CHARACTERIZING SPARSE CONNECTIVITY PATTERNS IN NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a novel way of reducing the number of parameters in the storage-hungry fully connected classification layers of a neural network by using pre-defined sparsity, where the majority of connections are absent prior to starting training. Our results indicate that convolutional neural networks can operate without any loss of accuracy at less than 0.5% classification layer connection density, or less than 5% overall network connection density. We also investigate the effects of pre-defining the sparsity of networks with only fully connected layers. Based on our sparsifying technique, we introduce the ‘scatter’ metric to characterize the quality of a particular connection pattern. As proof of concept, we show results on CIFAR, MNIST and a new dataset on classifying Morse code symbols, which highlights some interesting trends and limits of sparse connection patterns.

1 INTRODUCTION

Neural networks (NNs) in machine learning systems are critical drivers of new technologies such as image processing, speech recognition, and autonomous driving. Modern NNs are gigantic in size and have millions of parameters, such as the ones described in Alexnet (Krizhevsky et al., 2012), Overfeat (Sermanet et al., 2013) and Zeiler & Fergus (2013). They therefore require an enormous amount of memory and silicon processing during usage. Optimizing a network to improve performance typically involves adding more layers to make it deeper (Simonyan & Zisserman, 2015; Szegedy et al., 2015; He et al., 2016), which further exacerbates the problem of large storage complexity. While the convolutional (conv) layers in these networks do feature extraction, there are a few fully connected (FC) layers at the end performing actual classification. We shall henceforth refer to these classification layers as CLs, whether they are FC or not. Owing to their high density of connections, the majority of network parameters are concentrated in these layers. For example, Alexnet has a total of 13 layers, out of which the 3 CLs account for 95.7% of the network parameters (Zhang et al., 2016).

We shall refer to the spaces between CLs as CL junctions (or simply junctions), which are occupied by connections, or weights. Given the trend in modern NNs, we raise the question – “How necessary is it to have fully connected CLs?” or, in other words, “What if most of the junction connections never existed?” Would the resulting sparse network, when trained and tested, still give competitive performance?” As an example, consider a network with 2 CLs of 100 neurons each and the junction between them has 1000 weights instead of the expected 10,000. Then this is a sparse network with connection density of 10%. Given such a sparse architecture, a natural question to ask is “How can the existing 1000 weights be best distributed so that network performance is maximized?”

In this regard, the present work makes the following contributions. In Section 2, we formalize the concept of sparsity, or its opposite measure density, and explore its effects on different network types. We show that CL parameters are largely redundant and a network pre-defined to be sparse before starting training does not result in any performance degradation. For certain network architectures, this leads to CL parameter reduction by a factor of more than 450, or an overall parameter reduction by a factor of more than 20. In Section 2.4, we discuss techniques to distribute connections across junctions when given an overall network density. We find that to optimize performance, junctions closer to the output need to be denser than those closer to the input. Finally, in Section 3 we formalize pre-defined sparse connectivity patterns using adjacency matrices and introduce the
scatter metric. We show results indicating that scatter is a quick and useful indicator of how good a sparse network is, and infer that randomly connected sparse patterns optimize performance.

2 PRE-DEFINED SPARSITY

As an example of the footprint of modern NNs, AlexNet has a weight size of 234 MB and requires 635 million arithmetic operations only for feedforward processing [Zhang et al., 2016]. It has also been shown that NNs, particularly their FC CLs, have an excess of parameters and tend to overfit to the training data [Denil et al., 2013], resulting in inferior performance on test data. The following paragraph describes several previous works that have attempted to reduce parameters in NNs.

Dropout (deletion) of random neurons [Srivastava et al., 2014] incurs the disadvantage of having to train multiple differently configured networks, which are finally combined to regain the original full size network. [Chen et al., 2015] randomly forced the same value on collections of weights, but acknowledged that “a significant number of nodes [get] disconnected from neighboring layers.” Other sparsifying techniques such as pruning and quantization [Han et al., 2016; 2015; Zhou et al., 2016; Gong et al., 2014] first train the complete network, and then perform further computations to delete parameters. [Sindhwani et al., 2015] attempted to impose structure on network parameters by using low rank matrices. [Srinivas et al., 2016] proposed a regularizer to reduce parameters in the network, but acknowledged that this increased training complexity. In general, all of these architectures deal with the complete non-sparsified CLs at some point of time during their usage cycle and therefore, fail to permanently solve the parameter explosion problem of modern NNs.

