Using a Self-Organizing Map to Cluster World Development Indicators Data

Yuan Shen Li and Il Shan Ng, May 28, 2017

1 Introduction

We implement a self-organizing map to conduct clustering on a World Development Indicators data set. Our goal is to group countries based on their similarities across a diverse set of development indicators in the hope of (i) finding a more meaningful way to classify countries rather than using the oversimplified labels of developing or developed, or first, second or third-world, and (ii) learning more about the self-organizing map as a powerful solution to unsupervised learning problems. In Section 2, we describe the basic training algorithm for the self-organizing map. We then consider extensions of this basic algorithm in Section 3 and explain how they might improve the performance of the self-organizing map. In Section 4, we describe the data as well as any pre-processing steps that preceded the clustering. We then present the results of our analysis and assess any gains from implementing the extensions in Section 5, before concluding with suggested avenues for further work on our analysis in Section 6.

2 Training Algorithm

2.1 Map architecture

In its most abstract form, a self-organizing map (SOM) is an artificial neural network that performs dimensionality reduction by finding a non-linear transformation from a feature space of arbitrary dimension to a discrete 2D topological space. The output of interest from this algorithm is a 2D map that can be visualized to identify clusters within the data.

Figure 1 shows the basic architecture of an SOM. It comprises two layers of neurons: an input layer and an output layer. The input layer consists of as many neurons as there are features in the data set, and so represents an F-dimensional space, where F is the number of features. The output layer has a user-specified size $J \times K$, and represents the 2D topological space that we are interested in. Each output neuron is fully connected to all input neurons. The strength of these connections is measured by connection weights known as *models*, which parameterize the non-linear transformation from the feature to the topological space. During training, each output neuron is selectively tuned to certain patterns in the input data. Over many iterations, the output layer adopts a conformation in the feature space that minimizes the distance (in the traditional sense) between the input data points and the output neurons, while preserving topological ordering. A useful analogy for the output layer is that of a fishing net which tries to envelop clusters but whose links between neurons cannot be severed or interchanged. The rest of this section details the four main steps of the training algorithm and explains how the final 2D map is created.

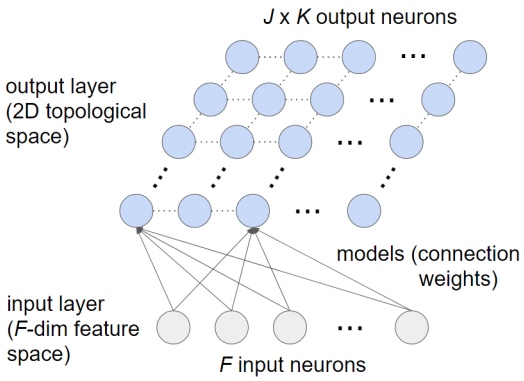


Figure 1: Self-organizing map architecture

2.2 Initialization step

As with most machine learning algorithms, SOM training begins with an initialization step. The simplest initialization method involves setting the models (connection weights) to small random values, though we consider smarter initialization methods in Section 3. Let m_{jkf} be the jk^{th} output neuron's model for the f^{th} feature of the data set. Conceptually, the non-linear transformation can be represented by a 3D $J \times K \times F$ matrix $\mathbf{M}$. In our implementation, however, we store these models in a 2D array and reshape the rows of this array to obtain $\mathbf{M}$.

2.3 Competitive step

Unlike traditional neural networks which use error-correction learning (backpropagation and gradient descent), an SOM applies competitive learning in which output neurons compete among themselves to respond to certain patterns in the input layer. We can model this competitive behavior by using a *discriminant function* to determine which neuron "wins" each data point. Specifically, given an F-dimensional data point $\vec{x_i}$, each neuron computes its discriminant

$$d_{jk}(\vec{x_i}) = \sum_{f=1}^{F}(x_{if} - m_{jkf})$$

and the neuron with the lowest discriminant "wins" the data point. In other words, the neuron whose model comes closest to the input vector (measured by distance in the traditional sense) gets activated and learns the input pattern. By repeatedly feeding in the full data set one point at a time, we can transform the continuous feature space of arbitrary dimension into a discrete 2D map consisting of neurons that themselves store continuous values.

2.4 Cooperative step

In addition to the winning neuron, neighboring neurons also get activated and learn the input pattern in proportion to their proximity to the winning neuron. To model this cooperative behavior, we define a Gaussian *topological neighborhood*

$$T_j = \exp\left[\frac{-D_j^2}{2\sigma_t^2}\right],$$

where D_j is the topological distance between neuron j and the winning neuron obtained by treating the output layer as a grid of cell size one. This topological neighborhood has several important properties: it is symmetric around and achieves its maximum on the winning neuron, and tails off monotonically as D_j increases. The size of the neighborhood is controlled by a time-dependent parameter σ_t that should be decreased over time to ensure convergence of the training algorithm. Kohonen (2013) recommends that σ_t be initialized to $\sigma_0 = \frac{1}{2}\max(J, K)$ i.e. half of the larger dimension of the map, and we follow Bullinaria (2004) by using an exponential decay $\sigma_t = \sigma_0 \exp[-t/\tau_\sigma]$ where τ_σ is the characteristic decay time. Kohonen (2013) suggests specifying a floor for σ_t that equals half the distance between any two adjacent nodes, which in our grid of cell size one, equals $\frac{1}{2}$. This floor guards against the complete elimination of any cooperative behavior. The chosen value of τ_σ influences the performance of the SOM greatly, and so we experiment with and present results for different values of τ_σ as part of parameter fine-tuning.

2.5 Adaptive step

After determining the winning neuron and weighting the amount that neighboring neurons learn by their proximity to the winning neuron, we now specify how this learning occurs in the form of a parameter update. Given an F-dimensional data point $\vec{x_i}$, we update the model of each output neuron using the equation

$$\Delta m_{jkf} = \alpha_t \cdot T_j \cdot (x_{if} - m_{jkf}), \tag{1}$$

where α_t is a time-dependent learning rate. Essentially, for each model, we are weighting the difference between the input vector and the current value of the model by the learning rate and the proximity of the neuron to the winning neuron, and applying this change to the current model values. Hence, a neuron learns more of the input pattern (i.e. experiences a larger model update) if it is closer to the winning neuron, and if its model values differ from the input vector by a lot. In practice, we anneal the learning rate α_t to prevent the models from thrashing about too wildly after many iterations so as to achieve convergence. Following Bullinaria (2004), we use an exponential decay $\alpha_t = \alpha_0 \exp[-t/\tau_\alpha]$, where $\alpha_0 = 1$ is the initial learning rate and τ_α is the characteristic decay time. Like τ_σ, we fine-tune the chosen value of τ_α to ensure convergence, but do not present results for this fine-tuning due to space constraints.

