

---

# Differentiable Sparsification for Deep Neural Networks

---

Anonymous Author(s)

Affiliation

Address

email

## Abstract

1 This is an appendix for *Differentiable Sparsification for Deep Neural Networks*.  
2 Source codes for the proposed approaches are provided in the supplementary  
3 material (source.zip).

## 4 1 Sparsity Regularizer

5 In our approach, different sparsity patterns can be derived by adopting different norms for a regularizer.  
6 For example, an individual component (ex. channel) can be removed by  $l_1$ -norm and a group of  
7 components or an entire module (ex. layer) can be zeroed-out by  $l_{2,1}$ -norm. Note that we do not need  
8 to manually implement different updating rules as in the proximal gradient approach. We just need to  
9 change a regularization term in an objective function. The implementation example with TensorFlow  
10 is in Listing 1.

```
11 1  
12 2 def safe_l2_norm(tensor, axis=None, keepdims=None, name=None):  
13 3     @tf.custom_gradient  
14 4     def norm(x):  
15 5         y = tf.norm(x, 2, axis, keepdims, name)  
16 6  
17 7         def grad(dy):  
18 8             ex_dy = tf.expand_dims(dy, axis) if axis else dy  
19 9             ex_y = tf.expand_dims(y, axis) if axis else y  
20 0             # for numerical stability, add a small constant  
21 1             return ex_dy * (x / (ex_y + 1e-19))  
22 2  
23 3         return y, grad  
24 4  
25 5     return norm(tensor)  
26 6  
27 7 alpha_mag = tf.nn.relu(abs_alpha - beta*alpha_l1)  
28 8 alpha = tf.math.sign(alpha) * alpha_mag  
29 9  
30 0 if norm == 'l1': #l1-norm  
31 1     reg = tf.reduce_sum(alpha_mag)  
32 2 elif norm == 'group': #l2-group norm  
33 3     alpha_mags = tf.reshape(alpha, shape=[len(channels), -1])  
34 4     reg = tf.reduce_sum(safe_l2_norm(alpha_mags, axis=-1))
```

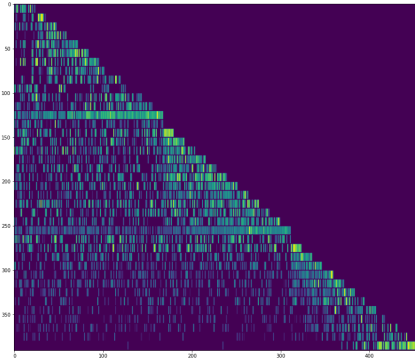
Listing 1: Codes for Regularizer

35 Figure 1 and 2 show the sparsified structures of DenseNet-K12 with 40 layer, which are trained with  
36  $l_1$ -norm and  $l_{2,1}$ -norm. Each row corresponds with one hidden layer. For the sparse regularization  
37 with  $l_{2,1}$ -norm, we make a group of 12 channels such that layer-wise connection can be learned.

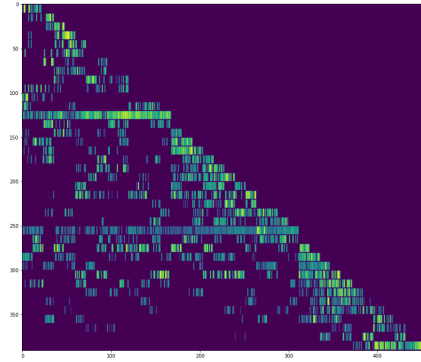
Fig. 1 shows the sparsified structures of hidden layers in channel-wise. A pixel-like thin short strip represents the magnitude of a scale parameter in the batch normalization. Fig. 2 shows the sparsified structures in group-wise. Each square block represents a group of 12 channels. One block is added at a time as a layer proceeds from top to bottom since each layer outputs new 12 channels. An input layer generates 24 channels and thus the first row has 2 blocks. The number of survived channels is colored by the magnitude of a block, whose brightness is proportionate to the number of non-zero channels within a group. Table 1 shows the sparsity rates for a base model, Network-slimming(NS) [7] and the proposed method (DS). Although the two cases of the proposed approach have similar channel-sparsity rates, the learned structures are very different. The experimental results show that the proposed sparse parameterization is not limited to a particular norm and it can learn different structures by simply changing norms.

Table 1: Sparsity Rate on CIFAR-10, DenseNet-40-K12.

