

digitLearning

Generated by Doxygen 1.8.11

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Class Index	3
2.1	Class List	3
3	Namespace Documentation	5
3.1	cProfiler Namespace Reference	5
3.1.1	Detailed Description	5
3.1.2	Function Documentation	5
3.1.2.1	main()	5
3.2	main Namespace Reference	5
3.2.1	Detailed Description	6
3.2.2	Function Documentation	6
3.2.2.1	main()	6
3.3	src.argumentsManager Namespace Reference	6
3.3.1	Detailed Description	6
3.3.2	Variable Documentation	7
3.3.2.1	POSSIBLE_GRAD_DESC_FACT_FUNC	7
3.4	src.externalFunc Namespace Reference	7
3.4.1	Detailed Description	7
3.4.2	Function Documentation	7
3.4.2.1	Constant(n, value=1)	7
3.4.2.2	CostFunction(output_layer, perfect_output)	7

3.4.2.3	DerCostFunction(output_layer, perfect_output)	7
3.4.2.4	isfloat(string)	8
3.4.2.5	NegPower(n, value=1.3)	8
3.4.2.6	progressbar(it, prefix="*****", size=60)	8
3.5	src.neuralNetwork Namespace Reference	8
3.5.1	Detailed Description	8
3.6	src.squishingFunc Namespace Reference	8
3.6.1	Detailed Description	9
3.6.2	Function Documentation	9
3.6.2.1	DerReEU(x)	9
3.6.2.2	DerReLU(x)	9
3.6.2.3	DerSigmoid(x)	9
3.6.2.4	InvReEU(x)	9
3.6.2.5	InvReLU(x)	9
3.6.2.6	InvSigmoid(x)	9
3.6.2.7	ReEU(x)	10
3.6.2.8	ReLU(x)	10
3.6.2.9	Sigmoid(x)	10
4	Class Documentation	11
4.1	src.argumentsManager.ArgsManager Class Reference	11
4.1.1	Detailed Description	12
4.1.2	Constructor & Destructor Documentation	12
4.1.2.1	__init__(self, list_args)	12
4.1.3	Member Function Documentation	12
4.1.3.1	analyseArgs(self, list_args)	12
4.1.3.2	checkBatchesSizeArg(self, arg)	12
4.1.3.3	checkGradientDescentFactorArg(self, arg)	12
4.1.3.4	checkInitArg(self, arg, main_dir)	12
4.1.3.5	checkLearningSizeArg(self, arg)	12
4.1.3.6	checkNeuralNetworkArg(self, document)	12

4.1.3.7	checkRepeatArg(self, arg)	13
4.1.3.8	checkSaveArg(self, main_dir)	13
4.1.3.9	checkSquishingFunctionsArg(self, arg)	13
4.1.3.10	checkTestingSizeArg(self, arg)	13
4.1.3.11	display(self)	13
4.1.3.12	displayErrorGrad(self, str_name_func, str_value, function)	13
4.2	src.neuralNetwork.NeuralNetwork Class Reference	13
4.2.1	Detailed Description	14
4.2.2	Constructor & Destructor Documentation	14
4.2.2.1	__init__(self, len_layers, squishing_funcs, dir_load)	14
4.2.3	Member Function Documentation	14
4.2.3.1	calculateNegGradient(self, in_out_layers)	14
4.2.3.2	calculateNegGradientNEO(self, in_out_layers, gdfactor)	15
4.2.3.3	derivativeCostToParam(self, index, a, der_func_z, der_cost_to_a)	15
4.2.3.4	generateAllLayers(self, input_layer)	15
4.2.3.5	generateInputLayer(self, output_layer)	15
4.2.3.6	generateOuputLayer(self, input_layer)	16
4.2.3.7	inform(self, args, error_rate, average_cost)	16
4.2.3.8	initializeEmptyDParamArrays(self)	16
4.2.3.9	initializeWeightsBiases(self, dir_load)	16
4.2.3.10	save(self, dir_save)	16
4.2.3.11	test(self, testing_data)	17
4.2.3.12	train(self, training_data, batch_size, gradientDescentFactor, repeat)	17
4.2.3.13	trainNEO(self, training_data, batch_size, gradientDescentFactor, repeat)	17
	Index	19

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

cProfiler	5
main	5
src.argumentsManager	6
src.externalFunc	7
src.neuralNetwork	8
src.squishingFunc	8

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

src.argumentsManager.ArgsManager	11
src.neuralNetwork.NeuralNetwork	13

Chapter 3

Namespace Documentation

3.1 cProfiler Namespace Reference

Functions

- def `main` ()

3.1.1 Detailed Description

Interpreter to use cProfile and analyse the results.
Use to optimize code by analyzing the execution time of each function.

