Sparse Deep Learning

Merging double sparsity with Deep NN
Final Project - Sparse and redundant representations

Elad Hoffer

March 10, 2015

1 Introduction

Inspired by their success in signal processing, we aim to leverage recent techniques to constrain a network to learn and use sparse representations in order to possibly improve computational efficiency and accuracy. This can be done by using a structured dictionary (such as one acquired by DCT or wavelets) to create intermediate sparse representations of the input, and then learning their coefficients using the network weights. These could then be constrained to have sparse values, effectively mimicking the double-sparsity model introduced by [7]. To investigate the usage of these technique, we will use known vision benchmarks to compare sparsity of the representations attained, the model accuracy and computational cost to known results achieved using deep convolutional networks.

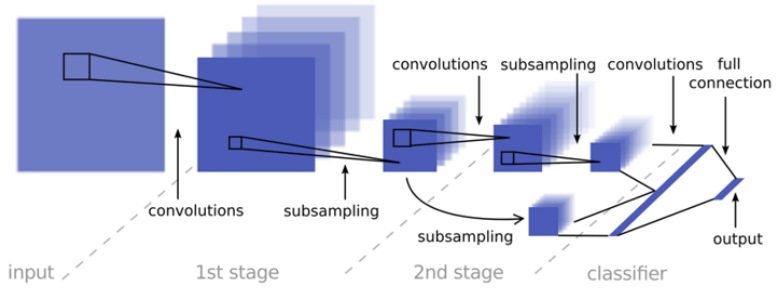
2 Deep learning overview

Deep learning is a set of machine learning algorithms based on neural network. These algorithms mostly concern multi-layered hierarchies (hence "Deep") that aim at finding abstract features representations from vast information. These representations can then be used for any feature based learning schemes such as classification.
The most widely used optimization technique used for neural network training is "Stochastic Gradient Descent" (SGD), which is performed on the network weight's gradients, after these were calculated using the *Back-Propagation* procedure.
Despite of their performance on many tasks, much is yet unknown on what makes these models so successful [9]. Recent usages indicated that some of this success is

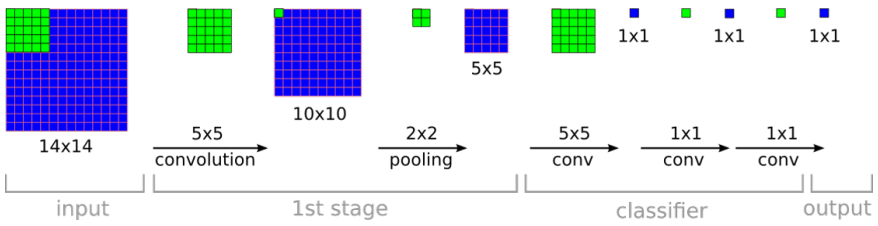
due to the usage of sparse intermediate feature embedding [8, 5], although those are generally not explicitly sought for.

2.1 Convolutional Neural Networks (CNN)

Deep convolutional neural network 1 represent the main approach of deep learning in computer vision tasks[3]. The main premise is that stationary properties of images allows the use of a reduced set of parameters that needs to be learned. Convolutional layers consists of a set of trainable weight kernels, that produce their output value by cross-correlating with input data 1b. This way, every spatial patch in the image is weighted using the same shared kernels.
Another main layer of CNNs is the pooling layer. Pooling layers are used to reduce the dimensionality of the input while gaining scale and shift invariance to small amounts. This is done by simply pooling (or down-sampling) spatial regions of the input by taking the average (Average-Pooling), max (Max-Pooling) or other variant ..
Most CNNs are comprised from a few (2-5) layers of Convolution+Pooling followed up by a fully-connected layers (1-2) and a classifier.



(a) CNN Structure



(b) Feature maps computation

Figure 1: Convolutional neural network

3 Double-Sparsity Deep Learning

We now begin to assemble the building blocks for our proposed double-sparsity deep learning model.