2.6 Convergence

Training the SOM involves looping through the competitive, cooperative and adaptive steps for each data point, wrapping around the data set for as many iterations is necessary for convergence. Although it is possible to implement a criterion for convergence based on how small the parameter updates have become, we choose instead to run the training algorithm for a fixed number of iterations. Doing so has two benefits: firstly, we avoid premature termination of the algorithm that may occur because the updates according to Equation (1) do not monotonically decrease in size over time; secondly, by fixing the number of iterations across all experiments, we can determine if certain extensions are successful in preventing the models from thrashing too much, which would improve convergence time.

2.7 Visualizing the output

We visualize the output of an SOM using a U-matrix. A U-matrix can be thought of as a heat map of the topological space, with the y- and x-axes corresponding to neuron location along the J and K dimensions respectively. Each cell of the U-matrix is assigned a color intensity value that is proportional to the average feature space distance of that particular neuron to its four adjacent neighbors (or two/three neighbors for corner/edge neurons). Darker cells in the U-matrix correspond to neurons which are more isolated in feature space. Patches of light cells, on the other hand, indicate a cluster of neurons which are close to one another in feature space.

Hence, to pick out clusters in our SOM, we simply visually identify light patches in the U-matrix surrounded by darker borders. To help with this, when presenting our results, we overlay the the identities of the input vectors at the location of their respective *best matching units*, i.e. the neuron which wins the input vector after convergence.

3 Extensions

3.1 Linear initialization

The random initialization method described in Section 2.1 may lead to slow convergence or even worse, cause the algorithm to converge to local minima (Akinduko and Mirkes, 2012). We follow Kohonen (2001) and try out *linear initialization*, which involves finding the first two principal component vectors of the data set and initializing the models in a linear fashion along the subspace spanned by these two vectors (i.e. taking a linear combination of these two vectors). The idea behind this technique is to spread out the initial models along the two basis vectors that account for the largest proportion of the total observed variance in the data set. Implementation-wise, we first create a covariance matrix from the data set and perform eigenvalue decomposition on this matrix. We then select the two vectors with the largest eigenvalues and compute linear combinations of these vectors.

3.2 Step neighborhood function

Tan et al. (2006) suggests an alternative non-Gaussian neighborhood function: a step function that allows all neurons within a specified radius of the winning neuron to learn *as much as* the winning neuron, and prevents all neurons outside of this neighborhood from learning anything. Let r be this radius. Then, the step function applied to the j^{th} neuron is

$$T_j = \begin{cases} 1 & \text{if } D_j \leq r \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$

where D_j is the topological distance between the j^{th} neuron and the winning neuron. We suspect that this all-or-nothing principle for learning will drastically change how our algorithm detects clusters, and we compare the results produced by both neighborhood functions in Section 5. Though not in the definition, in practice we decrease r over time using exponential decay.

4 Data and Data Pre-processing

We obtained the World Development Indicators data set from Kaggle, a popular platform that hosts data science competitions. Although the original data set had data on 1344 different indicators for 247 countries over a 56 year period (1960 to 2015), we had to take only a subset of the data to both reduce the dimensionality and exclude missing data. In particular, we determined that year 2014 had the least missing data across all 1344. We then chose the 100 indicators with the highest data availability, and restricted our analysis to the 161 countries that had complete data for these 100 indicators.[1] This data cleaning was performed using R.

Before training the SOM, we pre-process the data by normalizing it. This involves centering and scaling the values for each indicator by subtracting from each value the column-wise mean and dividing the result by the column-wise standard error. Doing so makes our algorithm more numerically stable by reducing the chances of it encountering over and underflows. Lastly, we use the subset of the MNIST handwritten digits data set provided in Homework #6 as a basic test for whether our algorithm works, though our main goal is to cluster the indicators data set.

5 Results

5.1 Basic testing with digits data

To verify that our basic SOM algorithm works, we tested it on artificially-generated data (from bivariate normal distributions) and the handwritten digits data set from Homework #6. See Figure 3 in Appendix A for the U-matrix of digit clustering.

Overall, after sufficient tweaking of parameters (discussed in previous sections and also below), we were able to get satisfactory clustering of the digits. In particular, "easily distinguishable" digits such as 0, 5 and 6 are isolated in distinct regions in the U-matrix with clear borders. Not surprisingly, clusters of digit pairs or triplets that are often confused, such as 3 vs. 8 and 7 vs. 4 vs. 9, are located close to each other with relatively indistinct decision boundaries. This is a direct consequence of the proximity of these similar images in feature space.

Finally, several clusters of digits were "cut off" by the border (e.g. clusters of 1 and 2). The border of the SOM is a particularly problematic region, as it is often distorted due to the folding of the interior of the map. This leads to a significant increase in the inter-neuron distance between the edge neurons, also known as *border effects* (Kohonen, 2013). In our tests, border effects were most severe along the right edge; for this reason, we chose to omit the right edge neurons when displaying the U-matrix.

5.2 Varying σ's characteristic decay time τ_σ

We switch to the World Development Indicators data set for parameter fine-tuning and testing of our extensions. For fine-tuning, we experimented with τ_σ values of $\{500, 1000, 3000, 10000\}$ iterations while fixing τ_α at 5000 iterations. A subset of the results are presented in Figures 4 to 6 in Appendix A. Appendix B gives a table that links each country code with the corresponding country's name.

We observed two significant patterns in the results. First, for small sigma decay times ($\tau_\sigma \leq 3000$), we observed the formation of "island clusters" around each data point. Each island consists of about 10 closely clustered neurons with the data point at the approximate center. Furthermore, there are clearly defined borders separating the islands. This small islands structure disappears when we increase τ_σ (see Figure 6). The individual islands coalesce into big superclusters with less clearly defined boundaries.