Practical use to open all the information :
- `atom $(find ./profile -name "*.txt")`

3.1.2 Function Documentation

3.1.2.1 def cProfiler.main ()

Main function.

3.2 main Namespace Reference

Functions

- def `main` ()

3.2.1 Detailed Description

Required for this to work:

- python3 (while coding 'python3 --version' = Python 3.5.2)
- numpy (install by doing 'pip3 install numpy')

Recommended packages for the code :

- linter by steelbrain
- linter-pylint by AtomLinter
- minimap by atom-minimap

Practical stuff: (using atom editor)

- atom \$(find . -name "*.csv")
- atom \$(find ./profile -name "*.txt")

Documentation about docstring convention in Python :

<http://sametmax.com/les-docstrings/>

3.2.2 Function Documentation

3.2.2.1 def main.main ()

Main function. It calls everything to make the whole thing work

3.3 src.argumentsManager Namespace Reference

Classes

- class [ArgsManager](#)

Variables

- int **SIZE_INPUT** = 784
- int **SIZE_OUTPUT** = 10
- int **SIZE_TRAINING** = 60000
- int **SIZE_TESTING** = 10000
- int **REPETITION_LIMIT** = 1000
- list **POSSIBLE_ARGS_WITHOUT_PARAM** = ["-S", "-v", "-l"]
- list **POSSIBLE_ARGS_WITH_PARAM** = ["-bs", "-sf", "-gdf", "-r", "-ls", "-ts", "-init=S"]
- **ALL_POSSIBLE_ARGS** = POSSIBLE_ARGS_WITH_PARAM+POSSIBLE_ARGS_WITHOUT_PARAM
- list **POSSIBLE_SQUISHING_FUNC** = ["Sigmoid", "ReLU", "ReLU"]
- list **POSSIBLE_GRAD_DESC_FACT_FUNC**

3.3.1 Detailed Description

File used to manage the arguments given by the user in the main function. As they are a lot of parameters possible it is better to decompose the code. Moreover, it will be easier to add and/or modify (the behaviour of) some parameters by creating a class to manage them.

3.3.2 Variable Documentation

3.3.2.1 list src.argumentsManager.POSSIBLE_GRAD_DESC_FACT_FUNC

Initial value:

```
1 = ["NegPower{anyPosFloat}",
2   "Constant{anyPosFloat}"]
```

3.4 src.externalFunc Namespace Reference

Functions

- def [NegPower](#) (n, value=1.3)
- def [Constant](#) (n, value=1)
- def [CostFunction](#) (output_layer, perfect_output)
- def [DerCostFunction](#) (output_layer, perfect_output)
- def [isfloat](#) (string)
- def [progressbar](#) (it, prefix="", size=60)

3.4.1 Detailed Description

File used to separate the code and therefore make it cleaner by separating some functions in that file when that makes sense.

3.4.2 Function Documentation

3.4.2.1 def src.externalFunc.Constant (n, value = 1)

Generate a constant gradient descent factor.

3.4.2.2 def src.externalFunc.CostFunction (output_layer, perfect_output)

Generate the cost array. To calculate the cost, you just have to compute `sum(CostFunction(output_layer, perfect_output))`.

Inputs :

-> output_layer, perfect_output : two NUMPY ARRAYS with the same size.

Output :

<- : NUMPY ARRAY which size is equal to the size of the inputs

3.4.2.3 def src.externalFunc.DerCostFunction (output_layer, perfect_output)

Generate an numpy array where each of its composant is the derivative of the cost array.

Exact same inputs and output as the function above.