3.1 Visual Dictionaries

For many computer-vision tasks, a dictionary of visual atoms is used to represent the data. Significant amount of research has also shown that the use of sparse dictionary representations can lead to efficient and accurate results in various tasks. We will now follow the notation

$$x = D\gamma$$

where x is the represented data, D is the underlying dictionary, and γ are the dictionary coefficient.
There exist two main approaches for choosing the dictionary D. The first method is to use predefined dictionaries such as ones created by DCT, Wavelets, Curvelets etc. These structured dictionaries have the benefit of needing to optimize only the coefficients (low number of variables) and exhibit low compute time (as the number of computation needed is sub-quadratic, usually of $O(n \log n)$ complexity). The second main approach is using machine learning techniques to train and infer the needed dictionary D. This approach requires an extensive period of training and longer inference time (as the dictionary has no pre-determined structure), but usually achieves better results on the final objective.
We note that convolutional networks can be grasped as a dictionary learning method, where each convolutional kernel is in fact a dictionary atom. With every feed-forward in a CNN, a correlation operation is performed on each region of the input against the current kernels. Thus we essentially learn a set of kernels (dictionary) that can best describe our data at hand.

3.2 Double sparsity model

The double sparsity model suggested by [7] addresses the learned vs structured dictionary dilemma by suggesting a tradeoff between the two. By using a pre-defined structured dictionary Φ , and an additional set of sparse coefficients A, the needed dictionary D is formed such that

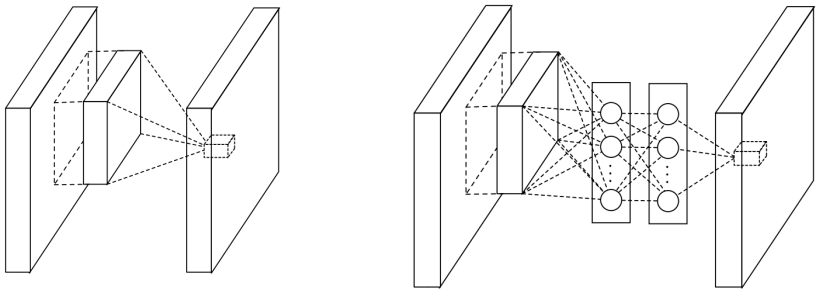
$$D = \Phi A$$

where A is constrained to be sparse - $\|a_i\|_0^0 < p$ for some p, where a_i are the columns of A. This leads to data representation

$$x = \Phi A\gamma$$

where both Φ and A are sparse . The double sparsity model allows us to learned useful, adaptive, dictionaries with good results, but with a lower computational cost that accompanies structured dictionaries.
Using this model on denoising and inpainting tasks, the authors demonstrated good results while allowing for efficient implementation both for training and inference. We note that original paper used a K-SVD variant to learn the dictionary coefficient, and as such is not applicable (at least not trivially) to the training of convolutional networks that we discuss here. Nevertheless we will demonstrate shortly how the core ideas used for the double-sparsity model can be applied to deep network and on different objective - classification of objects.

3.3 Network-In-Network Model



(a) Linear convolution layer (b) Network-In-Network layer

Figure 2: Convolutional layer to NiN module comparison

One recent model of convolutional network is the *Network-In-Network* model [4]. This model replaces the usual convolutional layer building-block 2a, with the *conv-mlp layer* 2b, which is essentially a convolution $k \times k$ layer followed by two successive 1×1 convolutional layers. This essentially creates feature maps that are linear combinations of the matching pixels in previous layer. It is important to note the between each such layer is a non-linear $ReLU$ activation layer. The complete NiN model (3) includes 3 instances of such layers with intermediate poolings, followed by a global pooling layer and a final classifier.

3.4 Putting it all together

Returning to our original topic, we can now see how the NiN architecture can fit nicely into the double-sparsity model to produce sparse, low-dimensional and partially structured kernels for deep convolutional networks. Replacing the learned

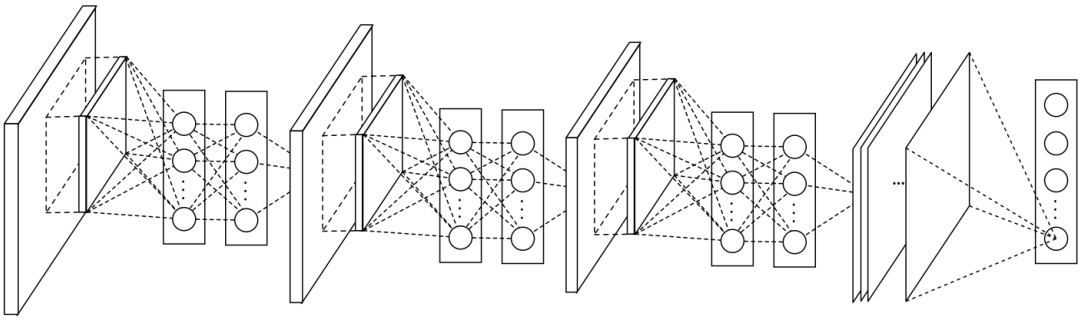


Figure 3: Network-In-Network structure

kernels $k \times k$ in the NiN module 2b, and training only the following 1×1 kernels, we essentially create

