Graph-based Isometry Invariant Representation Learning: Supplementary material

Renata Khasanova [1] Pascal Frossard [1]

In the supplementary material, we present in more detail the training procedure of the TIGraNet and define the partial derivatives, required to compute back-propagation through the newly introduced layers. We also provide a more complete analysis of the network behavior with additional experiments on the small MNIST-012 dataset.

1. Training

1.1. Back-propagation details

We use supervised learning and train our network so that it maximizes the log-probability of estimating the correct class of training samples via logistic regression. Overall, we need to compute the values of the parameters in each convolutional and in fully-connected layers. The other layers do not have any parameter to be estimated. We train the network using a classical back-propagation algorithm and learn the parameters using ADAM stochastic optimization (Kingma & Ba, 2014).

We provide more details here about the computation that are specific to our new architecture. We refer the reader to (Rumelhart et al., 1988) for more details about the overall training procedure. The back-propagation in the spectral convolutional layer is performed by evaluating the partial derivatives with respect to the parameters $\alpha : \alpha \in \mathbb{R}^{K_{l-1} \times M}$ of the spectral filters, and to the parameters $\beta : \beta \in \mathbb{R}^{K_{l-1}}$ of the feature map construction. The partial derivatives read

$$\frac{\partial E}{\partial \alpha^l_{i,m}} = \sum_{k=0}^{K_{l-1}} \beta^l_k \left[\mathcal{L}^m|_{\mathcal{N}_i^{l-1}} \right] y_k^{l-1} \frac{\partial E}{\partial z_i^l}, \tag{1}$$

$$\frac{\partial E}{\partial \beta^l_j} = \sum_{m=0}^{M} \alpha^l_{i,m} \left[\mathcal{L}^m|_{\mathcal{N}_i^{l-1}} \right] y_j^{l-1} \frac{\partial E}{\partial z_i^l}, \tag{2}$$

where E is the negative log-likelihood cost function, $z_i^l = y_i^l$ is the output feature map of layer l, K_{l-1} denotes the number of feature maps at the previous layer of the network, M is the polynomial degree of the convolutional filter and $\mathcal{L}$ is the Laplacian matrix. Then, we further need to compute the partial derivatives with respect to the previous feature maps as follows

$$\frac{\partial E}{\partial y_j^{l-1}} = \beta^l_j \sum_{m=0}^{M} \alpha^l_{i,m} \left[\mathcal{L}^m|_{\mathcal{N}_i^{l-1}} \right] \frac{\partial E}{\partial z_i^l}. \tag{3}$$

Our new dynamic pooling layers, as well as our statistical layer do not have parameters to be trained. Similarly to the max-pooling operator our dynamic pooling layer permits back-propagation through the active nodes since the gradient is 0 for the non-selected nodes and not zero for the chosen ones. Further, the statistical layer back-propagates the gradients as follows:

$$\frac{\partial E}{\partial t_{i,k}} = \frac{1}{N} \frac{\partial E}{\partial \mu_{i,k}}, \tag{4}$$

$$\frac{\partial E}{\partial t_{i,k}} = \frac{2(N-1)}{N^2} \sum_{i=1}^{N} (t_{i,k} - \mu_{i,k}) \frac{\partial E}{\partial \sigma^2_{i,k}}, \tag{5}$$

where $\mu_{i,k}, \sigma^2_{i,k}$ are the inputs to the first fully-connected layer and the outputs of the statistical layer. The derivatives $\partial E / \partial \tilde{z}_i$ are then computed as:

$$\frac{\partial E}{\partial \tilde{z}_i} = \sum_{k=0}^{K_{max}} \frac{\partial E}{\partial t_{i,k}} \frac{\partial t_{i,k}}{\partial \tilde{z}_i}, \tag{6}$$

where $\partial t_{i,k} / \partial \tilde{z}_i$ are simply the derivatives of Chebyshev polynomials (Shuman et al., 2011) with maximum order K_{max}. Please note that we use the non-linear absolute function $|t_{i,k}|$ before statistical layer, therefore, the gradient at $t_{i,k} = 0$ is not defined. In practice, however, we set it to 0, which gives us a nice property of encouraging some feature map values to be 0 and favors sparsity.

Finally, the parameters of the fully-connected layers are trained in a classical way, similarly to the training of fully-connected layers in ConvNet architectures (Rumelhart et al., 1988).

[1]Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. Correspondence to: Renata Khasanova <renata.khasanova@epfl.ch>, Pascal Frossard <pascal.frossard@epfl.ch>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

1.2. Filter initialization

The initialization of the system may have some influence on the actual values of the parameters after training. We have chosen to initialize the parameters $\alpha_{i,m}^{l}$ of our spectral convolutional filters so that the different filters uniformly cover the full spectral domain. We first create a set of Z overlapping rectangular functions $w(\lambda, a_i, b_i)$

$$w(\lambda, a_i, b_i) = \begin{cases} 1 & \text{if } a_i < \lambda < b_i, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The non-zero regions for all functions have the same size, and the set of functions covers the full spectrum of the normalized laplacian $\mathcal{L}$, i.e., $[0, 2]$. We finally approximate each of these rectangular functions by a M-order polynomial, which produces a set of initial coefficients $\alpha_{i,m}^{l}$ that are used to define the initial version of the spectral filter $\mathcal{F}_i^l$.

Finally, the initial values of the parameters β in the spectral convolutional layer are distributed uniformly in $[0, 1]$ and those of the parameters in the fully-connected layers are selected uniformly in $[-1, 1]$.

