LOCAL REPRESENTATION LEARNING WITH CONVOLUTIONAL AUTOENCODER

Michael P. Kenning, Xianghua Xie, Michael Edwards, Jingjing Deng

Swansea University, Swansea, UK

ABSTRACT

Very recent advances in deep learning methods have seen such representation learning approaches expand to domains which exhibit irregular spatial topologies. The approach of deep learning on graphs has seen increasing study, with generalization of localized filtering providing deep learning benefits in numerous fields where spatial relationships do not reside on a Cartesian grid. In this paper we present a graph-based convolutional autoencoder, and assess the contribution of its several components towards encoding quality. A graph-based convolutional operator is used to learn localized filtering operations for graph-wise encoding. An evaluation of the proposed method is provided on a version of MNIST that exists in an irregular topology that violates the array-like input required for conventional convolutional autoencoder methods.

1. INTRODUCTION

Representation learning approaches with deep neural networks have had a revolutionary impact on addressing numerous recognition problems. By learning appropriate feature representations, the need for domain expert insight and hand-crafted feature-extractors is greatly reduced [1]. Traditional Convolutional Neural Networks (CNNs) [2] learn localized feature descriptors which reside on a regular grid, leading to large performance gains in image and volume domain problems. However, a large number of application domains reside on irregular spaces which violate the array input domain assumption of the regular convolutional operation. For example, the convolution operator is well defined in the regular topology of the Cartesian grid, providing a kernel implementation which can be applied and optimized within CNN architectures and a suitable learning scheme. The kernel has a compact support, enabling it to learn local features from the input which can be translated across the input domain. In irregular domains the assumption of a grid-based input is inappropriate; thus the definition and translation of a localized filter is non-trivial [3].

The development of deep learning in the irregular domains has tried to explore this under-represented field, in attempts to identify mechanisms for learning spatially localized features from irregularly spaced input features. A number of structures exist to represent data which resides in an irregular domain [4]. The use of graphs represents the spatial structure of the problem domain as a set of vertices and edges. Each localization in the input domain is represented as a vertex in the graph structure, with weighted edges describing the relation between such vertices. Signal processing on graphs has developed into a field of research in which common signal processing techniques are generalized to an irregular domain, including filtering, downsampling and spectral transforms [5]. The study of graph signal processing approaches to representation learning on graph structures has lead to two main areas of research: graph-wise and vertex-wise approaches. Graph-wise methods treat a given observation of the whole graph as a single observation. Such approaches are comparable to image classification problems, in which an image is treated as a signal residing on the graph of a two-dimensional grid. Alternatively, vertex-wise techniques treat signals observed at a given vertex on the graph individually and can be compared to dense prediction problems such as segmentation in image processing.

Autoencoders [6] that were originally designed for unsupervised latent representation learning have been likewise translocated from the regular to the irregular domain. Graph-Based Autoencoders (Graph-AEs) are proposed to learn abstract feature embedding for irregularly spaced domain applications for better generalization performance. *Guo et al.* [7] proposed a Graph-AE which combines a Graph-Based Convolutional Neural Network (Graph-CNN) and an Autoencoder (AE) to learn discriminative feature representation from noise-degraded measurements of electroencephalography (EEG) and magnetoencephalography (MEG) scans taken from a network of sensors across the scalp. The extracted features are further generalized by a subsequent AE, whereas the Graph-CNN and AE are two consecutive and separated components. Similarly, *Litany et al.* [8] adopted the same graph-based learning schema for a three-dimensional body-meshes completion task, where a Variational Autoencoder (VAE) was used to learn a generative model on latent feature space for synthesizing structure at missing locations on a mesh. *Wang et al.* [9] proposed a Marginalized Graph-AE for spectral graph-clustering on a generalized latent feature space given by the output of the last layer of a Graph-AE.

Limited work has been conducted on Graph-Based Convolutional Autoencoders (Graph-CAEs), which utilize local features extracted using graph convolution operators to reconstruct graph signals and further learn generalized local latent feature representation. While the aforementioned examples show great promise to generalize graph features globally, in this paper, we present insight into the design of a *Graph-CAE*, which learns latent representation on localized eigen-graph in the spectral domain using graph-based convolution, and more importantly learning generalized features locally using a Convolutional Autoencoder (CAE).