We considered the latter case ideal for our purposes, since it successfully groups related countries into a single cluster. In the former case, we suspect that the island formation was a symptom of overfitting. Interestingly, regardless of τ_σ, certain countries were always close together in topological space. Examples include the group of Eastern European countries (POL, CZE, SVK, LTU, EST) as well as the group of Central/South American countries (MEX, COL, PER, DOM, PAN).

[1] The SOM algorithm can be made robust to data points with missing values for some features, but this would involve a significant amount of extra work because we use functions from numerical libraries that assume the absence of missing data. To ensure that we devote more time to analysis of our algorithm, we choose to forgo this aspect and reduce the size of the data set.

5.3 Step neighborhood function

Our results with the step neighborhood function are shown in Figures 7 to 9. We tested the step function with various decay times for the threshold radius r: $\{\infty(\text{no decay}), 500, 1000, 3000, 10000\}$ iterations, while keeping τ_α constant at 5000 iterations.

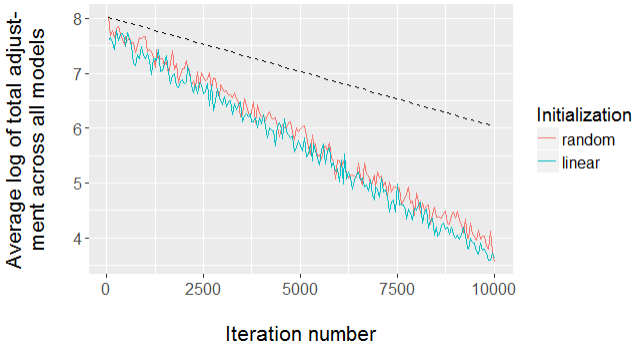
The U-matrices of SOM trained with the step neighborhood function appear drastically different from that trained with the gaussian function. Instead of globular clusters with relatively smooth boundaries, the step function results in tight circular neighborhoods with hard boundaries (indicated by the circular "coffee-stain" patterns in the U-matrix). This is because all neurons within the winner's neighborhood adapts equally in magnitude to the input, instead of having adaptation fall off with topological distance. Clearly, this results in a more contrived final ordering for the neurons.

We still observed cluster formation using the step function. However, a lot of clusters are relegated to the borders of the SOM. This is again due to the magnitude of the adaptation of the winner and its neighbors. Since the threshold radius r starts out at half of the largest SOM dimension, a random input point in the beginning could "pull" the large majority of the neurons towards it; following that, any other distant input point would only be able to attract the most distant neurons, i.e. the ones along the SOM border. This property of the step function is clearly disadvantageous since a lot of topological space is "wasted"; instead of being well spread out, a significant portion of the input points are marginalized.

5.4 Linear Initialization

Figure 2 shows the convergence of the SOM training for both random and linear initialization, by plotting the log of total adjustments made to all models according to Equation (1), averaged over intervals of 200 iterations each. The dashed line is an approximate envelope curve that shows the minimal rate of decay in model adjustments that is imposed by the decreasing learning rate. We see that the blue line for linear initialization is consistently further away from the envelope curve (i.e. lower) than is the red line for random initialization. This shows that with linear initialization, the adjustments to the models at any given iteration is smaller, which implies faster convergence under linear initialization, which squares well with the theory.

Figure 2: Convergence for both initialization methods with $\tau_\sigma = 10000$ and $\tau_\alpha = 5000$



6 Further Work

Due to its ubiquity and utility in data analysis, there is a wealth of extensions available to the SOM. In this section, we provide a concise and non-exhaustive list of directions where we could possibly extend our project.

First, the aforementioned border effects is a major issue, since the topological order of the data is severely distorted near the SOM borders. Experiments in literature suggest that using a hexagonal (instead of square) array with an oblong structure minimizes border defects (Kohonen, 2013). Furthermore, it is also possible to implement cyclic maps, where edges of the SOM are topologically connected. In this case, the map acquires a spherical or toroidal topology which completely removes the issue of border defects. However, this also means that it becomes challenging to represent the SOM accurately as a 2D surface.

Another possible extension is the growing SOM (Drittenback *et al.*, 2000). The traditional SOM relies on a fixed network architecture, requiring the user to preemptively set the dimensions of the map. In the growing SOM, an additional *growing* step after the adaptive step adds new rows/columns of neurons based on a pre-defined error threshold q. Hence, starting from a 2×2 map, the final SOM is hierarchically constructed by repeatedly growing the width and length of the map.

References

- Akinduko, Ayodeji A., and Evgeny M. Mirkes. "Initialization of self-organizing maps: principal components versus random initialization. A case study." arXiv preprint arXiv:1210.5873 (2012).
- Bullinaria, John A. "Self Organizing Maps: Fundamentals." Taken from http://www.cs.bham.ac.uk/ jxb/NN/l16.pdf
- Kohonen, Teuvo. "Essentials of the self-organizing map." Neural Networks 37 (2013): 52-65.
- Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar. "Introduction to Data Mining." (2006).
- Vesanto, Juha, and Esa Alhoniemi. "Clustering of the self-organizing map." IEEE Transactions on neural networks 11.3 (2000): 586-600.
- Drittenbach, Michael, Dieter Merkl, and Andreas Rauber. "The growing hierarchical self-organizing map." Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on. Vol. 6. IEEE, 2000.

