

Deep Clustering of Mobile Network Data with Sparse Autoencoders

Márton Kajó*, Benedek Schultz†, Georg Carle*

*Technical University of Munich, Department of Informatics

Email: {kajo, carle}@net.in.tum.de

†Nokia Bell Labs

Email: schultz.benedek@gmail.com

Abstract—Unsupervised machine learning methods, such as clustering algorithms could be powerful tools for automation. By simplifying data through structuring, these algorithms can help network management use-cases where autonomous agency or elevated levels of cognition is required. Recent developments in deep learning allow clustering algorithms to gain unprecedented insight into the data, creating meaningful clusters as a result. In this paper, we propose a Sparse Clustering Autoencoder, capable of autonomously encoding cell behavior into a graph-like representation we call Network State Transition Graphs. We compare our proposed algorithm against other deep learning-based clustering algorithms, and demonstrate its utility on data from a real, large-scale mobile network deployment.

Index Terms—Clustering, Sparseness, Autoencoder, Deep Learning, Network Management Automation, Cognitive Autonomous Networks

I. INTRODUCTION

The automation of mobile network management is traditionally done by numerous Self-Organizing Network (SON) functions working together [1], which typically follow hand-engineered rules, using information from a smaller sub-portion of the network. Such automation is able to only realize rigid and simple logic without insight or autonomy, which might not be sufficient for the management of the increasingly complex 5G and future mobile network generations. Cognitive Autonomous Networks (CANs) [2] aim to overcome these limitations, by envisioning network management functions that process large amounts of varied data, collected from many different parts and layers of the network.

Deep learning algorithms are key enablers to realizing such cognitive automation. In the CAN concept deep learning is specifically used to bring intuition/agency into the automation, to make the system capable of reacting to unforeseen circumstances, or as an exploratory tool to formulate understanding without the need for external (human) inputs. Although in the last few years, deep learning has achieved astonishing results in the category of supervised learning, in many of the envisioned uses the expected output is not available as training data, making supervised learning unfeasible.

Clustering, i.e. the formulation of groups based on similarity, is one of the most powerful tools for such use-cases, and the most often utilized method of unsupervised learning. In our nomenclature, clustering is more than just simple grouping; it

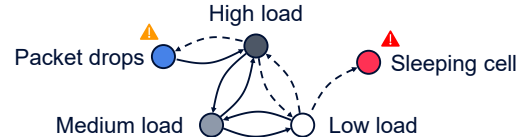


Fig. 1. Illustration of Network State Transition Graph

requires understanding of the underlying logic in the data, so that clusters can be based on deep and meaningful connections between the observations. Because large volumes of multi-dimensional data are hard to comprehend, it is preferable to model the data as a finite number of examples, with descriptive labels attached to them. This simplification and abstraction can benefit machine automation, in much the same way as it helps human understanding.

The use case we consider here is the modeling of network behavior as transitions between *network states*. Here individual network elements exist in a shared state-space, where they are modeled as either being in a state, or moving from one state to another. The possible state-space can be envisioned as a *network state transition graph*, where states are nodes, and transitions are connections between these nodes. An illustration of a network state transition graph can be seen on Fig. 1.

In this paper, we propose a Deep Neural Network (DNN) called Sparse Clustering Autoencoder (SCA). It is an autoencoder especially well-fit to autonomously define network state transition graphs by realizing clustering. Section II explains our design considerations and the implementation details. Section III discusses related work both from the network management, as well as the deep learning perspective, with a numerical comparison to similar algorithms. Section IV shows an example of using SCA to build a network state graph on real network data. Finally, Section V concludes the paper.