2. METHODOLOGY

2.1. Overview

In this section we discuss the components of the proposed Graph-CAE, namely the convolution units and pooling layers. Convolution in irregular domains poses a challenge to researches, since there is no guarantee of the spatial regularity that traditional methods assume. Pooling is also precluded by the spatial-irregularity property of graphs. Instead, we utilize graph signal processing approaches to generalize convolution and pooling operations to the graph representation of the irregular spatial domain problems.

2.2. Convolution on Graphs

An AE is a neural network that learns more compact representations of its input. More formally, if x is a data point and $f(x)$ is an AE, then $f(x) = r$ where r is a reconstruction of the input x; that is, $r \approx x$. Ideally r should approximate x as closely as possible. An AE consists of two parts: an encoder g and a decoder f: $f(x) = h(g(x)) = r$. The function g is able to encode more compact, generalized features, which was the original purpose of the AEs. The function h reconstructs the image from the encoding. AEs are more often used for generative modeling [10, p.499], as with VAE [11]. The convolution operator in the CAE enables the learning of generalised features locally.

CNNs operate on the assumption that input data are regularly structured. Two-dimensional images, for example, can be represented as a matrix of pixel intensities. Kernels detect features by exploiting the statistical properties of the grid, namely stationarity and compositionality [4]. Kernels can only operate on regular structures such as grids, and generalized graphs do not share this property. For similar reasons, multi-scale dyadic clustering or pooling is non-trivial [5]. Figure 1 illustrates this fact.

The challenge is how to define convolution and pooling in the graph domain. Over the last half-decade these problems have been subjected to greater attention [5]. In the direction of convolution, the main intuition is that the graph embeds knowledge of the spatial relationship between vertices. One such graph-based approach is to transform the spatial graph-signals into the frequency domain of the graph Laplacian. Convolution theorem states that convolution in the spatial domain can be expressed as a multiplication in the frequency domain [12]. Spectral filters, like kernels in the regular domain, have more compact support than the signal. A learned filter approximates the behavior of convolution, accentuating and attenuating signals in the frequency space.

Let $\mathcal{G} = \langle V, E\rangle$ denote a connected and undirected graph with $n_v \in \mathbb{N}_{>0}$ vertices $V = \{v_0, v_1, \ldots, v_{n_v-1}\}$ and pairs of vertices corresponding to edges $E = \{\langle v_i, v_j\rangle \mid \omega(\langle v_i, v_j\rangle) > 0\}$ where $\omega : E \rightarrow \mathbb{R}_{>0}$. The vertex signal is a function $f_c : V \rightarrow \mathbb{R}^c$ where c is the number of channels on the graph. The edge weights are described by an $n_v \times n_v$ weight matrix W, where $W_{i,j} = \omega(\langle v_i, v_j\rangle)$. The Laplacian matrix is $\mathcal{L} := D - W$, where D is an $n_v \times n_v$ diagonal matrix where the ith entry is a sum of the edge weights incident to the ith vertex; stated formally: $D_{i,i} = \sum_{j=0|j\neq i}^{n-1} W_{i,j}$.

Eigenvalue decomposition on $\mathcal{L}$ yields $U^\top \Lambda U$, since $\mathcal{L}$ is square and symmetric. U is an $n_v \times n_v$ matrix of orthonormal, column eigenvectors. Transposition and multiplication with the $n_v \times n_c$ graph signal $f_c(V)$ transforms the graph-signals to the frequency domain. Forward Fourier transformation is defined as

$$\phi(f_c(V)) = U^\top f_c(V) = \tilde{f}_c(V), \tag{1}$$

while reverse Fourier transformation is defined as

$$\phi^{-1}(\tilde{f}_c(V)) = UU^\top f_c(V). \tag{2}$$

Thus $\phi^{-1}(\phi(f_c(V)))$ the identity function.