Appendix A: U-matrices

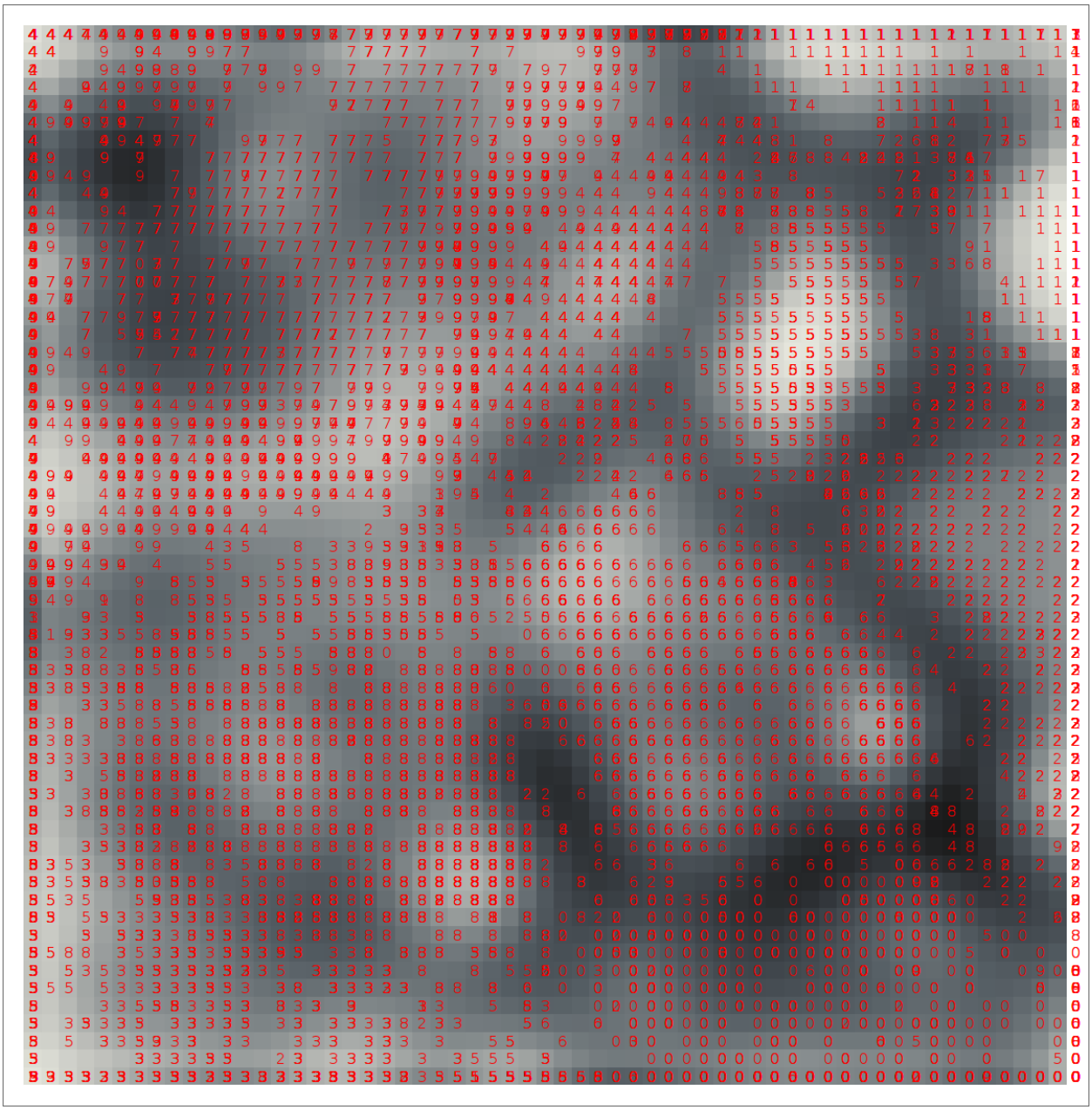


Figure 3: U-matrix of SOM applied to digits data with random initialization, gaussian neighborhood function and sigma decay time τ_σ of 10000 iterations.

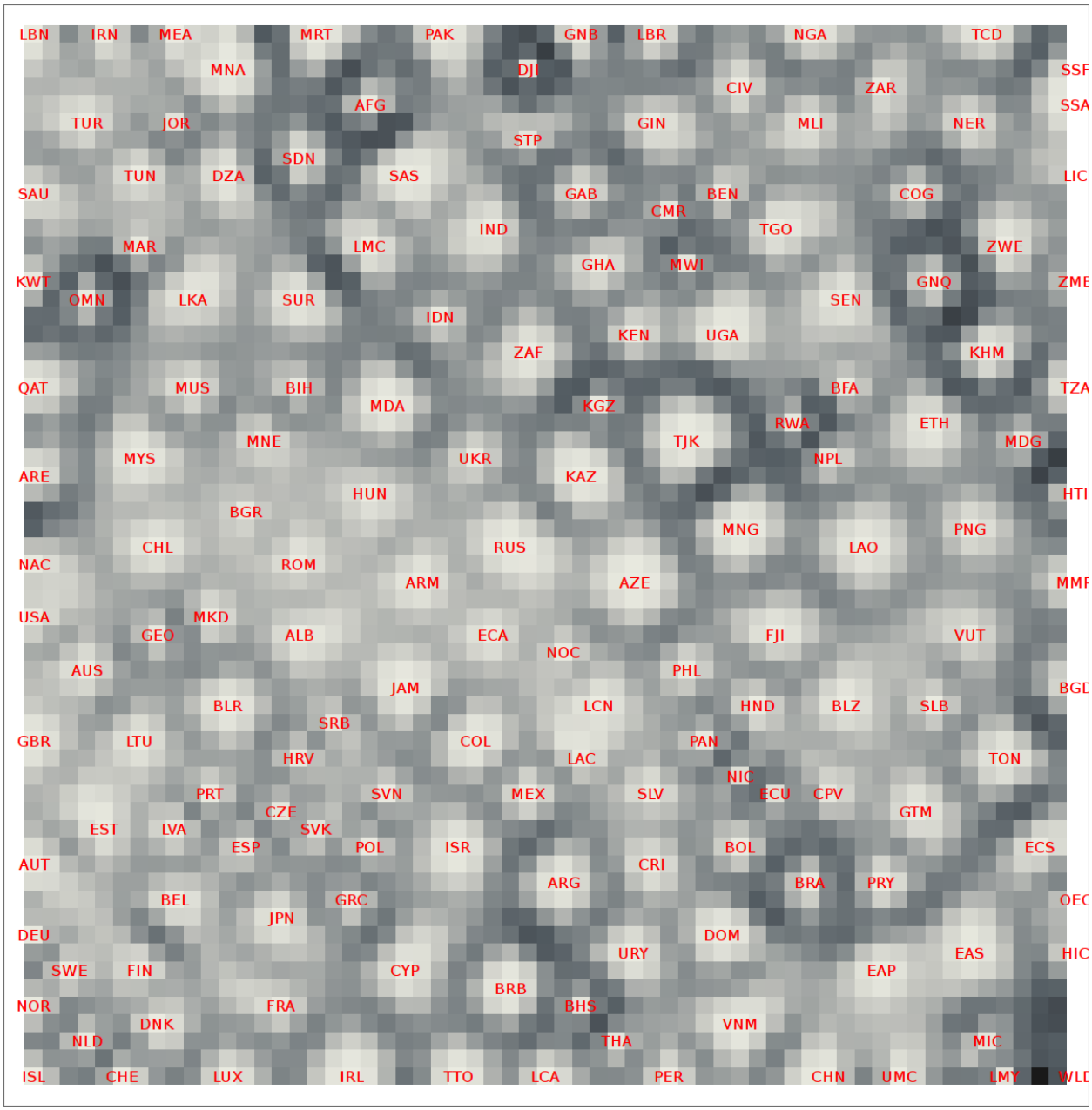


Figure 4: U-matrix of SOM with random initialization, gaussian neighborhood function and sigma decay time τ_σ of 1000 iterations.

