NOTE ON INITIALIZATION OF WEIGHTS AND BIAS WHEN TRAINING NEURAL NETWORKS OF ANY TYPE

In Matlab for training NN the weights and bias are initialized randomly, except for the binary perceptron they are initialized at 0. This means that in each running the starting weights and bias are quite different.

For example to generate 4 random numbers in [0 1] two times

```
>> x=rand(1,4)

x =    0.6324    0.0975    0.2785    0.5469

>> y=rand(1,4)

y =    0.9575    0.9649    0.1576    0.9706
```

And y is very different from x.

If we want to study experimentally the influence for example of the number of layers, or the number of neurons per layer, or training function, etc., it should be convenient to initialize the weighs and bias in the same values in order to eliminate the influence of different starting points that may counterbalance the influence of what we are researching.

To make that happen, the function rng (random number generator) can be used. This function fixes the seed of the random number generator algorithm. When Matlab starts the seed is 0, but the seed changes in each run of rand.

For example

```
>> rng(0)                                    makes the seed 0

>> x=rand(1,4)

x =    0.8147    0.9058    0.1270    0.9134

>> rng(0)                                    makes again the seed 0

>> y=rand(1,4)

y =    0.8147    0.9058    0.1270    0.9134
```

And y is equal to x.

The same happens with rands that generates in [- 1 1]

```
>> rng(0)

>> x=rands(1,4)

x =    0.6294    0.8116   -0.7460    0.8268

>> rng(0)

>> y=rands(1,4)

y =    0.6294    0.8116   -0.7460    0.8268
```

And x is equal to y.

When we are comparing different NNs with respect to other factors than weights and bias, it can be convenient to have the same seed in consecutive train calls. For that it is enough to write

```
rng(0)
```

before calling the train function, for example

```
% train with the same initialization of weights and bias

rng(0);

[NET,TR] = train(NET,X,T);

% every time train is executed, the weights and bias are initialized in the same values.
```

This applies also to trainNetwork in deep learning:

```
rng(0);

net = trainNetwork(XTrain, YTrain, layers, options);
```

Note however than when the network is fixed, and we want to find the best local minimum, then rng should not be used. Different initializations will converge to different local minima, leading eventually to better results.

For more >>help rng

ADC.30Oct2020