II. SPARSE CLUSTERING AUTOENCODERS

The clustering method proposed here is based on an autoencoder, a neural network topology used for data compression, feature extraction or denoising. Autoencoders are trained in an unsupervised manner, their target being the perfect reconstruction of the original observations without any additional labeled information. Perfect reconstruction is not trivial, as the autoencoder is forced by its own topology

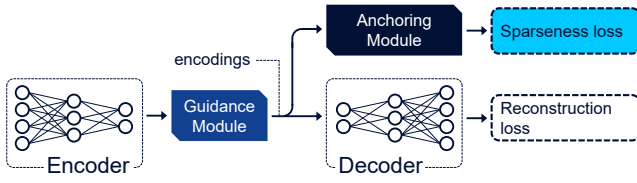


Fig. 2. Overview of Sparse Clustering Autoencoders

to encode the observations into a lower dimensional space, and be able to reconstruct (decode) them from this simplified representation. The encoding and decoding is learned together at training, during which the autoencoder tries to minimize a *reconstruction loss*, measured in the data-space between the original and the reconstructed observations.

Our additions to the autoencoder are two modules:

- The **anchoring** module, which calculates an additional *clustering loss* term on the encoded representation during training, so that the autoencoder is forced to learn a clustered representation;
- The **guidance** module, which constrains activations to a specific range, and helps in balancing cluster populations.

The anchoring module only plays a role during training, by indirectly affecting the encoded activations through a calculated *sparseness loss*. Contrarily, the guidance module is present in the network both during training and inference, and directly modifies activations. The building blocks of the proposed neural network can be seen on Fig. 2.

A. From Sparseness to Clustering

In the following, we will refer to an encoded representation of an observation as an encoded vector $Q = \{q_1, q_2, \dots, q_D\}$, or simply as an *encoding*. An encoding is made up of individual numerical values q_i , $i \in [1, D]$, also called *activations*. If the vectors propagated through a neural network contain values other than 0 in only a few places, it is said that a neural network uses *sparse* activations. Activation sparseness can be enforced for speeding up computation, or as a regularization method. In the case of SCA, we use activation sparseness combined with a confinement to a regular *simplex* S (Fig. 3) on the encoding vector Q . The simplex in a D -dimensional space is a convex hull, in the intersection of the $D - 1$ -dimensional affine subspace Θ and the D -dimensional unit cube. On the lower extreme of sparsity, where activations are 1-sparse, these restrictions collect encodings around the unit vectors, thereby forming clusters in the encoding space. We will refer to the unit vector positions as *base points*. To represent the sparseness of the encoding vector, we define the parameter s . Let $C_{nz}(Q)$ be the number of elements of vector Q that are non zero. Then, s is defined as:

$$s = C_{nz}(Q) - 1$$

The enforcement of activation sparseness in a neural network has to be done gradually, in order not to overtly disturb the already formed logic in the net [3]. The transition from no sparseness to full sparseness can be seen as a gradual limitation

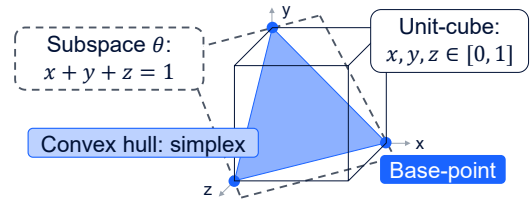


Fig. 3. The simplex of convex combinations

of the dimensionality of the space the encodings can use, a limitation of freedom. Our definition of the sparseness value s is equal to this dimensionality. A baseline restriction is present from the start in s , as points within the simplex already adhere to the sparseness $s = D - 1$. Lower s values limit activations to sub-hulls of the simplex. As an example, in 4 dimensions, the simplex is a 3-dimensional regular tetrahedron, with an s -value of 3. Lowering this s to 2, 1, and then finally 0, the activations are restricted to the faces, edges, and vertices of the tetrahedron respectively. The lowest sparseness value achievable is always 0, at which point the encodings are forced into the base points (restricted to 1-hot vectors), essentially realizing clustering. An illustration of this gradual limitation of freedom can be seen on Fig. 4. All through the process, the activations retain their convex combination-like qualities, staying within the simplex.

