Virtual Material

This notebook shows how to use the virtual material through Nascence API. It is using the java wrapper that is suited to access the VM.

```
Needs["JLink`"];
```

Java wrapper wrapper :)

The VM is expected to be running at the given machine and port. Path is the path to the compiled VM classes.

```
vmConnect[host_, port_, path_] :=
 Module[{vmJars = Join[FileNames[path <> "/dist/*"], FileNames[path <> "/lib/*"]]},
  javaBin = "/usr/lib/jvm/java-6-sun/bin/java";
  ReinstallJava[CommandLine = javaBin];
  AddToClassPath /@ vmJars;
  JavaNew["nascence.vm.io.MathClient", host, port]
 ]

vmDisconnect[vmClient_] := vmClient@closeConnection[]
```

Example usage

First, import this library :

In[115]:=
```
Import["vmWrapper1.m"]
```

Start the server first

```
vmHost = "localhost";
vmPort = 9090;

vmPath = "git/NASCENCE/vm";

vmClient = vmConnect[vmHost, vmPort, vmPath]
```

```
« JavaObject[nascence.vm.io.MathClient] »
```

Check who we are talking to (this is also test if the board / VM works). VM outputs st. like `Virtual Material - Elman Recurrent Neural Network`:

```
vmClient@getMotherboardID[]
```

```
Virtual Material - Elman Recurrent Neural Network
```

Try a simple experiment with a VM generated at random (8 inputs, 32 hidden nodes, 8 outputs, max weights of 5.0, 0.9 connection probability, no CW-RNN, no recurrency (feedforward MLP only)):

```
vmClient@programmeVarElmanRandom[8, 32, 8, 5.0, 0.9, False, False]
```

Now test the material generating input data (list of input lists), let' s use the last (7 th pin - VM uses 0 - based indexing) pin as the VM output and evaluate the VM. The wrapper internally uses real numbers. One has to specify the amplitude, which is multiplied by the data in order to server the VM integer values via the nascence API. The output is, again, scaled by the amplitude value. Setting up amplitude to 255 will make values between 0 and 1 to have 255 distinct values. Here is the 10 input vectors of length 7.

```
MatrixForm[Round[data = RandomReal[{0, 1}, {10, 7}], 0.01]]
```

$$\begin{pmatrix} 0.05 & 0.76 & 0.21 & 0.24 & 0.65 & 0.56 & 0.07 \\ 0.82 & 1. & 0.93 & 0.71 & 1. & 0.7 & 0.36 \\ 0.8 & 0.85 & 0.26 & 0.27 & 0.59 & 0.65 & 0.57 \\ 0.47 & 0.45 & 0.7 & 0.98 & 0.81 & 0.38 & 0.4 \\ 0.98 & 0.86 & 0.36 & 0.21 & 0.23 & 0.43 & 0.45 \\ 0.93 & 0.7 & 0.01 & 0.83 & 0.52 & 0.71 & 0.69 \\ 0.76 & 0.14 & 0.05 & 0.95 & 0.86 & 0.74 & 0.37 \\ 0.63 & 0.32 & 0.38 & 0.76 & 0.27 & 0.87 & 0.94 \\ 0.99 & 0.97 & 0.93 & 0.75 & 0.62 & 0.49 & 0.29 \\ 0.84 & 0.97 & 0.91 & 0.32 & 0.26 & 0.1 & 0.47 \end{pmatrix}$$

Now specify the input pins :

```
inputPins = Range[0, 6]
amplitude = 255;
```

```
{0, 1, 2, 3, 4, 5, 6}
```

and evaluate the VM, the result should be a 2D array again, that contains vectors of length 1 only:

```
MatrixForm[
 Round[result = vmClient@evaluateArray[data, amplitude, inputPins], 0.001]]
```

$$\begin{pmatrix} 0.98 \\ 1. \\ 0.996 \\ 0.996 \\ 1. \\ 1. \\ 0.996 \\ 0.996 \\ 1. \\ 1. \end{pmatrix}$$

```
vmDisconnect[vmClient]
```