2. TIGraNet Analysis

We analyze the performance of our new architecture on the MNIST-012 dataset. We first give some examples of feature maps that are produced by our network. We then illustrate the spectral kernels learned by our system, and discuss the influence of dynamic pooling.

We first confirm the transformation invariant properties of our architecture. Even though our classifier is trained on images without any transformations, it is able to correctly classify rotated images in the test set, since our spectral convolutional layer learns filters that are equivariant to isometric transformations. We illustrate this in Fig. 1, which depicts several examples of feature maps y_i^2 from the second spectral convolutional layer for randomly rotated input digits in the test set. Each row of Fig. 1 corresponds to images of a different digit, and we see that the corresponding feature maps are very close to each other (up to the image rotation) even when the rotation angle is quite large. This confirms that our architecture is able to learn features that are preserved with rotation, even if the training has been performed on non-transformed images. Despite important similarities in feature maps of rotated digits, one may however observe some slightly different values for the intensity. This can be explained by the fact that rotated versions of the input images may differ a bit from the original images due to interpolation artifacts.

Fig. 2 then shows the spectral representation of the kernels learned for the first two spectral convolutional layers

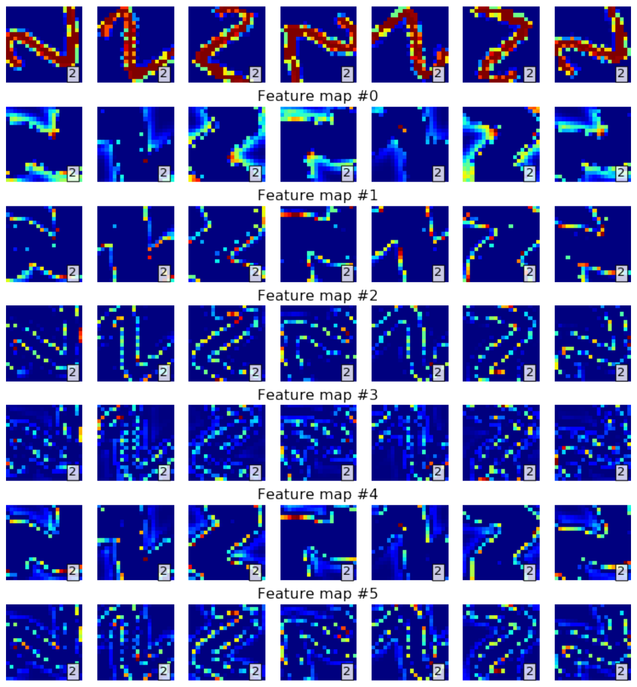

Figure 1. **Feature maps** from the second spectral convolutional layer for test images that are rotated versions of an image of the digit '2'. The predicted label for each of the images is further shown in the right bottom corner of each image.

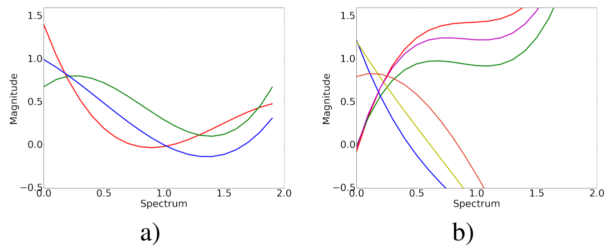

Figure 2. **Sample trained filters** in the spectral domain for (a) first and (b) second convolutional layers. Different colors represent different filters on each of the layers.

of our network. As expected, the network learns filters that are quite different from each other in the spectral domain but that altogether cover the full spectrum. They permit to efficiently combine information in the different bands of frequency in the spectral representation of the input signal. Generally, the filters in the upper spectral convolutional layers are more diverse and represent more complicated features than those for the lower ones.

Finally, we look at the influence of the new dynamic pooling layers in our architecture. Recall that dynamic pooling is used to reduce the network complexity and to focus on the representative parts of the input signal. Fig. 3 depicts the intermediate feature maps of the network for sample test images. We can see that after each pooling operation

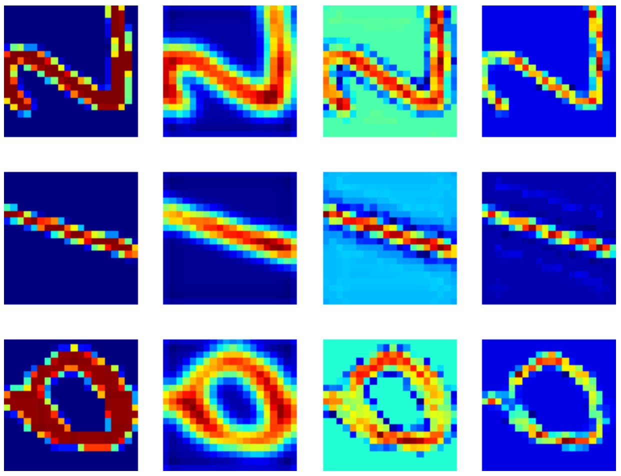

Figure 3. **Feature maps after pooling** Each row shows different digits. The left most column depicts the original images, while the other columns show the features maps after dynamic pooling at the first, second and third layers respectively. The degree of the polynomial filters has been set to $M = 3$ for each layer in this experiment.

the signal is getting more and more sparse, while preserving the structure of the data that is important for discriminating images in different classes. That shows that our dynamic pooling operator is able to retain the important information in the feature maps constructed by the spectral convolutional layers.

References

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *arXiv Preprint*, 2014.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Shuman, D. I., Vandergheynst, P., and Frossard, P. Chebyshev polynomial approximation for distributed signal processing. In *IEEE International Conference on Distributed Computing in Sensor Systems*, pp. 1–8, 2011.