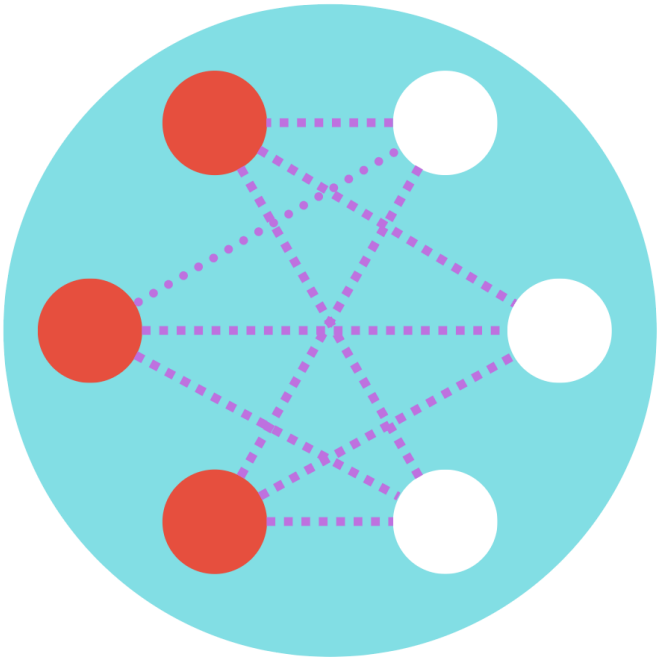


ANN4j

ANN4j - Artificial Neural Networks for Java

ANN4j is a java package that provides Object oriented Neural Networks for making *Explainable Networks*. Object Oriented Network structure is helpful for observing each and every element the model. This package is developed for XAI research and development.

READ COMPLETE DOCUMENTATION HERE

Table of Contents

About

ANN4j - Creating observable object-oriented neural networks for better Explainable AI.

ANN4j is a java package that provides object oriented functionality to neural networks. It implements multilayer perceptrons in java by using Objects instead of matrix multiplications. Every neuron is treated as a seperate object. While this kind of implementation is highly inefficiant when compared to matrix multiplications, this implementation will help research in the fields of Explainable AI. Explainable AI aims at making the model interpretable. By pausing and observing the neural net at different stages, researchers can study neural networks more efficiantly. Indivisual

observable interfaces are more easy to observe then matrices. Operations which are difficult to perform on matrices can be performed more easily using this technique.

Features

- Observable implementation for Artificial Neural Networks (ANN)
- XAI method for relevance propagation
- Stochastic/batch gradient descent
- No hardcoded implementations lets researchers change the parameters as they want.
- Plug and play mnist type data. Other Data files can be handeled via extension