2.1 OUR METHODOLOGY

Our attempt to simplify NNs is to pre-define the level of sparsity, or connection density, in a network prior to the start of training. This means that our network always has lesser connections than its FC counterpart; the weights which are absent never make an appearance during training or inference. In our notation, a NN will have \( J \) junctions, i.e. \( J + 1 \) layers, with \( \{N_1, N_2, \cdots, N_{J+1}\} \) being the number of neurons in each layer. \( N_i \) and \( N_{i+1} \) are respectively the number of neurons in the preceding (left) and succeeding (right) layers of junction \( i \). Every left neuron has a fixed number of edges going from it to the right, and every right neuron has a fixed number of edges coming into it from the left. These numbers are defined as fan-out \( (fo_i) \) and fan-in \( (fi_i) \), respectively. For a conventional FC layer, \( fo_i = N_{i+1} \) and \( fi_i = N_i \). For our sparse architecture, \( fo_i < N_{i+1} \) and \( fi_i < N_i \), such that \( N_i \times fo_i = N_{i+1} \times fi_i = W_i \), the number of weights in junction \( i \). Having a fixed \( fo_i \) and \( fi_i \) ensures that all neurons in a junction contribute equally and none of them get disconnected, since that would lead to loss of information. The connection density in junction \( i \) is given as \( W_i / (N_i N_{i+1}) \) and the overall CL connection density is defined as \( \left( \sum_{i=1}^{J} W_i \right) / \left( \sum_{i=1}^{J} N_i N_{i+1} \right) \).

Note that earlier works such as [Dey et al., 2017b; a] have proposed hardware architectures that leverage pre-defined sparsity to speed up training. However, a complete analysis of methods to pre-define connections, its possible gains on different kinds of modern deep NNs and a test of its limits via a metric quantifying its goodness has been lacking. [Bourely et al., 2017] introduced a metric based on eigenvalues, but only ran limited tests on MNIST. The following subsections analyze the characteristics of our method of pre-defined sparsity in more detail.

2.2 NETWORK EXPERIMENTS

We experimented with networks operating on 4 datasets of different dimensionalities: a) CIFAR10 and b) CIFAR100 image classification, both of which have 3D image inputs consisting of width, height and features (depth), c) MNIST digit classification, which has 2D image inputs having width and height, and d) Morse code symbol classification, a new dataset described in Dey (2017). Each input in d) has a width of 64 values representing Morse code symbols, and there are 64 output classes representing different English letters, numbers, and symbols.
2.2.1 CIFAR

We used the original CIFAR10 and CIFAR100 datasets without data augmentation. Our network has 6 conv layers with number of filters equal to [64, 64, 128, 128, 256, 256]. Each has window size 3x3. The outputs are normalized before applying ReLU non-linearity. A max-pooling layer of pool size 2x2 succeeds every pair of conv layers. This structure finally results in a layer of 4096 neurons, which is followed by the CL layers. We used the Adam optimizer and ReLU activation for the CLs.

Our results in Section 2.2.2 indicate that later CL junctions (i.e. closer to the outputs) should be denser than earlier ones (i.e. closer to the inputs). Moreover, since most networks have a tapering structure where $N_i$ monotonically decreases as $i$ increases, more parameter savings can be achieved by making earlier layers less dense. Accordingly, we picked the CL junction densities as given in Table 1. Note that the phrase ‘conv3’ denotes 3 CL junctions corresponding to a CL neuron configuration of (4096, 512, 256, 16) for CIFAR10 and (4096, 512, 256, 128) for CIFAR100. For ‘conv2’, the only difference is that the 256-neuron layer is missing. ‘MNIST CL’ and ‘Morse CL’ refer to the CL only networks described in Sections 2.2.2 and 2.2.3, for which we have only shown some of the more important configurations in Table 1.