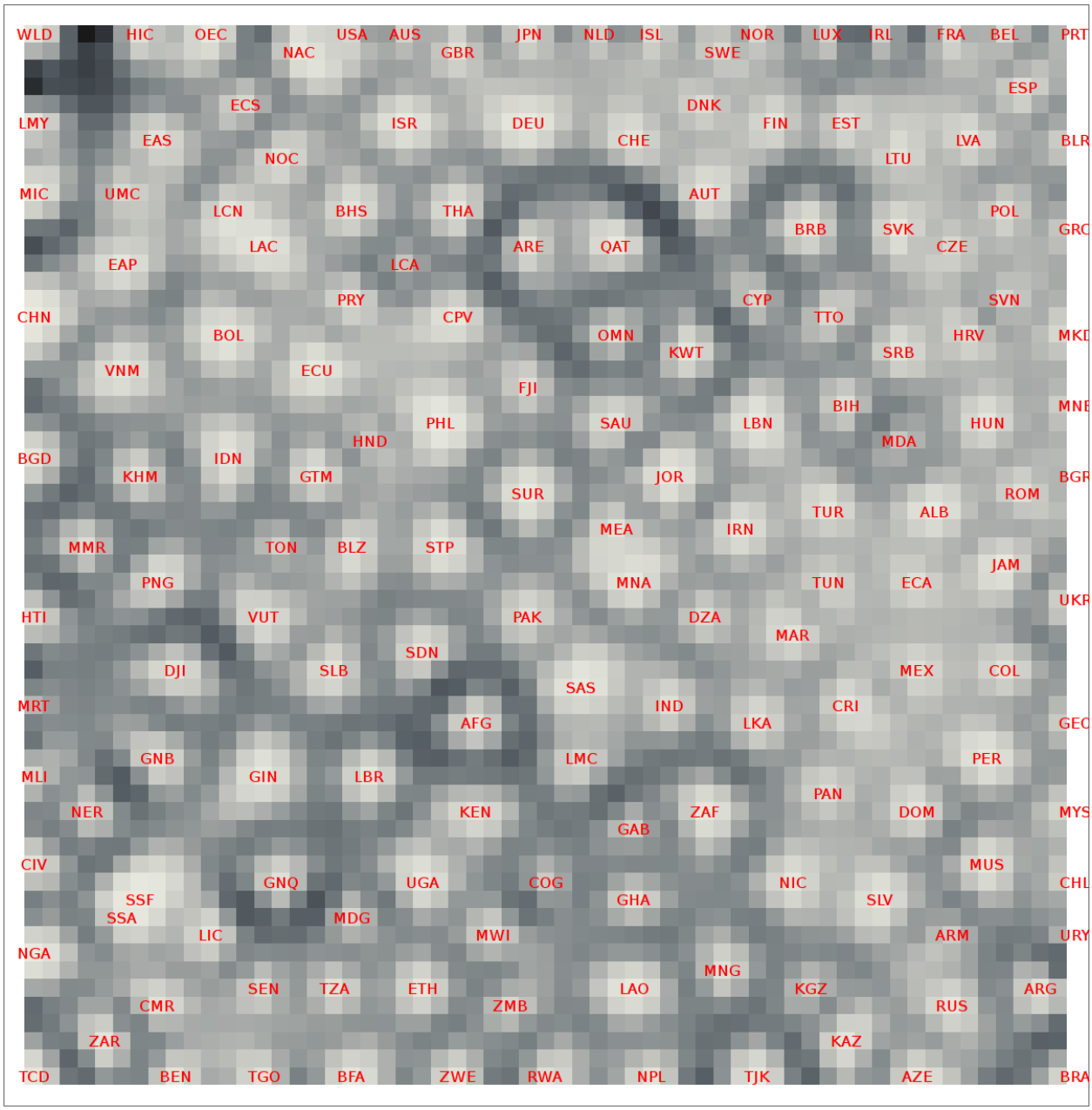


Figure 5: U-matrix of SOM with random initialization, gaussian neighborhood function and sigma decay time τ_σ of 3000 iterations.