3.4.2.4 `def src.externalFunc.isfloat (string)`

Function useful to test whether or not a string is a float or not.

3.4.2.5 `def src.externalFunc.NegPower (n, value = 1.3)`

Generate a natural gradient descent factor.
The slope seems good for value = 1.3.

Inputs :

```
-> n          : INT in [0, +inf[
-> value      : FLOAT in ]0, +inf[
```

Output :

```
<-          : FLOAT in ]0, 1[. Return value^(-n).
```

3.4.2.6 `def src.externalFunc.progressBar (it, prefix = " ", size = 60)`

Function used to have access to progress without having to use the package `progressbar2`.
Example of use

3.5 `src.neuralNetwork` Namespace Reference

Classes

- class [NeuralNetwork](#)

Variables

- int **SIZE_INPUT** = 784
- int **SIZE_OUTPUT** = 10

3.5.1 Detailed Description

File `neuralNetwork.py` is used to create and handle a Neural Network as an object.

3.6 `src.squishingFunc` Namespace Reference

Functions

- def [Sigmoid](#) (x)
- def [InvSigmoid](#) (x)
- def [DerSigmoid](#) (x)
- def [ReLU](#) (x)
- def [InvReLU](#) (x)
- def [DerReLU](#) (x)
- def [ReLU](#) (x)
- def [InvReLU](#) (x)
- def [DerReLU](#) (x)

3.6.1 Detailed Description

File used to store different function used to train the neural network.

3.6.2 Function Documentation

3.6.2.1 `def src.squishingFunc.DerReEU (x)`

Derivative of Rectified Exponential Unit function.
Function that returns a value between]0, 1] if the entry x is in [0, +inf] otherwise it returns 0.
In the case of x = 0, return 1/2. It is not mathematically true, however in practical in works well because it is the value that makes sense.

3.6.2.2 `def src.squishingFunc.DerReLU (x)`

Derivative of Rectified Linear Unit function.
If x > 0, return 1, if x < 0 return 0.
In the case of x = 0, return 1/2. It is not mathematically true, however in practical in works well because it is the value that makes sense.

3.6.2.3 `def src.squishingFunc.DerSigmoid (x)`

Derivative of Sigmoid function.
x in [-inf, +inf] and return a value in]0, 1[.
 $\text{Sigmoid}'(x) = (1/2) (1/(\cosh(x)+1))$

3.6.2.4 `def src.squishingFunc.InvReEU (x)`

Inverse Rectified Exponential Unit function.
Function that returns a value between]0, +inf[and we suppose the entry x in]0, 1[.
 $y = 1 - \exp(-x) \Leftrightarrow x = -\ln(1-y)$

3.6.2.5 `def src.squishingFunc.InvReLU (x)`

Inverse Rectified Linear Unit function.
Return a value in [0, +inf] if we suppose the entry x in [0, +inf].

3.6.2.6 `def src.squishingFunc.InvSigmoid (x)`

Inverse Sigmoid function.
x in]0, 1[and return a value in [-inf, +inf].
 $y = 1/(1 + \exp(-x)) \Leftrightarrow x = \ln(y) - \ln(1-y)$

3.6.2.7 `def src.squishingFunc.ReEU (x)`

Rectified Exponential Unit function.
Kind of a mix between sigmoid and ReLU.
Function that returns a value between $[0, 1[$ if the entry x is in $[0, +\infty]$ otherwise it returns 0.
Used this method because it is faster than `np.maximum(0, x)`.

3.6.2.8 `def src.squishingFunc.ReLU (x)`

Rectified Linear Unit function.
The idea is that there is a real activation in neurals from our brains. This function is inspired by biological researchs. If there is no activation, here $x < 0$, we return 0. This means that the neuron doesn't emit any output signal. However if $x \geq 0$, the neuron emit a signal that correspond to the entry.

Beware : this function tends to work well, however neurons in each layer except the first one are not values in $]0, 1[$ but in $]0, +\infty[$ when using this function.

3.6.2.9 `def src.squishingFunc.Sigmoid (x)`

Sigmoid function.
Oldschool way to train networks.
Base function used to train neural network but unefficient
 x in $[-\infty, +\infty]$ and return a value in $]0, 1[$.