Model	C. Sparsity(%)		G. Sparsity(%)		Top-1 Error(%)		Parmas	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
Base	00.0	0.0	00.0	0.0	5.92	0.20	$1.1 \times 10^6$	0.0
NS	60.0	0.4	6.0	0.9	5.71	0.16	$4.8 \times 10^5$	$7.7 \times 10^2$
DS- $l_1$	60.3	0.4	6.4	0.3	5.72	0.16	$4.7 \times 10^5$	$3.5 \times 10^3$
DS- $l_{2,1}$	60.6	0.4	56.5	0.5	6.28	0.25	$4.7 \times 10^5$	$5.0 \times 10^3$

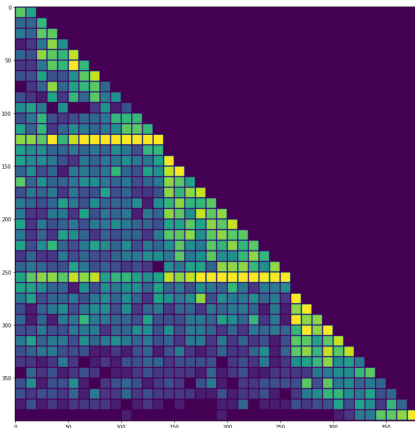


(a)  $l_1$ -norm

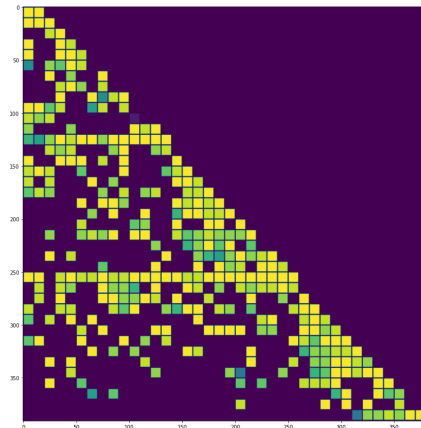


(b)  $l_{2,1}$ -group norm

Figure 1: Sparsified connection in channel-wise view. DenseNet-40-K12.



(a)  $l_1$ -norm



(b)  $l_{2,1}$ -group norm

Figure 2: Sparsified connection in group-wise view. DenseNet-40-K12.

## 2 Rectified Gradient Flow

### 2.1 Implementation

Listing 2 shows TensorFlow implementation of the rectified gradient flow. The learning method can be implemented by simply switching `tf.nn.relu` to `rgf_relu`.

```

53 1
54 2 @tf.custom_gradient
55 3 def rgf_relu(x):
56 4     o1 = tf.nn.relu(x) # forward-pass
57 5     o2 = tf.keras.activations.elu(x, alpha=0.1) # backward-pass
58 6
59 7     def grad(dy): # the gradient of elu is used in backward-pass
60 8         return tf.gradients(o2, [x], grad_ys=[dy])
61 9
62 10    return o1, grad
63 11
64 12 #alpha_mag = tf.nn.relu(abs_alpha - sig_beta*alpha_l1)
65 13 alpha_mag = rgf_relu(abs_alpha - sig_beta*alpha_l1)

```

Listing 2: Codes for Rectified Gradient Flow

### 2.2 Analysis on Gradient Flow

Assume that an output  $y$  of a neuron in a hidden layer is written as

$$y(\mathbf{x}) = \sum_{i=1}^n a_i f_i(\mathbf{x}; \mathbf{w}_i),$$

where  $\mathbf{x}$  denotes an input from a proceeding layer,  $\mathbf{w}_i$  model parameters for component  $f_i$ , and  $a_i$  an edge or an architecture parameter. A loss function can be denoted by

$$\mathcal{L}(a(\alpha), f(\mathbf{x}; \mathbf{w})).$$

Let  $a_i$  be a function of  $\alpha$ ,

$$a_i(\alpha_i) = \text{sign}(\alpha_i) (|\alpha_i| - \sigma(\beta))_+,$$

which is made simpler for analysis than the main paper. The gradients can be written as

$$\frac{\partial \mathcal{L}}{\partial \alpha_i} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial a_i(\alpha_i)}{\partial \alpha_i} \cdot f_i(\mathbf{x}, \mathbf{w}_i), \quad (1)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} = \frac{\partial \mathcal{L}}{\partial y} \cdot a_i(\alpha_i) \cdot \frac{\partial f_i(\mathbf{x}; \mathbf{w}_i)}{\partial \mathbf{w}_i}, \quad (2)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial y} \cdot \sum_j^n \left( a_j(\alpha_j) \cdot \frac{\partial f_j(\mathbf{x}; \mathbf{w}_j)}{\partial \mathbf{x}} \right). \quad (3)$$