$$D_l = \Phi_c W_l$$

where $\Phi_c \in \mathbb{R}^{(F_l \times k \times k) \times F_c}$ are predefined and fixed DCT dictionary atoms of spatial size $k \times k$ with $|\Phi_c| = F_l \cdot F_c$, that are using F_l feature maps to produce F_c new feature maps using the DCT kernels, and $W_l \in \mathbb{R}^{F_c \times F_{l+1}}$ are trainable weights that will produce the subsequent layer. Thus we are effectively creating a double-sparsity induced deep network, where we can control the sparsity of the coefficient by regularizing the L_1 norm of the weights (as a smooth replacement for the required low L_0 objective). Training the newly formed network will create create layers that are formed from structured dictionaries such as DCT or wavelets, while allowing their coefficients to be optimized by the SGD procedure.

4 Experiments

The double-sparsity induced model based on the NiN is built by pre-defining all $k \times k$ convolutional layers (with $k > 1$) to be a fixed over-complete DCT dictionary 9. These DCT kernels are not trained, and the only trainable parameter is the bias term for each feature map. As such, the new model has much less trainable parameter which amount to about $\frac{1}{7}$th of the original NiN model 4.

	Original NiN model	Double-Sparsity model
#Parameters	966986	160010
Ratio measure	1	0.165

Figure 4: Comparing number of trainable parameters

The network is trained on classification task using the negative-log-likelihood loss defined by

$$E_{NLL}(y, y_t) = \sum_{i=1}^{\#Classes} y_t^{(i)} \log \frac{y_t^{(i)}}{y^{(i)}} = -\log y^{(c)}$$

where

$$y^{(i)} = \Pr(c = i|x), \quad y_t^{(i)} = \mathbb{I}[c = i]$$

Because the network is trained using the SGD algorithm, regularization is introduced to the optimization procedure by adding the needed regularized measured multiplied by a predefined coefficient. The used loss is then

$$Loss(x, y_t) = E_{NLL}\left(Net(x; w), y_t\right) + \frac{\mu}{2} \cdot \|w\|_2 + \zeta \cdot \|w\|_1$$

where μ and ζ are the L_2 and L_1 regularization coefficient respectively.
Training by SGD is done by following the simple training rule:

$$w_{t+1} = w_t - \epsilon \cdot \nabla_w Loss(x, y_t)$$

where ϵ is the *learning rate*. This will translate in our case to the expression:

$$w_{t+1} = w_t - \epsilon \cdot [\nabla_w E_{NLL} + \mu \cdot w + \zeta \cdot sign(w)]$$

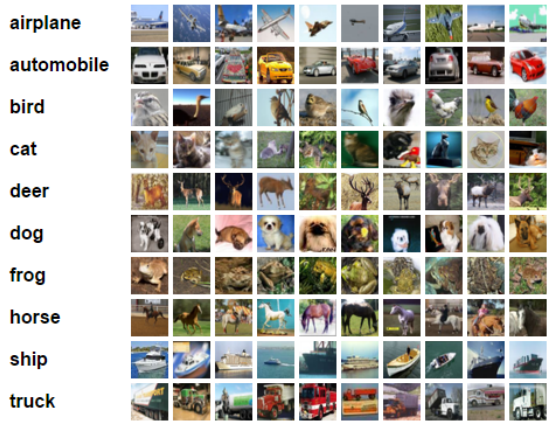
An additional *momentum* factor (noted as η) is usually introduced into the learning rule, producing

$$w_{t+1} = \eta \cdot (w_t - w_{t-1}) + (1 - \eta) \left[w_t - \epsilon \cdot \nabla_w E_{NLL} + \mu \cdot w + \zeta \cdot sign(w)\right]$$

