# CITK: Computational Intelligence Toolkit

## version 0.1b

Dmytro Androsov, Volodymyr Sydorskyy

27.12.2020

# Contents

# 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 | 0.281 | 0.279 |

# 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_i) for all x_i 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_i)) for x_i 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_i)-exp(-x_i))/(exp(x_i)+exp(-x_i)) for x_i 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 multiplicationbetween 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 multiplicationbetween 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 minimuzing 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

# Python Module Index

## c