

# CITK: Computational Intelligence Toolkit

version 0.1

Dmytro Androsov, Volodymyr Sydorskyy

20.12.2020



# Contents

Welcome to citk's documentation!	1
Quickstart	1
Installation	1
Examples	1
Result Table	1
API reference	2
citk package	2
Submodules	2
citk.functions module	2
citk.layer module	4
citk.losses module	7
citk.model module	8
citk.optimisers module	8
citk.utils module	10
Module contents	10
Indices and tables	10
Index	11
Python Module Index	13



# Welcome to citk's documentation!

## Quickstart

CITK is an ultimate package for State-Of-The-Art CI algorithmes, such as ANN, GMDH and Fuzzy Nets

## Installation

```
git clone https://github.com/tupoylogin/Neural_Net_Genetic_Alg.git
cd Neural_Net_Genetic_Alg
pip install . (or pip install -e . to enable edit mode)
```

## Examples

- Multilayer Perceptron trained with Genetic Algorithm
- Multilayer Perceptron trained with SGD
- Multilayer Perceptron trained with Genetic Algorithm and then with SGD
- Multilayer Perceptron trained with Conjugate SGD
- ANFIS Neural Net trained with SGD
- GMDH Neural Net with Layer Hypersearch trained with SGD
- Fuzzy GMDH Neural Net with Layer Hypersearch trained with SGD

## Result Table

All experiments are carried on [Boston dataset](#)

Using such preprocessing: - Quantile Transform on Target (n\_quantiles=300, output\_distribution="normal") - Standard Scaling of features

Test/Train splitting: - test size - 20% - use histogram bins stratification

[Data preparation code](#)

Metric - MSE on normalized data

Exepriment name	Train score	Test score
MLP+Genetic	0.455	0.645
MLP+SGD	0.323	0.590
MLP+(Genetic->SGD)	0.284	0.558
MLP+Conjugate SGD	0.367	0.563
ANFIS+SGD	0.621	0.768
GMDH+SGD	0.191	0.386
FuzzyGMDH+SGD	0.459	0.628

## API reference

### citk package

#### Submodules

#### citk.functions module

citk.functions.BellMembership (x: numpy.ndarray, c: numpy.ndarray, a: numpy.ndarray) → numpy.ndarray  
Bell Membership Function

Parameters:

- **x** (*np.ndarray*) – Input array.
- **c** (*np.ndarray*) – Centroid array.
- **a** (*np.ndarray*) – Bandwith.

Returns:  $1 / (1 + ((x - c)**2)/a**2)$ .

Return type: np.ndarray

citk.functions.GaussianMembership (x: numpy.ndarray, c: numpy.ndarray, a: numpy.ndarray) → numpy.ndarray  
Gaussian Membership Function

Parameters:

- **x** (*np.ndarray*) – Input array.
- **c** (*np.ndarray*) – Centroid array.
- **a** (*np.ndarray*) – Bandwith.

Returns:  $\exp(-((x - c)**2)/a**2)$ .

Return type: np.ndarray

citk.functions.GaussianRBF (x: numpy.ndarray, c: numpy.ndarray, r: numpy.ndarray) → numpy.ndarray  
Gaussian radial basis activation

Parameters:

- **x** (*np.ndarray*) – Input array.
- **c** (*np.ndarray*) – Centroid array.
- **r** (*np.ndarray*) – Standard deviation array.

Returns:  $res = res = np.exp(-||x-c||**2/(2*r**2))$ .

Return type: np.ndarray

citk.functions.Linear (x: numpy.ndarray) → numpy.ndarray  
Linear activation

Parameters: **x** (*np.ndarray*) – Input array.

Returns: Copy of input.

Return type: np.ndarray

ReLU : rectified linear unit activation function. Sigmoid : sigmoid activation function. Tanh : hyperbolic tangent activation function.

