# Interpreting a Decomposition Method for Sparse Nonconvex Optimization

**Shane T. Barratt** [1] **Rishi Sharma** [1]

## Abstract

Optimization methods based on derivative information fail to produce sparse solutions to nonconvex problems with an additive $\ell_1$ penalty, as the objective function is not differentiable. In this paper, we derive an optimization algorithm, based on the alternating direction method of multipliers updates, for producing (approximate) solutions to nonconvex problems with an $\ell_1$ penalty. We additionally propose a variant of the algorithm for when a stochastic gradient method is used to optimize the nonconvex objective, which is particularly relevant in large-scale machine learning problems. We interpret the resulting algorithm in the context of neural networks, and connect it to the synaptic pruning that occurs in the human brain during sleep. We apply our algorithm, and its variant, to the problem of learning a neural network with sparse weight matrices, a problem that is hard to solve with gradient and proximal-gradient based techniques.

## 1. Introduction

Neural connections in the human brain are sparse; though we have billions of neurons, each neuron is connected to at most ten thousand other neurons. Insofar as we take inspiration from learning in the human brain, it guides us to construct models with an inductive bias for sparsity. Further, it has been seen that enforcing a parsimonious representation of a prediction function can act as an effective regularizer, an application of the principle of Occam's Razor.

In this work we derive an optimization algorithm for producing sparse (approximate) solutions to general nonconvex optimization problems, noting that similar efforts have been made recently in the context of neural networks (Kiaee et al., 2016; Zhang et al., 2018). We also explore the possibility that our algorithm, which is based on the alternating direction method of multipliers (ADMM), helps explain the biological processes involved in recent discoveries about the synaptic pruning that occurs in human brains during sleep.

Our setting is the following optimization problem

$$\text{minimize} \quad l(x) + \lambda \|x\|_1 \tag{1}$$

where $x \in \mathbf{R}^n$ is our optimization variable and $l : \mathbf{R}^n \to \mathbf{R}$ is a differentiable nonconvex function. It is well known that the $\ell_1$ norm in the objective in (1) can induce sparsity in the vector $x$ (Donoho, 2006). This is an unconstrained optimization problem, but because the objective is not differentiable, standard gradient and hessian-based algorithms that assume differentiability, for example, gradient descent and Newton's method, will fail to produce sparse solutions. To alleviate this, we will decompose the problem into two sub-problems, one that is differentiable, and one that is easy to solve. To do this, we introduce the variable $z \in \mathbf{R}^n$ and convert (1) into the equivalent (equality-constrained) optimization problem

$$\begin{array}{ll} \text{minimize} & l(x) + \lambda \|z\|_1 + (\rho/2)\|x - z\|_2^2 \\ \text{subject to} & x = z \end{array} \tag{2}$$

where we have enforced that $x = z$ via the equality constraint, and introduced a penalty parameter $\rho > 0$. It is easy to see that this new problem is equivalent to (1). The augmented Lagrangian (Hestenes, 1969) for (2) is then

$$\mathcal{L}_\rho(x, z, y) = l(x) + \lambda \|z\|_1 + y^T(x - z) + \frac{\rho}{2}\|x - z\|_2^2 \tag{3}$$

where $y \in \mathbf{R}^n$ is the Lagrange dual variable. By introducing the variable $u = (1/\rho)y$, we can rewrite (3) as

$$\mathcal{L}_\rho(x, z, u) = l(x) + \lambda \|z\|_1 + (\rho/2)\|x - z + u\|_2^2 \tag{4}$$

plus a constant that does not depend $x$ or $z$. The alternating direction method of multipliers (ADMM) is an algorithm that solves convex optimization problems by decomposing them into easier to solve pieces (Gabay & Mercier, 1975; Boyd et al., 2011). Starting with initial iterates $(x_0, z_0, y_0)$, the ADMM updates for our $\ell_1$-regularized nonconvex prob-

[1]Department of Electrical Engineering, Stanford University, Stanford, CA. Correspondence to: Shane Barratt <sbarratt@stanford.edu>.

*Submitted to the ICML 2018 Workshop on Modern Trends in Nonconvex Optimization for Machine Learning*, Stockholm, Sweden, PMLR 80, 2018. 