Usage

Download

Releases

The package can be imported after download. `import ann4j.*;`

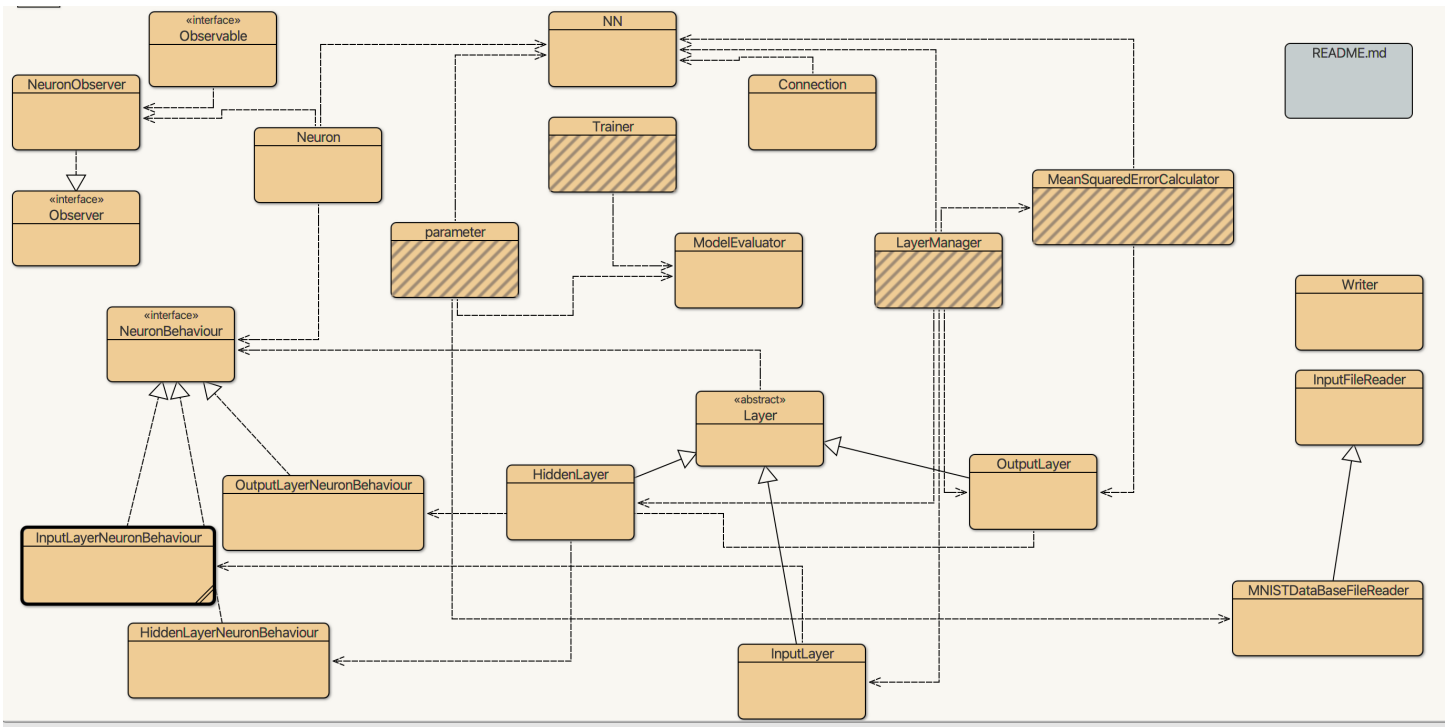
Requirements and Dependancies

None. This package is made using 100% Pure Java. The java package requires java 5.0+. No other requirements are required. Recommended to use the latest version of Java. Java download link

Design

Exaustive information can be found on our wiki

Class Diagram



Classes and Methods

Hierarchy For All Packages

Package Hierarchies:
ann4j

Class Hierarchy

- java.lang.**Object**
  - ann4j.**Connection**
  - ann4j.**HiddenLayerNeuronBehaviour** (implements ann4j.NeuronBehaviour)
  - ann4j.**InputFileReader**
    - ann4j.**MNISTDataBaseFileReader**
  - ann4j.**InputLayerNeuronBehaviour** (implements ann4j.NeuronBehaviour)
  - ann4j.**Layer**
    - ann4j.**HiddenLayer**
    - ann4j.**InputLayer**
    - ann4j.**OutputLayer**
  - ann4j.**LayerManager**
  - **Main**
  - ann4j.**MeanSquaredErrorCalculator**
  - ann4j.**ModelEvaluator**
  - ann4j.**Neuron** (implements ann4j.Observable)
  - ann4j.**NeuronObserver** (implements ann4j.Observer)
  - ann4j.**NN**
  - ann4j.**OutputLayerNeuronBehaviour** (implements ann4j.NeuronBehaviour)
  - ann4j.**parameter**
  - ann4j.**Trainer**
  - ann4j.**Writer**

Interface Hierarchy

- ann4j.**NeuronBehaviour**
- ann4j.**Observable**
- ann4j.**Observer**

Design patterns

- Strategy Pattern - NeuronBehaviour
- Observer pattern - NeuronObserver
- Template pattern - Trainer
- Singleton Pattern - NeuronBehaviour concrete classes

Training

Setting parameters

- Setting the output file to be output.txt and enabling command line logging.

```
parameter.setOutputFile("output.txt", true);
```

- Setting the number of neurons in each layer.

```
parameter.setLayerArray(784, 32, 16, 16, 26);
```

- Setting the training file to be emnist-letters-train.csv and the file type

```
parameter.setTrainingFileReader("emnist-letters-train.csv", "mnist");
```