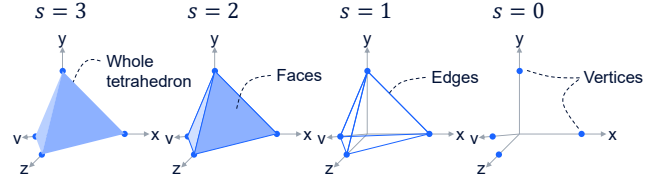


Fig. 4. Gradual limitation of freedom

B. Anchoring Module - Sparseness Loss Calculation

Sparsity in neural networks is usually enforced by the masking of activations, through multiplication by 0. In many cases, this operation means quite a radical transformation of the propagated vectors, and as such can easily degrade the established logic in the net. In SCA, we invoke sparseness through a softer, more forgiving enforcement, by introducing a *sparseness loss* on the encodings. To calculate the sparseness loss, we calculate an anchor point $\tilde{Q}$ for every encoding Q , which is the orthogonal projection of Q onto the closest sub-hull of S . The anchor describes the position where the encoded point should be according to the current sparseness setting (s). The sparseness loss is then calculated as:

$$l_{sparse} = dist(Q, \tilde{Q}),$$

where $dist$ is the Euclidean or L_2 distance.

Calculating the projection is a non-trivial task, because for $s < D - 1$ there are multiple sub-hulls of the simplex, from which we would need to find the closest one for each encoding. In theory, this means that we would need to compute an intractable $\binom{D}{s}$ number of projections. In order to escape this

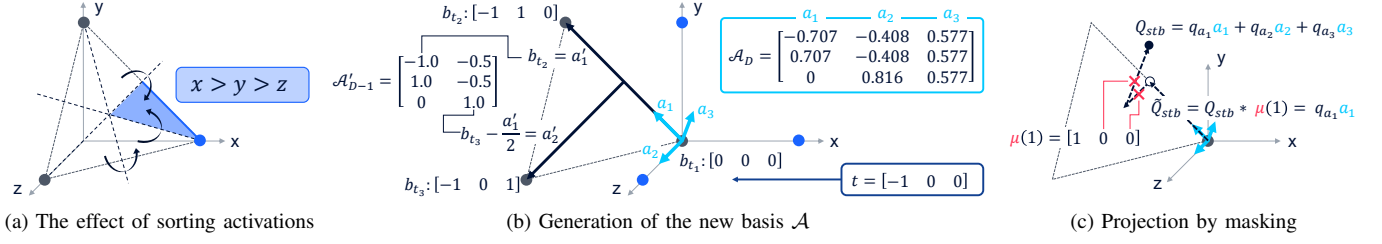


Fig. 5. The core steps in the generation of anchor point $\tilde{Q}$

complexity, we *sort* the activations in a descending order for every Q , and work in this sorted space. From a geometric view, sorting rotates and mirrors all parts of the simplex into a single section (illustrated on Fig. 5a), where only a single projection suffices to virtually implement any projection needed for calculating $\tilde{Q}$.

By using a translation of $-b_1$, the previously affine subspace Θ becomes a real subspace of $\mathbb{R}^D$. Within this translated space, the anchor point $\tilde{Q}$ can be effectively calculated for every level of sparseness s , by changing to a new orthonormal basis $\mathcal{A} = \{a_1, a_2, \dots, a_D\}$. The calculation of the new basis can be simply done by using Gramm-Schmidt orthogonalization on the points of B . This yields $\mathcal{A}_{D-1} = \{a_1, a_2, \dots, a_{D-1}\}$, with an additional base a_D calculated as an orthonormal vector to $\mathcal{A}_{D-1}$, so that $\mathcal{A} = \mathcal{A}_D$ becomes a fully-ranked orthonormal basis (Fig. 5b).

The sorting and translation has to be done on the encodings which we wish to project. After the translation, we can apply the base change to $\mathcal{A}$. In this modified basis the projection to the s -sparse subspace can be done by masking the last $D - s$ bases. This is done similar to regular sparseness enforcement, by multiplying the activations with the mask $\mu(s) = (1, \dots, 1, 1, 0, \dots, 0)$, in which the first s elements are 1, and the rest 0 (Fig. 5c). This procedure yields the anchor point $\tilde{Q}$ in the sorted, translated and rebased space.