As an example, consider the first network in ‘Cifar10 conv2’ which has $f_{o1} = f_{o2} = 1$. This means that the individual junction densities are (4096 × 1)/(4096 × 512) = 0.2% and (512 × 1)/(512 × 16) = 6.3%, to give an overall CL density of (4096×512×1)/(4096×512+512×16) = 0.22%. In other words, while the FC network would have been 100% dense with 2,097,152 + 8192 = 2,105,344 CL weights, our pre-defined sparse network always operates using 4096 + 512 = 4608 weights, which is 457 times less. Note that the sparse CL weights in each junction are distributed as fixed, but randomly generated patterns, with the constraints of fixed fan-in and fan-out.

<table>
<thead>
<tr>
<th>Net</th>
<th>Junction fan-outs</th>
<th>CL Junction Densities (%)</th>
<th>Overall CL Density (%)</th>
<th>Net</th>
<th>Junction fan-outs</th>
<th>CL Junction Densities (%)</th>
<th>Overall CL Density (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cifar10</td>
<td>1</td>
<td>0.2,6.3</td>
<td>0.22</td>
<td>Cifar10</td>
<td>1</td>
<td>0.2,4.6.3</td>
<td>0.22</td>
</tr>
<tr>
<td>conv2</td>
<td>1.8</td>
<td>0.2,50</td>
<td>0.39</td>
<td>conv3</td>
<td>1.18</td>
<td>0.2,4.50</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.2,16</td>
<td>0.2,8,100</td>
<td>0.41</td>
</tr>
<tr>
<td>Cifar100</td>
<td>1.1</td>
<td>0.2,0.8</td>
<td>0.21</td>
<td>Cifar100</td>
<td>1.1</td>
<td>0.2,4.0.8</td>
<td>0.22</td>
</tr>
<tr>
<td>conv2</td>
<td>1.8</td>
<td>0.2,6.3</td>
<td>0.38</td>
<td>conv3</td>
<td>1.1,16</td>
<td>0.2,4.13</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td>1.32</td>
<td>0.2,25</td>
<td>0.95</td>
<td></td>
<td>1.2,32</td>
<td>0.2,8,25</td>
<td>0.59</td>
</tr>
<tr>
<td>MNIST</td>
<td>1.5</td>
<td>0.1,50</td>
<td>0.29</td>
<td>MNIST</td>
<td>4.2</td>
<td>38,100</td>
<td>38.3</td>
</tr>
<tr>
<td>conv2</td>
<td>4.10</td>
<td>0.5,100</td>
<td>0.83</td>
<td>CL</td>
<td>2.5</td>
<td>1.8,50</td>
<td>2.39</td>
</tr>
<tr>
<td></td>
<td>16,10</td>
<td>2,100</td>
<td>2.35</td>
<td>Morse CL</td>
<td>512,32</td>
<td>50,50</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 1 shows the results for CIFAR. Subfigures (a), (b), (d) and (e) show classification performance on validation data as the network is trained for 30 epochs. The different lines correspond to different overall CL densities. Subfigures (c) and (f) show the best validation accuracies after 1, 5 and 30 epochs for the different CL densities.

Notice from subfigures (c) and (f) that the sparse networks generally start training quicker, as evidenced by their higher accuracies compared to FC after 1 epoch of training. This trend continues for the first few epochs, but eventually the FC network catches up. It is important to note that the final accuracies (the numbers at the top of each column) show negligible performance degradation for these extremely low levels of density, not to mention some cases where they outperform FC. Note that the final accuracies stayed almost constant after 20 epochs as evidenced by the other subfigures.

2.2.2 MNIST

We used 2 different kinds of networks when experimenting on MNIST (no data augmentation). The first was a conv network with 2 conv layers having 32 and 64 filters of size 5x5 each, each followed by a 2x2 max pooling layer. This results in a layer of 3136 neurons, which is followed by 2 CLs having 784 and 10 neurons, i.e. 2 junctions overall. Activations and the optimizer are the same as in Section 2.2.1. Fig. 2(a) and (b) show the results. Due to the simplicity of the overall network,
performance starts degrading at higher densities compared to CIFAR. However, a network with CL density 2.35% still gives better accuracy than FC. Note that the total number of weights is 0.11M for this network, i.e. (conv + sparse CLs), which is only 4.37% of the original (conv + FC CLs).

