Multilayer Self Organising Map for Clustering of Atomistic Samples

Aquistapace F.

Facultad de Ciencias Exactas y Naturales, UNCuyo

February 21, 2022

Introduction

This software implements a multilayer architecture of Self Organising Maps (also known as a Kohonen Networks) for clustering of atomistic samples through unsupervised learning. It is written in Python 3.8.8 and depends on the external packages NumPy (version 1.20.1) [1] and Pandas (version 1.2.4) [2]. The core of the algorithm is a neural network composed of an input layer and an output layer, where the number of output neurons determines the number of groups the atoms are going to be classified into. Other software that implements the Kohonen Network can be found on the work by J. Troncoso on applying a SOM for cluster analysis and lattice defects detection [3]. Although no previous implementations of a multilayer architecture of SOMs for defects detection has been found in the literature.

Description

The `input_params.py` file contains all the information needed to initialize the algorithm. The software requests the user a training file, and a list of files to be analysed after the training process is completed. Furthermore, the files fed to the algorithm must be dump files such as those generated by the software LAMMPS [4], or at least have a line with the following format:

```
ITEM: ATOMS id x y z feature_1 feature_2 ...
```

Although the amount of layers and neurons is determined by the user, the specific architecture of the algorithm is determined by the training process, since not every neuron of a given layer will be used. Once the first layer has been trained, the algorithm assigns a SOM in the second layer to every active neuron in the first layer. This process is repeated for subsequent layers.

Layers

The layers of the architecture have to be independently specified, and each layer requires the following parameters:

- `features`: Per-atom features that are going to be used for the analysis.
- `scaling`: Individual scaling method for each feature.
- `f`: Fraction of the input data to be used when training the network, must be between 0 and 1.
- `SIGMA`: Maximum value of the σ function, present in the neighbourhood function.
- `ETA`: Maximum value of the η funtion, which acts as the learning rate of the network.
- `N`: Number of output neurons of the SOM, this is the number of groups the algorithm will use when classifying the atoms in the sample.

- `Batched`: Whether to use batched or serial learning for the training process.
- `batch_size`: In case the training is performed with batched learning.

Training

The training process of each SOM of the architecture can be reduced to the following steps for serial learning:

1. Initialize the weights of the SOM to random values between 0 and 1.
2. Normalize and shuffle the training data.
3. Select a point from the training data.
4. Find the neuron closest to the selected point, this neuron is referred to as the Best Matching Unit (BMU). The distance between a given point and all the neurons in the SOM is obtained efficiently through Eq. 1:

$$D^2 = X \cdot \vec{x}^T - 2(\vec{x} \cdot W)^T + diag(W^T W)^T \tag{1}$$

   Where $\vec{x}$ is the data point, W is the weight matrix of the SOM, and X is a matrix defined as $X_{mj} = (\vec{x})_j, \forall m \in \{1, ..., N\}$ (N being the number of output neurons of the SOM). The superscript T refers to the transpose of a matrix. For $N = 2$ the distances for each neuron are estimated independently, since this is more efficient than Eq. 1 for that case.

5. Update weights of the SOM through Eq. 2:

$$w_{mn}^t = w_{mn}^t + \eta(t) h_m(t)(x_n - w_{mn}^t) \tag{2}$$

   Where w_{mn}^t is the n-th component of the weight of the m-th neuron at iteration t. Also, $\eta(t)$ is the learning rate and $h_m(t)$ is the neighbourhood function for the m-th neuron, both expressed as functions of the iteration number t. Although a variety of functions can be used for $\eta(t)$ and $h(t)$, this software implements those given by Eqs. 3 and 4:

$$\eta(t) = \frac{\eta_0}{t}, t \geq 1 \tag{3}$$

$$h_m(t) = \exp\left(\frac{|\vec{w}_m - \vec{w}_{BMU}|}{\sigma(t)^2}\right) \tag{4}$$

   Where η_0 is the initial value of $\eta(t)$, $\vec{w}_m$ and $\vec{w}_{BMU}$ are the weights of the m-th neuron and the BMU respectively, and $\sigma(t)$ is a function given by Eq. 5:

$$\sigma(t) = \frac{\sigma_0}{t}, t \geq 1 \tag{5}$$

   With σ_0 being the initial value of $\sigma(t)$.

