Learning Sparse Neural Networks via ℓ_0 and Tℓ_1 by a Relaxed Variable Splitting Method with Application to Multi-scale Curve Classification

Fanghui Xue and Jack Xin[✉]

Department of Mathematics, UC Irvine, Irvine, CA 92697, USA
{fanghuix,jack.xin}@uci.edu

Abstract. We study sparsification of convolutional neural networks (CNN) by a relaxed variable splitting method of ℓ_0 and transformed-ℓ_1 (Tℓ_1) penalties, with application to complex curves such as texts written in different fonts, and words written with trembling hands simulating those of Parkinson's disease patients. The CNN contains 3 convolutional layers, each followed by a maximum pooling, and finally a fully connected layer which contains the largest number of network weights. With ℓ_0 penalty, we achieved over 99% test accuracy in distinguishing shaky vs. regular fonts or hand writings with above 86% of the weights in the fully connected layer being zero. Comparable sparsity and test accuracy are also reached with a proper choice of Tℓ_1 penalty.

Keywords: Convolutional neural network · Sparsification · Multi-scale curves · Classification

1 Introduction

Sparsification of neural networks is one of the effective complexity reduction methods to improve efficiency and generalizability [3,4]. In this paper, we sparsify convolutional neural networks (CNN) for classifying curves with multi-scale structures. Such curves arise in hand writings of people with neurological disorders e.g. Parkinson disease (PD) patients, and in neuropsychological exams. Distinguishing hand writings of normal and PD subjects computationally will greatly help diagnosis and reduce physicians' workload in evaluations.

People with PD tend to lose control of their hands, and their writing or drawing shows oscillatory behavior as shown in Fig. 2, a century old image available online. Such oscillatory features can be learned during CNN training. Since we do not have large amount of PD hand writings, we shall generate on the computer a large number of oscillatory shapes that mimic shaky writings of PD subjects. Indeed, we found that CNN is quite successful for this task and can reach accuracy as high as 99% on our synthetic data set with three convolution layers and one fully connected layer as shown in Fig. 1. However, we also found

that there is a lot of redundancy in the weights of the trained CNNs, especially in the fully connected layer where we aim to significantly sparsify the network weights with minimal loss of accuracy.

Since the natural sparsity promoting penalty ℓ_0 is discontinuous, we shall adopt the relaxed variable splitting method (RVSM, [3]) for network sparsification. Even though Lipschitz continuous penalties such as ℓ_1 and transformed-ℓ_1 [2,8] are almost everywhere differentiable, the splitting approach [3] is more effective for enforcing sparsity than directly placing a penalty function inside the stochastic gradient descent (SGD) algorithm. The RVSM is also much simpler than the statistical ℓ_0 regularization approach in [4]. A systematic comparison with [4] will be conducted elsewhere.

The rest of the paper is organized as follows. In Sect. 2, we review RVSM for ℓ_0, transformed-ℓ_1, and ℓ_1 penalties and present a convergence theorem. A new critical point condition is introduced for the limit. We apply RVSM to CNNs for multi-scale curve classification. In Sect. 3, we describe our data set, CNN architecture and training, the CNN performance in terms of network accuracy and sparsity. We compare weight distributions of sparse and non-sparse networks. Concluding remarks are in Sect. 4.

2   Sparse Neural Network Training Algorithm

When training neural networks, one minimizes a penalized objective function of the form:

$$l(w) := f(w) + \lambda\, P(w),$$

where $f(w)$ is a standard loss function in neural network models such as cross entropy [7], and $P(w)$ is a penalty function. In SGD, the expected loss f is replaced by an empirical loss over batches of training samples [7]. In this section, we shall consider the expected loss function f which has better regularity than the empirical loss functions [6], and is more conducive to analysis. In the actual training, SGD and the sample averaged empirical loss function will be implemented. The standard penalty is ℓ_2 norm, also known as weight decay. However, ℓ_2 penalty cannot reduce the number of redundant parameters, resulting in a network with on the order of millions of nonzero weights. Thus we turn to ℓ_0 penalty, which produces zero weights during training [4], however leads to a non-convex discontinuous optimization problem. In [4], a statistical approach is proposed to regularized ℓ_0. In this paper, we utilize the Relaxed Variable Splitting Method (RSVM) studied in [3] for a neural network regression problem. RSVM is much simpler to state and implement than [4]. To this end, let us consider the following objective function for parameter $\beta > 0$:

$$\mathcal{L}_\beta(u, w) = f(w) + \lambda\, P(u) + \frac{\beta}{2}\|w - u\|_2^2.$$