Chapter 4

Class Documentation

4.1 src.argumentsManager.ArgsManager Class Reference

Public Member Functions

- def [__init__](#) (self, list_args)
- def [analyseArgs](#) (self, list_args)
- def [checkNeuralNetworkArg](#) (self, document)
- def [checkBatchesSizeArg](#) (self, arg)
- def [checkSquishingFunctionsArg](#) (self, arg)
- def [displayErrorGrad](#) (self, str_name_func, str_value, function)
- def [checkGradientDescentFactorArg](#) (self, arg)
- def [checkRepeatArg](#) (self, arg)
- def [checkLearningSizeArg](#) (self, arg)
- def [checkTestingSizeArg](#) (self, arg)
- def [checkInitArg](#) (self, arg, main_dir)
- def [checkSaveArg](#) (self, main_dir)
- def [display](#) (self)

Public Attributes

- **neural_network**
- **batches_size**
- **squishing_funcs**
- **squishing_funcs_str**
- **grad_desc_factor**
- **grad_desc_factor_str**
- **repeat**
- **learning_size**
- **testing_size**
- **dir_save**
- **dir_load**
- **to_display**
- **to_info**

4.1.1 Detailed Description

Class used to manage arguments.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 `def src.argumentsManager.ArgsManager.__init__ (self, list_args)`

Initialize a object ArgsManager.

4.1.3 Member Function Documentation

4.1.3.1 `def src.argumentsManager.ArgsManager.analyseArgs (self, list_args)`

Method used to analyse the inputs of the main function.
Used to make sure that args are correct but also to assign them.

4.1.3.2 `def src.argumentsManager.ArgsManager.checkBatchesSizeArg (self, arg)`

Check the optional argument batch size.

4.1.3.3 `def src.argumentsManager.ArgsManager.checkGradientDescentFactorArg (self, arg)`

Check the optional argument gradient descent factor.

4.1.3.4 `def src.argumentsManager.ArgsManager.checkInitArg (self, arg, main_dir)`

Method used to check if the arg for the -init=S
pre-parameter is good.

4.1.3.5 `def src.argumentsManager.ArgsManager.checkLearningSizeArg (self, arg)`

Check the optional argument learning size.

4.1.3.6 `def src.argumentsManager.ArgsManager.checkNeuralNetworkArg (self, document)`

Check the required argument neural network document.

4.1.3.7 `def src.argumentsManager.ArgsManager.checkRepeatArg (self, arg)`

Check the optional argument repeat.

4.1.3.8 `def src.argumentsManager.ArgsManager.checkSaveArg (self, main_dir)`

Method used to check the save argument.

4.1.3.9 `def src.argumentsManager.ArgsManager.checkSquishingFunctionsArg (self, arg)`

Check the optional argument squishing functions.

4.1.3.10 `def src.argumentsManager.ArgsManager.checkTestingSizeArg (self, arg)`

Check the optional argument testing size.

4.1.3.11 `def src.argumentsManager.ArgsManager.display (self)`

Method used to display all the argument given by the user.
Mostly useful to debug and see whether or not the argument argument well identify.

4.1.3.12 `def src.argumentsManager.ArgsManager.displayErrorGrad (self, str_name_func, str_value, function)`

Used to simplify the code in the checkGradientDescentFactorArg function.

The documentation for this class was generated from the following file:

- /home/alexandre/Documents/python/digitLearning/src/argumentsManager.py

4.2 src.neuralNetwork.NeuralNetwork Class Reference

Public Member Functions

- `def __init__ (self, len_layers, squishing_funcs, dir_load)`
- `def initializeWeightsBiases (self, dir_load)`
- `def initializeEmptyDParamArrays (self)`
- `def trainNEO (self, training_data, batch_size, gradientDescentFactor, repeat)`
- `def calculateNegGradientNEO (self, in_out_layers, gdfactor)`
- `def train (self, training_data, batch_size, gradientDescentFactor, repeat)`
- `def derivativeCostToParam (self, index, a, der_func_z, der_cost_to_a)`
- `def calculateNegGradient (self, in_out_layers)`
- `def generateOutputLayer (self, input_layer)`
- `def generateAllLayers (self, input_layer)`
- `def generateInputLayer (self, output_layer)`
- `def save (self, dir_save)`
- `def test (self, testing_data)`
- `def inform (self, args, error_rate, average_cost)`

Public Attributes

- **nb_layer**
- **len_layers**
- **weights**
- **biases**
- **squishing_funcs**

4.2.1 Detailed Description

Class neural network.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 `def src.neuralNetwork.NeuralNetwork.__init__(self, len_layers, squishing_funcs, dir_load)`

Initialize an object NeuralNetwork.