Graph convolution is a linear combination of a graph signal with a smooth filter $k \in \mathbb{R}^{n_v \times n_i \times n_o}$, where n_i and n_o are respectively the number of input and output filters. The filter is obtained by axis-aligned interpolation of the set of n_θ trainable parameters $\theta \in \mathbb{R}^{n_w \times n_i \times n_o}$. It yields a new mapping $\bar{f}_{n_c} : V \rightarrow \mathbb{R}^{n_c}$. Convolution is formally defined as

$$\tilde{f}^o_{n_o}(V) = U \sum_{i=0}^{n_i} U^\top f^i_{n_i}(V) \odot k^o_{n_o}, \tag{3}$$

where $1 \leq i < n_i$, $0 \leq o < n_o$, $f_c^{(n)}$ is the nth channel of the graph-signals of vertices V, and $k^o_{n_o}$ is the filter kernel for the oth channel of $\bar{f}_{n_o}$. The kernel k an interpolation $\mathcal{K}$ of n_θ tracked weights θ: $k = \mathcal{K}\theta$. Observe that, analogous to convolution, the filter's support is compact: $|\theta| \ll |k|$. The more compact the support, the smoother the filter. This decision also reduces the number of weights in the system while also exploiting statistical stationality. In our case, we use bicubic interpolation, but other methods are equally applicable, such as spline interpolation.

2.3. Pooling

Due to the Hadamard product (3) and the assumption of a fixed graph inhered in the Laplacian matrix, graph convolution does not pool or downsample the data. Pooling is useful for reducing computational burden, but more importantly feature generalization. In Graph-CNNs, pooling is implemented as a separate layer, with a graph-coarsening scheme. Graph coarsening introduces a hierarchical understanding of the graph, where each coarsening produces a higher level of understanding of the spatial relations of clusters of nodes.

Coarsening $\mathcal{G} = \langle V, E \rangle$ to $\hat{\mathcal{G}} = \langle \hat{V}, \hat{E} \rangle$ reduces $|V|$ and $|E|$ by a cut metric, a measure of the modularity of the clusters. This is an NP-hard problem with literature exploring the problem [13]. For this paper, we considered three coarsening strategies: Kron reduction, Graclus multilevel clustering and Algebraic Multigrid (AMG) clustering. Kron reduction has been used by Edwards and Xie in a Graph-CNN for human-action recognition [14]. The strategy assumes a bipartite graph, however, which does not apply to all graph inputs. Graclus coarsening was used by Defferrard et al. [15] and accommodates partitioned graphs well. Graclus is a greedy coarsening-scheme, lending itself to efficient implementation on a Graphics Processing Unit (GPU) [16]. Although, unlike AMG, Kron reduction and Graclus do not provide a solution for *un*coarsening, which is absolutely necessary for an autoencoder. For this reason, we decided on AMG, which like Graclus can be efficiently implemented on a GPU and has already been used successfully in a Graph-CNN [17]. Figure 2 demonstrates a two-level pooling using AMG with coarsening factor $\beta = 0.05$.

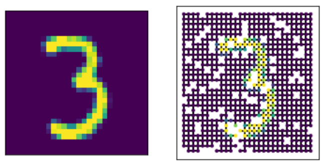
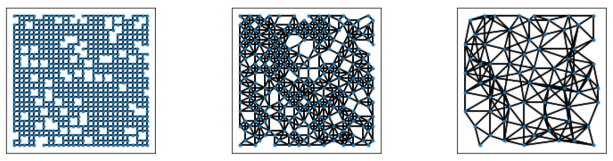


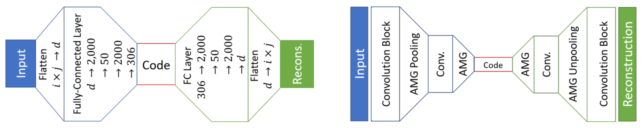
Fig. 1. The same image represented as a two-dimensional image and an irregular graph. The translational property does not exist for graphs, as there is no guarantee of spatial regularity, precluding the application of CNNs.



(a) Level 0 pooling. (b) Level 1 pooling. (c) Level 2 pooling.

Fig. 2. Our neural network implements two pooling layers with a coarsening factor of 0.05. The level 0 pooling is our subsampled graph, the input of the network. After level 1 pooling, the number of nodes in the graph is reduced from 700 to 176. After level 2 pooling, the number of nodes is further reduced to 66.