If  $|\alpha_i| < \sigma(\beta)$ ,  $\partial a_i / \partial \alpha_i$  in Eq. (1) becomes zero and  $\alpha_i$  does not have a learning signal. If `elu` [1] is employed in the backward pass, it can generate approximated gradient for  $\alpha_i$ . Regardless of whether `elu` is used in the backward pass or not,  $\mathbf{w}_i$  and  $\mathbf{x}$  do not receive a learning signal through  $a_i$  since  $a_i = 0$  in Eq. (2) and (3). This leads to a similar learning mechanism proposed in DNW [8], where the gradients flows to zeroed-out (hallucinated) edges but does not through them.

If we define  $a$  as in the main paper,

$$a_i(\alpha) = \text{sign}(\alpha_i) (|\alpha_i| - \sigma(\beta) \|\alpha\|_1)_+,$$

the gradient is written as

$$\frac{\partial \mathcal{L}}{\partial \alpha_i} = \frac{\partial \mathcal{L}}{\partial y} \cdot \sum_j^n \left( \frac{\partial a_j(\alpha)}{\partial \alpha_i} \cdot f_j(\mathbf{x}, \mathbf{w}_j) \right).$$

Even if *elu* [1] is not used in the backward pass, the gradient for  $\alpha_i$  is still generated through others. However, a learning signal can be generated more eagerly with *elu*.

### 3 Channel Pruning in Convolutional Network

We compare network-slimming (NS) [7] and our proposed method (DS). We ran each experiments 5 times and showed the average and the standard deviation. We controlled the value of  $\lambda$  such that it has similar pruning rate with that of the network-slimming approach. Table 2 and 3 show experimental results on DenseNet with 100 layers [5], and Table 4 and 5 on ResNet with 164 layers [2, 3]. In the tables, sparsity denotes a pruning rate, i.e, the number of removed channels in hidden layers. When the sparsity rate is relative low, they have similar error rates but the gap increases as the sparsity rate does.

Table 2: Performance on CIFAR-10, DenseNet-100-BC-K12.

Model	Sparsity(%)		Top-1 Error(%)		Parmas		FLOPs	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
Base	00.0	0.0	5.44	0.11	$7.6 \times 10^5$	0.0	$5.8 \times 10^8$	0.0
NS	60.0	0.0	5.40	0.14	$3.7 \times 10^5$	$3.3 \times 10^2$	$2.5 \times 10^8$	$3.0 \times 10^6$
NS	70.0	0.0	6.53	0.19	$2.9 \times 10^5$	$9.3 \times 10^2$	$1.9 \times 10^8$	$5.0 \times 10^6$
NS	75.0	0.0	7.29	0.27	$2.5 \times 10^5$	$1.0 \times 10^3$	$1.6 \times 10^8$	$3.3 \times 10^6$
NS	80.0	0.0	8.39	0.28	$2.0 \times 10^5$	$2.0 \times 10^3$	$1.4 \times 10^8$	$3.4 \times 10^6$
DS	60.4	0.4	5.42	0.20	$3.6 \times 10^5$	$3.5 \times 10^3$	$2.5 \times 10^8$	$3.4 \times 10^6$
DS	70.3	0.1	5.77	0.09	$2.7 \times 10^5$	$2.3 \times 10^3$	$1.8 \times 10^8$	$1.6 \times 10^6$
DS	75.7	0.2	6.05	0.26	$2.2 \times 10^5$	$2.6 \times 10^3$	$1.5 \times 10^8$	$3.0 \times 10^6$
DS	80.4	0.1	6.64	0.11	$1.7 \times 10^5$	$0.6 \times 10^3$	$1.3 \times 10^8$	$2.0 \times 10^6$

Table 3: Performance on CIFAR-100, DenseNet-100-BC-K12.