To allow exact zero-valued weight coefficients, we replaced the L_1 update rule with the more definite form of

$$Shrink(w_t) = \begin{cases} w_t^{(i)} - \zeta \cdot sign(w_t^{(i)}) & \text{if } |w_t^{(i)}| > \zeta \\ 0 & \text{else} \end{cases}$$

Additional experiments were done using lower than 1 norms (0.5,0.75) which didn't prove useful and so were dropped from this discussion.
We experimented with 2 datasets. The first is *Cifar10* ([2]), consisting of 60000 32x32 color images of 10 classes (of which 50000 are used for training only, and 10000 for test only). The second is the *Street-View-House-Numbers (SVHN)* of [6] consisting of 600000 32x32 color images of house-number digits 0-9. Training was done using the mentioned SGD with L_1 regularization over the negative-log-likelihood objective and accuracy was check every epoch on the isolated test set. The deep double-sparsity model was implemented and trained using the Torch7 environment ([1]).



(a) Cifar10 Dataset

(b) SVHN Dataset

Figure 5: Datasets used in this work

5 Results

We now turn to compare the double-sparsity induced model with the original NiN results. We note that when replicating the results of the NiN paper, we received slightly lower accuracies, probably due to training technique and learning rate modification. We will henceforth compare our results to the replicated accuracies and to those appearing in the paper. Best results for the sparsity induced model were gained using L_1 coefficient $\zeta = 10^{-5}$ and those appear on 6.

Dataset	Original NiN model	Double-Sparsity model
Cifar10	87.6%	74.2%
SVHN	97.65%	95.47%

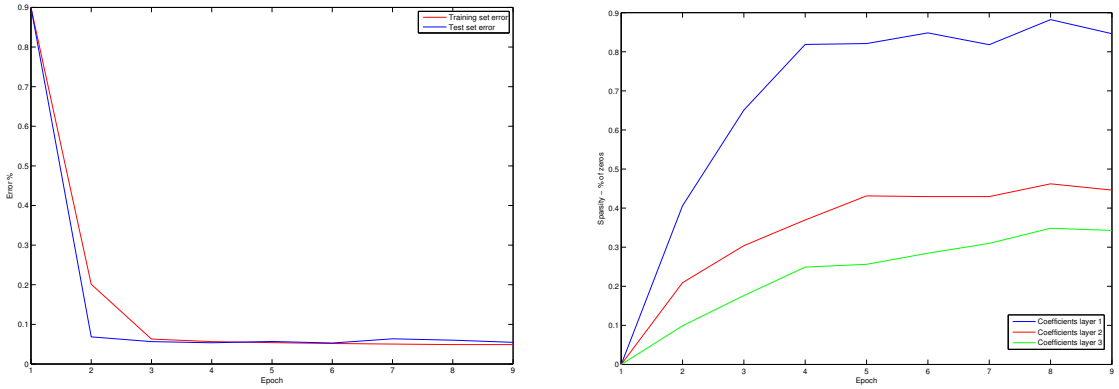
Figure 6: Classification accuracy (no data augmentation)

We can accumulate the number of zero valued parameters in the coefficient layers (layers following a DCT convolution layer) to judge how sparse these layers are 7.

Dataset	Layer #1 Sparsity	Layer #2 Sparsity	Layer #3 Sparsity
Cifar10	25.47%	10.43%	44.5%
SVHN	84.63%	44.62 %	34.28%

Figure 7: Percent of zero coefficient for each coeff layer

We can see the performance drop varies with respect to each dataset, and whereas the Cifar10 dataset suffered a significant 14% drop, the SVHN had only 2% percent drop in accuracy while using very low number of non-zero parameters. These results shows not only the complexity difference between the two datasets, but also the ability of the double-sparsity network to achieve descent accuracy with pre-defined kernels and low number of parameters. We feel that further investigation and experiments can lead to competitive results while gaining huge computational efficacy. Another notable finding is available when taking a closer look at the training result for the SVHN task at 8a 8b. Here we can see that in the training process even after the error has reached its lower bound, we can further train the weights to achieve higher level of sparsity, that are useful for later computational efficiency at inference time.