- 1 : Sort activations: $Q_s = \text{sort}(Q)$
- 2 : Translate: $Q_{st} = Q_s - t$
- 3 : Mask basis: $\tilde{\mathcal{A}} = \mathcal{A} * \mu(s)$
- 4 : Change to new basis: $\tilde{Q}_{stb} = Q_{st} \tilde{\mathcal{A}}^T$
- 5 : Compute loss: $l_{\text{sparse}} = \text{dist}(Q_{stb}, \tilde{Q}_{stb})$

Fig. 6. Shortened procedure for computing l_{sparse}

If the user has no need of the anchor points (for visualization, or debugging) in the original encoding space, the reverse operations of base change, translation and sorting can be omitted. This is because, although a general basis change can be non-distance-preserving, the change between two orthonormal bases is an isometry, which preserves distances. This is also true for translation. Coordinate sorting would not be distance preserving between encodings, but it is distance preserving between every encoding and its corresponding anchor. This allows us to calculate the loss immediately in the rebased space as a function of Q_{stb} and $\tilde{Q}_{stb}$. Additionally, for further speedup, the masking operation can be done to the basis $\mathcal{A}$

itself instead of the encodings, which avoids a significant amount of computation. The steps for this final, shortened procedure are listed on Fig. 6.

C. Guidance Module

The anchoring module provides the sparseness loss, but does not directly interact with the encodings. Contrarily, the guidance module, which is located in the middle of the autoencoder and precedes the anchoring module, does directly change the activations. One of the tasks of this module is to limit the encodings to the range of $[0, 1]$, by using a sigmoid nonlinearity on the activations. The sigmoid nonlinearity ensures that the encodings fall within the unit hypercube in $\mathbb{R}^D$, turning the linear projection onto the affine subspace Θ into a projection onto the simplex.

Many clustering algorithms have explicit mechanisms, often an additional loss, that tries to even out cluster populations. Without any such additional mechanisms, our method usually shows reduced performance due to the encoding not utilizing all the available clusters for representation. Instead of an additional loss, we use a weight-shared batch-normalization layer before the sigmoid nonlinearity. Internally this layer centers the activations around the origin, by implementing:

$$y_{\text{batchnorm}} = \frac{x - \text{mean}(x)}{\text{std}(x)} * p_{\text{scale}} + p_{\text{offset}}.$$

In this equation x denotes the input, while p_{scale} and p_{offset} are learnable parameters of the batch normalization layer. The idea behind batch-normalization is that the centering effect helps in the early training by easing discovery of the activation space, while later in the training the network is able to counteract possibly unwanted normalization by separately changing the learnable parameters for any feature [4]. In our case, to aid the even distribution of encodings among clusters all through the training, the learnable parameters are shared across features. This retains the centering effect of the batch normalization layer, but gives the network enough flexibility to adhere to the required sparseness.

D. Training

Training of the SCA can be split into 3 phases. In the *exploration phase* the sparseness parameter s is held at the initial $D - 1$ value, which only limits the activations onto the simplex, leaving a large amount of freedom for the autoencoder to explore the encoding space and establish a