When scalar is passed, scalar is returned, so it is recommended to convert scalar into 1-d array instead.

```
>>> x = np.array([[1, -2], [-3, 4]])
>>> y = np.array([-3.])
>>> Linear(x)
array([[1, -2],
       [-3, 4]])
>>> Linear(y)
array([-3.])
```

citk.functions.LogSigmoid (x: numpy.ndarray) → numpy.ndarray

Natural log of sigmoid function.

citk.functions.Poly (x: numpy.ndarray, deg: int, type: Optional[str] = 'full')

citk.functions.ReLU (x: numpy.ndarray) → numpy.ndarray

Rectified Linear Unit (ReLU) activation

Parameters: **x** (*np.ndarray*) – Input array.

Returns: Array of element-wise maximum(0, x<sub>i</sub>) for all x<sub>i</sub> in a.

Return type: np.ndarray

Linear : linear activation function. Sigmoid : sigmoid activation function. Tanh : hyperbolic tangent activation function.

When scalar is passed, scalar is returned, so it is recommended to convert scalar into 1-d array instead.

```
>>> x = np.array([[1, -2], [-3, 4]])
>>> y = np.array([-3.])
>>> ReLU(x)
array([[1, 0],
       [0, 4]])
>>> ReLU(y)
array([0.])
```

citk.functions.Sigmoid (x: numpy.ndarray) → numpy.ndarray

Sigmoid activation

Parameters: **x** (*np.ndarray*) – Input array.

Returns: res = 1/(1+exp(-x<sub>i</sub>)) for x<sub>i</sub> in x.

Return type: np.ndarray

ReLU : rectified linear unit activation function. Linear : identity activation function. Tanh : hyperbolic tangent activation function.

When scalar is passed, scalar is returned, so it is recommended to convert scalar into 1-d array instead.

```
>>> x = np.array([[0, np.inf], [-np.inf, 0]])
>>> y = np.array([-0.])
>>> Sigmoid(x)
array([[0.5, 1.],
       [0., 0.5]])
>>> Sigmoid(y)
array([0.5])
```

citk.functions.Sum (x: numpy.ndarray) → numpy.ndarray

Basic sum along rows.

citk.functions.Tanh (x: numpy.ndarray) → numpy.ndarray

Hyperbolic tangent activation

Parameters: **x** (*np.ndarray*) – Input array.

Returns: res = (exp(x<sub>i</sub>)-exp(-x<sub>i</sub>))/(exp(x<sub>i</sub>)+exp(-x<sub>i</sub>)) for x<sub>i</sub> in x.

Return type: np.ndarray

ReLU : rectified linear unit activation function. Linear : identity activation function. Sigmoid : sigmoid activation function.

When scalar is passed, scalar is returned, so it is recommended to convert scalar into 1-d array instead.

```
>>> x = np.array([[0, np.inf], [-np.inf, 0]])
>>> y = np.array([-0.])
>>> Tanh(x)
array([[0., 1.],
       [-1., 0.]])
>>> Tanh(y)
array([0.5])
```

## citk.layer module

`class citk.layer.BaseLayer (nonlinearity: Callable[[Any], numpy.ndarray], *args, **kwargs)`

Bases: `object`

All custom layers should be inherited from this class.

Parameters: **parser** (*WeightsParser*) – Weights Parser

`build_weights_dict (*args)`

Builds Weight Dictionary

`forward (*args, **kwargs)`

Performs forward pass logic of layer

*property* parser

`class citk.layer.Conv2D (kernel_shape: Tuple[int], num_filters: int, mode: str, nonlinearity: Callable[[Any], numpy.ndarray], **kwargs)`

Bases: `citk.layer.BaseLayer`

Useful for image classification tasks.

`build_weights_dict (input_shape: Tuple[int]) → Union[int, Tuple[int]]`

Weights builder

Input\_shape: Input shape.

Returns: union object (number\_of\_weights, \_output\_shape)

Return type: union

`conv_output_shape (A, B)`

`forward (inputs: numpy.ndarray, param_vector: numpy.ndarray) → numpy.ndarray`

Forward pass method

Inputs: Input matrix.

Param\_vector: Vector of network's weights.

Returns: Result of convolution

Return type: np.ndarray

`class citk.layer.Dense (size: int, nonlinearity: Callable[[Any], numpy.ndarray], **kwargs)`

Bases: `citk.layer.BaseLayer`

The essential building block of an ANN.

`build_weights_dict (input_shape)`

Weights builder

Input\_shape: Input shape.