Model	Sparsity(%)		Top-1 Error(%)		Parmas		FLOPs	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
Base	00.0	0.0	24.00	0.23	$7.9 \times 10^5$	0.0	$5.8 \times 10^8$	0.0
NS	60.0	0.0	25.02	0.26	$3.9 \times 10^5$	$2.0 \times 10^3$	$2.4 \times 10^8$	$4.9 \times 10^6$
NS	70.0	0.0	28.02	0.75	$3.1 \times 10^5$	$1.5 \times 10^3$	$1.9 \times 10^8$	$4.0 \times 10^6$
NS	75.0	0.0	29.66	0.51	$2.8 \times 10^5$	$1.1 \times 10^3$	$1.6 \times 10^8$	$4.7 \times 10^6$
NS	80.0	0.0	31.50	0.46	$2.4 \times 10^5$	$7.9 \times 10^2$	$1.3 \times 10^8$	$2.7 \times 10^6$
DS	60.5	0.2	24.86	0.20	$3.6 \times 10^5$	$2.9 \times 10^3$	$2.4 \times 10^8$	$5.0 \times 10^6$
DS	70.4	0.4	26.37	0.15	$2.7 \times 10^5$	$3.5 \times 10^3$	$1.8 \times 10^8$	$5.4 \times 10^6$
DS	75.3	0.3	27.20	0.06	$2.3 \times 10^5$	$2.9 \times 10^3$	$1.5 \times 10^8$	$3.0 \times 10^6$
DS	80.3	0.4	28.12	0.23	$1.8 \times 10^5$	$3.5 \times 10^3$	$1.3 \times 10^8$	$1.1 \times 10^7$

Table 4: Performance on CIFAR-10, ResNet-164.

Model	Sparsity(%)		Top-1 Error(%)		Parmas		FLOPs	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
Base	00.0	0.0	5.13	0.05	$1.7 \times 10^6$	0.0	$5.0 \times 10^8$	0.0
NS	60.0	0.0	5.49	0.06	$1.1 \times 10^6$	$7.9 \times 10^3$	$2.9 \times 10^8$	$3.0 \times 10^6$
NS	70.0	0.0	6.67	0.19	$8.1 \times 10^5$	$1.4 \times 10^4$	$2.2 \times 10^8$	$3.4 \times 10^6$
DS	60.7	0.3	5.33	0.15	$8.9 \times 10^5$	$1.7 \times 10^4$	$2.6 \times 10^8$	$3.4 \times 10^6$
DS	70.5	0.3	5.68	0.21	$5.9 \times 10^5$	$1.8 \times 10^4$	$1.9 \times 10^8$	$5.3 \times 10^6$

Table 5: Performance on CIFAR-100, ResNet-164.

Model	Sparsity(%)		Top-1 Error(%)		Parmas		FLOPs	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
Base	00.0	0.0	23.23	0.28	$1.7 \times 10^6$	0.0	$5.0 \times 10^8$	0.0
NS	60.0	0.0	24.71	0.24	$1.2 \times 10^6$	$5.3 \times 10^3$	$2.5 \times 10^8$	$4.8 \times 10^6$
NS	70.0	0.0	27.91	0.36	$1.1 \times 10^6$	$3.7 \times 10^3$	$1.9 \times 10^8$	$5.7 \times 10^5$
DS	59.6	1.0	24.52	0.08	$1.0 \times 10^6$	$3.1 \times 10^4$	$2.4 \times 10^8$	$1.1 \times 10^7$
DS	70.8	1.5	25.44	0.40	$6.3 \times 10^5$	$1.9 \times 10^4$	$1.8 \times 10^8$	$1.3 \times 10^7$

## 89 4 Discovering Neural Wirings

90 We compare Discovering Neural Wirings(DNW) [8] and our proposed method(DS).  
 91 MobileNetV1 ( $\times 0.25$ ) [4] is employed as a base model and our implementation closely follows that  
 92 of DNW. DNW chose the value of  $k$  such that a final learned model has similar Mult-Adds with the  
 93 base model, and we also set the value of  $\lambda$  in the same manner. We ran each experiments 5 times and  
 94 showed the average and the standard deviation. Table 6 and 7 show the experimental results.