**Algorithm 1** ADMM

**Input:** Initial iterates $x_0$, $z_0$, $u_0$
**repeat**
$x^{k+1} = \text{argmin}_x \left(l(x) + (\rho/2)\|x - z^k + u^k\|_2^2\right)$
$z_i^{k+1} = S_{\lambda/\rho}(x_i^{k+1} + u_i^k)$
$u^{k+1} = u^k + x^{k+1} - z^{k+1}$
**until** $x_k, z_k, u_k$ converged

lem are then

$$x^{k+1} := \underset{x}{\text{argmin}} \left(l(x) + (\rho/2)\|x - z^k + u^k\|_2^2\right) \quad (5)$$

$$z^{k+1} := \underset{z}{\text{argmin}} \left(\lambda\|z\|_1 + (\rho/2)\|x^{k+1} - z + u^k\|_2^2\right) \quad (6)$$

$$u^{k+1} := u^k + x^{k+1} - z^{k+1}. \quad (7)$$

It is worth noting that ADMM need not converge to a global, nor even local solution, even though it provably does in the convex case. However, the nice part about ADMM is that our problem is now decomposed into three updates, two of which are trivial. The $x$-update (5) requires solving a nonconvex optimization problem with a $\ell_2$ regularization term that is now differentiable, meaning we can use gradient and hessian-based methods. The $z$-update (6) is simply the proximal operator for the $\ell_1$ norm, which is given by

$$z_i^{k+1} = S_{\lambda/\rho}(x_i^{k+1} + u_i^k) \quad (8)$$

where $S_\kappa(a) := (a - \kappa)_+ - (-a - \kappa)_+$ is the soft thresholding operator. Finally, the $y$-update (7) is simply a vector addition. The full ADMM algorithm is summarized in Algorithm 1.

## 2. Sparse Neural Networks

Consider a neural network $f : \mathbf{R}^p \to \mathbf{R}^m$ parameterized by a vector $\theta \in \mathbf{R}^n$. Suppose we are given samples $\{(x_i, y_i)\}_{i=1}^N$ from a joint probability distribution. Fitting the neural network to these samples plus a $\ell_1$-regularization penalty, that imposes sparsity on the weights, fits perfectly into our framework. In the case of regression, our loss function is the $\ell_2$ loss, or $l(\theta) = \frac{1}{N}\sum_{i=1}^N \|f(x_i;\theta) - y_i\|_2^2$. In the case of classification, $y$ and $f(x;\theta)$ are probability distributions, and our loss function is the cross-entropy loss, or $l(\theta) = -\frac{1}{N}\sum_{i=1}^N \sum_{j=1}^m y_{ij} \log f_j(x_i;\theta)$.

To apply ADMM to this problem, we need to make two copies ($z$ and $u$) of the original network parameters $x$. The updates proceed as follows. The first network $x$ (the base network) is updated by minimizing $l$ plus an $\ell_2$ penalty (weighted by $\rho/2$) for deviating from $z - u$. This can be viewed as essentially a prior distribution on the weights of the network governed by the subtraction of the other two networks. The second network $z$ (the sparse network) is updated by soft-thresholding the network $x + u$, resulting in a (possibly) sparse and shrunken network $z$. The third network $u$ (the price network) is updated by adding the difference of the other two networks ($x - z$) to itself, and accounts for the difference in the two networks. Each network plays its own role until convergence, and in the end, the sparse network results in a sparse solution to the $\ell_1$-regularized optimization problem.

## 3. Connection to Dreaming

Sleep is ubiquitous among animals in the biological world, from fruit flies to birds to humans, suggesting that it serves an essential function in survival. Attempts to explain the neural process that makes sleep critical to survival have only recently begun to show promise (after all, periodic unconsciousness would appear to be a detriment not an asset). One newly developed explanation that has received attention is the synaptic homeostasis hypothesis (Tononi & Cirelli, 2006), which posits that the purpose of sleep is to prune synaptic connections between neurons, weakening the connections or removing them altogether. During waking life, new synaptic connections are constantly being formed in response to stimuli, but maintaining these synaptic connections requires energy. Thus, they argue that by pruning connections during sleep, our brain returns to energetic homeostasis (roughly speaking). Another explanation based on synaptic pruning suggests its use by the brain to improve its signal-to-noise ratio—removing extraneous connections that do not contribute to learning—thus explaining improved memory and cognition due to sleep (Walker, 2009).

