NetSim®

Accelerate Network R & D

Clustering in WSN using Self-Organizing Map Neural Network

Version 15.0

A Network Simulation & Emulation Software

By

TETCOS LLP

Contents

Software: NetSim Standard v15.0, Visual Studio 2022, MATLAB
Note: Deep Learning Toolbox is mandatory.

Project download link:

https://github.com/NetSim-TETCOS/SOM_Clustering_in_WSN_v15.0/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and setup the Project in NetSim:

https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects

1 Objective

The goal of this project is to maximize the lifetime of a Wireless Sensor Network using Self-Organizing Map (SOM) based Neural Network algorithms for cluster head selection.

2 Introduction

We define the lifetime of a WSN as the time at which the power of half the sensors reaches zero (also called half-life of the network). Initially, all sensors start with a fixed amount of energy. Subsequently, energy is consumed during transmission, reception, and idle states. Packets are transmitted from sensors to their cluster head sensor and then forwarded to the sink node through other cluster heads. The selection of the cluster heads is done using SOM. All MAC/PHY layer simulations are carried out using NetSim while the cluster head selection using the SOM algorithm is done using MATLAB.

3 Self-Organizing Map based Neural Network

We use a 2-Dimensional SOM to get a k sized cluster from n sensors located in 2D space, using distance as a metric for clustering.

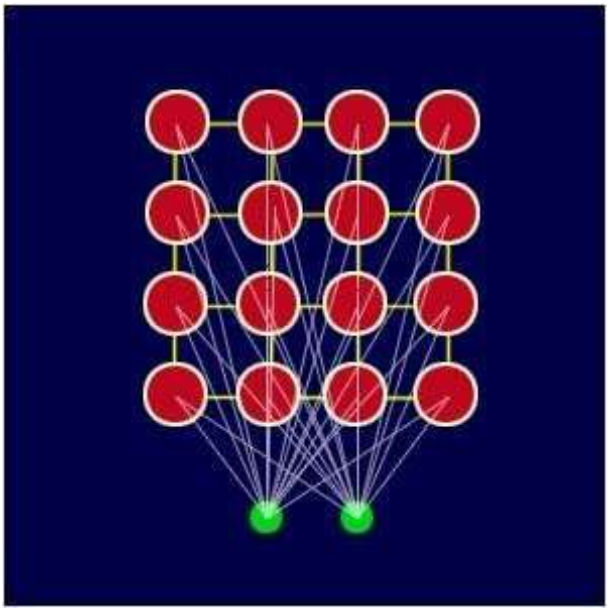

Figure 3-1: *Neural network schematic with k 2D lattice points, where red nodes represent lattice points, green neurons denote the input layer, and connections between them illustrate the links.*

As shown in the figure above, a neural network is created from k 2D lattice points (also known as nodes), each of which is connected with the input layer. Each link has an associated weight. As the input vectors are 2D points here, there are 2 neurons in the input layer of the neural network. Each node has a topological position (x coordinate and y coordinate) and also a weight vector of 2 dimensions (one weight for each dimension).

So, with input vectors and weight vectors, the SOM algorithm explained below orders the weight vectors in a way that represents similarities with input vectors.

The algorithm consists of the following steps:

1. Each node's weight is randomly initialized.
2. Choose an input vector and find the node whose weight vector is closest to the chosen point. The most common method to calculate distance is finding the Euclidean distance. This node is called the BMU (Best Matching Unit).
3. The neighborhood of the BMU is defined as all the nodes lying within its radius of influence. The number of neighbors decreases over time because the radius of influence decreases over time.
4. The weight vector associated with the neighbor node (and BMU too) is updated using the following equation:

$$iw(q) = iw(q-1) + \alpha(p(q) - iw(q-1))$$

Where $p(q)$ is the input vector chosen, $iw(q-1)$ is the weight vector associated with node i before the update, and $iw(q)$ is the updated value of the weight vector.

5. Repeat from step 2 until the iteration limit has been reached.

The above procedure is repeated for a large number of iterations (chosen as 200 in our example).

There are k output nodes in the neural network, where each output node is associated with some pattern or cluster in the input points.

Each point is passed through the network, and if the i^{th} output node has the highest value, then this point belongs to cluster i.