Table 6: Performance on CIFAR-10, Discovering Neural Wirings

Model	Top-1 Error(%)		Parmas		Mult-Adds	
	Avg.	Std.	Avg.	Std.	Avg.	Std.
MobileNetV1( $\times 0.25$ )	13.44	0.24	$2.2 \times 10^5$	0.0	$3.3 \times 10^6$	0.0
No Update( $\times 0.225$ )	13.86	0.27	$2.2 \times 10^5$	$3.7 \times 10^1$	$4.5 \times 10^6$	$3.7 \times 10^4$
DNW( $\times 0.225$ )	10.30	0.20	$1.8 \times 10^5$	$6.7 \times 10^1$	$3.1 \times 10^6$	$4.6 \times 10^4$
$\lambda \times 10^{-3}$	Proximal Gradient with $l_1$ -norm					
1.875	11.64	0.41	$2.4 \times 10^4$	$9.0 \times 10^2$	$3.7 \times 10^6$	$1.3 \times 10^5$
1.950	11.95	0.42	$2.3 \times 10^4$	$1.5 \times 10^3$	$3.7 \times 10^6$	$2.6 \times 10^5$
2.225	12.17	0.44	$2.1 \times 10^4$	$9.4 \times 10^2$	$3.3 \times 10^6$	$1.7 \times 10^5$
2.325	12.50	0.28	$1.9 \times 10^4$	$5.7 \times 10^2$	$3.2 \times 10^6$	$1.1 \times 10^5$
2.400	12.66	0.38	$1.8 \times 10^4$	$1.1 \times 10^3$	$3.0 \times 10^6$	$6.4 \times 10^4$
$\lambda \times 10^{-3}$	Proximal Gradient with $l_{1,2}$ -norm					
1.250	12.15	1.17	$1.0 \times 10^5$	$1.3 \times 10^4$	$3.7 \times 10^6$	$1.2 \times 10^5$
1.375	13.14	0.42	$9.6 \times 10^4$	$1.2 \times 10^4$	$3.6 \times 10^6$	$2.1 \times 10^5$
1.500	13.62	0.56	$9.6 \times 10^4$	$1.6 \times 10^4$	$3.4 \times 10^6$	$8.6 \times 10^4$
1.625	14.12	0.62	$8.7 \times 10^4$	$8.6 \times 10^3$	$3.3 \times 10^6$	$1.6 \times 10^5$
1.750	15.09	1.21	$8.7 \times 10^4$	$1.5 \times 10^4$	$3.2 \times 10^6$	$2.0 \times 10^5$
$\lambda \times 10^{-5}$	DS-No Rectified Gradient					
1.125	10.20	0.08	$6.9 \times 10^4$	$8.9 \times 10^2$	$3.6 \times 10^6$	$4.9 \times 10^4$
1.375	10.55	0.23	$6.1 \times 10^4$	$5.7 \times 10^2$	$3.4 \times 10^6$	$4.5 \times 10^4$
1.625	10.96	0.18	$5.6 \times 10^4$	$1.7 \times 10^3$	$3.2 \times 10^6$	$5.3 \times 10^4$
1.875	11.06	0.45	$5.1 \times 10^4$	$2.4 \times 10^2$	$3.1 \times 10^6$	$3.7 \times 10^4$
2.000	11.05	0.25	$4.9 \times 10^4$	$8.0 \times 10^2$	$3.0 \times 10^6$	$3.4 \times 10^4$
$\lambda \times 10^{-5}$	DS-Rectified Gradient					
6.0	9.04	0.10	$5.3 \times 10^4$	$7.0 \times 10^2$	$3.5 \times 10^6$	$4.8 \times 10^4$
6.5	9.28	0.38	$5.0 \times 10^4$	$9.8 \times 10^2$	$3.5 \times 10^6$	$7.8 \times 10^4$
7.0	9.36	0.27	$4.7 \times 10^4$	$8.4 \times 10^2$	$3.3 \times 10^6$	$6.7 \times 10^4$
7.5	9.32	0.19	$4.5 \times 10^4$	$3.2 \times 10^2$	$3.3 \times 10^6$	$8.5 \times 10^4$
8.0	9.66	0.07	$4.2 \times 10^4$	$9.0 \times 10^2$	$3.1 \times 10^6$	$6.0 \times 10^4$