While the biological mechanisms for the tagging and pruning of synaptic connections remain the subject of much research (Rogerson et al., 2014), the algorithm that this mechanism implements is still largely unknown. We believe our method for pruning neural networks can help guide investigation into a biologically plausible pruning algorithm. Though we are not arguing that the method itself is biologically plausible, we are offering this method as a computationally sensible starting point for investigation the biological mechanism.

Within this framework, we may view the process of dreaming due to re-stimulation of synaptic pathways during sleep as the process of calculating the error between the true weights and a pruned version in the update in (7). We may then view (6) as the process of synaptic tagging and pruning itself. We know that this process occurs repeatedly during sleep until the synaptic connections are pruned approximately back to pre-sleep levels, at which point we are ready to wake up and learn again. This suggests also that the synaptic pruning algorithm is run repeatedly until convergence or a stopping criterion is reached, as in Algorithm 2.

**Algorithm 2** SGD-ADMM($K$)

**Input:** Initial iterates $x_0$, $z_0$, $u_0$
**repeat**
  Perform $K$ SGD steps on $l(x) + (\rho/2)\|x - z^k + u^k\|_2^2$
  $z_i^{k+1} = S_{\lambda/\rho}(x_i^{k+1} + u_i^k)$
  $u^{k+1} = u^k + x^{k+1} - z^{k+1}$
**until** $x_k, z_k, u_k$ converged

## 4. Integration with Stochastic Gradient Descent

A standard technique to accelerate ADMM is to terminate the iterative method used to perform the (possibly expensive) $x$-update early. In the convex case, ADMM still converges with inexact $x$-updates. Though there are no guarantees for the nonconvex case, there are still reasons for why we might want inexact $x$-updates. One reason is that it results in more efficient updates, and thus might lead to faster convergence.

When we are solving large-scale nonconvex machine learning problems, our loss function is additive in all the training examples, or $l(x) = \sum_{i=1}^N l_i(x)$. If we were to fully optimize this loss function, we would heavily overfit to the training data. The batched stochastic gradient method (SGD) and its accelerated variants are extremely effective for minimizing loss functions of these forms, and has many other pleasing qualities, such as opportunities for parallelism and performing implicit regularization (Bottou et al., 2018). SGD is an anytime algorithm, so if we are using it for the $x$-updates, we can run it for a fixed number of steps, or perform a few passes over our training data, and then proceed onto our updates for $z$ and $u$. To further reduce overfitting, we can use a validation set and perform early-stopping every iteration based on our loss on the validation set. This procedure is summarized in Algorithm 2, and allows for $\ell_1$ regularization terms during neural network training, at only the cost of having three copies of the network parameters. (Note that only $x$ has to lie on the primary computing device, as the $z$ and $u$ updates are trivial $O(n)$ operations.)

## 5. Cardinality Constraints

The optimization problem with a cardinality constraint on the vector $x$

$$\begin{array}{ll} \text{minimize} & l(x) \\ \text{subject to} & \mathbf{card}(x) \leq c \end{array} \tag{9}$$

where $x \in \mathbf{R}^n$ is the optimization variable, $\mathbf{card}(x)$ counts the number of nonzero elements in $x$, and $l$ is a differentiable nonconvex function, also has a corresponding ADMM algorithm. Sparing the details of the derivation, see, *e.g.*, Boyd et al. (2011), the updates have the form

$$x^{k+1} := \operatorname*{argmin}_x \left(l(x) + (\rho/2)\|x - z^k + u^k\|_2^2\right) \tag{10}$$