The topology function of the k nodes and the distance function used to evaluate the distance between a sensor and a node can be chosen from a given set of values, as described below.

3.1 Topology

The neurons in the layer of an SOM are arranged originally in physical positions according to a topology function. The functions `gridtop`, `hextop`, or `randtop` can arrange the neurons in a grid, hexagonal, or random topology.

3.1.1 Grid Topology (`gridtop`)

The `gridtop` topology starts with neurons in a rectangular grid of dimensions which you may specify.

Suppose you want to classify n points into k clusters. Then you can start with k neurons arranged in a rectangular grid of dimensions $[k_1\ k_2]$ such that $k_1 \times k_2 = k$.

For example: `pos = gridtop([2, 3])` gives

$$\texttt{pos} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 2 & 2 \end{pmatrix}$$

Suppose you had chosen dimensions to be [3, 2], then you would get the following configuration of neurons: `pos = gridtop([3, 2])` gives

$$\texttt{pos} = \begin{pmatrix} 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

3.1.2 Hexagonal Topology (`hextop`)

In `hextop` topology, neurons are initially arranged in a hexagonal pattern.

For example, a 2-by-3 pattern of `hextop` neurons is generated as follows: `pos = hextop([2, 3])` gives

$$\texttt{pos} = \begin{pmatrix} 0 & 1.0000 & 0.5000 & 1.5000 & 0 & 1.0000 \\ 0 & 0 & 0.8660 & 0.8660 & 1.7321 & 1.7321 \end{pmatrix}$$

`hextop` is the default pattern for SOM networks generated by `selforgmap`.

3.1.3 Random Topology (`randtop`)

The `randtop` function creates neurons in a random pattern in the specified dimensions.

For example: `pos = randtop([2, 3])` gives

$$\texttt{pos} = \begin{pmatrix} 0 & 0.42 & 0.29 & 0.87 & 0.07 & 0.43 \\ 0 & 0.01 & 0.26 & 0.48 & 1.32 & 1.33 \end{pmatrix}$$

3.2 Distance Functions

Distances between neurons are calculated from their positions with a distance function. There are four distance functions:

- `dist` — the Euclidean distance from a neuron to a point.
- `boxdist`
- `linkdist` — the number of links, or steps, that must be taken to get to the neuron under consideration.
- `mandist` — calculates the Manhattan distance between points.

3.3 Creating a Self-Organizing Map Neural Network (`selforgmap`)

SOM is created using the `selforgmap` function, whose syntax is as given below.

```
selforgmap(dimensions, coversteps, initNeighbour,
    topologyFunction, distanceFunction)
```

Where the parameters can take the following values:

1. `dimensions` is a row vector of dimension sizes of the initial neurons. Default value = [8 8].
2. `coversteps` is the number of training steps to cover the whole input dataset initially (default = 100).
3. `initNeighbour` is the size of the initial neighbourhood (default = 3).
4. `topologyFunction` is the initial topology of neurons (default = `'hextop'`).
5. `distanceFunction` is the neuron distance function (default = `'linkdist'`).

Suppose you want to cluster n points located in 2D space into k clusters based on Euclidean distance. Let x be a matrix with dimension $2 \times n$ which contains the coordinates of the points.

```
net = selforgmap([2 k/2], 100, 3, 'gridtop', 'dist');
```

You can set the number of iterations the neural network will train using

```
net.trainParam.epochs = 1000;
```

The network is trained using `train(network, dataset)` as

```
net = train(net, x);
```

To get the cluster id of the points by passing them as input to the learnt neural network,

```
y = net(x);
```

y is a $4 \times n$ matrix. The i^{th} column of y is the output for the i^{th} point, and all the entries in the column are zero except one, which is the cluster to which that point belongs – more precisely, the node which is the cluster head of the i^{th} point.

To get the cluster id in the range $(1, k)$,

```
IDX = vec2ind(y);
```

Where IDX is a vector of length n.

Now we have to get the geometrical centroid of each cluster, which can be obtained by iterating through all the points that belong to that cluster and finding the mean of their position vectors.

On running the above code, a GUI Deep Learning Toolbox window appears, showing several visualizations of the network that is learnt, such as SOM Topology, SOM Neighbour Connections, SOM Neighbour Distances, SOM Input Planes, SOM Sample Hits, and SOM Weight Positions.