Inputs :

-> entry : STRING variable that is the name of a txt document.
This document contains all the information concerning the layers and their sizes. It will be used as followed :
"./main.py information.txt"
ex of entry : network1.txt

4.2.3 Member Function Documentation

4.2.3.1 `def src.neuralNetwork.NeuralNetwork.calculateNegGradient(self, in_out_layers)`

Method used to train the neural network.

Inputs :

-> inOutLayers : a TUPLE that contains two numpy arrays
the first numpy array has a length of $28 \times 28 = 784$
each element is in $[0, 1]$ 0 means a dark pixel
and 1 means a white pixel
the second numpy array has length of 10.
This is the best output that could be obtain when
we test the neural network with the according image

Output :

<- (dweights, dbiases) : TUPLE of LISTS.
The first one contains NUMPY MATRIX for all the
weight matrix in the neural network.
The second one contains NUMPY ARRAY for all the
biases array in the neural network.

4.2.3.2 `def src.neuralNetwork.NeuralNetwork.calculateNegGradientNEO (self, in_out_layers, gdfactor)`

Method used to train the neural network.

Inputs :

-> inOutLayers : a TUPLE that contains two numpy arrays
 the first numpy array has a length of $28 \times 28 = 784$
 each element is in $[0, 1]$ 0 means a dark pixel
 and 1 means a white pixel
 the second numpy array has length of 10.
 This is the best output that could be obtain when
 we test the neural network with the according image

4.2.3.3 `def src.neuralNetwork.NeuralNetwork.derivativeCostToParam (self, index, a, der_func_z, der_cost_to_a)`

Optimized calculation for derivative of the cost function according
 to the following parameters : bias, weight and a.

Complexity : $(\text{column}+1) * \text{row}$

4.2.3.4 `def src.neuralNetwork.NeuralNetwork.generateAllLayers (self, input_layer)`

Method used when training the neural network model by giving it a
 array that contains all the pixels from an handwriting image.
 Each step consists in calculating $f(A_i x_i + b_i) = x_{(i+1)}$.
 A_i is a weight matrix, b_i is a biases vector, x_i is the neural
 vector or layer of index i and $x_{(i+1)}$ of index $(i+1)$.

Input :

-> input_layer : NUMPY ARRAY which len is 784.
 it contains the color of pixel (white / black) with
 number notation from 0 to 1 (everything was divided
 by 255)

Output :

<- values_layers : LIST of NUMPY ARRAY for each layer in the neural
 network (they all have different sizes) which size
 is $\text{nb_layer} + 2$.
 This will be used to calculate the
 negative gradient and therefore to do the back
 propagation. Thus $\text{values_layers}[\text{self.nb_layer}+1]$ is
 a NUMPY ARRAY of size 10.
 ex : $[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$ in the best
 case scenario if the input is a handwriting five.

<- z_values : LIST of NUMPY ARRAY for each layer in the neural
 network minus one (except the first one).
 Thus its size is $\text{nb_layer}+1$.

4.2.3.5 `def src.neuralNetwork.NeuralNetwork.generateInputLayer (self, output_layer)`

Method used to generate an input of the neural network.
 This input can be used after to reconstitute an image thanks to the
 function `writeMNISTImage` in `mnistHandwriting.py` using
 Python Image Library.
 Each step consists in calculating $x_i = A_i^{-1} [f^{-1}(x_{(i+1)}) - b_i]$.
 A_i is a weight matrix, b_i is a biases vector, x_i is the neural
 vector or layer of index i and $x_{(i+1)}$ of index $(i+1)$.