$$z^{k+1} := \Pi_S(x^{k+1} + u^k) \tag{11}$$

$$u^{k+1} := u^k + x^{k+1} - z^{k+1} \tag{12}$$

where $S = \{x \mid \mathbf{card}(x) \leq c\}$ and $\Pi_S(\cdot)$ denotes projection onto $S$. Projection onto $S$ involves keeping the largest $c$ magnitude elements and zeroing out the rest. It is useful to note that the ADMM algorithm here is the exact same as in the $\ell_1$ case, but in this case, a fixed number of the elements of $x + u$ are zeroed out, rather than a dynamic number depending on $\lambda/\rho$. One can easily replace $S_{\lambda/\rho}$ in Algorithm 1 and Algorithm 2 with $\Pi_S$ and get the equivalent algorithm for cardinality-constrained minimization of a nonconvex objective. This can be useful in the context of neural networks, because we can directly optimize for neural networks that satisfy a "storage" constraint, or compress/prune a learned neural network while still attaining accuracy. This algorithm was applied to compressing pretrained networks (Kiaee et al., 2016; Zhang et al., 2018), and a similar algorithm was applied to imposing constraints that the weights are $n$-bit integers to perform compression (Leng et al., 2017).

## 6. Numerical Experiments

We now apply the ADMM algorithms to the problem of learning sparse neural networks, first to directly minimizing the training criterion, and then with the SGD algorithm to test generalization. All of the code for our experiments is available online, along with a PyTorch (Paszke et al., 2017) implementation that can be applied to any model with only a few lines[1]. (See also the Supplementary Materials for PyTorch pseudocode.)

### 6.1. Direct Minimization

Consider the following one-layer neural network $f : \mathbf{R}^n \to \mathbf{R}^m$

$$y = W_2\sigma(W_1 x + b_1) + b_2 \tag{13}$$

where $W_1$, $b_1$, $W_2$, and $b_2$ are the appropriate sizes. We first generate data from a random one-layer neural network with sparse weights, or where $W_1$, $b_1$, $W_2$, and $b_2$ are drawn from the standard normal distribution and each entry is randomly set to 0 with probability 80% to create a sparse neural network. We create a regression dataset of $N = 1000$ pairs $(x, y)$ where $x$ comes from the standard normal distribution and $y$ is given by (13) with i.i.d. standard normal noise. To attempt to reconstruct this sparse neural network, or a sparse approximation to it, we attempt solve the following

[1] [Link released upon acceptance]

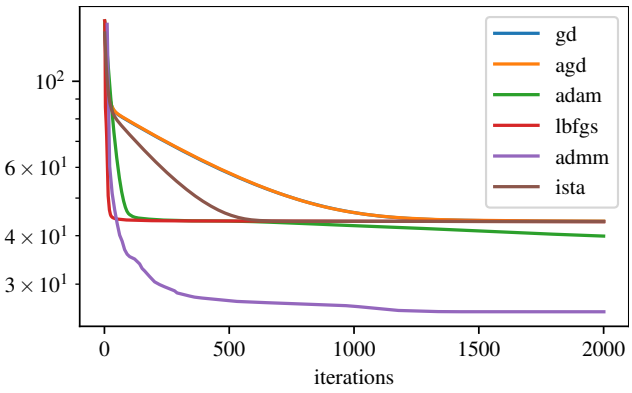


*Figure 1.* Loss per iteration for the six algorithms compared.

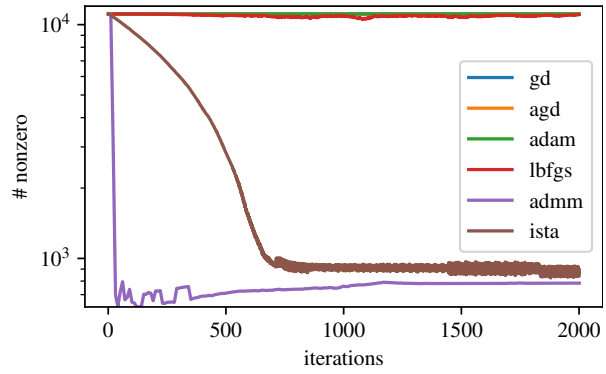


*Figure 2.* The quantity $\mathbf{card}(z)$ per iteration for the six algorithms compared.

$\ell_1$ regularized regression problem

$$\text{minimize} \quad l(\theta) + \lambda\|\theta\|_1 \tag{14}$$

where $\theta = [W_1, b_1, W_2, b_2]$ are the neural network weights, and $l(\theta) = \frac{1}{2N}\|f(x_i;\theta) - y_i\|_2^2$. We compare six methods—gradient descent, accelerated gradient descent, Adam, LBFGS, ISTA, and our ADMM-based method—on their ability to minimize (14) with $\lambda = .08$. To be clear, we do not care about generalization performance, but rather direct minimization of the loss function. For the three gradient-based methods, we ignore the non-differentiability when $\theta_i = 0$ and simply use what can be viewed as a subgradient (but is not technically a subgradient), or

$$g \in \nabla_\theta l(\theta) + \partial(\lambda\|\theta\|_1)$$

where $+$ denotes set addition. All the algorithms are implemented in the python library PyTorch.