Table 7: Performance on CIFAR-100, Discovering Neural Wirings

Model	Top-1 Error(%)		Parmas		Mult-Adds	
	Avg.	Std.	Avg.	Std.	Avg.	Std.
MobileNetV1( $\times 0.25$ )	43.78	0.54	$2.4 \times 10^5$	0.0	$3.4 \times 10^6$	0.0
No Update( $\times 0.225$ )	40.50	0.30	$3.1 \times 10^5$	$3.8 \times 10^1$	$4.6 \times 10^6$	$3.6 \times 10^4$
DNW( $\times 0.225$ )	34.18	0.37	$2.6 \times 10^5$	$5.0 \times 10^2$	$3.3 \times 10^6$	$4.7 \times 10^4$
$\lambda \times 10^{-3}$	Proximal Gradient with $l_1$ -norm					
2.1	35.27	0.65	$1.3 \times 10^5$	$4.5 \times 10^3$	$3.9 \times 10^6$	$2.6 \times 10^5$
2.2	36.00	0.70	$1.2 \times 10^5$	$3.9 \times 10^3$	$3.6 \times 10^6$	$9.5 \times 10^4$
2.3	35.21	0.32	$1.2 \times 10^5$	$4.1 \times 10^3$	$3.6 \times 10^6$	$1.4 \times 10^5$
2.4	36.57	0.53	$1.2 \times 10^5$	$4.3 \times 10^3$	$3.4 \times 10^6$	$1.9 \times 10^5$
2.5	36.81	0.41	$1.1 \times 10^5$	$3.5 \times 10^3$	$3.2 \times 10^6$	$2.0 \times 10^5$
$\lambda \times 10^{-3}$	Proximal Gradient with $l_{1,2}$ -norm					
1.00	35.97	0.59	$3.0 \times 10^5$	$7.7 \times 10^3$	$4.4 \times 10^6$	$1.2 \times 10^5$
1.25	37.52	0.84	$2.7 \times 10^5$	$6.3 \times 10^3$	$3.9 \times 10^6$	$1.2 \times 10^5$
1.50	38.45	0.29	$2.6 \times 10^5$	$5.5 \times 10^3$	$3.7 \times 10^6$	$1.1 \times 10^5$
1.75	39.63	0.54	$2.5 \times 10^5$	$1.1 \times 10^4$	$3.5 \times 10^6$	$1.4 \times 10^5$
2.00	41.35	0.86	$2.5 \times 10^5$	$9.7 \times 10^3$	$3.3 \times 10^6$	$1.6 \times 10^5$
$\lambda \times 10^{-5}$	DS-No Rectified Gradient					
2.00	35.51	0.62	$1.8 \times 10^5$	$5.4 \times 10^2$	$3.7 \times 10^6$	$4.8 \times 10^4$
2.25	35.26	0.50	$1.7 \times 10^5$	$7.8 \times 10^2$	$3.5 \times 10^6$	$4.6 \times 10^4$
2.50	35.68	0.47	$1.7 \times 10^5$	$7.5 \times 10^2$	$3.4 \times 10^6$	$4.3 \times 10^4$
2.75	35.32	0.28	$1.6 \times 10^5$	$5.2 \times 10^2$	$3.3 \times 10^6$	$2.5 \times 10^4$
3.00	35.77	0.36	$1.6 \times 10^5$	$1.0 \times 10^3$	$3.2 \times 10^6$	$2.3 \times 10^4$
$\lambda \times 10^{-4}$	DS-Rectified Gradient					
0.850	32.16	0.18	$2.0 \times 10^5$	$3.0 \times 10^3$	$3.8 \times 10^6$	$1.2 \times 10^5$
0.925	32.84	0.44	$1.9 \times 10^5$	$4.8 \times 10^3$	$3.6 \times 10^6$	$8.6 \times 10^4$
1.000	32.78	0.83	$1.8 \times 10^5$	$3.3 \times 10^3$	$3.5 \times 10^6$	$1.2 \times 10^5$
1.125	32.92	0.60	$1.6 \times 10^5$	$4.6 \times 10^3$	$3.3 \times 10^6$	$6.8 \times 10^4$
1.250	33.80	0.58	$1.6 \times 10^5$	$4.4 \times 10^3$	$3.1 \times 10^6$	$8.9 \times 10^4$

## 5 Learning relationship between Nodes in Graph

### 5.1 Data

Table 8: Summary of Experiment Data

Area	# Road Segments	Collection Method	Collection Period	Time Interval	Time Horizon for Prediction
See Fig. 3	170	Taxi with GPS	Jan.-Oct.	15 min.	1 (15 min.)

A summary of the data reported in the main paper is shown in Table 8. There were approximately 70,000 probe taxis operating in the metropolitan area where the data were collected. As the taxis operated in three shifts, more than 20,000 probes on average ran at any point. The traffic speed data covers 4,663 links in the area, which includes major arterials. Speed data from probes in the metropolitan area were aggregated every 5 minutes for each road segment; if no probe passed a link during a 5-minute period, the speed was estimated based on speeds in the previous time periods or from the same time periods in the past. After the preprocessing, the raw data were once again aggregated in 15-minute periods to maximize forecasting utility. The 15-minute period is the standard on which the highway capacity manual is based. As no information was available regarding probe counts that contribute to averaging speed data, three speeds for a 5-minute period were averaged without weight. Among the 4,663 links for which speed data were available, we selected 170 links

108 in the subarea (see Fig. 3) of the metropolitan; it was expected that the speed data collected in this  
 109 region would be free from missing observations as it is the busiest region in the metropolitan area,  
 110 and most taxi drivers congregate in the region to find passengers. Following the generic conventions  
 111 of machine learning, we used data from the previous eight months to train the proposed model and  
 112 reserved the data for the latter two months to test the trained model.