Returns: Union object (number\_of\_weights, \_output\_shape)

Return type: union

`forward (inputs, param_vector)`

Forward pass method

Inputs: Input matrix.

Param\_vector: Vector of network's weights.

Returns: Nonlinearity applied to matrix multiplication between weights and input

Return type: np.ndarray



```

class citk.layer.Fuzzify (num_rules: int, msf: Callable[[Any], numpy.ndarray], nonlinearity: Callable[[Any],
numpy.ndarray] = <function Linear>, **kwargs)
    Bases: citk.layer.BaseLayer
    Main block for ANFIS-type networks

    build_weights_dict (input_shape)
        Weights builder

        Input_shape: Input shape.

        Returns: Union object (number_of_weights, _output_shape)
        Return type: union

    forward (inputs, param_vector)
        Forward pass method

        Inputs: Input matrix.
        Param_vector: Vector of network's weights.

        Returns: Result of fuzzy-consequence
        Return type: np.ndarray

class citk.layer.FuzzyGMDHLayer (poli_type: str, nonlinearity: Callable[[Any], numpy.ndarray], msf: Callable[[Any],
numpy.ndarray], **kwargs)
    Bases: citk.layer.BaseLayer
    Building block of FGMDH pipeline. Here we combined GMDH functionality and embed it into TSK controller

    build_weights_dict (input_shape)
        Weights builder

        Input_shape: Input shape.

        Returns: Union object (number_of_weights, _output_shape)
        Return type: union

    forward (inputs, param_vector)
        Forward pass method

        Inputs: Input matrix.
        Param_vector: Vector of network's weights.

        Returns: Result of fuzzy-consequence over polynome of input
        Return type: np.ndarray

class citk.layer.GMDHDense (size, degree, nonlinearity: Callable[[Any], numpy.ndarray], **kwargs)
    Bases: citk.layer.BaseLayer

    build_weights_dict (input_shape)
        Builds Weight Dictionary

    static calc_input_shape (input_size: int, deg: int) → int

    forward (inputs, param_vector)
        Performs forward pass logic of layer

class citk.layer.GMDHLayer (poli_type: str, nonlinearity: Callable[[Any], numpy.ndarray], **kwargs)
    Bases: citk.layer.BaseLayer
    Building block of GMDH pipeline.

    build_weights_dict (input_shape)
        Weights builder

```

Input\_shape: Input shape.

Returns: Union object (number\_of\_weights, \_output\_shape)

Return type: union

forward (inputs, param\_vector)

Forward pass method

Inputs: Input matrix.

Param\_vector: Vector of network's weights.

Returns: Polynome of input.

Return type: np.ndarray

*class* citk.layer.LSTM (units, size, \*\*kwargs)

Bases: [citk.layer.BaseLayer](#)

build\_weights\_dict (input\_shape)

Weights builder

Input\_shape: Input shape.

Returns: Union object (number\_of\_weights, \_output\_shape)

Return type: union

forward (inputs, param\_vector)

Forward pass method

Inputs: Input matrix.

Param\_vector: Vector of network's weights.

Returns: Result of LSTM operations.

Return type: np.ndarray

*class* citk.layer.MaxPool (pool\_shape, nonlinearity: Callable[[Any], numpy.ndarray], \*\*kwargs)

Bases: [citk.layer.BaseLayer](#)

Max Pooling layer

build\_weights\_dict (input\_shape: Tuple[int])

Weights builder

Input\_shape: Input shape.

Returns: union object (number\_of\_weights, \_output\_shape)

Return type: union

forward (inputs: numpy.ndarray, param\_vector: numpy.ndarray)

Forward pass method

Inputs: Input matrix.

Param\_vector: Vector of network's weights. (ingored)

Returns: Result of pooling

Return type: np.ndarray

*class* citk.layer.RBFDense (hidden: int, out: int, \*\*kwargs)

Bases: [citk.layer.BaseLayer](#)

Building block of RBF-network

build\_weights\_dict (input\_shape)

Weights builder

Input\_shape: Input shape.

Returns: Union object (number\_of\_weights, \_output\_shape)

Return type: union

forward (inputs, param\_vector)

Forward pass method

Inputs: Input matrix.

Param\_vector: Vector of network's weights.