4 Interfacing WSN Simulation in NetSim with SOM Algorithm running in MATLAB

SOM based clustering is implemented in NetSim by interfacing with MATLAB for the purpose of running the SOM algorithm. The sensor coordinates are fed as input to MATLAB, and the Self-Organizing Map neural network algorithm implemented in MATLAB is used to dynamically perform clustering of the sensors into n number of clusters.

In addition to clustering, we also determine the cluster head of each cluster mathematically in MATLAB. The distance of each sensor from the centroid of the cluster to which it belongs is calculated. Then the sensor which has the least distance is elected as the cluster head.

From MATLAB we get the cluster id of each sensor, the cluster heads of each cluster, and the size of each cluster.

All the above steps are performed periodically, as defined by the implementation. Each time, the cluster members and the cluster heads are determined based on the current position, and they are not fixed.

The code required for the mathematical calculations done in MATLAB is written to a `som_optimization.m` file, available in the `MATLAB` folder under `bin_x64` of `SOM_Optimization_Workspace`.

A `SOM_Clustering.c` file is added to the DSR project, which contains the following functions:

- `fn_NetSim_som_clustering_CheckDestination()` — Used to determine whether the current device is the destination.
- `fn_NetSim_som_clustering_GetNextHop()` — Statically defines the routes within the cluster and from the cluster to the sink node. Returns the next hop based on the static routing that is defined.
- `fn_NetSim_som_clustering_IdentifyCluster()` — Returns the cluster id of the cluster to which a sensor belongs.
- `fn_NetSim_som_clustering_run()` — Calls the MATLAB interfacing function and passes the inputs from NetSim, i.e. the sensor coordinates, number of clusters, and sensor count.
- `fn_netsim_som_form_clusters()` — Assigns each sensor to its respective cluster based on the cluster ids obtained from MATLAB.
- `fn_netsim_assign_cluster_heads()` — Assigns the cluster heads for each cluster based

on the cluster head ids obtained from MATLAB.

- `fn_NetSim_som_Clustering_Init()` — Initializes all parameter values.

4.1 Static Routing

Static routing is defined in such a way that the sensors in the cluster send packets to the cluster head. The cluster head then directly sends the packets to the destination (sink node).

- If the current sensor is the source device and is not a cluster head, its next hop is its cluster head.
- If the current sensor is the source device and is a cluster head, its next hop is the destination (the sink node).
- If the current sensor is not the source, the packet is sent to the destination (the sink node).

5 Steps to run SOM Clustering Code in NetSim

1. Import the workspace `SOM_Optimization_Workspace`.
2. Add the following MATLAB install directory path to the Environment PATH variable:

`<MATLAB_INSTALL_DIRECTORY>\bin\win64`

For example: `C:\Program Files\MATLAB\R2021b\bin\win64`

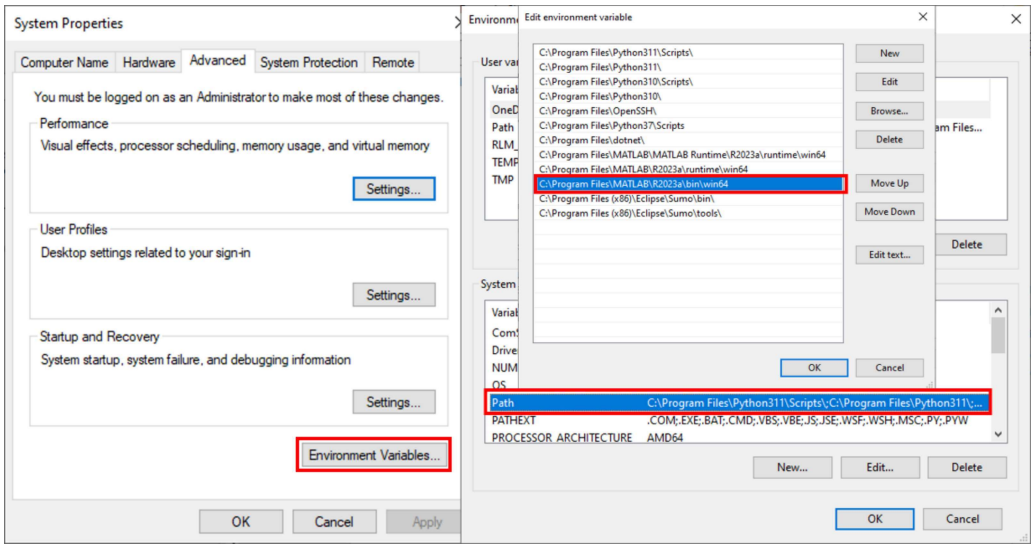

Figure 5-1: *Environment variable PATH.*

If the machine has more than one MATLAB version installed, the directory for the target platform must be ahead of any other MATLAB directory (for instance, when compiling a 64-bit application, the directory of the MATLAB 64-bit installation must be first on the PATH).

3. Open Command Prompt as administrator and execute the command `matlab -regserver`. This registers MATLAB as a COM automation server and is required for NetSim to start the MATLAB automation server during runtime.

4. Open the source code in Visual Studio by going to **Your Work** → **Source Code** and clicking the **Open Code** button on the NetSim Home Screen window.

5. Under the DSR project in Solution Explorer, you will find the `SOM_Clustering.c` file, which contains source code related to interactions with MATLAB and handling clustering in NetSim.

6 Example

1. Run NetSim in administrative mode.

2. `SOM_Optimization_Workspace` comes with a sample configuration that is already saved. To open this example, go to **Your Work** and click on **Som__clustering__Example**.

3. The saved network scenario consists of 64 sensors uniformly placed in the grid environment, along with a sink node, forming a Wireless Sensor Network. Traffic is configured from each sensor node to the sink node.

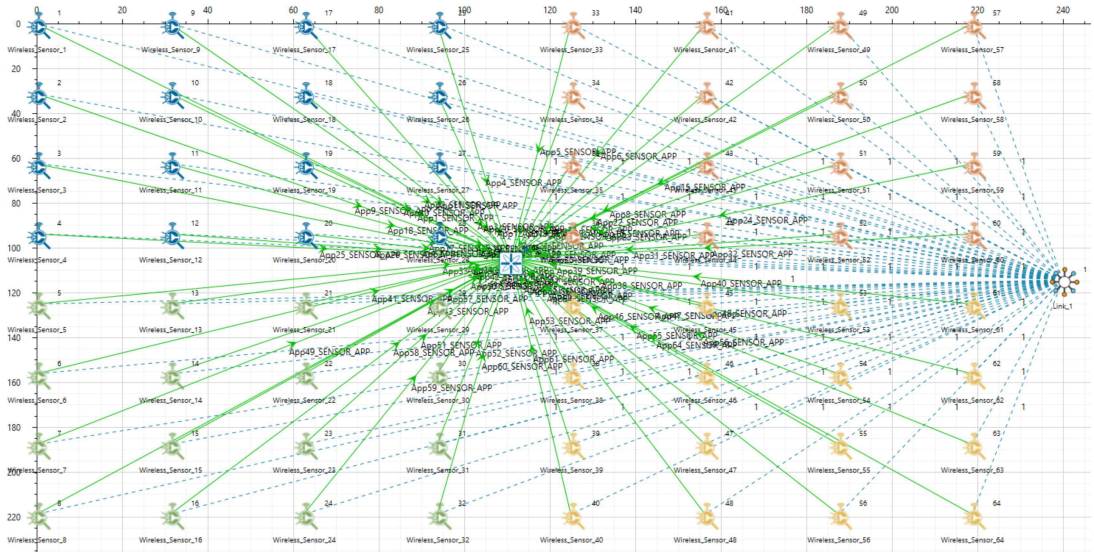

Figure 6-1: *Network topology.*

4. Run the simulation and press any key to continue.

5. This opens the `MatlabInterface.exe` console window. You will observe that as the simulation starts in NetSim, MATLAB gets initialized, and a graph associated with energy consumption in the sensor network is plotted during runtime.

6. There are two algorithms implemented to find the best clusters and cluster heads: one uses SOM with distance as the metric, and the other is a modified version of the first algorithm, where a function of both remaining power and distance from the cluster head is minimized over all the sensors in the cluster, to get the cluster head with the least distance from the geometrical centroid of the cluster and the maximum remaining power.