Let η be the learning rate. We minimize $\mathcal{L}_\beta(u, w)$ with the RVSM algorithm below where the u step is thresholding and the w step is gradient descent followed by a normalization:

Algorithm 1. RVSM

Initialize u^0, w^0 randomly.
while *not converged* **do**
$\qquad u^{t+1} \leftarrow \arg\min_u \mathcal{L}_\beta(u, w^t)$
$\qquad \hat{w}^{t+1} \leftarrow w^t - \eta\nabla f(w^t) - \eta\beta(w^t - u^{t+1})$
$\qquad w^{t+1} \leftarrow \frac{\hat{w}^{t+1}}{\|\hat{w}^{t+1}\|}$
end

The main theorem of [3] guarantees the convergence of RVSM algorithm under some conditions on the parameters (λ, β, η) and initial weights in case of one convolution layer network and Gaussian input data. The latter conditions are used to prove that the loss function f has Lipschitz gradient away from the origin. Assuming that the Lipschitz gradient condition holds for f, we adapt the main result of [3] into:

Theorem 1. *Suppose that f is bounded from below, and satisfies the Lipschitz gradient inequalities: $\|\nabla f(x) - \nabla f(y)\| \le L_1 \|x - y\|$, and $|f(x) - f(y) - \langle \nabla f(x), x - y \rangle| \le L_2 \|x - y\|^2$, $\forall (x, y)$ with $\|x\| \ge \delta_0$, $\|y\| \ge \delta_0$ for some positive constants δ_0, L_1, and L_2. Then there exists a positive constant $\eta_0 = \eta_0(\delta_0, L_1, L_2, \beta) \in (0, 1)$ so that if $\eta < \eta_0$, the Lagrangian function $\mathcal{L}_\beta(u^t, w^t)$ is descending and converging in t, with (u^t, w^t) of RVSM algorithm satisfying $\|(u^{t+1}, w^{t+1}) - (u^t, w^t)\| \to 0$ as $t \to +\infty$, and subsequentially approaching a limit point $(\bar{u}, \bar{w})$.*

For the ℓ_0 penalty, our objective function (the Lagrangian) becomes

$$\mathcal{L}_\beta(u, w) = f(w) + \lambda\|u\|_0 + \frac{\beta}{2}\|w - u\|_2^2.$$

In this case, we simply obtain

$$u^{t+1} = \arg\min_u \mathcal{L}_\beta(u, w^t) = H_{\lambda/\beta}(w^t),$$

where H_γ is the hard-thresholding operator [1]. On each component

$$H_\gamma(w_i) = \begin{cases} 0 & \text{if } |w_i| \le \sqrt{2\gamma} \\ w_i & \text{if } |w_i| > \sqrt{2\gamma}. \end{cases} \tag{1}$$

For the ℓ_1 case, it is also clear that

$$u^{t+1} = S_{\lambda/\beta}(w^t),$$

where S_γ is the soft-thresholding operator [2]

$$S_\gamma(w_i) = \begin{cases} w_i + \gamma & \text{if } w_i \le -\gamma \\ 0 & \text{if } |w_i| < \gamma \\ w_i - \gamma & \text{if } w_i \ge \gamma. \end{cases} \tag{2}$$

We also consider the transformed ℓ_1 (TL1) penalty [8], which nicely interpolates the ℓ_0 and ℓ_1 penalties:

$$\rho_a(x) = \frac{(a+1)|x|}{a+|x|}$$

to each component of a vector, where a is a positive parameter. It is clear that

$$\lim_{a \to 0^+} \rho_a(x) = I_{\{x \neq 0\}}, \quad \lim_{a \to +\infty} \rho_a(x) = |x|.$$