Returns: Nonlinearity applied to matrix multiplication between weights and input

Return type: np.ndarray

`class citk.layer.SimpleRNN (units, size, **kwargs)`

Bases: `citk.layer.BaseLayer`

build\_weights\_dict (input\_shape)

Weights builder

Input\_shape: Input shape.

Returns: Union object (number\_of\_weights, \_output\_shape)

Return type: union

forward (inputs, param\_vector)

Forward pass method

Inputs: Input matrix.

Param\_vector: Vector of network's weights.

Returns: Result of RNN operations.

Return type: np.ndarray

`class citk.layer.WeightsParser`

Bases: `object`

add\_weights (name: str, shape: Tuple[int])

Helper tool to add weights to ANN Layers

Parameters:

- **name** (*str*) – name of layer/weights set.
- **shape** (*tuple*) – shape of layer/weights set.

get (vect: numpy.ndarray, name: str)

Helper tool to parse weights from ANN Layers

Parameters:

- **vect** (*np.ndarray*) – vector of weights.
- **name** (*str*) – name of layer/weights set.

## `citk.losses module`

`citk.losses.Huber (y_true: numpy.ndarray, y_pred: numpy.ndarray, d: Optional[float] = 1.0) → float`

Huber Loss

`citk.losses.MAE (y_true: numpy.ndarray, y_pred: numpy.ndarray) → float`

Mean Average Loss

`citk.losses.MSE (y_true: numpy.ndarray, y_pred: numpy.ndarray) → float`

Mean Squared Loss

## citk.model module

```
class citk.model.FFN (input_shape: Tuple[int], layer_specs: List[citk.layer.BaseLayer], loss: Callable[[...],
numpy.ndarray], **kwargs)
```

Bases: `object`

```
eval (input: numpy.ndarray, output: numpy.ndarray) → float
```

Evaluate network on given input

Parameters: **inputs** (*np.ndarray*) – Input vector.

Output: Desired output

Returns: Loss value

Return type: float

```
fit (optimiser: citk.optimisers.BaseOptimizer, train_sample: Tuple[numpy.ndarray], validation_sample:
Tuple[numpy.ndarray], batch_size: int, epochs: Optional[int] = None, verbose: Optional[bool] = None,
load_best_model_on_end: bool = True, minimize_metric: bool = True)
```

Fit network on given input

Optimiser: Algorithm to use for minimizing loss.

Parameters:

- **train\_sample** (*tuple*) – Train pair (X, y).

- **validation\_sample** (*tuple*) – Validation pair (X, y).

Batch\_size: Batch size.

Epochs: Number of epochs.

Returns: Tuple (trained\_model, loss\_history)

Return type: union[`FFN`, dict]

```
frac_err (X, T)
```

```
loss (W_vect: numpy.ndarray, X: numpy.ndarray, y: numpy.ndarray, omit_reg: bool = False) → numpy.ndarray
```

Loss function constructor

W\_vect: Network weights vector.

X: Input vector.

Y: Desired network response.

Omit\_reg: Omit regularization flag. Default is False

```
predict (inputs: numpy.ndarray) → numpy.ndarray
```

Predict method

Parameters: **inputs** (*np.ndarray*) – Input vector.

Returns: Network response.

Return type: np.ndarray

## citk.optimisers module

```
class citk.optimisers.BaseOptimizer (*args, **kwargs)
```

Bases: `object`

All custom optimizers should inherit this class

```
apply (loss: Callable[[...], float], graph: List[citk.layer.BaseLayer])
```

Apply optimizer

```
class citk.optimisers.ConjugateSGDOptimizer (eta: float = 0.001, **kwargs)
```

Bases: `citk.optimisers.BaseOptimizer`

Conjugate Stochastic Gradient Descent Optimiser

`apply` (loss: Callable[[numpy.ndarray, numpy.ndarray, numpy.ndarray, Optional[Dict[str, float]]], float], input\_tensor: numpy.ndarray, output\_tensor: numpy.ndarray, W: numpy.ndarray, \*\*kwargs)

Perform one step of Conjugate SGD

Parameters:

- **loss** (*callable*) – Loss fitness function to minimize
- **input\_tensor** (*np.ndarray*) – Global input to FFN, i.e. your X variable
- **output\_tensor** (*np.ndarray*) – Desired FFN response, i.e. your Y variable
- **W** (*np.ndarray*) – Initial network weights

Returns: union (reached tolerance flag, corrected weights, loss value)

Return type: Union [np.ndarray, float]

`class citk.optimisers.GeneticAlgorithmOptimizer` (num\_population: int, k: int = 5, \*\*kwargs)

Bases: `citk.optimisers.BaseOptimizer`

Vanilla Genetic Algorithm.

