Install and Use Instructions

How to install

1. You will need Java JDK 7+ and sbt 0.13.13 or newer.

   OpenJDK – for Linux:

   http://openjdk.java.net/install/

   Java (Oracle) – for any platform (Linux also):

   http://www.oracle.com/technetwork/java/javase/downloads/index.html

   sbt:

   http://www.scala-sbt.org/0.13/docs/Installing-sbt-on-Linux.html

   http://www.scala-sbt.org/0.13/docs/Installing-sbt-on-Windows.html

   The MSI installer is no longer supported, as far as I know, so you have to download and unpack the zip file.

   http://www.scala-sbt.org/0.13/docs/Installing-sbt-on-Mac.html

2. You can check if **sbt** works simply typing "sbt" in the command line:

```
bash-4.3# sbt
> sbt-version
[info] 0.13.8
```

3. Download the ANNA project, either through GIT:

```
git clone https://github.com/makingthematrix/ann.git
```

   or by downloading the zip file:

   https://github.com/makingthematrix/ann/archive/SOSWithBlock_1.0.zip

   The current stable branch is SOSWithBlock_1.0 .

4. Go to the main project directory ("ann") and type:

```
sbt compile
```

   sbt will download and install Scala and Akka if you haven't done it before. Be patient.

Then type:

```
sbt console
```

You will see the welcome screen.

How to use

You can use pre-constructed networks:

1. **sos**
2. **dotAndLine**

and utility methods for network construction:

3. **delayGate(delay: Int)**
4. **signalSum(requestedSum: Int)**

All four give you an object of the type NetData which you can then pass to the setup method:

```
> setup(sos)
> setup(dotAndLine)
> setup(delayGate(2))
> setup(signalSum(3))
```

This will build the network and display the list of neuron ids.

The setup method works only for these four networks. If you want to build one of your own, please look into the code to learn how to use **NetBuilder** and then call **initializeNetwork(netData: NetData)**.

Please note that the setup method adds triggers to output neurons (different for each network). If you build a network by hand, you have to do this by hand as well.

The output neurons for the networks are, respectively:

1. **sos**: S2 (sends 'S' to the output) and O2 (sends 'O')
2. **dotAndLine**: DOT3 (sends '.') and LINE2 (sends '-')
3. **delayGate(_)**: DG3 (sends '1')
4. **signalSum(_)**: SS2 (sends '1')

You can also print the JSON form of each network, e.g.:

```
> print(delayGate(2))
```

After the setup, you can send input vectors to the network and see how it handles them. Since all four networks have only one input neuron, you can use a utility method and send the whole input vectors' sequence in form of a string, eg.:

```
> send("100010001000110011001100100010001000")
```

The string will be parsed into signals which will be sent to the input neuron and from there to other neurons in the network. Then you can see the output by typing:

```
> output
```

If you want to send longer input sequences, please modify the maximum number of iterations:

```
> setMaxIterations(_)
```

The variable was introduced in order to prevent infinite processing which is possible with customized networks. The default number is 100. You can check it by typing:

```
> maxIterations
```

During the processing the console will display logs from the neurons, about what signals they received and sent, their internal state, etc. You can control logs from which neurons you want to see, by typing:

```
> LOG.allow(neuronId: String)
> LOG.removedAllowedId(neuronId: String)
> LOG.clearAllowedIds()
```

Do that between the setup and the send methods. The next setup will reset the list.

Maciej Gorywoda (gorywodamaciej@gmail.com)

November 12, 2016