By solving the problem with TL1 penalty, we can also get a thresholding operator $T_{a,\gamma}$ in closed form [8]:

$$T_{a,\gamma}(w_i) = \begin{cases} 0 & \text{if } |w_i| \leq t \\ g_{a,\gamma}(w_i) & \text{if } |w_i| > t, \end{cases} \tag{3}$$

where

$$g_{a,\gamma}(x) = \operatorname{sgn}(x) \left\{ \frac{2}{3}(a+|x|) \cos\left(\frac{\phi(x)}{3}\right) - \frac{2a}{3} + \frac{|x|}{3} \right\}$$

and $\phi(x) = \arccos\left(1 - \frac{27\gamma a(a+1)}{2(a+|x|)^3}\right)$. Here the parameter t depends on γ as:

$$t = \begin{cases} \gamma \frac{a+1}{a} & \text{if } \gamma \leq \frac{a^2}{2(a+1)} \\ \sqrt{2\gamma(a+1)} - \frac{a}{2} & \text{if } \gamma > \frac{a^2}{2(a+1)}. \end{cases} \tag{4}$$

Remark 1. It follows from the Theorem above that the limit point $(\bar{u}, \bar{w})$ satisfies the *equilibrium* equations for the ℓ_0, ℓ_1 and transformed-ℓ_1 penalties respectively as:

$$\bar{u} = H_{\lambda/\beta}(\bar{w}), \text{ or } S_{\lambda/\beta}(\bar{w}), \text{ or } T_{a,\lambda/\beta}(\bar{w});$$
$$\nabla f(\bar{w}) = -\beta(\bar{w} - \bar{u}). \tag{5}$$

The system (5) serves as a novel "critical point condition". This is particularly useful in the ℓ_0 case where the Lagrangian function $\mathcal{L}_\beta(u, w)$ is discontinuous in u.

3   Experimental Results

We apply the RVSM algorithm to convolutional neural networks to see how it brings about a sparse network. After training, the $\bar{w}$ is sparse with small components removed, and it serves as the network weight for inference. In the following experiment, we consider a convolutional neural network of 3 layers and a data set of 100×100 binary images. What we care about is the percentage of the weights which are zero after training the sparse network. Many of the algorithms can result in a sparsity of over 90%, which means only less than
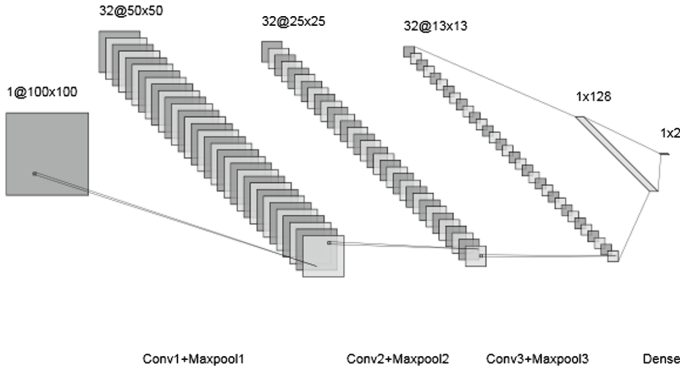
Fig. 1. CNN architecture in this study.

10% of the parameters contribute to the model. This makes our model far more efficient than the original one without regularization.

In order to find out how the weights are distributed in each layer, we go through the structure of the network. Figure 1 shows the number of nodes in each layer, from which we can simply calculate the number of weights needed to connect the nodes.[1] We apply 32 3×3 filters to the initial image to get the first convolutional layer, which results in $32 \times 3 \times 3 = 288$ weights. Similarly, each of the second and the third convolutional layer contains $32 \times 32 \times 3 \times 3 = 9216$ weights, if we apply 32 3×3 filters again. After each convolutional layer, we add one max pooling layer with a 2×2 filter and a stride of 2. The dimension of each image is not changed after each convolution, since we have applied padding. But it is reduced by a half on both the width and the height after max pooling because of a stride of 2. Thus the dimension of the image is reduced from 100×100 to 50×50, to 25×25 and finally to 13×13. So this produces $13 \times 13 \times 32 \times 128 = 692224$ weights when constructing a dense layer of 128 nodes. Finally, $128 \times 2 = 256$ weights are used to connect the dense layer to the output layer of 2 nodes, if our goal is to classify the images into two categories. From the above discussion, we notice that 97.3% of the weights are concentrated to the dense layer. We will see that most of them contribute nothing to the model after we train the sparse network.