6. Repeat steps 3, 4 and 5 for every point in the training data.

For the case of batched learning, the training process of each SOM is given by the following steps:

1. Initialize the weights of the SOM to random values between 0 and 1.
2. Normalize and shuffle the training data.
3. Select a batch of data of size B from the training data.
4. Find the BMU for every point in the batch through Eq. 1.
5. Update weights of the SOM through Eq. 6:

$$w_{mn}^{t} = w_{mn}^{t} + \eta(t)(H_{mn}(t) - w_{mn}^{t}) \tag{6}$$

with $H_{mn}(t)$ defined by equation 7:

$$H_{mn}(t) = \frac{\sum_{i=1}^{B} h_m(b_i, t) x_{i,n}}{\sum_{i=1}^{B} h_m(b_i, t)} \tag{7}$$

Where b_i is the winning neuron associated with the data point $\vec{x}_i$, and $h_m(b_i, t)$ is just the neighbourhood function for the m-th neuron, with respect to b_i. The definition of $H_{mn}(t)$ was inspired by the work of Matsushita and Nishio [5].

6. Repeat steps 3, 4 and 5 for every batch in the training data.

Before being fed to a SOM, the data is normalized using one of three available methods for every feature. These methods are called 'normal', 'standard' and 'robust', and are described respectively by Eqs. 8, 9 and 10:

$$C_{norm} = \frac{C - C_{min}^{train}}{C_{max}^{train} - C_{min}^{train}} \tag{8}$$

where C_{norm} is the normalized data, C is the original data and C_{min}^{train} and C_{max}^{train} are the minimum and maximum values in the training data, respectively;

$$C_{stand} = \frac{C - \bar{C}^{train}}{\sigma_{C^{train}}} \tag{9}$$

where C_{stand} is the standardized data, $\bar{C}^{train}$ is the mean of the training data and $\sigma_{C^{train}}$ is the standard deviation of the training data;

$$C_{robust} = \frac{C - C_{median}^{train}}{IQR} \tag{10}$$

where C_{robust} is the robust normalized data, C_{median}^{train} is the median of the training data and IQR is the Inter-Quartile Range, given by the difference between the 75% and the 25% quartiles.

It is important to note that the training data of a given SOM is the data for which the associated neuron in the previous layer is the BMU, and that this data can be reduced by means of the `f` parameter of the SOM's layer.

Classification and Output

After every layer has been trained, the algorithm proceeds to classify the atoms in every file requested. The classification process for a SOM simply consists on finding the BMU for every point in the data. Notice that for every layer but the first, the information fed to the SOM is the collection of points for which the associated neuron in the previous layer is the BMU. Furthermore, the data is passed through the same preprocessing as in the training stage, with the relevant parameters of each normalization method already determined by the training data.

The results found by each layer are saved and mapped according to one of two available mappings, 'godel' and 'linear'. The 'godel' mapping is based on the encoding process developed by Kurt Gödel, that maps a set of n integer numbers $\{x_1, x_2, ..., x_n\}$ to a single unique integer G defined by Eq. 11:

$$G = \prod_{i=1}^{n} p_i^{x_i} \tag{11}$$

where p_i is the i-th primer number, with $p_1 = 2$. Based on this, for a given layer l and a given data point, the mapping can be applied by building a set with the predictions of the previous layers for said point, $\{x_1, ..., x_l\}$ and returns the corresponding integer as the result of the classification for layer l. To do this efficiently, the algorithm simply takes the mapped result of the point for layer $l-1$ and multiplies it by $p_l^{x_l}$. This mapping method is only recomended for small architectures, that have few layers and few neurons per layer, and for cases when the results need to be in integer form for a particular reason.

On the other hand, the 'linear' method is based on a linear interpolation of the previous layer, to assign the results of the current layer. This process is described by Eq. 12:

$$L = A_x + |B_x - A_x| \cdot \frac{x_{max}}{x_{max} + 1} \cdot \frac{x - x_{min}}{x_{max} - x_{min}} \tag{12}$$

where x is the obtained result, L is the mapped result for the current layer, A_x is the mapped result for the previous layer for x, B_x is the immediately superior mapped result of the previous layer with respect to A_x (or immediately inferior result in case A_x is the maximum value for the previous layer), and x_{max} and x_{min} are the maximum and minimum values obtained for the current layer (before mapping) respectively. When this method is selected, the results of the first layer are simply mapped to integer values starting from zero.

The format of the output files matches the format of the input file, and the name of each output file is simply the name of its corresponding input file with the prefix `SOM_` added at the beginning.

References

[1] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. *Nature 585*, 357–362 (2020). DOI: 0.1038/s41586-020-2649-2.

[2] McKinney, W. Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference, Volume 445* (2010).

[3] Troncoso, J. F. ClasSOMfier: A neural network for cluster analysis and detection of lattice defects. *arXiv e-prints,* (2020).

[4] Plimpton, S. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics, 117*(1), 1-19 (1995).

[5] Matsushita, H., Nishio, Y. Batch-learning self-organizing map with weighted connections avoiding false-neighbor effects. *The 2010 international joint conference on neural networks (IJCNN) (pp. 1-6). IEEE*, (2010).