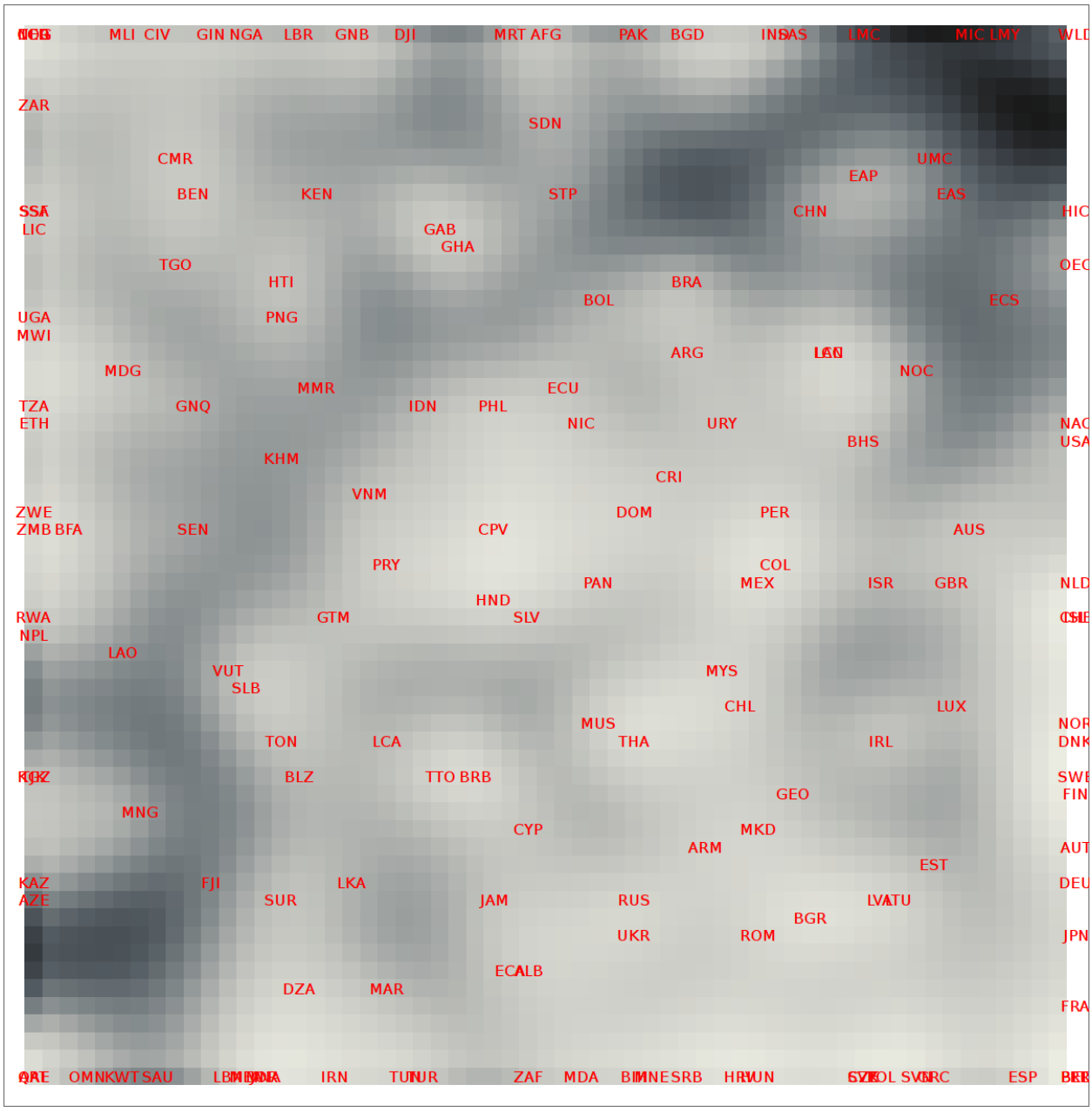


Figure 6: U-matrix of SOM with random initialization, gaussian neighborhood function and sigma decay time τ_σ of 10000 iterations.

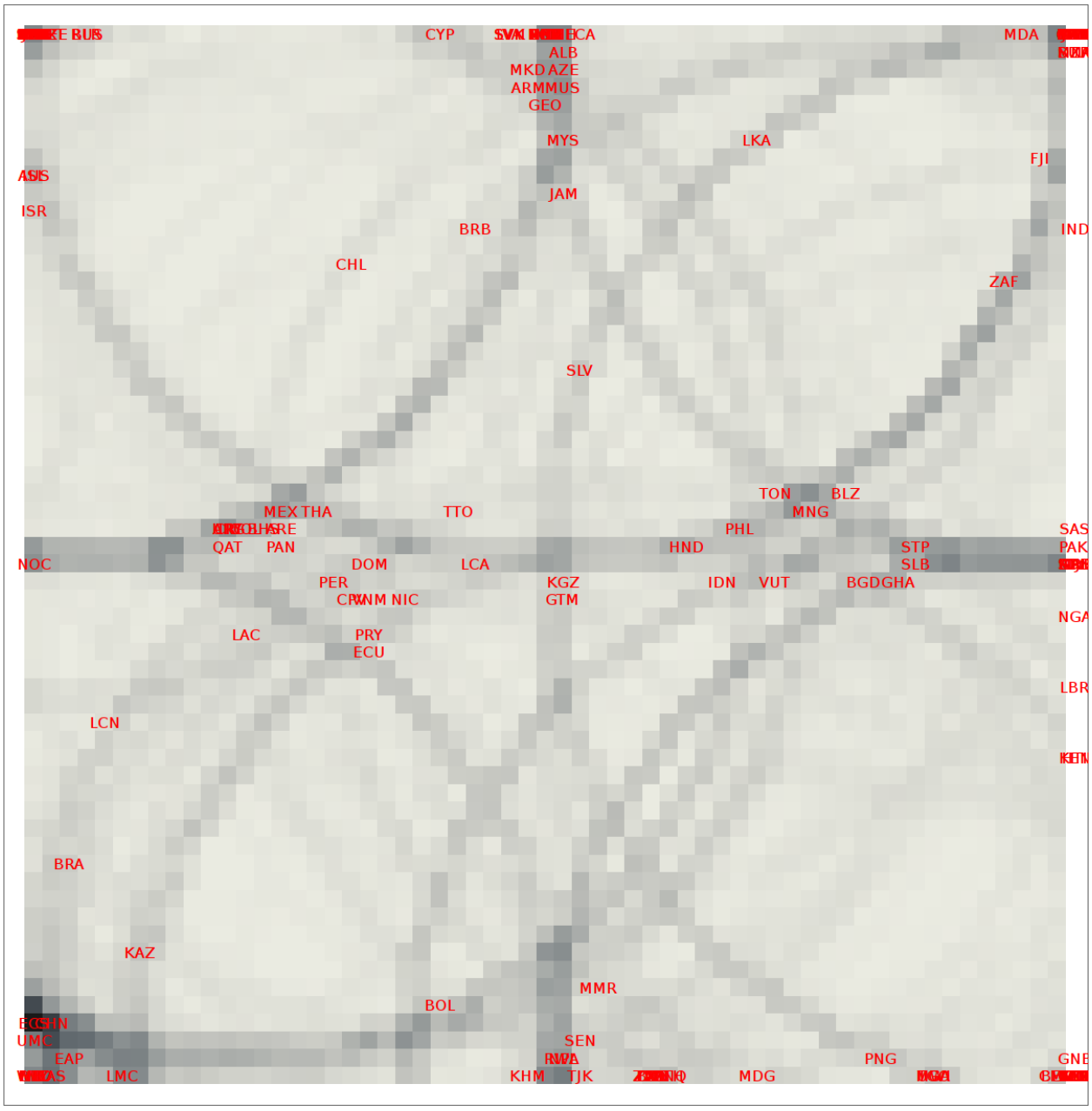


Figure 7: U-matrix of SOM with random initialization, step neighborhood function and no threshold decay ($\tau_r = \infty$).