The first data set we use is the images of the handwritten alphabet by Parkinson's disease (PD) patients and normal handwritten alphabet. We know that many PD patients may suffer from tremors in their daily life and work. One remarkable feature is that the words they write can be much shakier than the normal, which can be used to distinguish a PD patient during diagnosis. Figure 2[2] shows one real example of handwritten sentence by a PD patient.

---

[1] When generating the figure, we used a tool by Alex Lenail available at http://alexlenail.me/NN-SVG/LeNet.html.

[2] https://en.wikipedia.org/wiki/Micrographia_(handwriting).

Fig. 2. Handwritten sentence by a PD patient.

From our point of view, these two writing styles – normal vs. shaky – can be treated as two fonts. There is one Parkinson's font available on the internet,[3] which contains the whole alphabet of the 52 uppercase and lowercase letters. We simulate a training set of 5,000 observations and a test set of 1,000 observations by adding some rotations, affine transformations and elastic distortions [5]. As we have mentioned, this is a data set of 100×100 binary images, of which some samples are shown in Fig. 3. Though our model is used to distinguish the letters written by a Parkinson's disease patient in this single experiment, it can be simply applied to classify any other fonts.
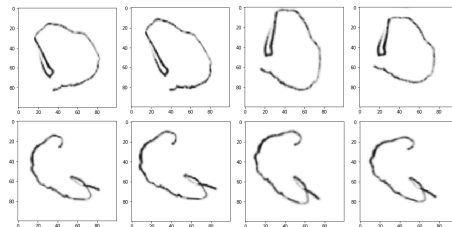

Fig. 3. Sample images of PD patients' handwriting.

As most of the redundancy appears in the dense layer, we apply the threshold step of the algorithm to the weights in dense layer only. This is because if we use the same λ and β in all the layers, the proportion of zero weights in the convolutional layers might be high, where the zero weights can indeed grade the model. Compared to the dense layer of 700,000 weights, there is not much freedom to modify the convolutional layer of 10,000 weights. Too much sparsity leads to a sizable loss of accuracy.

In our models, we have the freedom to set the thresholding parameters, namely β, λ and a. A higher threshold usually means more sparsity, since more weights are forced to zero by the threshold. From the formula (1) and (2) for the ℓ_0 and ℓ_1 penalties, it is clear that the larger λ is and the smaller β is, the higher the threshold γ will be. Given the same thresholding parameter γ, the ℓ_0 model may result in a sparser model than ℓ_1, since its threshold is a square root of γ,

which is higher. From the formula (3) and (4) for the TL1 penalty, the smaller a is, the higher the threshold is. As discussed in the previous section, when a goes to infinity, TL1 becomes ℓ_1. When a goes to 0, it becomes ℓ_0. So as to achieve more sparsity, we may choose a small a.

Our algorithm converges quickly after a few iterations. In most of the cases, it obtains an accuracy of 95% and a sparsity of 60% after 10 epochs. The accuracy soon goes up to 98% within 20 epochs, while some models achieve a sparsity of around 90% eventually. Figure 4 shows the convergence of the training algorithm.
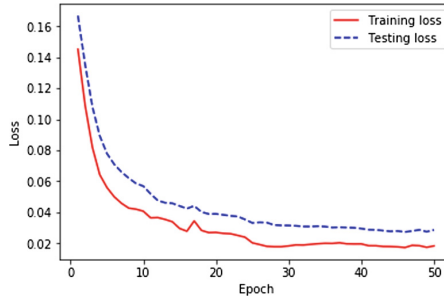

Fig. 4. Training and testing loss functions vs. epochs.