The parameter settings of the six algorithms that were compared are described in the Supplementary Materials, however, we refer the reader to the source code for all of the details. We ran all of the algorithms for exactly 2000 steps. To be fair, for the ADMM method, we only allowed 2000 total inner iterations of LBFGS, so it had a similar wall-clock time. The loss per iteration for the four algorithms is displayed in Figure 1 and the number of nonzero elements per iteration is displayed in Figure 2. While the five compared methods fail to reduce the objective value below 48 or so, the ADMM-based method is able to reduce it all the way down to 36. Similarly, only ADMM and the ISTA method produce sparse solutions with about 8.3 % of $x$ being nonzero. From this experiment, we can conclude that an ADMM-based method is superior than many other common methods for direct minimization of a differentiable nonconvex loss plus an $\ell_1$ penalty.

## 7. Sparse Convolutional Neural Network

In order to evaluate the ability of our algorithm to learn sparse neural networks, as well as the generalization properties of Algorithm 2, we train sparse convolutional neural networks using the SGD-based ADMM on the MNIST dataset (LeCun, 1998). (MNIST is composed of 50k training examples and 10k test examples coming from 10 classes.) We use the following network topology

```
conv1 = Conv2d(1, 10, kernel_size=5)
conv2 = Conv2d(10, 20, kernel_size=5)
conv2_drop = Dropout2d()
fc1 = Linear(320, 50)
fc1_drop = Dropout()
fc2 = Linear(50, 10)
```

with max pooling and rectified linear activation functions. This network has 21480 parameters. For SGD, we use a batch size of 100 and a learning rate of .01. We evaluate the cardinality-constrained version of SGD-ADMM($K$) with $K = 50$ and set the overall allowed cardinality $c = 2148$, which corresponds to a 10x compression of the original network[2]. With this setup, we were able to achieve a test accuracy of 96.75 %, whereas the original network achieves a test accuracy of 99 %. Although lower, the test accuracy is impressive for a network with only 2148 parameters.

## 8. Conclusion

We derived a general method for sparsifying approximate solutions to non-convex optimization problems based on ADMM. This method has been known in the context of sparse neural networks, and we presented a possible connection between this method and the brains mechanisms for achieving synaptic homeostasis.

[2]The compressed network only takes up 8.19KB of space, which is about the size of an email message.

## Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1656518.

## References

Beck, Amir and Teboulle, Marc. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

Bottou, Léon, Curtis, Frank E., and Nocedal, Jorge. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

Boyd, Stephen, Parikh, Neal, Chu, Eric, Peleato, Borja, Eckstein, Jonathan, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Byrd, Richard H, Lu, Peihuang, Nocedal, Jorge, and Zhu, Ciyou. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

Donoho, David L. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

Gabay, Daniel and Mercier, Bertrand. *A dual algorithm for the solution of non linear variational problems via finite element approximation*. Institut de recherche d'informatique et d'automatique, 1975.

Hestenes, Magnus R. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5): 303–320, 1969.

Kiaee, Farkhondeh, Gagné, Christian, and Abbasi, Mahdieh. Alternating direction method of multipliers for sparse convolutional neural networks. *arXiv preprint arXiv:1611.01590*, 2016.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

LeCun, Yann. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

Leng, Cong, Li, Hao, Zhu, Shenghuo, and Jin, Rong. Extremely low bit neural network: Squeeze the last bit out with admm. *arXiv preprint arXiv:1707.09870*, 2017.

Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in PyTorch. 2017.

Rogerson, Thomas, Cai, Denise J, Frank, Adam, Sano, Yoshitake, Shobe, Justin, Lopez-Aranda, Manuel F, and Silva, Alcino J. Synaptic tagging during memory allocation. *Nature Reviews Neuroscience*, 15(3):157, 2014.

