TreeConnect: A Sparse Alternative to Fully Connected Layers

Oliver Richter
Department of Information Technology and Electrical Engineering
ETH Zurich
Zurich, Switzerland
richtero@ethz.ch

Roger Wattenhofer
Department of Information Technology and Electrical Engineering
ETH Zurich
Zurich, Switzerland
wattenhofer@ethz.ch

***Abstract*—We present a generally applicable tree-like sparse multilayer architecture that has a balanced connection from all input neurons to all output neurons. If the ratio between input and output neurons is fixed, the parameters required by our architecture scale with $\mathcal{O}(n^{1.5})$ as compared to $\mathcal{O}(n^2)$ in a fully connected layer, where n is the number of input neurons. Our sparse 2-layer architecture performs similar and/or superior when compared to its fully connected 1-layer and 2-layer counter parts on the IMDB review sentiment classification task, the Reuters news categorization task and the CIFAR-10 image classification task.**

***Index Terms*—Deep Learning, Neural Networks, Sparsity, Text Sentiment Analysis, Text Categorization, Image Classification**

I. Introduction

With the continuous increase in computation power and the success of deep learning in recent years, we see a trend to apply neural networks to ever more complex tasks. A prime example here is image classification, where problems shifted from handwritten digit classification (MNIST [1]), gray-scale images of 28x28 pixels with 10 classes, to general image classification, e.g., ImageNet [2], RGB images of 469x387 pixels on average with 1000 classes. To solve these complex tasks, we often require a high dimensional latent representation such that the neural network has enough internal representation power to learn the complex relations. Meanwhile, a full connectivity between high dimensional internal representations requires large amounts of memory and computation power. A full connectivity often also leads to networks with too much capacity which over-fit the training set. Sparsity is therefore one of the success factors of convolutional neural networks [3], which are applicable to a variety of tasks [4] and used in all state-of-the-art modern architectures for image classification [5]–[12]. However, up to now, sparse architectures either try to introduce problem specific priors [3], [4], [13]–[16] or focus on reducing the number of connections by a certain factor [17]. In contrast, the question we seek to answer with this paper is, how sparse we can make a network under the constraint of a full connectivity between all input and output neurons. An extreme variant is to connect each input and output node with just a single path. Or, from the vantage point of an output respectively input node, connecting to all the input respectively output nodes with just a tree. Can such a sparse network still learn, or is there a price to be paid? We show that the generally applicable sparse architecture we propose has a performance similar or superior to fully connected layers while using much fewer parameters.

II. Related Work

Denil et al. [18] show that many deep neural network architectures experience immense redundancy in their parameterization. Since sparsity in neural networks limits the memory and computation required to compute a given inference, many approaches are discussed in the literature which we roughly divide into three categories:

To the first category we refer to as *a posteriori* sparsity, but more often it is referred to as pruning [19]–[21]. The goal of a posteriori sparsity is to prune unused connections after the training in an effort to reduce the network size without losing too much accuracy when compared to the full network. Han et al. [21] show that popular neural network architectures can be compressed by 35x to 49x without a drop in accuracy, suggesting that many connections in the neural networks are not needed in the first place. A posteriori sparsity is basically for free, as one does not risk much by simply removing redundant edges.

The second category we call *learned* or *dynamic* sparsity. Castellano et al. [20] and Han et al. [22] suggest to prune during training, while Louizos et al. [23] propose to use a learned L_0 regularization enforcing sparsity during training. Pruning during training is more risky than pruning a posteriori as one does not yet know whether weights will remain low, or whether they might be needed in later training. Besides enforcing hard sparsity constraints during training, any weight decay regularization gradually decreases the unused weights to 0 and thereby sparsifies the network as well [24]. Glorot et al. [25] attribute the success of rectified linear units as non-linear activation functions also to the sparse representations they introduce. Randomized approaches, that introduce sparsity on the representations (Dropout, [26]) or connections (DropConnect, [27]) during training also became popular because of their regularization and generalization capabilities.

In contrast to introducing sparsity during or after training, we focus on developing a sparse network architecture *a priori*,

i.e., *before* training. A priori sparsity (pruning before training) is the most tricky of the three, since the network is not allowed to develop connections where it might need them. Nevertheless it is frequently done, most prominently in the domain of convolutional neural networks (CNNs) [3]. Some work, e.g., [6], [7], only uses convolutional layers for image classification. Besides the success of CNNs, recent work [14]–[16] suggests to group neurons into capsules for structured entity representations, hence limiting the connectivity between capsules. Most recent architecture suggestions [5]–[12] focus mainly on the CNN part of the network. In this paper however, we take a closer look at the fully connected layers without weight sharing, which many architectures (e.g. [5], [8]–[12]) use for the final classification. While problem specific architectures (e.g., [13]) can improve performance, we aim in this work to develop a generally applicable sparse architecture. Closest related to our work is the work of Bourely et al. [17], who propose several architectures for sparsity within one layer as alternative to fully connected layers and show that sparsity can increase generalization due to a limited network capacity which is not as prone to over-fitting. In contrast to Bourely et al. we propose a multilayer sparse architecture that guarantees a balanced connectivity between input and output neurons. We hypothesize and show in our experiments that an unbalanced connectivity introduced by random sparsity can result in a bigger performance variation across training runs and worse performance overall, since the architecture is more sensitive to initialization. Also, while Bourely et al. only use one learning rate in their experiments, we find that different architecture types need different learning rates for optimal performance.