Table 1 shows our results of sparsity and testing accuracy. It verifies what we discussed on the thresholding parameter. That is, when the threshold grows higher, the sparsity also grows correspondingly. When a is less than 0.1, we achieve a sparsity of 86%, while the accuracy remains high. The key point should be noticed is that these sparse networks achieve almost the same, or even better accuracy than the non-sparse model. Thus we affirm that around 90% of the parameters are redundant, as they hardly contribute to the accuracy of the model.

Another data set we consider is the images of normal vs. shaky planar shapes like triangles and quadrangles (not necessarily convex). It can be viewed as another demonstration of PD patients' handwriting, as what they draw are somehow shaky, likewise the letters they write. This data set of 100×100 binary images is simulated by adding random noise to the normal planar shapes. Figure 5 shows some sample images of our shapes. The results on this data set are similar to those of the first data set, as shown in Table 2. So RVSM also achieves high accuracy and sparsity on multi-scale planar curve data.

More properties of our sparse networks are as follows. First, there is a remarkable difference in distributions of the weights between the sparse and non-sparse models. For the sparse model, most of the weights are zero, while the rest are very close to zero. So its distribution looks like a vertical line plus some noise on the interval close to zero. In our example of non-sparse model, it also has a peak at zero. However, very few weights are exactly zero. Many of them are merely close to zero, while a large proportion are far away from zero. What's more, the

distribution of this non-sparse model seems to be bell shaped. The distributions are shown in Fig. 6, where the weights are normalized for better viewing.
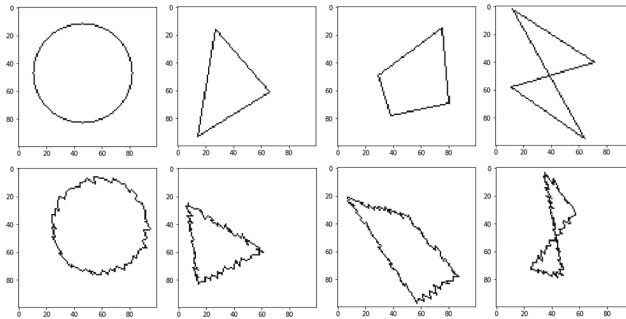

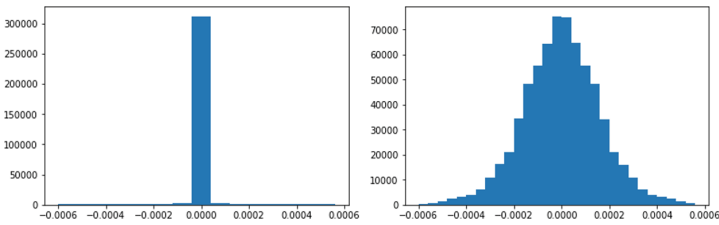**Fig. 5.** Normal vs. shaky shapes.

Fig. 6. Distribution of weights: Sparse vs. Non-sparse networks

What we also notice is that, RVSM performs much better than applying SGD directly to the TL1 penalized loss functions. As shown in Table 3, most of the normalized weights in the SGD model are distributed between 10^{-5} and 10^{-3}. It seems there is no apparent criterion to judge if a weight of 10^{-4} should be set to zero or it does contribute to the network. However, for the RVSM method when $a = 0.01$, it is clear that 8.7% of the weights are greater than 10^{-4} and 84.9% of the weights are less than 10^{-10}. There is a significant gap between the two scales of 10^{-4} and 10^{-10}, which makes it reasonable to set all the weights less than 10^{-10} to zero. This leads to a network of 84.9% sparsity. Another point worth mentioning is that applying SGD directly to the penalized loss function may hurt the accuracy a lot at $a = 0.01$, resulting in 96.7% accuracy for the model. This is because when a is small, the penalized term behaves like ℓ_0, which renders the objective function nearly singular. RVSM resolves this issue by making the penalty implicit to a thresholding process, which gives an accuracy of 99.5%.

Table 4 shows another interesting phenomenon. Since the weights are randomly initialized with mean zero, there is roughly even split of plus/minus signs

Table 1. Testing sparsity and accuracy for the data of alphabets.