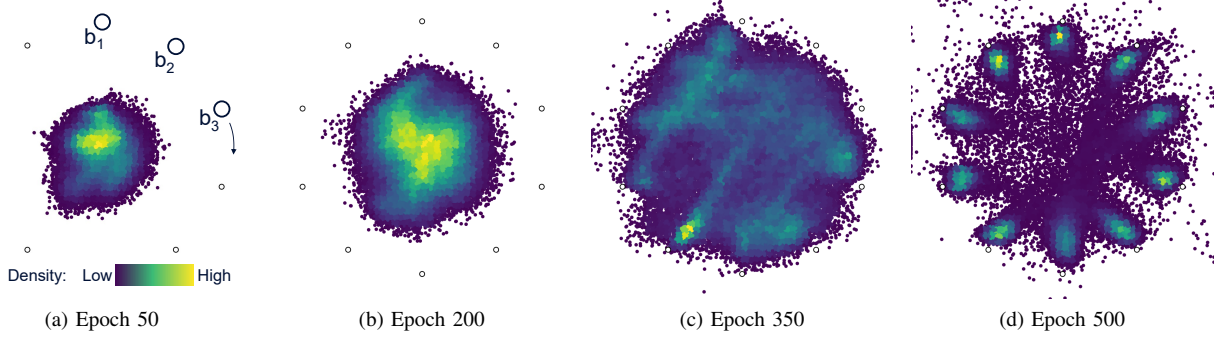


Fig. 7. Projected scatter plots of the encodings during training

mapping that can later be focused into a clustering. In the *clustering phase*, the s parameter is gradually decreased to encourage the grouping of encodings. The *refinement phase* then allows the network to smooth out imperfections that might have arisen during the clustering phase.

During the development of SCA, we often used scatter-plots of the encoded activations, to make sure the network trained correctly. We found the best way to plot encodings is to project them into a 2-dimensional space so that the base points are arranged on a circle going around clockwise. The projection can be misleading, as one has to keep in mind that all base points are equidistant from each other in the encoding space, however, the plots give a good impression of how the encoding develops as sparseness is increased. An example of these plots can be seen on Fig. 7.

III. RELATED WORK AND COMPARISON

A. Clustering in Mobile Network Management

Clustering has been an important tool for network management use-cases where autonomous agency is required, because the conditions with which the system will be working are unknown at the time the system is designed. One group of such use-cases fall under the Self-healing aspect of SON [1]. Self-healing is envisioned as one of the final stages towards complete autonomy, where SONs are able to detect, diagnose, and recover from previously known and even unknown failures in the network. The first step of this process is the detection of anomalous behavior:

- Static anomalous cell behavior can be caused by erroneous Configuration Management (CM) settings, especially so when the settings are autonomously changed by multiple SON functions. In [5], the Hierarchical Dirichlet Process (HDP) clustering is used to detect CM changes that cause a degradation, and to possibly roll back these changes.

- Dynamic anomalies in cell behavior can be caused by strange user behavior, software bugs, or hardware failure. In [6], such anomalies are detected with the help of the Merge Growing Neural Gas (MGNG) clustering algorithm. A later improvement to this clustering method, called Fixed Resolution Growing Neural Gas (FRGNG) is proposed in [7].

- A specific use-case for anomaly detection in mobile networks is detection of sleeping cells. In [8], the k -Means algorithm is used in an encoded space formed by Diffusion

Maps on simulated radio measurements, in order to detect degraded or sleeping cells.

- We also proposed a clustering algorithm called Bounding Sphere Quantization (BSQ) [9], a modification of the k -Means algorithm specifically targeted for anomaly detection or data exploration purposes. Although the BSQ algorithm should not be seen as deep learning, an implementation thereof that uses common tools from deep learning software packages is proposed in [10].

Recently, clustering plays a major role in many of the processes involved in the envisioned CAN modules. One such module is Environmental Modeling and Abstraction (EMA) [11], where k -Means or BSQ is used in an encoded space in order to implement a dynamic mapping between network states and human defined labels. These labeled states are then used as inputs to Cognitive Functions (CFs), the CAN equivalent of SON functions.

While some of the clustering methods above are very complex, none of them are deep learning-based. The mentioned uses-cases rely on precise models for sensing specific network states. Shallow models generated by statistical methods are prone to under- or overfitting in many cases. Deep learning-based clustering could be a better fit for these systems, by allowing for a modeling that focuses on deeper underlying logic in the data, and is in turn not prone to sensitivity issues.