The second network we used for MNIST only had CLs, with 784 input, 112 hidden and 10 output neurons, i.e. 2 junctions overall. Activations are ReLU and the optimizer used is stochastic gradient descent (SGD). Figure 2(c) shows that performance drops off at higher densities for this MNIST network as compared to the one with conv layers. However, a density of 38.29% (circled) gives same accuracy as the FC case, while a 75% dense network slightly outperforms FC. This leads us to believe that even for such a CL only network, the storage and computational complexity can be reduced by 2-3x.

2.2.3 Morse

The Morse code dataset presents a harder challenge for sparsity because it has 64 output classes and the distinctions between inputs belonging to different classes is small. Unlike MNIST where an image of a 7 will have a considerably different 784-input pattern than a 3, the 64-input pattern for a Morse codeword ‘. . . . ’ can be easily confused with the codeword ‘. . . .’ . The smaller size of inputs means that each input neuron is more significant and so performance degrades more quickly as density is reduced. Our network had 64 input and output neurons and 1024 hidden layer neurons, i.e. 3 CLs and 2 junctions. We again used ReLU activations and SGD. The results are shown in Fig. 3(a). Note that the network can be made 50% dense, i.e. half the parameters saved, with negligible degradation in accuracy.
2.3 Analyzing the Results of Pre-defined Sparsity

Our results indicate that for deep networks having several conv layers, there is severe redundancy in the CLs. As a result, they can be made extremely sparse without hampering network performance, which leads to huge memory savings. If the network only has CLs, the amount of density reduction achievable without performance degradation is smaller. This can be explained using the argument of relative importance. For a network which extensively extracts features and processes its raw input data via conv filters, the input to the CLs can already substantially discriminate between inputs belonging to different classes. As a result, the importance of the CLs’ functioning is lesser as compared to a network where they process the raw inputs.

The computational savings by sparsifying CLs, however, are not as large because the conv layers dominate the computational complexity. Other types of NNs, such as restricted Boltzmann machines, have higher prominence of CLs than CNNs and would thus benefit more from our approach.

Table 2 shows the overall memory and computational gains obtained from pre-defining CLs to be sparse for our networks. The number of sparse CL parameters (params) are calculated by taking the minimum overall CL density at which there is no accuracy loss. Note that the number of operations (ops) for CL layers is nearly the same as their number of parameters, hence are not explicitly shown.

<table>
<thead>
<tr>
<th>Net</th>
<th>CLs / Conv Total Layers</th>
<th>Conv Params (M)</th>
<th>Conv Ops (B)</th>
<th>FC CL Params (M)</th>
<th>Sparse CL Params (M)</th>
<th>Overall Param % Reduction</th>
<th>Overall Op % Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morse CL</td>
<td>2/2</td>
<td>0</td>
<td>0.13</td>
<td>0.07</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>MNIST CL</td>
<td>2/2</td>
<td>0</td>
<td>0.07</td>
<td>0.03</td>
<td>61.7</td>
<td>61.7</td>
<td>61.7</td>
</tr>
<tr>
<td>MNIST conv2</td>
<td>2/6</td>
<td>0.05</td>
<td>1.24</td>
<td>0.06</td>
<td>95.63</td>
<td>18.29</td>
<td></td>
</tr>
<tr>
<td>Cifar10 conv2</td>
<td>2/17</td>
<td>1.15</td>
<td>0.15</td>
<td>2.11</td>
<td>0.005</td>
<td>64.63</td>
<td>1.35</td>
</tr>
<tr>
<td>Cifar100 conv2</td>
<td>2/17</td>
<td>1.15</td>
<td>0.15</td>
<td>2.16</td>
<td>0.02</td>
<td>64.76</td>
<td>1.38</td>
</tr>
<tr>
<td>Cifar10 conv3</td>
<td>3/18</td>
<td>1.15</td>
<td>0.15</td>
<td>2.23</td>
<td>0.009</td>
<td>65.83</td>
<td>1.43</td>
</tr>
<tr>
<td>Cifar100 conv3</td>
<td>3/18</td>
<td>1.15</td>
<td>0.15</td>
<td>2.26</td>
<td>0.013</td>
<td>65.99</td>
<td>1.45</td>
</tr>
</tbody>
</table>