- Setting the testing file

```
parameter.setTestingFileReader("emnist-letters-test.csv", "mnist");
```

- Setting the learning rate for weights

```
parameter.setLearningRate(1);
```

- Setting the learning rate for the bias to 1.

```
parameter.setBiasLearningRate(1);
```

- Setting the epsillion value for the relevance propagation algorithm.

```
parameter.setEpsillion(0);
```

- Setting the batch size

```
parameter.setBatchsize(10);
```

- Setting the rectification function.

```
parameter.setRectificationFunction("sigmoid");
```

Training the Model

- Creating a new instance of the Trainer class.

```
Trainer myTrainer = new Trainer();
```

- Training the network with 88800 samples for n epochs

```
myTrainer.train(m, n);
```

- Creating a new instance of the NeuronObserver class this class will observe the neurons and respond when every parameter is changed.

```
NeuronObserver myNeuronObserver = new NeuronObserver();
```

- Testing the network with 9990 samples.

```
myTrainer.test(9990);
```

- Adding the neuron at layer 1 and index 31 to be observed.

```
myNeuronObserver.addNeuronToBeObserved(1, 31);
```

Evaluating the model

- Training accuracy

```
myTrainer.getModelEvaluator().getTrainingAccuracy();
```

- Testing accuracy

```
myTrainer.getModelEvaluator().getTrainingAccuracy();
```

- Confusion Matrix

```
myTrainer.getModelEvaluator().printConfusionMatrix();
```

XAI

- xai algorithm for relevance propagation.

```
myTrainer.relevancePropagate(2, 3);
```

- xai algotithm for most significant input neurons

```
myTrainer.forwardPropagatewithExclusionInputLayerOnKSamples(2);
```

Observable methods

In ANN4j, every neuron is an object of its own. Every Neuron can be observed by the NeuronObserver class when the values are updated. NeuronObserver class can be extended as per the requirement of the parameters to be observed. Neurons objects can also be obtined and observed independantly.

- Get a neuron object from a layer.

```
myTrainer.getLayerManager().getLayer(layerNum).getNeuron(neuronNum));
```

- Get activation of a neuron

```
neuron.getActivation();
```

- Get bias of the neuron

```
neuron.getBias();
```

- Get arraylist of the left or right connections of the neuron

```
neuron.leftConnections;
neuron.rightConnections;
```

- Get weight of a connection

```
connection.getWeight();
```

Output

Output can be seen in the file specified by the parameter class. The write class can be used to write any user defined Strings to the file.

```
Writer.write();
```

This is an example of the output generated by training and observing a neuron.

```
Training accuracy in epoch 0 is 10.66891891891892
Testing accuracy 28.92892892892893
The neuron 31 in layer 1 has been updated by forward propagation
Neuron #31 has activation 6.567825572210979E-4
```

```
The neuron 31 in layer 1 has been updated by forward propagation
Neuron #31 has activation 0.003181628304117291

Testing accuracy 28.93314651721377
The neuron 0 in layer 2 has been updated by forward propagation
Neuron #0 has activation 0.25373727956231534

The neuron 0 in layer 2 has been updated by forward propagation
Neuron #0 has activation 0.7220061457959416

Testing accuracy 28.94736842105263
```

Data format

Default format

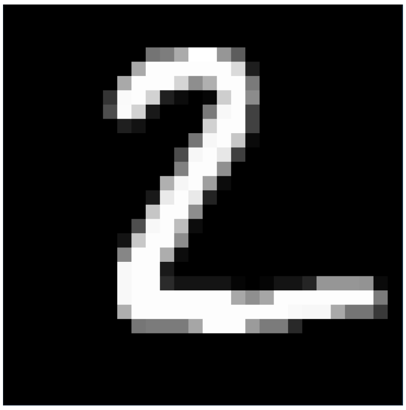
The default format for the package is MNIST format.

File type CSV consisting of the following

- 1 Label (Expected number)
- n pixel weights n must match number of input neurons.