B. Deep Clustering

Although deep learning has typically been applied for supervised learning tasks, unsupervised tasks have not been completely ignored, with Deep Clustering algorithms being the most a recent development. Deep clustering algorithms utilize deep autoencoder topologies, sometimes using generic Multilayer Perceptrons (MLPs), but more often incorporating specialized methods, such as Convolutional Neural Networks (CNNs) or Deep Belief Networks (DBNs). Clustering is achieved similarly to our proposed method, by adding an additional criteria to the model formulation during training, where otherwise the goal of the network is to best reconstruct the original observations. A major distinguishing factor of our work is that none of the algorithms produce encodings that can be understood as graphs. A good overview of the recent deep clustering algorithms can be found in [12].

The algorithms are compared on datasets where class associations are known. Performance is measured against the original

class associations, with metrics that are invariant to cluster association permutations. One such metric is Normalized Mutual Information (NMI), which takes the value 0 in case of no connection between cluster and class associations, and 1 if there is a perfect mapping between the two. Following is a table from [12] (Table I), that compares the performance of our algorithm with recent deep clustering algorithms, measured as the NMI on the MNIST dataset.

TABLE I
DEEP CLUSTERING PERFORMANCE COMPARISON

Method	Architecture	Clustering alg.	NMI
<i>k</i> -Means	-	-	0.481
DEC	MLP	Network est. params.	0.800
DCN	MLP	Network est. params.	0.810
SCA	CNN	-	0.839
UMMC	DBN	<i>k</i> -Means	0.864
CCNN	CNN	<i>k</i> -Means	0.876
JULE	CNN	Agglomerative clust.	0.915
DEPICT	CNN	Network est. params.	0.916
DBC	CNN	<i>k</i> -Means	0.917
TAX	CNN	Network est. params.	0.923

The performance of our algorithm is highlighted with blue. Although other algorithms did perform better, we see our method as competitive. We do not consider this performance to be the absolute best achievable, as there remain certain quirks, as well as a very irregular performance between trainings, which we hope to eliminate in the future.

IV. EXAMPLE USE OF SCA ON NETWORK DATA

This section is meant as a demonstration of the ease of understanding gained by working with meaningful clusters on multi-dimensional network management data. In this case, we have chosen to show how to explore network data to get a general overview of the network, and look for anomalous/misconfigured cells. To show an example of this, we have trained an SCA on Performance Management (PM) data captured from a real mobile network. The dataset contains roughly 3 months worth of hourly measurements from more than 2000 cells, consisting of 17 Key Performance Indicators (KPIs). We can group these KPIs into 5 groups based on meaning, as shown on Table II:

TABLE II
KPI GROUPS

Users	Number of users, connection attempts/releases
Volume / throughput	Data volume, PRB utilization, layer throughput
Latency	Latency on radio
Radio conditions	RSRP/Q, CQI
Voice	Voice data volume, number of QCI 1 users

The neural net used for this clustering was a deep MLP autoencoder consisting of fully-connected layers, batch normalization layers and Leaky Rectified Linear Unit (ReLU) nonlinearities. The encoder and the decoder were mirrored equivalents, both consisting of 4 layers.

We chose to use 16 clusters for this demonstration, as too many clusters would be hard to visualize or work with in the

confines of this paper. The choice is a little arbitrary, as in a typical scenario the user has no notion of how many real classes to expect in the data. Unfortunately, this downside is shared by many other clustering approaches. Generally, the more clusters are used, the more homogeneous the clusters will be, and the right choice for the number of clusters depends on a tradeoff between homogeneity and complexity of the outcome.

The sparsity parameter s was continuously lowered from 15 to 0.5 during training. This creates an encoding, that somewhat allows for binary combination of two network states, which helps in the identification of used transitions between the states, but generally encourages the activations to be close to the centroids. The final projection of the encoded activations can be seen on Fig. 8, together with the corresponding network state graph.