2.4 Distributing Individual Junction Densities

In Fig. 3(a) we showed that performance drops off for the Morse network as the density of connections is decreased below 25%. Note that this network has symmetric junctions since each will have $64 \times 1024 = 65,536$ weights to give a total of 131,072 weights in the FC case. Consider an example where overall density of 50% (i.e. 65,536 total weights) is desired. This can be achieved in multiple ways, such as making both junctions 50% dense, i.e. 32,768 weights in each. Here we explore if individual junction densities contribute equally to network performance.

Figure 3(b) and (c) sweep junction 1 and 2 connectivity densities on the x-axis such that the resulting overall density is fixed at 25% for (b) and 50% for (c). The black vertical line denotes where the densities are equal. Note that peak performance in both cases is achieved to the left of the black line,
such as in (c) where junction 2 is 75% dense and junction 1 25%. This suggests that later junctions need more connections than earlier ones. See Appendix Section 5.1 for more details.

3 Connectivity Patterns

We now introduce adjacency matrices to describe junction connection patterns. Let $A_i \in \{0, 1\}^{N_i+1 \times N_i}$ be the (simplified) adjacency matrix of junction $i$, such that element $[A_i]_{j,k}$ indicates whether there is a connection between the $j$th right neuron and $k$th left neuron. $A_i$ will have $f_i 1$’s on each row and $f_0 1$’s on each column. These adjacency matrices can be multiplied to yield the effective connection pattern between any 2 junctions $X$ and $Y$, i.e. $A_{X:Y} = \prod_{i \in Z_{\geq 0}}^{X} A_i \in \mathbb{Z}^{N_Y+1 \times N_X}$, where element $[A_{X:Y}]_{j,k}$ denotes the number of paths from the $k$th neuron out of $N_X$ to the $j$th neuron out of $N_Y+1$. For the special case where $X = 1$ and $Y = J$ (total number of junctions), we obtain the input-output adjacency matrix $A_{1:J}$. As a simple example, consider the $\{8, 4, 4\}$ network shown in Fig. 4 where $f_0 = 1$ and $f_0 = 2$, which implies that $f_1 = f_2 = 2$. $A_1$ and $A_2$ are adjacency matrices of single junctions. The input-output adjacency matrix $A_{1:2}$ is formed by multiplying $A_2$ with $A_1$, and we obtain $f_0_{1:2} = f_0_1 f_0_2 = 2$ and $f_{1:2} = f_1 f_2 = 4$. Note that this equivalent junction 1:2 is only an abstract concept that aids visualizing how neurons connect from the inputs to the outputs. It has no relation to the overall network density.

![Figure 4: An example of adjacency matrices and equivalent junctions.](image)

We now attempt to characterize the quality of a sparse connection pattern, i.e. we try to find the best possible way to connect neurons to optimize performance. Since sparsity gives good performance, we hypothesize that there exists redundancy / correlated information between left neurons of a junction which contain either raw features or processed feature maps. Intuitively, we assume that the left neurons can be grouped into windows depending on the dataset dimensionality. MNIST would have 2D windows, for example, a window of input neurons could be the top-left 16th of the image, i.e. the leftmost 7 pixels from each of the topmost 7 rows. The next window would be the next group of 7x7 until the 16th window is at the bottom right. CIFAR would have 3D windows, each containing a fraction of the image and its features. Given such windows, we will try to maximize the number of windows to which each right neuron connects, the idea being that each right neuron should get some information from all portions of the image in order to capture global view. As an example, consider the MNIST output neuron representing digit 2. Let’s say the sparse connection pattern is such that when the connections to output 3 are traced back to the input layer, they all come from the top half of the image. This would be undesirable since the top half of an image of a 2 can be mistaken for a 3. A good sparse connection pattern will try to avoid such scenarios by spreading the connections to any right neuron across as many input windows as possible. The problem can also be mirrored so that every left neuron connects to as many different right windows as possible. This ensures that local information from left neurons is spread to different parts of right neurons.

