorized form and explicitly displays it.

Usage

```
view_nn(nn)
```

Arguments

- nn A neural network., i.e. a list of lists of W and b .

Value

The original neural network, invisibly, after printing its layers.

Examples

```
c(5, 6, 7, 9) |>
  create_nn() |>
  view_nn()
Sqr(2.1, 0.1) |> view_nn()

Xpn(3, 2.1, 1.1) |> view_nn()
Pwr(2.1, 0.1, 3) |> view_nn()
```

Xpn	<i>The Xpn function</i>
-----	-------------------------

Description

The Xpn function

Usage

```
Xpn(n, q, eps)
```

Arguments

- n A non-negative Taylor truncation index.
- q A finite real number greater than 2.
- eps A finite positive approximation parameter. Extreme combinations of q and eps that cannot be represented by R doubles are rejected.

Value

A neural network that approximates e^x for real x when given appropriate n, q, ε and instantiated with ReLU activation at point x .

References

Definition 2.28 in Rafi S., Padgett, J.L., Nakarmi, U. (2024) Towards an Algebraic Framework For Approximating Functions Using Neural Network Polynomials <https://arxiv.org/abs/2402.01058>

Examples

```
Xpn(3, 2.25, 0.25)
Xpn(3, 2.2, 0.2) |> inst(ReLU, 1.5)
```

Index

`%comp%` (`comp`), [4](#)
`%nn_sum%` (`nn_sum`), [17](#)
`%slm%` (`slm`), [27](#)
`%srm%` (`srm`), [29](#)
`%stk%` (`stk`), [30](#)

`Aff`, [3](#), [5](#), [32](#)

`comp`, [4](#), [30](#)
`Cpy`, [3](#), [5](#), [32](#)
`create_block_diagonal`, [5](#)
`create_nn`, [6](#)
`Csn`, [7](#)

`dep`, [4](#), [7](#)
`draw_nn`, [8](#)

`Etr`, [9](#)

`generate_random_matrix`, [10](#)
`graphics::plot.default()`, [8](#)

`hid`, [10](#)

`i`, [11](#)
`Id`, [11](#), [30](#)
`inn`, [12](#)
`inst`, [13](#)
`inst()`, [25](#)
`is_nn`, [14](#)

`lay`, [15](#)

`MC`, [16](#)
`Mxm`, [17](#)

`nn_sum`, [17](#)
`nn_sum()`, [18](#)
`nn_sum_many`, [18](#)
`Nrm`, [19](#)

`out`, [20](#)

`param`, [20](#)
`Phi`, [21](#)
`Phi_k`, [22](#)
`Pnm`, [22](#)
`Prd`, [23](#)
`Pwr`, [24](#)

`realize_nn`, [25](#)
`ReLU`, [25](#), [26](#), [33](#)

`Sigmoid`, [26](#), [26](#), [33](#)
`slm`, [27](#)
`Sne`, [7](#), [28](#)
`Sqr`, [28](#)
`srm`, [29](#)
`stk`, [30](#)
`stk_many`, [31](#)
`Sum`, [3](#), [5](#), [32](#)

`Tanh`, [26](#), [32](#)
`Tay`, [33](#)
`Trp`, [34](#)
`Tun`, [34](#)
`Tun()`, [18](#), [30](#), [31](#)

`validate_nn(is_nn)`, [14](#)
`view_nn`, [35](#)

`Xpn`, [36](#)