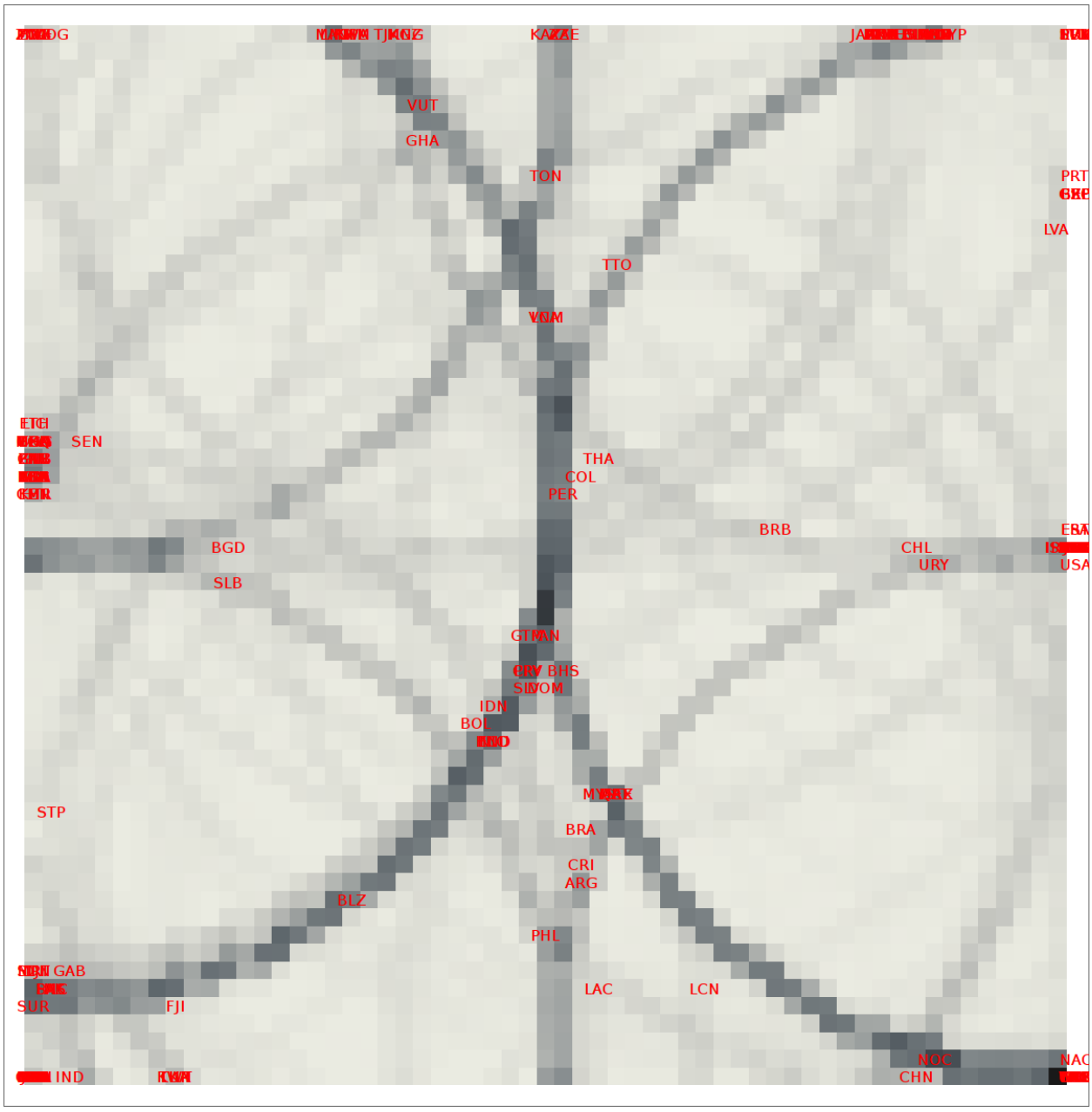


Figure 8: U-matrix of SOM with random initialization, step neighborhood function and threshold decay time τ_r of 1000 iterations.