The difference between our model and Guo *et al.*'s is the connectedness of the autoencoder: ours uses convolutional layers, whereas theirs uses stacked, fully-connected autoencoders. Additionally, we use AMG pooling to compact the feature space (see Figure 3). Our encoder consists of a convolutional block, a pooling layer, another convolutional block and another pooling layer. Our decoder is a mirror-image of the encoder.



(a) Guo *et al.*'s model. (b) Our model.

Fig. 3. A graphical comparison between Guo *et al.*'s model and ours. The proposed model uses convolutional layers, whereas Guo *et al.* use fully-connected layers.

3. EXPERIMENTAL RESULTS

Our Graph-CAE was implemented in TensorFlow 1.4.0 on an NVIDIA GeForce 1080Ti. Unless otherwise modified for experimental purposes, we fixed our hyperparameters to the following: The neural network was optimized using the Adam optimizer with a learning rate of 0.001 and a batch-size of 256 samples. For pooling, the coarsening factor was 0.05. The dropout factor was 20%. We used 10 tracked weights, 10 output filters for each convolution layer in the first convolution block and 20 in the second, and one convolutional unit per block. To measure the autoencoder's performance, we used the Charbonnier loss-function (4),

$$L(x, \tilde{x}) = \sqrt{x^2 - \tilde{x}^2} + \epsilon^2 \tag{4}$$

where ϵ is a small, insignificant value intended to prevent a vanishing gradient, which in our case was set to 0.001.

For our experiments, we used the MNIST dataset, given its adoption as a benchmark dataset. The testing and validation sets were 45,000 and 5,000 images large respectively. Before training the neural network, the data was converted to a grid and irregularized by removing a random subset of nodes from the grid (see Figure 1). The graph was fixed across all experiments.

Adding convolution units increases the capacity of the neural network. The requisite capacity for good encoding (Figure 4) was exceeded when we increased the number of convolution units beyond 1 unit, leading to the model overfitting the training data. This low number is likely due to the simplicity of the MNIST dataset; had the dataset been CIFAR10 or CIFAR100, for instance, then we might see the neural network optimize at a different value.

Increasing the number of tracked weights in the neural network per convolution unit yielded diminishing returns

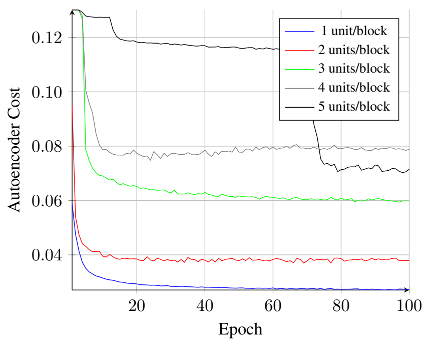


Fig. 4. The size of the convolution blocks, i.e., the number of convolution units per block, has a dramatic effect on the encoding. Increasing the number of convolution units per block increases the capacity of the neural network to such a point that it poorly generalizes to unobserved data.

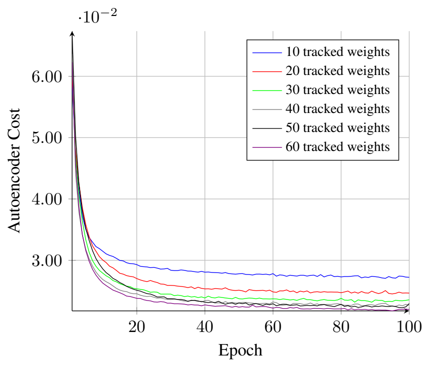


Fig. 5. Increasing the number of tracked weights in the neural network yields diminishing returns.

(Figure 5). Nonetheless, with 60 tracked weights the neural network performed at its best. Interestingly, increasing the number of weights did not have a very significant effect on encoding quality. More tracked weights increase the capacity of the network, but coarsens the filter. It is likely that we did not increase the number of weights to such a level that would impede generalization.

Varying the number of filters, too, had a lesser positive effect on the encoding quality. Increasing the number of filters in the first convolution block yielded diminishing returns (Figure 6), as there are a finite number of low level features in the MNIST dataset, and increasing the number of filters does not change this fact. Increasing the number of filters at a higher level, i.e., after the first pooling, likewise yielded diminishing returns (Figure 7), but the improvement in encoding quality is far greater than filter-size increases in the first convolution block. This is likely a consequence of the pooling, as more general features are learned here.