The window size is chosen to be the minimum possible such that the ideal number of connections from or to it remains integral. The example from Fig. 4 is reproduced in Fig. 5. Since $f_1 = 2$, 

Figure 5: Window adjacency matrices and scatter. Green neurons indicate ideal connectivity. The hidden layer is split into 2 to show separate constructions of $A_{w1}^l$ and $A_{w1}^r$.

Figure 6: Construction of window adjacency matrices.

the inputs must be grouped into 2 windows so that ideally 1 connection from each reaches every output neuron. If instead the inputs are grouped into 4 windows, the ideal number would be half of a connection, which is not achievable. In order to achieve the minimum window size, we let the number of left windows be $f_l$ and the number of right windows be $f_r$. So in junction $i$, the left window size is $w_{il} = N_i/f_l$ neurons and the right window size is $w_{ir} = N_i/f_r$ neurons. Then we construct left- and right-window adjacency matrices $A_{w1}^{il} \in \mathbb{Z}_{N_i+1 \times w_{il}}$ and $A_{w1}^{ir} \in \mathbb{Z}_{N_i \times w_{ir}}$ by summing up entries of $A_i$ as shown in Fig. 6. The window adjacency matrices describe connectivity between windows and neurons on the opposite side. Ideally, every window adjacency matrix for a single junction should be the all 1s matrix, which signifies exactly 1 connection from every window to every neuron on the opposite side. Note that these matrices can also be constructed for multiple junctions, i.e. $A_{X,Y}^{w1}$ and $A_{X,Y}^{w1}$, by multiplying matrices for individual junctions. See Appendix Section 5.2 for more discussion.

3.1 Scatter

Scatter is a proxy for the performance of a NN. It is useful because it can be computed in a fraction of a second and used as a filter to reject networks which will likely yield poor accuracy without having to train them (which, for 30 epochs on a GPU, generally takes more than an hour). To compute scatter, we count the number of entries greater than or equal to 1 in the window adjacency matrix. The reason behind treating entries greater than 1 as 1 is that if a particular window gets more than its fair share of connections to a neuron on the opposite side, then it is depriving some other neuron from getting its fair share. So this should not be looked on as a good thing by having a greater value for that particular element. The scatter is the average of the count, i.e. for junction $i$:

$$S_{if} = \frac{1}{w_{il}N_{i+1}} \sum_{j=1}^{N_{i+1}} \sum_{k=1}^{w_{il}} \mathbb{I}(A_{w1}^{il}_{j,k} \geq 1), \quad S_{ir} = \frac{1}{N_iw_{ir}} \sum_{j=1}^{N_i} \sum_{k=1}^{w_{ir}} \mathbb{I}(A_{w1}^{ir}_{j,k} \geq 1).$$

(1)
Subscripts $f$ and $b$ denote forward and backward, indicating the direction of data flow for a particular window-neuron combination. As an example, we consider $A^{\text{init}}_{1,1}$ in Fig. 5 which has a scatter value $S_{1f} = 6/8 = 0.75$. The other scatter values can be computed similarly to form the scatter vector $\vec{S} = [S_{1f}, S_{1b}, S_{2f}, S_{2b}, S_{f}, S_{b}]$, the final 2 values corresponding to junction 1:2. Notice that $\vec{S}$ will be all 1s for an FC network, which is the ideal case. Incorporating sparsity leads to reduced $\vec{S}$ values. The final scatter metric $S \in [0,1]$ is the minimum value in $\vec{S}$, i.e. 0.75 for Fig. 5. Our experiments indicate that any low value in $\vec{S}$ leads to bad performance, so we picked the critical minimum value.