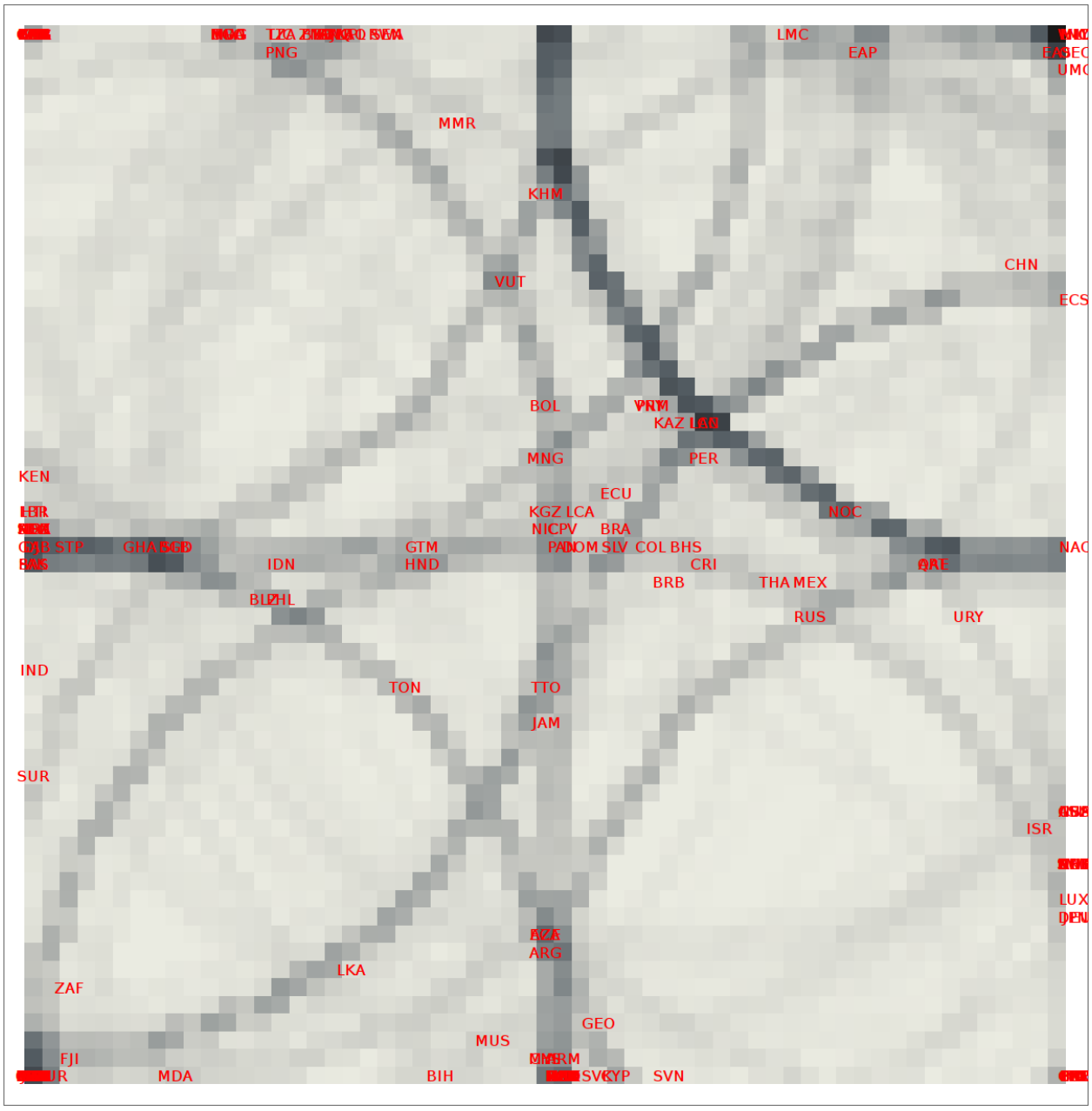


Figure 9: U-matrix of SOM with random initialization, step neighborhood function and threshold decay time τ_r of 10000 iterations.

Appendix B: Country Codes

Code	Country	Code	Country
AFG	Afghanistan	LBR	Liberia
ALB	Albania	LTU	Lithuania
DZA	Algeria	LMY	Low & middle income
ARG	Argentina	LIC	Low income
ARM	Armenia	LMC	Lower middle income
AUS	Australia	LUX	Luxembourg
AUT	Austria	MKD	Macedonia, FYR
AZE	Azerbaijan	MDG	Madagascar
BHS	Bahamas, The	MWI	Malawi
BGD	Bangladesh	MYS	Malaysia
BRB	Barbados	MLI	Mali
BLR	Belarus	MRT	Mauritania
BEL	Belgium	MUS	Mauritius
BLZ	Belize	MEX	Mexico
BEN	Benin	MEA	Middle East & North Africa (all income levels)
BOL	Bolivia	MNA	Middle East & North Africa (developing only)
BIH	Bosnia and Herzegovina	MIC	Middle income
BRA	Brazil	MDA	Moldova
BGR	Bulgaria	MNG	Mongolia
BFA	Burkina Faso	MNE	Montenegro
CPV	Cabo Verde	MAR	Morocco
KHM	Cambodia	MMR	Myanmar
CMR	Cameroon	NPL	Nepal
TCD	Chad	NLD	Netherlands
CHL	Chile	NIC	Nicaragua
CHN	China	NER	Niger
COL	Colombia	NGA	Nigeria
ZAR	Congo, Dem. Rep.	NAC	North America
COG	Congo, Rep.	NOR	Norway
CRI	Costa Rica	OMN	Oman
CIV	Cote d'Ivoire	PAK	Pakistan
HRV	Croatia	PAN	Panama
CYP	Cyprus	PNG	Papua New Guinea
CZE	Czech Republic	PRY	Paraguay
DNK	Denmark	PER	Peru
DJI	Djibouti	PHL	Philippines
DOM	Dominican Republic	POL	Poland
EAS	East Asia & Pacific (all income levels)	PRT	Portugal
EAP	East Asia & Pacific (developing only)	QAT	Qatar
ECU	Ecuador	ROM	Romania
SLV	El Salvador	RUS	Russian Federation
GNQ	Equatorial Guinea	RWA	Rwanda
EST	Estonia	STP	Sao Tome and Principe
ETH	Ethiopia	SAU	Saudi Arabia
ECS	Europe & Central Asia (all income levels)	SEN	Senegal

Code	Country	Code	Country
ECA	Europe & Central Asia (developing only)	SRB	Serbia
FJI	Fiji	SVK	Slovak Republic
FIN	Finland	SVN	Slovenia
FRA	France	SLB	Solomon Islands
GAB	Gabon	ZAF	South Africa
GEO	Georgia	SAS	South Asia
DEU	Germany	ESP	Spain
GHA	Ghana	LKA	Sri Lanka
GRC	Greece	LCA	St. Lucia
GTM	Guatemala	SSF	Sub-Saharan Africa (all income levels)
GIN	Guinea	SSA	Sub-Saharan Africa (developing only)
GNB	Guinea-Bissau	SDN	Sudan
HTI	Haiti	SUR	Suriname
HIC	High income	SWE	Sweden
NOC	High income: non-OECD	CHE	Switzerland
OEC	High income: OECD	TJK	Tajikistan
HND	Honduras	TZA	Tanzania
HUN	Hungary	THA	Thailand
ISL	Iceland	TGO	Togo
IND	India	TON	Tonga
IDN	Indonesia	TTO	Trinidad and Tobago
IRN	Iran, Islamic Rep.	TUN	Tunisia
IRL	Ireland	TUR	Turkey
ISR	Israel	UGA	Uganda
JAM	Jamaica	UKR	Ukraine
JPN	Japan	ARE	United Arab Emirates
JOR	Jordan	GBR	United Kingdom
KAZ	Kazakhstan	USA	United States
KEN	Kenya	UMC	Upper middle income
KWT	Kuwait	URY	Uruguay
KGZ	Kyrgyz Republic	VUT	Vanuatu
LAO	Lao PDR	VNM	Vietnam
LCN	Latin America & Caribbean (all income levels)	WLD	World
LAC	Latin America & Caribbean (developing only)	ZMB	Zambia
LVA	Latvia	ZWE	Zimbabwe
LBN	Lebanon		