Sutskever, Ilya, Martens, James, Dahl, George, and Hinton, Geoffrey. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, pp. 1139–1147, 2013.

Tononi, Giulio and Cirelli, Chiara. Sleep function and synaptic homeostasis. *Sleep Medicine Reviews*, 10(1): 49–62, 2018/05/21 2006. doi: 10.1016/j.smrv.2005.05.002. URL http://dx.doi.org/10.1016/j.smrv.2005.05.002.

Walker, Matthew. The role of sleep in cognition and emotion. *Annals of the New York Academy of Sciences*, 1156(1): 168–197, 2009.

Zhang, Tianyun, Ye, Shaokai, Zhang, Kaiqi, Tang, Jian, Wen, Wujie, Fardad, Makan, and Wang, Yanzhi. A systematic dnn weight pruning framework using alternating direction method of multipliers. *arXiv preprint arXiv:1804.03294*, 2018.

# Supplementary Materials

## A. Six Algorithms Compared

**Gradient Descent (GD)** In gradient descent, the update moves the parameter in the negative direction of the gradient. We use a constant step size $\alpha_k = 1 \times 10^{-3}$.

**Accelerated Gradient Descent (AGD)** We use accelerated gradient descent based on the formula for Nesterov momentum given in (Sutskever et al., 2013). We use a momentum factor of $0.9$ and the same step size as gradient descent.

**Adam** Adam is an adaptive gradient method that computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients (Kingma & Ba, 2014). We refer the reader to the paper for more details. We use the same constant step size, and set $\beta_1 = .9$, and $\beta_2 = .999$.

**Limited Memory BFGS (LBFGS)** Limited Memory BFGS (LBFGS) is a first-order optimization method that approximates the Hessian (Byrd et al., 1995). We use a step size of $.5$.

**Iterative Soft-Thresholding Algorithm (ISTA)** The iterative soft-thresholding algorithm (ISTA) is the proximal gradient method and reduces to gradient updates and then soft-thresholding (Beck & Teboulle, 2009). We use the same constant step size as the gradient method.

**ADMM** We use Algorithm 1, and use LBFGS (described above) with a step size of $.5$ to approximately solve for the $x$-update for a maximum of $10$ iterations. We also set $\rho = 1$. Importantly, we evaluate the loss of the $z$ at every iteration, not $x$.

## B. PyTorch Implementation

There is a clean implementation of the algorithm in the python library PyTorch that can be applied to any network with relative ease.

First we create 2 copies of the network:

```
rho = 1.0
xnet = build_net()
znet = copy.deepcopy(model)
unet = copy.deepcopy(model)
xzv = zip(xnet.parameters(),\
  znet.parameters(),unet.parameters())
```

To add the loss $\rho/2\|x - z + u\|_2^2$ to the variable `loss`, all we do is:

```
for xzv:
  loss += rho/2*torch.norm(xparam-\
  zparam.detach()+uparam.detach())**2
```

Then to update $x$ and $u$, all we do is:

```
ss = torch.nn.Softshrink(lamb/rho)
for xzv:
  zparam.data = ss(xparam + uparam)
  uparam.data = uparam + xparam - zparam
```

## C. Sparsity Pattern of Convolutional Network

The sparsity patterns for the second experiment are displayed in Figure 3 and Figure 4.

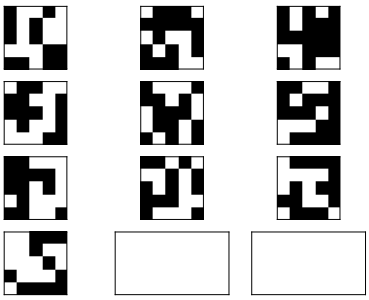

*Figure 3.* Sparsity pattern of convolutional filters in first layer (black = nonzero).

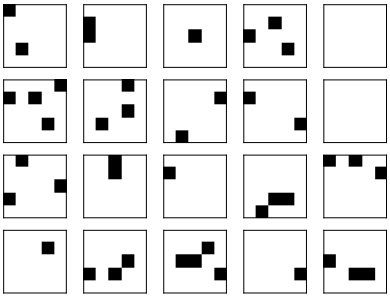

*Figure 4.* Sparsity pattern of convolutional filters in second layer (black = nonzero).