3.2 Analysis and Results of Scatter

We ran experiments to evaluate scatter using a) the Morse CL only network with $f_o = 128, 8$, b) an MNIST CL only network with $(1024, 64, 16)$ neuron configuration and $f_o = 1, 4$, and c) the ‘conv2’ CIFAR10 network with $f_o = 1, 2$. We found that high scatter indicates good performance and the correlation is stronger for networks where CL layers have more importance, i.e. CL only networks as opposed to conv. This is shown in the performance vs. scatter plots in Figure 7, where (a) and (b) show the performance predicting ability of scatter better than (c). Note that the random connection patterns used so far have the highest scatter and occur as the rightmost points in each subfigure. The other points are obtained by specifically planning connections. We found that when 1 junction was planned to give corresponding high values in $\vec{S}$, it invariably led to low values for another junction, leading to a low $S$. This explains why random patterns generally perform the best.

Figure 7: Network performance vs. scatter for CL only networks of (a) Morse (b) MNIST, and convolutional network with 2 CL junctions of (c) CIFAR10. All minimum values that need to be considered to differentiate between connection patterns are bolded.

$\vec{S}$ is shown alongside each point. When $S$ is equal for different connection patterns, the next minimum value in $\vec{S}$ needs to be considered to differentiate the networks, and so on. Considering the Morse results, the leftmost 3 points all have $S = \frac{1}{8}$, but the number of occurrences of $\frac{1}{8}$ in $\vec{S}$ is 3 for the lowest point (8% accuracy), 2 for the second lowest (12% accuracy) and 1 for the highest point (46% accuracy). For the MNIST results, both the leftmost points have a single minimum value of $\frac{0.64}{1.0}$ in $\vec{S}$, but the lower has 2 occurrences of $\frac{1}{4}$ while the upper has 1. In summary, although we defined $S$ as a single value for convenience, there may arise cases when other elements in $\vec{S}$ are important.

4 Conclusion and Future Work

This paper discusses the merits of pre-defining sparsity in CLs of neural networks. This leads to significant reduction in parameters and, in several cases, computational complexity as well, without any loss of performance. In general, the smaller the fraction of CLs in a network, the more redundancy there exists in their parameters. If we can achieve similar results (i.e., 0.2% density) on Alexnet for example, we would obtain 95% reduction in overall parameters. Coupled with hardware acceleration designed for pre-defined sparse networks, we believe our approach will lead to more aggressive exploration of network structure. In particular, network connectivity can be guided by the scatter metric, which is closely related to network performance, and by optimally distributing connections across junctions. Future work would involve extension to convolutional layers.
REFERENCES


5 Appendix

5.1 More on Distributing Individual Junction Densities

Section 2.4 showed that when overall CL density is fixed, it is desirable to make junction 2 denser than junction 1. It is also interesting to note, however, that performance falls off more sharply when junction 1 density is reduced to the bare minimum as compared to treating junction 2 similarly. This is not shown in Fig. 3 due to space constraints. We found that when junction 1 had the minimum possible density and junction 2 had the maximum possible while still satisfying the fixed overall, the accuracy was about 36% for both subfigures (b) and (c). When the densities were flipped, the accuracies were 67% for subfigure (b) and 75% for (c) in Figure 3.

5.2 Dense Cases of Window Adjacency Matrices

As stated in Section 3.1, window output matrices for several junctions can be constructed by multiplying the individual matrices for each component junction. Consider the Morse network as described in Section 3.2. Note that \( f_{o1:2} = 128 \times 8 = 1024 \) and \( f_{i1:2} = 8 \times 128 = 1024 \). Thus, for the equivalent junction 1:2 which has \( N_1 = 64 \) left neurons and \( N_3 = 64 \) right neurons, we have \( f_{o1:2} > N_3 \) and \( f_{i1:2} > N_1 \). So in this case \( w_{il} \) and \( w_{ir} \) will be rounded up to 1, and both the ideal window adjacency matrices \( A_{wil}^{w1} \) and \( A_{wlr}^{w2} \) will be all 16’s matrices since the ideal number of connections from each window to a neuron on the opposite side is \( 1024/64 = 16 \). This is a result of the network having sufficient density so that several paths exist from every input neuron to every output neuron.