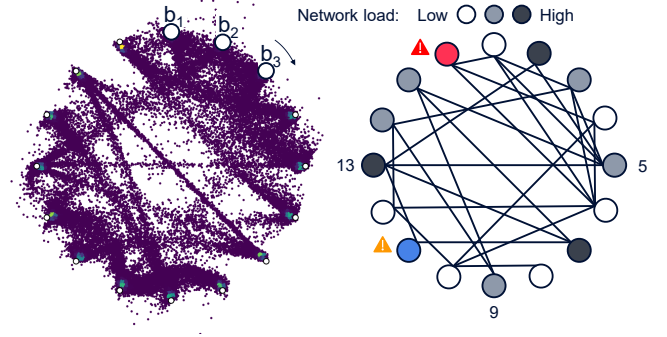


Fig. 8. Network data encoded by SCA as activations (left), and the equivalent network state graph (right)

Each observation was hard-assigned to the state with the largest activation in its encoding. The connections in the network state graph were established by looking at the network state transition sequences in the data: a connection is drawn between two states, if the number of transitions between the two clusters is above a certain threshold. There is a clear distinction in the number of transitions between often used and seldom used transitions, with often used transitions numbering in the 100 thousands range, while seldom used ones only represented in the hundreds. We chose the cutoff threshold to be 1000 transitions, with which the network state graph visually matches the scatter plot of the encoded activations.

The prototypes (or centroids) of the states are obtained by decoding the 1-hot encoding vectors in the encoding space back to the original KPI-space. By looking at what these prototypes represent in the original KPI-space (in the form of bar-charts), we were quickly able to deduce that most of the states correspond to different levels of network load, but represent mostly normal operation. On Figure 8, the states are colored from white to dark grey depending on the network load. However, states 11 and 16 stood out, showing strange behavior (highlighted as dark blue and red respectively). The KPI-space prototypes of these states can be seen on Fig. 9. All KPIs are shown as individually normalized features, with their mean being at 0 and standard variation at 1, thus, the vertical axis holds values in standard deviation.

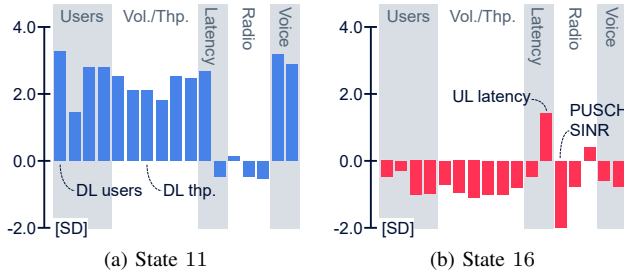


Fig. 9. Anomalous network state prototypes

State 11 shows extremely high user numbers and throughput in the Downlink (DL), one which is not represented in any other state. State 16, on the other hand, shows a situation with low cell load, but unreasonably high Uplink (UL) latency, and abysmal UL radio conditions. It is interesting to note, that state 11 only has transitions from and to high load states, while state 16 only low load states, which is also represented on Fig. 9. Another important aspect to note is that out of all states, state 11 and 16 have the largest amount of self-transitions, meaning that these states were usually not temporary, rather, cells spent longer, continuous timeframes therein.

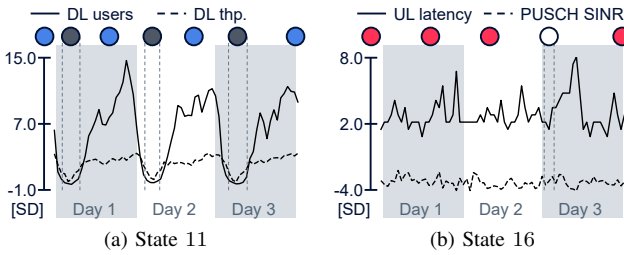


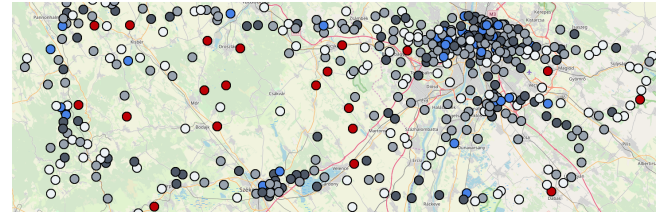
Fig. 10. Anomalous network state sequences