λ	β	a	Penalty	Sparsity (%)	Accuracy (%)
0.0005	0.1	0	ℓ_0	86.1	99.4
		0.01	TL1	87.6	99.0
		0.1	TL1	85.8	99.7
		1	TL1	78.1	99.3
		100	TL1	82.0	99.3
		∞	ℓ_1	76.5	99.0

Table 2. Testing sparsity and accuracy for the data of planar shapes.

λ	β	a	Penalty	Sparsity (%)	Accuracy (%)
0.0005	0.1	0	ℓ_0	90.2	99.9
		0.01	TL1	83.5	99.1
		0.1	TL1	87.6	99.8
		1	TL1	74.9	99.9
		100	TL1	75.0	99.9
		∞	ℓ_1	74.6	99.6

Table 3. Sparsity and accuracy: RVSM vs. Direct SGD for TL1 penalty

a	Algorithm	Sparsity (%) of 10^{-n} scale					Accuracy (%)
		10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-10}	
0.01	RVSM	99.7	96.0	91.3	88.6	84.9	99.5
0.01	SGD	99.9	99.9	45.9	5.44	10^{-5}	96.7
100	RVSM	99.9	97.5	92.7	88.5	80.3	99.3
100	SGD	99.9	99.7	48.1	6.68	10^{-5}	99.0

Table 4. Number of sign changes and relative % in kernels of convolutional layers.

a	Layer 1	Layer 2	Layer 3
0.01	72 (25.0%)	1120 (12.2%)	769 (8.34%)
1	45 (15.6%)	1133 (12.3%)	784 (8.51%)
100	35 (12.2%)	1001 (10.9%)	995 (10.8%)

in all layers. At the end of training, we counted the number of sign changes in the kernel of each convolutional layer, and found that more weights changed signs in the first convolutional layer than in the next two layers. This is consistent with the network filters structured towards low pass in depth after training.

4   Conclusions

In this paper, we have applied the RVSM algorithm to learn sparse neural networks. We have achieved an accuracy of 99% and a sparsity of 87% when training CNNs on a data set consisted of synthetic handwritten letters and planar curves by PD patients, and normal handwriting. We have also discussed the tuning of thresholding parameters, and verified the fact that a higher threshold can produce higher sparsity. What's more, our experiments show that the RVSM outperforms the direct application of SGD on the penalized loss function, in both sparsity and accuracy. The RVSM generates a significant gap between the weights of large scale and small scale, which acts as an indicator to show sparsity. In future work, we plan to explore a wider variety of PD patient data and more refined multi-class classification tasks.

References

1. Blumensath, T., Davies, M.: Iterative thresholding for sparse approximations. J. Fourier Anal. Appl. **14**(5–6), 629–654 (2008)
2. Daubechies, I., Michel, D., De Mol, C.: An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. Commun. Pure Appl. Math. **57**(11), 1413–1457 (2004)
3. Dinh, T., Xin, J.: Convergence of a relaxed variable splitting method for learning sparse neural networks via ℓ_1, ℓ_0, and transformed-ℓ_1 penalties (2018). arXiv:1812.05719
4. Louizos, C., Welling, M., Kingma, D.: Learning sparse neural networks through ℓ_0 regularization. In: ICLR (2018). arXiv:1712.01312v2
5. Simard, P., Steinkraus, D., Platt, J.: Best practices for convolutional neural networks applied to visual document analysis. In: Seventh International Conference on Document Analysis and Recognition, pp. 958–963. IEEE (2003)
6. Yin, P., Zhang, S., Lyu, J., Osher, S., Qi, Y-Y., Xin, J.: Blended coarse gradient descent for full quantization of deep neural networks. Res. Math. Sci. **6**(1), 14 (2019). arXiv:1808.05240
7. Yu, D., Deng, L.: Automatic Speech Recognition: A Deep Learning Approach. Signals and Communication Technology. Springer, New York (2015)
8. Zhang, S., Xin, J.: Minimization of transformed l_1 penalty: closed form representation and iterative thresholding algorithms. Comm. Math. Sci. **15**(2), 511–537 (2017)