`apply` (loss: Callable[[numpy.ndarray, numpy.ndarray, Optional[Dict[str, float]]], float], input\_tensor: numpy.ndarray, output\_tensor: numpy.ndarray, W: numpy.ndarray, \*\*kwargs)

Perform one step of GA

Parameters:

- **loss** (*callable*) – Inverse fitness function to minimize
- **input\_tensor** (*np.ndarray*) – Global input to FFN, i.e. your X variable
- **output\_tensor** (*np.ndarray*) – Desired FFN response, i.e. your Y variable
- **W** (*np.ndarray*) – Initial network weights

Returns: tuple (best individual so far, lowest loss so far)

Return type: Union[np.ndarray, float]

`static construct_genome` (W: numpy.ndarray, weight\_init: Callable[[...], numpy.ndarray])

Construct random population

Parameters:

- **layers\_list** (*list*) – Genotype, i.e. FFN template to mimic to.
- **weight\_init** (*callable*) – Weight distribution function.

Returns: Initalized weights.

Return type: np.ndarray

`static crossover` (ind\_1: numpy.ndarray, ind\_2: numpy.ndarray) → numpy.ndarray

Perform simple crossover

Parameters:

- **ind\_1** (*np.ndarray*) – FFN layers weights, first individual.
- **ind\_2** (*np.ndarray*) – FFN layers weights, second individual

Returns: Generated offsprings

Return type: np.ndarray

`static mutate` (ind: numpy.ndarray, mu: float = 0.1, sigma: float = 1.0, factor: float = 0.01) →

List[citk.layer.BaseLayer]

Perform simple mutation

Parameters:

- **ind** (*np.ndarray*) – FFN layers weights
- **mu** (*float*) – mean of distribution
- **sigma** (*float*) – scale of distribution
- **factor** (*float*) – scale factor of mutation

Returns: Generated individual

Return type: *np.ndarray*

`class citk.optimisers.SGDOptimizer (alpha: float = 0.0, eta: float = 0.001, **kwargs)`

Bases: `citk.optimisers.BaseOptimizer`

Stochastic Gradient Descent Optimizer

`apply (loss: Callable[[numpy.ndarray, numpy.ndarray, numpy.ndarray, Optional[Dict[str, float]]], float],  
input_tensor: numpy.ndarray, output_tensor: numpy.ndarray, W: numpy.ndarray, **kwargs)`

Perform one step of SGD

Parameters:

- **loss** (*callable*) – Loss fitness function to minimize
- **input\_tensor** (*np.ndarray*) – Global input to FFN, i.e. your X variable
- **output\_tensor** (*np.ndarray*) – Desired FFN response, i.e. your Y variable
- **W** (*np.ndarray*) – Initial network weights

Returns: tuple (reached tolerance flag, corrected weights, loss value)

Return type: Union [*np.ndarray*, float]

## `citk.utils module`

`citk.utils.concat_and_multiply (weights, *args)`

`citk.utils.gen_batch (dataset: Tuple[numpy.ndarray, numpy.ndarray], batch_size: int)`

`citk.utils.nCr (n, r)`

## Module contents

# Indices and tables

- `genindex`
- `search`

# Index

## A

`add_weights()` (citk.layer.WeightsParser method)  
`apply()` (citk.optimisers.BaseOptimizer method)  
(citk.optimisers.ConjugateSGDOptimizer method)  
(citk.optimisers.GeneticAlgorithmOptimizer method)  
(citk.optimisers.SGDOptimizer method)