Fig. 10 shows 3 days of two cells with a high amount of time spent in state 11 and 16 respectively. Although a large portion of the time is therein, state 11 turns out to be a somewhat temporary state, as cell A1 regularly leaves it. Contrarily, state 16 seems to be more permanent, as cell A2 was continuously in that state for almost 2 days. This can be verified also on Fig. 11, where the cells are shown on the map and colored according to their state. During peak hours (3:00 PM), state 11 is represented in many of the cells belonging to densely populated, urban areas, however, all cells leave this state in the off hours (3:00 AM). On the other hand, some cells in rural areas seem to be stuck in state 16 for long periods of time.

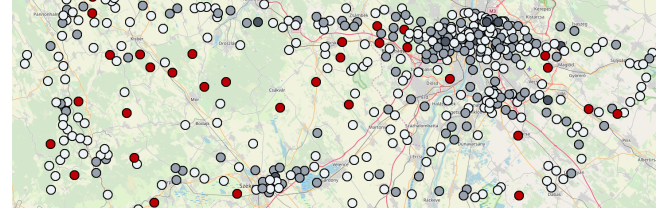
Overall, we conclude that state 11, although showing an extremely high load, does not represent a fault in itself. However, the cells that spend long, continuous timeframes in 16, should be checked for possible misconfiguration, or a potential Coverage and Capacity Optimization (CCO) issue.

V. CONCLUSION

In this paper, we have shown a deep neural network-based clustering algorithm, that is especially fit to translate network data into a network state transition graph. We have discussed



(a) 3:00 PM



(b) 3:00 AM

Fig. 11. Maps of cell states at two different timeslots

the implementation, explaining the details from a geometrical perspective. We have numerically evaluated and compared the algorithm with other, similar clustering methods on a common image dataset. We have also demonstrated the use of the algorithm on network management data, by undertaking a small data-exploration task.

REFERENCES

- [1] H. Seppo, S. Henning, and S. Cinzia, *LTE Self-organising Networks (SON): Network management automation for operational efficiency*. John Wiley & Sons, 2012.
- [2] S. S. Mwanje and C. Mannweiler, "Towards cognitive autonomous networks in 5g," in *Proceedings of ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*. IEEE, 2018.
- [3] A. Makhzani and B. Frey, "K-sparse autoencoders," *arXiv preprint arXiv:1312.5663*, 2013.
- [4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
- [5] G. F. Ciocarlie, C. Connolly, C.-C. Cheng, U. Lindqvist, S. Nováczki, H. Sanneck, and M. Naseer-ul Islam, "Anomaly detection and diagnosis for automatic radio network verification," in *International Conference on Mobile Networks and Management*. Springer, 2014, pp. 163–176.
- [6] B. Gajic, S. Nováczki, and S. Mwanje, "An improved anomaly detection in mobile networks by using incremental time-aware clustering," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 1286–1291.
- [7] S. Nováczki and B. Gajic, "Fixed-resolution growing neural gas for clustering the mobile networks data," in *International Conference on Engineering Applications of Neural Networks*. Springer, 2015, pp. 181–191.
- [8] F. Chernogorov, J. Turkka, T. Ristaniemi, and A. Averbuch, "Detection of sleeping cells in lte networks using diffusion maps," in *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*. IEEE, 2011, pp. 1–5.
- [9] M. Kajó, B. Schultz, J. Ali-Tolppa, and G. Carle, "Equal-volume quantization of mobile network data using bounding spheres and boxes," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–9.
- [10] M. Kajó, S. S. Mwanje, and S. Benedek, "Quantization techniques for the modern network automator."
- [11] S. S. Mwanje, M. Kajó, and S. Majumdar, "Environment modeling and abstraction of network states for cognitive functions."
- [12] E. Aljalbout, V. Golkov, Y. Siddiqui, M. Strobel, and D. Cremers, "Clustering with deep learning: Taxonomy and new methods," *arXiv preprint arXiv:1801.07648*, 2018.