Inputs :

```
-> output_layer : NUMPY ARRAY of size 10. Each element is in
    [0, 1]. ex : [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
```

Output :

```
<- new_array : NUMPY ARRAY which len is 784.
    it contains the color of pixel (white / black) with
    number notation from 0 to 1
```

4.2.3.6 `def src.neuralNetwork.NeuralNetwork.generateOuputLayer (self, input_layer)`

Method used to test the neural network model by giving it an array that contains all the pixels from an handwriting image. Use the same principle as `generateAllLayers` method.

Input :

```
-> input_layer : NUMPY ARRAY which len is 784.
    it contains the color of pixel (white / black) with
    number notation from 0 to 1 (everything was divided
    by 255)
```

Output :

```
<-output_layer : NUMPY ARRAY of size 10.
    ex : [0, 0, 0, 0, 1, 0, 0, 0, 0, 0] in the best
    case scenario if the input is a handwriting five.
```

4.2.3.7 `def src.neuralNetwork.NeuralNetwork.inform (self, args, error_rate, average_cost)`

Method to load information in the CSV file in the correspondent directory.

4.2.3.8 `def src.neuralNetwork.NeuralNetwork.initializeEmptyDParamArrays (self)`

Method used to do the initialization of `dw` and `db` for the method `train`.

4.2.3.9 `def src.neuralNetwork.NeuralNetwork.initializeWeightsBiases (self, dir_load)`

Method used to initialize the matrix weights and the vectors biases. If `load == None`, it means that this will not load weights and biases. Else `load == "dir/doc.txt"` load weight and biases from that doc

4.2.3.10 `def src.neuralNetwork.NeuralNetwork.save (self, dir_save)`

Method used to save the neural network. In fact, it simply writes every matrix weights and biases in the document named `doc_save`.

Moreover, this function also writes information about the size of the training data used to train the model during the execution and

4.2.3.11 `def src.neuralNetwork.NeuralNetwork.test (self, testing_data)`

Method used to test the neural network after its training.

4.2.3.12 `def src.neuralNetwork.NeuralNetwork.train (self, training_data, batch_size, gradientDescentFactor, repeat)`

Method used to train the neural network.

If `batch_size == 1` => individual training
Else => mini_batching training

Repeat is the number of repetition of learning for each batch.

4.2.3.13 `def src.neuralNetwork.NeuralNetwork.trainNEO (self, training_data, batch_size, gradientDescentFactor, repeat)`

Method used to train the neural network.

If `batch_size == 1` => individual training
Else => mini_batching training

Repeat is the number of repetition of learning for each batch.

The documentation for this class was generated from the following file:

