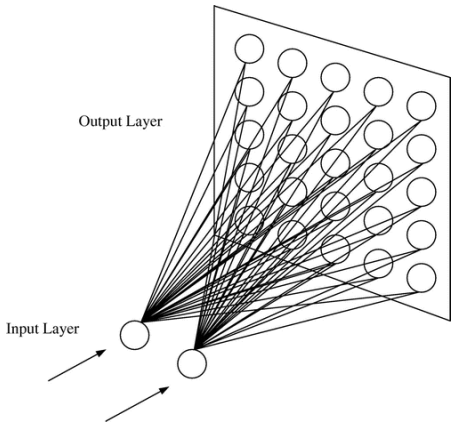
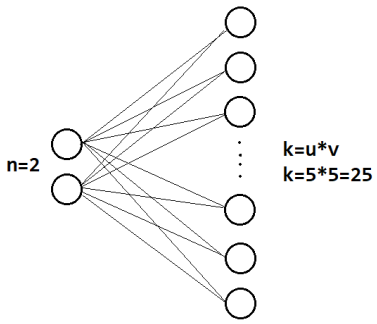
SOM map algorithm – for dummies

This is a brief overview of the self organizing map algorithm. The layout is:

One input layer of width *n*, where each cell is connected to an output matrix u*v. Each cell of the matrix has an *n* dimensional vector of weights in it:



The output matrix can be represented as a standard neural network where the output layer has width *k*:



The positions of the cells of the matrix are bound to a 2D grid independent from the input and weight dimensionality. **The positions NEVER change !**

Input

- A set of *n* dimensional vectors and a corresponding category to each vector

Output

- An output vector k where each node has a vector of weights *w*, which is *n* items wide

Helper calculations

- During the calculation we need the exponential decay function for the learning rate α and the neighbor radius λ:

$$\alpha_t = \alpha_i \left(\frac{\alpha_f}{\alpha_i}\right)^{\frac{t}{t_{max}}}$$

$$\lambda_t = \lambda_i \left(\frac{\lambda_f}{\lambda_i}\right)^{\frac{t}{t_{max}}}$$

variable indexes:

t – current iteration

t_max – all iterations count

i – initial value on the beginning of training

f – final desired value after training

- fixed number of epochs: t_{max}
  - i.e. time is $t \in \{1, 2, \ldots, t_{max}\}$
- for first epoch, parameter value is $\alpha_1 = \alpha_s$ (start)
- for last epoch, parameter value is $\alpha_{t_{max}} = \alpha_f$ (finish)
- geometric schedule: parameter value for epoch t is:

$$\alpha_t = \alpha_s \cdot \left(\frac{\alpha_f}{\alpha_s}\right)^{\frac{t-1}{t_{max}-1}} \qquad \lambda_t = \lambda_s \cdot \left(\frac{\lambda_f}{\lambda_s}\right)^{\frac{t-1}{t_{max}-1}}$$

Algorithm

1. Initialize all output neuron weight vectors with random small numbers

2. Select a random input vector (tip – iterate trough the input dataset in a shuffled way)

3. Find the output neuron k_i, which has the weights with the smallest Euclidean distance i^* to the input neuron x **in the n dimensional space**

$$i^* = \arg\min ||w_i - x||$$

4. Update the weight of the found neuron and all of it's neighbors based on the diameter function. The diameter function has the parameter j^* which represents the Euclidean distance in the **2D GRID !!! NOT THE n-Dimensional space**

Diameter function:

$$h = \exp(-\frac{(j^*)^2}{\lambda^2})$$

Weight update

$$w_i = w_i + \alpha . h . (x - w_i)$$

5. Update alpha and lambda based on the exponential decay function and repeat from point 2.

Calculate the error each iteration !

Quantization error- average distances of BMU's and their points

Avg adjustment – the delta of weight change of the BMU – weights before and after adjustment

quantization error = average distance of data point x_i to it's best matching neuron c_j:

$$E = \frac{1}{n} \sum_i min_j \|x_i - c_j\|$$

average amount of adjustment of a neurons at a time t – let $\Delta c_j(t) = c_j(t) - c_j(t-1)$:

$$A(t) = \frac{1}{k} \sum_j \|\Delta c_j(t)\|$$