## B

`BaseLayer` (class in citk.layer)  
`BaseOptimizer` (class in citk.optimisers)  
`BellMembership()` (in module citk.functions)  
`build_weights_dict()` (citk.layer.BaseLayer method)  
(citk.layer.Conv2D method)  
(citk.layer.Dense method)  
(citk.layer.Fuzzify method)  
(citk.layer.FuzzyGMDHLayer method)  
(citk.layer.GMDHDense method)  
(citk.layer.GMDHLayer method)  
(citk.layer.LSTM method)  
(citk.layer.MaxPool method)  
(citk.layer.RBFDense method)  
(citk.layer.SimpleRNN method)

## C

`calc_input_shape()` (citk.layer.GMDHDense static method)  
`citk`  
module  
`citk.functions`  
module  
`citk.layer`  
module  
`citk.losses`  
module  
`citk.model`  
module  
`citk.optimisers`  
module  
`citk.utils`  
module  
`concat_and_multiply()` (in module citk.utils)  
`ConjugateSGDOptimizer` (class in citk.optimisers)

`construct_genome()`  
(citk.optimisers.GeneticAlgorithmOptimizer static method)

`Conv2D` (class in citk.layer)  
`conv_output_shape()` (citk.layer.Conv2D method)  
`crossover()` (citk.optimisers.GeneticAlgorithmOptimizer static method)

## D

`Dense` (class in citk.layer)

## E

`eval()` (citk.model.FFN method)

## F

`FFN` (class in citk.model)  
`fit()` (citk.model.FFN method)  
`forward()` (citk.layer.BaseLayer method)  
(citk.layer.Conv2D method)  
(citk.layer.Dense method)  
(citk.layer.Fuzzify method)  
(citk.layer.FuzzyGMDHLayer method)  
(citk.layer.GMDHDense method)  
(citk.layer.GMDHLayer method)  
(citk.layer.LSTM method)  
(citk.layer.MaxPool method)  
(citk.layer.RBFDense method)  
(citk.layer.SimpleRNN method)  
`frac_err()` (citk.model.FFN method)  
`Fuzzify` (class in citk.layer)  
`FuzzyGMDHLayer` (class in citk.layer)

## G

`GaussianMembership()` (in module citk.functions)  
`GaussianRBF()` (in module citk.functions)  
`gen_batch()` (in module citk.utils)  
`GeneticAlgorithmOptimizer` (class in citk.optimisers)  
`get()` (citk.layer.WeightsParser method)  
`GMDHDense` (class in citk.layer)  
`GMDHLayer` (class in citk.layer)

## H

`Huber()` (in module citk.losses)

## L

[Linear\(\)](#) (in module [citk.functions](#))

[LogSigmoid\(\)](#) (in module [citk.functions](#))

[loss\(\)](#) ([citk.model.FFN](#) method)

[LSTM](#) (class in [citk.layer](#))

## M

[MAE\(\)](#) (in module [citk.losses](#))

[MaxPool](#) (class in [citk.layer](#))

[module](#)

[citk](#)

[citk.functions](#)

[citk.layer](#)

[citk.losses](#)

[citk.model](#)

[citk.optimisers](#)

[citk.utils](#)

[MSE\(\)](#) (in module [citk.losses](#))

[mutate\(\)](#) ([citk.optimisers.GeneticAlgorithmOptimizer](#)  
static method)

## N

[nCr\(\)](#) (in module [citk.utils](#))

## P

[parser\(\)](#) ([citk.layer.BaseLayer](#) property)

[Poly\(\)](#) (in module [citk.functions](#))

[predict\(\)](#) ([citk.model.FFN](#) method)

## R

[RBFDense](#) (class in [citk.layer](#))

[ReLU\(\)](#) (in module [citk.functions](#))

## S

[SGDOptimizer](#) (class in [citk.optimisers](#))

[Sigmoid\(\)](#) (in module [citk.functions](#))

[SimpleRNN](#) (class in [citk.layer](#))

[Sum\(\)](#) (in module [citk.functions](#))

## T

[Tanh\(\)](#) (in module [citk.functions](#))

## W

[WeightsParser](#) (class in [citk.layer](#))



# Python Module Index

## c

[citk](#)

[citk.functions](#)

[citk.layer](#)

[citk.losses](#)

[citk.model](#)

[citk.optimisers](#)

[citk.utils](#)