- `/home/alexandre/Documents/python/digitLearning/src/neuralNetwork.py`

Index

- `__init__`
 - `src::argumentsManager::ArgsManager`, 12
 - `src::neuralNetwork::NeuralNetwork`, 14
- `analyseArgs`
 - `src::argumentsManager::ArgsManager`, 12
- `cProfiler`, 5
 - `main`, 5
- `calculateNegGradient`
 - `src::neuralNetwork::NeuralNetwork`, 14
- `calculateNegGradientNEO`
 - `src::neuralNetwork::NeuralNetwork`, 14
- `checkBatchesSizeArg`
 - `src::argumentsManager::ArgsManager`, 12
- `checkGradientDescentFactorArg`
 - `src::argumentsManager::ArgsManager`, 12
- `checkInitArg`
 - `src::argumentsManager::ArgsManager`, 12
- `checkLearningSizeArg`
 - `src::argumentsManager::ArgsManager`, 12
- `checkNeuralNetworkArg`
 - `src::argumentsManager::ArgsManager`, 12
- `checkRepeatArg`
 - `src::argumentsManager::ArgsManager`, 12
- `checkSaveArg`
 - `src::argumentsManager::ArgsManager`, 13
- `checkSquishingFunctionsArg`
 - `src::argumentsManager::ArgsManager`, 13
- `checkTestingSizeArg`
 - `src::argumentsManager::ArgsManager`, 13
- `Constant`
 - `src::externalFunc`, 7
- `CostFunction`
 - `src::externalFunc`, 7
- `DerCostFunction`
 - `src::externalFunc`, 7
- `DerReEU`
 - `src::squishingFunc`, 9
- `DerReLU`
 - `src::squishingFunc`, 9
- `DerSigmoid`
 - `src::squishingFunc`, 9
- `derivativeCostToParam`
 - `src::neuralNetwork::NeuralNetwork`, 15
- `display`
 - `src::argumentsManager::ArgsManager`, 13
- `displayErrorGrad`
 - `src::argumentsManager::ArgsManager`, 13
- `generateAllLayers`
 - `src::neuralNetwork::NeuralNetwork`, 15
- `generateInputLayer`
 - `src::neuralNetwork::NeuralNetwork`, 15
- `generateOutputLayer`
 - `src::neuralNetwork::NeuralNetwork`, 16
- `inform`
 - `src::neuralNetwork::NeuralNetwork`, 16
- `initializeEmptyDParamArrays`
 - `src::neuralNetwork::NeuralNetwork`, 16
- `initializeWeightsBiases`
 - `src::neuralNetwork::NeuralNetwork`, 16
- `InvReEU`
 - `src::squishingFunc`, 9
- `InvReLU`
 - `src::squishingFunc`, 9
- `InvSigmoid`
 - `src::squishingFunc`, 9
- `isfloat`
 - `src::externalFunc`, 7
- `main`, 5
 - `cProfiler`, 5
 - `main`, 6
- `NegPower`
 - `src::externalFunc`, 8
- `POSSIBLE_GRAD_DESC_FACT_FUNC`
 - `src::argumentsManager`, 7
- `progressbar`
 - `src::externalFunc`, 8
- `ReEU`
 - `src::squishingFunc`, 9
- `ReLU`
 - `src::squishingFunc`, 10
- `save`
 - `src::neuralNetwork::NeuralNetwork`, 16
- `Sigmoid`
 - `src::squishingFunc`, 10
- `src.argumentsManager`, 6
- `src.argumentsManager.ArgsManager`, 11
- `src.externalFunc`, 7
- `src.neuralNetwork`, 8
- `src.neuralNetwork.NeuralNetwork`, 13
- `src.squishingFunc`, 8
- `src::argumentsManager`
 - `POSSIBLE_GRAD_DESC_FACT_FUNC`, 7

- src::argumentsManager::ArgsManager
 - `__init__`, 12
 - `analyseArgs`, 12
 - `checkBatchesSizeArg`, 12
 - `checkGradientDescentFactorArg`, 12
 - `checkInitArg`, 12
 - `checkLearningSizeArg`, 12
 - `checkNeuralNetworkArg`, 12
 - `checkRepeatArg`, 12
 - `checkSaveArg`, 13
 - `checkSquishingFunctionsArg`, 13
 - `checkTestingSizeArg`, 13
 - `display`, 13
 - `displayErrorGrad`, 13
- src::externalFunc
 - `Constant`, 7
 - `CostFunction`, 7
 - `DerCostFunction`, 7
 - `isfloat`, 7
 - `NegPower`, 8
 - `progressbar`, 8
- src::neuralNetwork::NeuralNetwork
 - `__init__`, 14
 - `calculateNegGradient`, 14
 - `calculateNegGradientNEO`, 14
 - `derivativeCostToParam`, 15
 - `generateAllLayers`, 15
 - `generateInputLayer`, 15
 - `generateOutputLayer`, 16
 - `inform`, 16
 - `initializeEmptyDParamArrays`, 16
 - `initializeWeightsBiases`, 16
 - `save`, 16
 - `test`, 16
 - `train`, 17
 - `trainNEO`, 17
- src::squishingFunc
 - `DerReLU`, 9
 - `DerReLU`, 9
 - `DerSigmoid`, 9
 - `InvReLU`, 9
 - `InvReLU`, 9
 - `InvSigmoid`, 9
 - `ReLU`, 9
 - `ReLU`, 10
 - `Sigmoid`, 10
- test
 - src::neuralNetwork::NeuralNetwork, 16
- train
 - src::neuralNetwork::NeuralNetwork, 17
- trainNEO
 - src::neuralNetwork::NeuralNetwork, 17