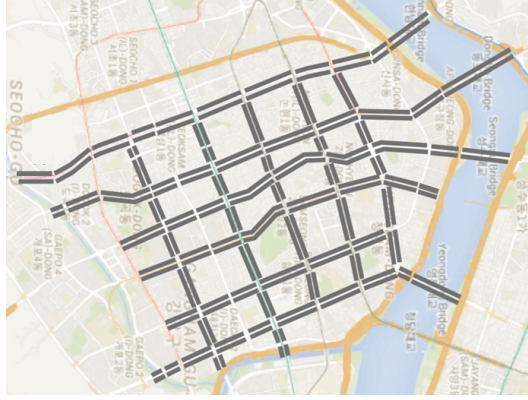


Figure 3: The gray lines represent 170 road links where the experimental data were collected.

## 113 5.2 Prediction Model with GCN

114 Figure 4 illustrates our GCN implementation. The outputs of hidden GCN blocks are concatenated  
 115 and fed into an output block. This was motivated by the dense connections of DenseNet [5]. The  
 116 structure of hidden GCN blocks were illustrated in the main paper. We trained the model for 500  
 117 epochs using Adam [6] with the defaulting parameter setting of TensorFlow. The initial learning rate  
 118 was set to 0.0005 and multiplied by 0.5 at epochs 400 and 450.

119 A training loss is defined by the mean relative error (MRE)

$$\frac{1}{N} \sum_{i=1}^N \frac{|\tilde{y}_i - y_i|}{y_i},$$

120 where  $N$  is the number of road segments;  $\tilde{y}_i$  and  $y_i$  denote an estimate and actual future observation  
 121 of travel speed on road segment  $i$ , respectively. We ran each experiment five times and selected the  
 122 median among the five lowest validation errors. We reported the test error from the epoch with the  
 123 median validation MRE. The report error was re-written in tables with mean absolute percentage  
 124 error (MAPE) for readers' accessibility, which is defined by multiplying MRE by 100.

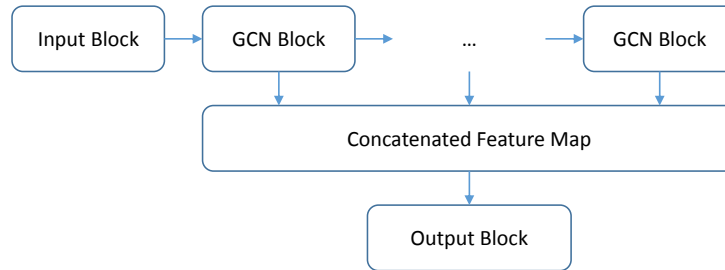


Figure 4: Structure of GCN. Graph convolutional neural network (GCN) structure. The GCN in our experiments consists of 5 GCN blocks. The output feature maps of GCN blocks are concatenated and then fed into an output block. Input and Output blocks consist of 3 fully connected layers, respectively.

### 5.3 Exclusive Sparsity Regularization

Table 9: Traffic speed prediction with GCN and  $l_{1,2}$ -norm regularization

(a) Proximal Gradient					(b) Proposed				
$\lambda$	# N.Z.	MAPE(%)	L. R.( $\times 100$ )		$\lambda$	# N.Z.	MAPE(%)	L.R.( $\times 100$ )	
			$k = 1$	$k = 2$				$k = 1$	$k = 2$
3	12,895	5.3898	57.24	62.42	3	6,170	5.3891	50.75	58.74
5	11,915	5.3933	55.48	60.90	5	5,692	5.3872	52.90	60.86
10	8,723	5.4894	33.39	40.35	10	5,009	5.3999	53.68	61.64
15	7,584	5.5395	13.37	18.05	15	4,704	5.3948	55.83	63.68