In terms of the effect on training time, the marginal increase in time for each additional convolution unit was on average 257 seconds, a factor of 1.68. Every additional 10 filters

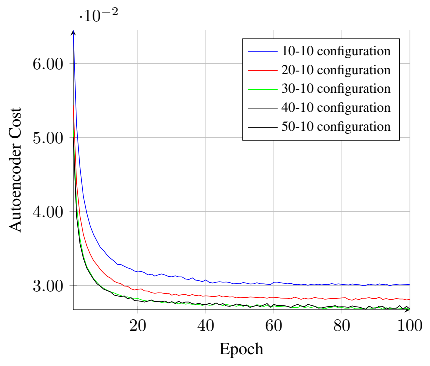


Fig. 6. We increased the number of filters in the convolution units of the first convolution block from 10 to 60. Like increasing the number of tracked weights, this yields diminishing returns.

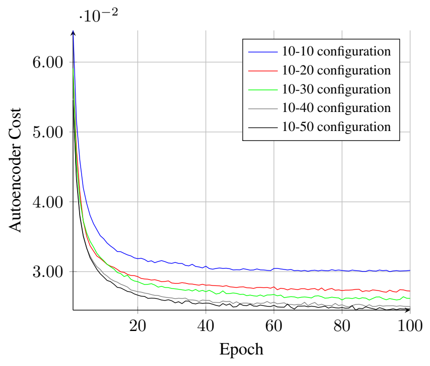


Fig. 7. We increased the number of filters in the convolution units of the first convolution block from 10 to 60. Like increasing the number of tracked weights, this yields diminishing returns. However, the number of filters in the units of the second block have a greater effect on the encoding quality than the number of filters in the units of the first block.

in the units of the first convolution block increased the time to train by 180 seconds on average, a factor of 1.34, and 204 seconds for each additional 10 filters to the units in the second block, a factor of 1.47. Increasing the number of weights did not affect the time to train significantly.

4. CONCLUSION

In this paper we examined convolutional autoencoding in irregular domains, specifically graphs. After describing the components of the neural network, we investigated the contributions to encoding quality by various aspects of the architecture. The number of convolutional layers in the neural network had a particularly stark effect on the encoding quality, as too many lead to too high capacity causing overfitting on the training set and poor generalization.

5. REFERENCES

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] Joan Bruna, Wjciech Zaremba, Arthur Szlam, and Yann LeCun, "Spectral Networks and Locally Connected Networks on Graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[4] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, July 2017.

[5] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, May 2013.

[6] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[7] Yiluan Guo, Hossein Nejati, and Ngai-Man Cheung, "Deep Neural Networks on Graph Signals for Brain Imaging Analysis," 2017.

[8] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia, "Deformable Shape Completion with Graph Convolutional Autoencoders," 2017.

[9] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang, "MGAE: Marginalized Graph Autoencoder for Graph Clustering," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17*. ACM Press.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT press, 2016.

[11] Carl Doersch, "Tutorial on Variational Autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[12] Ronald Newbold Bracewell and Ronald N Bracewell, *The Fourier Transform and Its Applications*, vol. 31999, McGraw-Hill New York, 1986.

[13] Cédric Chevalier and Ilya Safro, "Comparison of coarsening schemes for multilevel graph partitioning," in *Learning and Intelligent Optimization*, Thomas Stützle, Ed., Berlin, Heidelberg, 2009, pp. 191–205, Springer Berlin Heidelberg.

[14] Michael Edwards and Xianghua Xie, "Graph-Based CNN for Human Action Recognition from 3D Pose," *Deep Learning in Irregular Domains Workshop, British Machine Vision Conference*, 2017.

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[16] Bas Fagginger Auer and Rob H Bisseling, "Graph Coarsening and Clustering on the GPU," *Graph Partitioning and Graph Clustering*, vol. 588, pp. 223, 2012.

[17] Michael Edwards and Xianghua Xie, "Graph-Based Convolutional Neural Network," in *Proceedings of the British Machine Vision Conference*, 2015.