Example

```
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```



Here the image 2 is represented as an array of 28*28 pixels each value represents pixel activation.

Some datasets to test the package on (without extending mnist file reader)

1. MNIST Handwritten database
2. MNIST extended chracter database and Kaggle link
3. MNIST fashion data set
4. Kannada MNIST

References

- About MNIST database
- Preparing MNIST data

File rendering for other formats

ANN4j provides functionality to extend the InputFileReader to add file handling for various types of datasets apart from mnist type files. InputFileReader or MNISTFileReader can be extended by making relevant changes in file reading functions.

Constructor

The new file reader class must pass the filename to the super constructor.

```
super(filename);
```

Getting input

The `next()` method is responsible for reading new line input from the dataset. It must also act as a super setter method. It must set all values like `label` , `expectedOutputArray` and `inputArray`

```
public void next()
```

Getting label

This method must return the label (expected value of prediction).

```
public double getLabel()
```

Getting Input Neuron values

This method must return the values of neurons (expected value of prediction). Example for digit recognition fo digit two, the arraylist must contain 784 elements of the pixel values.

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
public ArrayList<Double> getInputArray()
```

Getting Output Neuron values

This method must return the expected values of output neurons (expected value of prediction). Example for digit recognition fo digit two, the arraylist can be `0,0,1,0,0,0,0,0,0,0`

This is dependant on the model and is a design decision.

```
public ArrayList<Double> getExpectedOutputArray()
```

Getting prediction from the output neurons

This method is used for obtaining the prediction value from the activations in the output neurons. In the digit recognitoin case as every input is mapped with same neuron it is the same. For example if neuron number 3 fires the highest, the model has predicted 3. But this needs to be overridden for different model configurations.

```
public double getPredictionFromNeuronNum(int mostSignificantNeuronNumAsPrediction)
```

Note-

Predicted neuron and prediction are different.

Predicted neuron is the neuron which is most significant in firing. The prediction is the value corresponding to that neuron.

Example Consider case of handwritten letters database. If the neuron 4 is most significant (glows brightest) and it corresponds to label D then the predicted neuron is 4 and prediction is D.

`getMostSignificantNeuronNumAsPrediction()` is a method in LayerManager class which helps to get the value of the neuron which fires the most.

Restarting the file

Creates a new instance of the file reader and starts all over again.

```
public void restart()
```

Setting the file reader

After the file reader custom class had been made, it can be passed to the parameter class using the methods

```
public static void setTrainingFileReader(InputFileReader inputFileReader);
public static void setTestingFileReader(InputFileReader inputFileReader){
```

Documentation

Please visit the documentation wiki page https://aatmaj-zephyr.github.io/ANN4jwiki/

Examples

- Example code https://github.com/Aatmaj-Zephyr/ANN4j/blob/main/Main.java
- Sample output https://github.com/Aatmaj-Zephyr/ANN4j/blob/3721148ec24371bf095e1394fe39fc471f391466/output.txt
- Sample output https://github.com/Aatmaj-Zephyr/ANN4j/blob/ef0f34b505e6e6316f94b5a660b9ef651582667d/output.txt

Other resources

- More about Artificial Neural Networks https://www.3blue1brown.com/topics/neural-networks
- Relevance propagation example https://towardsdatascience.com/indepth-layer-wise-relevance-propagation-340f95deb1ea
- Rectification functions https://www.quora.com/What-is-the-purpose-of-rectifier-functions-in-neural-networks

ANN4j Community

Raising an issue

Please feel free to suggest any changes or point out any errors by raising an issue here

Asking for help

For asking for clarification on any topic, raise an question issue here

Community

- Discussions
- Wiki
- Documentation

Contributing

Please read the contributing guidelines here. Everyone is free to contribute to this project.

Help spread the word

Are you using ANN4j in your research or project? If so, please let me know and I may add a link to your project or application and your logo to this repository. Also please consider starring this repository and following me.

Citing this package for research work

You can cite this repository using the following bibtex entry. Please update the date.

```
@misc{AatmajZephyr21:online,
author = {Aatmaj Mhatre},
title = {Aatmaj-Zephyr/ANN4j: Artificial Neural Networks for Java This package provide
howpublished = {\url{https://github.com/Aatmaj-Zephyr/ANN4j}},
month = {},
year = {},
note = {(Accessed on <date>)}
}
```

License

License notice