The  $l_p$ -norm with  $p < 1$  can induce strong competition within a group, but its closed form solution for the proximal operator is unknown. Among the norms which have the closed form solutions, the exclusive sparsity regularization can be a promising substitute  $l_p$ -norm with  $p < 1$  because it is also known that it promotes the sparsity or the competition within a group. We performed comparison experiments with the  $l_{1,2}$ -exclusive norm [10, 9]. As in the main paper, we made groups for the row and column vectors of an adjacency matrix. Note that the proximal operator should be applied on free variables; it cannot be applied on  $\gamma$  or  $a$  of an adjacency matrix. Thus, we did not apply the exponential function on  $\alpha$  and instead constrained the elements of an adjacency matrix to be non-negative by setting initial values as 1.0 and modifying the proximal operator as

$$\alpha_{g,i} \leftarrow \left( \alpha_{g,i} - \eta \lambda \|\alpha_g\|_1 \right)_+,$$

where  $\alpha_g$  represents the row or column vector of an adjacency matrix before the doubly-stochastic normalization. We applied the proximal operator at every mini-batch after updating model variables with the gradient of the prediction loss.

Similarly as above, we modified our proposed approach for comparison experiments. We did not employ the exponential function but applied the differentiable thresholding operation described in the main paper,

$$\tilde{\gamma}_{g,i} = \left( \alpha_{g,i} - \sigma(\beta_g) \cdot \|\alpha_g\|_1 \right)_+,$$

where  $\tilde{\gamma}_g$  represents the row or column vector of an adjacency matrix before the doubly-stochastic normalization. Then, it simultaneously optimized the prediction loss and the exclusive sparsity regularizer,

$$\mathcal{R}(\alpha) = \frac{1}{2} \sum_g \|\alpha_g\|_1^2 = \frac{1}{2} \sum_g \left( \sum_i |\alpha_{g,i}| \right)^2.$$

The regularization was applied on  $\alpha$  rather than  $\tilde{\gamma}$  in order to keep the consistency with the experiment on the proximal operator, where the regularization is applied on free variables. Table 9a and 9b show the experimental results with the proximal gradient and the proposed method, respectively. The proximal method learned the mapping between inputs and targets by reducing the prediction loss, but did not learn the relationship between nodes as well as our method. our method seeks the balance between both by directly optimizing the regularized objective and learns the relationship between nodes more effectively.

Note that it is more appropriate to apply the regularization on  $\gamma$  rather than  $\alpha$  to enable the threshold operator receive the learning signal directly from the regularization term and better learn how to control the trade-offs. To be certain, we also performed the experiments with the regularization on  $\tilde{\gamma}$  before the doubly-stochastic normalization:

$$\mathcal{R}(\tilde{\gamma}) = \frac{1}{2} \sum_g \|\tilde{\gamma}_g\|_1^2 = \frac{1}{2} \sum_g \left( \sum_i |\tilde{\gamma}_{g,i}| \right)^2.$$



Table 10: Traffic speed prediction with GCN and  $l_{1,2}$ -norm regularization on  $\tilde{\gamma}$

$\lambda$	# N.Z.	MAPE(%)	L.R.( $\times 100$ )	
			$k = 1$	$k = 2$
3	1,211	5.4089	70.47	77.04
5	1,161	5.4408	69.73	75.54
10	924	5.4695	70.66	76.24
15	871	5.4696	66.63	72.06

The experimental results with the exclusive sparsity regularization on  $\tilde{\gamma}$  are shown in Table 10. We can see that the direct regularization on  $\tilde{\gamma}$  allows to better learn the relationship between nodes.

Note that we could not apply the  $l_{1,2}$ -norm regularization on the row or column vectors of an adjacency matrix after the doubly-stochastic normalization as their values of  $l_{1,2}$ -norm are scaled to 1 by the normalization. Again, the most appropriate approach is that applying a regularization on the row or column vectors of an adjacency matrix after the doubly-stochastic normalization as in the main paper, but it is not possible for the proximal gradient as discussed above. It shows the flexibility of the proposed approach.

## References

- [1] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*, 2016.
- [2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.
- [3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.
- [4] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL <https://arxiv.org/abs/1704.04861>.
- [5] Gao Huang, Zhuang Liu, and Laurens van der Maaten. Densely connected convolutional networks. In *CVPR*, 2017.
- [6] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
- [7] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.
- [8] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. In *NeurIPS*, 2019.
- [9] Jaehong Yoon and Sung Ju Hwang. Combined group and exclusive sparsity for deep neural networks. In *ICML*, 2017.
- [10] Yang Zhou, Rong Jin, and Steven Chu-Hong Hoi. Exclusive lasso for multi-task feature selection. In *AISTATS*, 2010.