Self Organizing Maps

1. Neural Networks

A neural network contains a number of nodes (called units or neurons) connected by edges. Each link has a numerical weight associated with it. The weights can be compared to a long-term memory and actually the learning process for neural networks is to find (compute) these weights so that the network presents best results suited to training data. Some units have direct links with the environment and can be considered as input or output units. Other units that have no direct link with the outside are considered "hidden" units. Network weights are modified during the learning process so as to produce a network behavior (relations between inputs and outputs) as closely as the training data are.

Each unit has a set of inputs from other units, a set of outputs that are linked to other units, a threshold for activation and a method for computing the activation level of the unit for the next step based on the inputs and current weights. The idea is that each unit makes a calculation based on data received from local inputs (which can be network inputs or outputs from other units), without having an overview of what needs to be learned. In practice, most neural networks are software implemented for updating synchronous all network units at a fixed time sequence.

2. Neural networks architecture

Artificial Neural Network can be viewed as weighted directed graphs in which artificial neurons are nodes and directed edges (with weights) are connections between neurons. Based on the connections patterns (architecture), artificial neural networks ca be grouped into two categories: ***feed-forward*** networks (in which graphs have no loops – no connection from output neurons to inputs neurons) and ***recurrent*** networks (in which loops occurs because of the feedback connections).

In the most common family of feed-forward networks, called multilayer perceptron, the neurons are organized into layers that have unidirectional connections between them.

Different connectivity cause different networks behaviors. Generally speaking, ***feed-forward*** networks are *static*, that is, because they produce only one set of output values rather then a sequence of values from a given input. Feed-forward networks are memory-less in the sense that their response to an input is independent of the previous network state. ***Recurrent***, or feedback, networks, on the other hand, are dynamic systems. When a new input pattern is presented, the neurons outputs are computed. Because of the feedback paths, the inputs to each neuron are then modified, which leads the network to enter a new state.

3. Learning Rules

The ability to learn is a fundamental characteristic of intelligence. Although a precise definition of learning is difficult to formulate, a learning process in the artificial neural networks (ANN) context can be viewed as the problem of updating the network architecture and connection weights so that the network can efficiently perform a specific task. The network usually must learn the connection weights from available training patterns. Performance is improved over time by iteratively updating the network weights. ANN's ability to automatically learn from examples makes them attractive and exciting. Instead of following a set of rules specified by human experts, ANNs appear to learn basic rules (like input-output relations) from the given collection of representative examples. This is one of the major advantages of neural networks over traditional expert systems.

To understand or design a learning process, you must first have a model of the environment in which a neural network operates, that is, you must know what information is available to the network. We refer to this model as a learning paradigm. Second, you must understand how network weights are updated, that means, which learning rules govern the updating process. A learning algorithm refers to a procedure in which learning rules are used for adjusting the weights.

There are two main learning paradigms:

- ***supervised*** learning, or learning with a "teacher", the network is provided with a correct answer (output) for every input pattern. Weights are determined to allow the network to produce answers as close as possible to the known correct answers.
- ***unsupervised*** learning, or learning without a "teacher", does not require a correct answer associated with each input pattern in the training data set. It explores the underlying structure in the data, or correlations between patterns in the data, and organizes patterns into categories from these correlations.

There are four basic types of learning rules:

- **Competitive learning rule** - learning units compete among themselves for activation. As a result, only one output unit is active at any given time for a given example. This phenomenon is known as *"Winner Takes All"*. Competitive learning often clusters or categorizes the input data. Similar patterns are grouped by the network and represented by a single unit. This grouping is done automatically based on data correlations.
- **Error correction rules** – in the supervised learning paradigm, the network is given a desired

output for each input pattern. During the learning process, the actual output generated by the network may not be equal to the desired output. The basic principle of error correction learning rules is to use the error signal to modify the connection weights to gradually reduce this error.

- **BOLTZMANN learning** - Boltzmann machines are symmetric recurrent networks consisting of binary units (+1 for “on” and -1 for “off”). Boltzmann learning can be viewed as a special case of error-correction learning in which the error is measured not as the direct difference between desired and actual outputs, but as the difference between correlations among the outputs of two neurons under clamped and free running operating conditions.
- **HEBBIAN rule** - the oldest learning rule is Hebb's postulate of learning occurred in 1949 based on the following observation from neurobiological experiments: *"If neurons on both sides of synapse are activated synchronously and repeatedly, the synapse’s strength is selectively increased”*. An important property of this rule is that learning is done locally, that means, the changing for the synapse weight depends only on the activities of the two neurons connected by it.

4. Online / Offline Learning

Based on the updating time of the network weights, ANNs learn can be grouped into two categories: online learning and offline learning:

- **Offline learning** - for each input vector the changes of the synaptic coefficients are computed. These changes are applied to the network only after all input vectors from the training data were presented.
- **Online learning** – for each input vector the coefficients changes are computed and these changes are applied immediately to the network. In comparison with first method, this method is generally faster and can simple leave form some minimal local values for the error function.

5. Competitive Learning Networks

The simplest competitive learning network consists of a single layer of outputs units. Each output unit in the network is connected to all the input units via weights. Each output unit also is connected to all other output units via weights. As a result of competition, only the unit with the highest (or the smallest) net input become the winner.

In many neural networks the “winner take all” process is necessary to select, during learning, the neuron with maximum activation.