(a) Training and test error

(b) Sparsity of the coefficients layers

Figure 8: SVHN training values per epoch

5.1 Visualizing low-level kernels

One known way to visualize the outcome of a trained convolutional network is to view the form of kernels in its first layer. As this layer used to extract the most basic aspects in an image, its kernels are usually some form of gabor filters, which are known to capture low-level features of visual information such as edges, orientation, and color nuances. By visualizing the kernels D_1 in our model which

amounts to

$$D_1 = W_1^T \cdot \Phi_1,$$
$$W_1 \in \mathbb{R}^{F_l \times F_{l+1}},$$
$$\Phi_1 \in \mathbb{R}^{F_l \times F_l \times 5 \times 5}$$

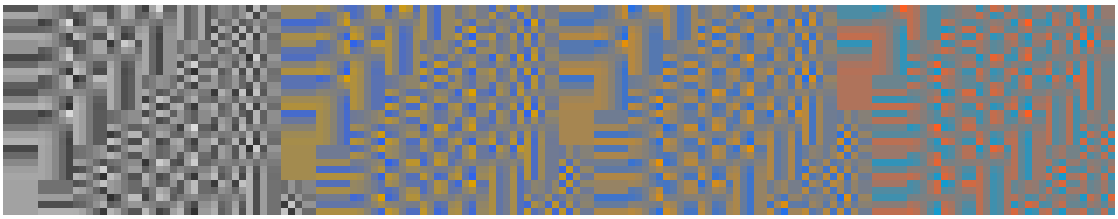
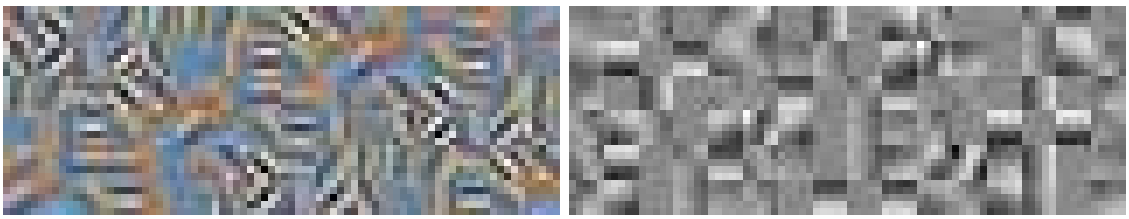


Figure 9: Overcomplete DCT 5×5 dictionary used

Where Φ_1 is the overcomplete 5×5 DCT dictionary 9, and W_1 is the first set of trainable parameters.
We can observe that indeed, the outcome of this double-sparse dictionary forms a set of Gabor-like filters 10. In fact, we can use this visualization to analyze the underlying representation needed for each dataset. Whereas the Cifar10 dataset is a more complex dataset with various objects and color dependant scenery, the data needed to classify a SVHN digit is color independent and is much simpler. We can view both these qualities in the visualized kernels, where the Cifar10 kernel have a wide variety of orientations and color-specific details 10a, the SVHN filters are much more simpler, with no color information available in them 10b.



(a) Learned dictionary for the Cifar10 dataset (b) Learned dictionary for the SVHN dataset

Figure 10: Learned low-level filters

5.2 Computational cost

In this work, the double sparsity layers were created by simply using a convolutional layer with pre-defined and fixed (untrainable) DCT filters. This was used to

simplify implementation, but in fact a much more optimized usage can be creating by noting that

- Structured dictionaries are faster to compute - as seen in [7] we can accelerate inference time using the known structure of the predefined dictionary.
- DCT kernel are separable - we can perform 1-dimensional convolution twice and allow for much better performance.
- Sparse coefficients can be translated to fewer kernels - we can see in 10b that some filters can be completely removed to allow faster inference time.

6 Summary and possible future research

In this work it was shown that the double-sparsity model can be embedded in the deep convolutional network framework. This can potentially allow for sparse-oriented models with lower number of tunable parameters, and structured dictionaries that can be efficiently used to accelerate computations. Further research is needed to improve upon those findings and allow better results that are competitive with state-of-the-art results of conventional deep networks. Another interesting direction is testing these models in image processing tasks such as denoising and inpainting, where deep learning has lately been shown to provide good results.

References

[1] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[2] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[3] Yann LeCun Yann LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010.

[4] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[5] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[6] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning.

[7] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Double sparsity: Learning sparse dictionaries for sparse signal approximation. *Signal Processing, IEEE Transactions on*, 58(3):1553–1564, 2010.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[9] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.