6. Self-Organizing Maps - SOM

This network, known also as the Kohonen network, after the professor who developed it (Theo Kohonen), is the most popular network for unsupervised learning and it is a special type of competitive learning network. In this network the input / output units are disposed into a grid (usually rectangular) in the data space (two dimensional spaces for our application for example) and the coefficients for connections (weights) between units from the output layer depend on "the distance" between them. Thus, if units that are considered closer have a positive influence and the units that are considered distant have an inhibitory influence. For each unit is defined a spatial neighborhood of it. The shape of the local neighborhood can vary based on the input data and can be square, rectangular or circular. During the competitive learning step all the weights vectors associated with the winner unit and its neighboring units are updated. This mode of updating encourages neighboring units to respond similarly with the winner unit for the same input vector. Because T. Kohonen has worked to develop the theory of competition, the competitive processing elements (network units) are often referred as Kohonen units.

The self organizing map (SOM) network architecture consists of a two-dimensional array of units, each connected to all input units. Let $\boldsymbol{w_{ij}}$ denote the *n*-dimensional vector associated with the unit at location *(i, j)* – in the 2D array (matrix). Each neuron computes the Euclidean distance between the current input vector *x* and all the stored weight vector $\boldsymbol{w_{ij}}$. For the winner neuron (and also for neurons in its vicinity) all weights are updated according to the following formula (Kohonen's formula). The weight is updated after each input vector and thus this network has an online learning:

$$\vec{w}_i(t+1) = \begin{cases} \vec{w}_i(t) + \alpha(t)(\vec{x} - \vec{w}_i(t)) & \text{for } i \in neighborhood \\ w_i(t) & otherwise \end{cases} \quad (1)$$

where

$w_i(t+1)$ represents the new weight for the neuron (to the next step)

$w_i(t)$ - represents the weight of neuron for the current step

x – represents the input vector

$\alpha(t)$ – represents the Learning rate

The idea of this algorithm is to seek the "winner" unit for each input vector, and the modification for the synaptic coefficients is done for the winner unit and also (typically with a lower factor α) for all units from the neighborhood of the winner unit. This method of modification for coefficients encourage neighboring units to respond similarly to the same input vector, the network is like a "map" of an input dataset.

6.1 Kohonen learning algorithm

Learning idea is to place the network somewhere in the data space and start to train it, with a large neighborhood and a large learning coefficient, that gradually decrease during learning, as follows:

1. Initialize weights to small random numbers; in our case I recommend that the output units to be placed as a grid in a two dimensional space. It is recommended to store these units in a matrix in order to easy compute the neighborhood of a neuron. In SOM network the weights of the output neurons are considered to be the position of the neuron in the input space. For a better view of the network "position" it is recommended that first time to place neurons symmetrically distributed throughout our data space representation and instead of displaying the neurons from the network to display on the screen only the edges between neurons from the first vicinity level – only the vertical and horizontal edges for each neuron;
2. Set the initial learning rate and neighborhood. Also specify a formula for computing the neighborhood and the learning rate (the formula needs to depend on the stage reached in learning);
3. Take an input vector (sample), and evaluate the distance between itself and all (units) neurons from the network;
4. The neuron that is closest to the current sample is considered to be the "winner" neuron (as in formula 2) updates its weights according to equation 1;

$$\left\|\mathbf{x}-\mathbf{w}_{\mathbf{m,n}}\right\| = \min_{i,j}\left\|\mathbf{x}-\mathbf{w}_{\mathbf{ij}}\right\| \qquad (2)$$

5. For all neurons found in the neighborhood of the winner neuron also update all their weights according to the Kohonen equation – (equation 1);
6. Repeat the steps 3-5 for all input vectors from the data set (in our case from all point from file);
7. Decrease the value of α (learning rate) and shrink the neighborhood;
8. Repeat steps 3 through 7 until the learning rate is less than a pre-specified threshold or a maximum number of iterations are reached.

In the SOM network the neighborhood plays an essential role. If neurons are disposed on a square grid and stored in a 2 dimensional array the neighborhood of a neuron is defined as all neighbor neurons based on the index in 2D array (matrix). For a winner neuron from the *(i, j)* position in the matrix (*i* and *j* are indexes in matrix) and *v* is the current value of neighborhood, the neighbors of the winner neuron are all neurons that have the index in the domain [i-*v*, i+*v*] and [j-*v*, j+*v*].

There are several formulas for computing the neighborhood and the learning rate at a age *t*, which strongly depends on the application. An example of such formula for the current application would be:

$$Neighbor(t) = 6.1 * e^{-\frac{t}{N}}$$

$$\alpha(t) = 0{,}1 * e^{-\frac{t}{N}}$$

where *t* is the current age (stage), N – total proposed number of iterations

Note:

- It is considered that an age (stage) was passed when all the input vectors from the dataset are processed (all point from our file).
- For the coefficient of Neighborhood is chosen a value 6.1 because we want to start with a large neighborhood at the beginning (recommended neighborhood be about 60-70% from the all neurons). The constant 6.1 was chosen for a network of 10x10 neurons.
- The parameter for learning rate (α) was choose the constant 0.1 because we want that the learning rate be small in order to descend quickly to 0.

Problem. The request for this lab is to implement the SOM algorithm for automatic finding the groups of points from the file that was generated in the first lab (file with points). It will serve as a clustering algorithm (unsupervised grouping of input data) and will show on screen both entry points and the current positions of neurons from the network, in order to have a visual view of “quality of learn”.

For a better view of the network "position" it is recommended that instead of displaying the neurons from the network to display on the screen only the edges between neurons on the first vicinity level (ie for the neuron from the ***(i, j)*** position will show its connections with neurons of the positions ***(i-1, j) (i +1, j) (i, j-1), (i, j+1)*** – only if these connections exist).