The literature also provides a large body of work [28]–[33] that introduces linear dependencies between weights to compress the computation and storage requirements of ordinary neural networks. Our architecture is similar in that it groups inputs together in the first layer. However, the non-linearity in the hidden layer allows it to approximate a larger set of functions while the balanced connectivity makes it less sensitive to initialization.

III. TreeConnect

In our proposed architecture, hidden nodes are shared among trees but no two paths between any input and output neuron exist. A small example 2-layer network can be seen in Figure 1.

This simple structure is, due to its full connectivity between input and output, generally applicable to all problems. We now describe the properties of 2-layer TreeConnect networks and then generalize to l-layer networks.

The number of trainable parameters N_{FC} in a fully connected layer with bias is $N_{FC} = n \cdot m + m$, where n is the number of input neurons and m is the number of output neurons. Similarly, the number of trainable parameters in a 2-layer fully connected network is $N_{2FC} = n \cdot h + h + h \cdot m + m$, where h is the number of neurons in the hidden layer. In a 2-layer TreeConnect network, the input layer neurons are

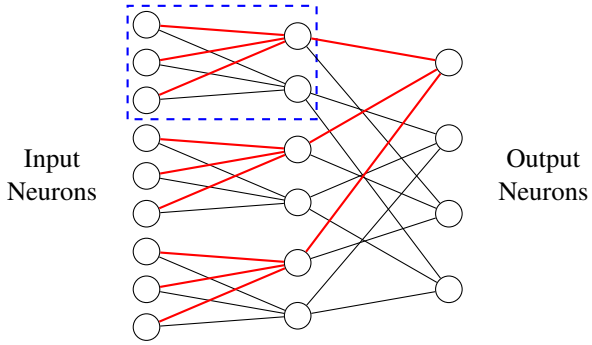

Fig. 1. Example 2-Layer TreeConnect network with 9 input neurons and 4 output neurons. The highlighted red connections show the tree structure when viewed from the output neuron while the dashed blue box shows a fully connected subnet in the first layer, also referred to as *channel* in this paper.

partitioned into c equally sized channel inputs, and partition-wise fully connected to the corresponding $\frac{h}{c}$ neurons of the hidden layer. Additionally, each output neuron is connected to c neurons of the hidden layer, one from each channel. Therefore, the total amount of trainable parameters in a TreeConnect network is

$$N_{TreeConnect} = c \cdot \frac{n}{c} \cdot \frac{h}{c} + h + c \cdot m + m = \frac{n \cdot h}{c} + h + c \cdot m + m$$

Note that by varying the hidden layer size h, we can trade off sparsity vs. hidden representation size. On one end of the spectrum, if $h = c$, the number of parameters in the TreeConnect network scales linearly with n, but the small hidden layer acts as a bottleneck in the information flow. On the other end of the spectrum, if $h = c \cdot m$, each output neuron is connected through a completely separated tree to all input neurons, i.e., no hidden neurons are shared among trees. However, in this case the number of parameters scales as $\mathcal{O}(n \cdot m)$, i.e., the architecture is not sparser than a fully connected layer.

If we define $r = \frac{m}{n}$ to be the compression ratio and assume $\sqrt{n}$ and $\sqrt{m}$ to be integer numbers, we can choose $c = \sqrt{n}$ and $h = \sqrt{n \cdot m} = n \cdot \sqrt{r}$ such that the number of parameters in the TreeConnect network is

$$N_{TreeConnect} = \frac{n^2 \cdot \sqrt{r}}{\sqrt{n}} + n \cdot \sqrt{r} + r \cdot n\sqrt{n} + r \cdot n$$

For fixed r this number scales as $\mathcal{O}(n^{1.5})$ as compared to $\mathcal{O}(n^2)$ for a fully connected layer. With $h = \sqrt{n \cdot m}$ the compression from input to hidden is equal the compression from hidden to output. We believe this to be a good trade-off between sparsity and hidden layer size.

Formally, for a given input activation $\vec{x}$, the output activation $\vec{y}$ of a 2-layer TreeConnect network can be computed as

$$\vec{z} = \sigma_z(\mathbf{W_1}\vec{x} + \vec{b_z})$$

$$z_i^* = z_{c \cdot i \bmod h} \text{ for } i \in \{0, ..., h-1\}$$

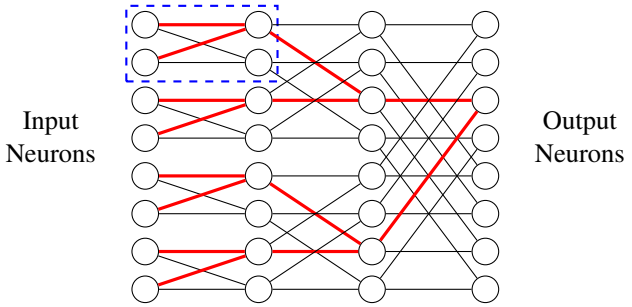

Fig. 2. Example 3-Layer TreeConnect network with 8 input neurons and 8 output neurons. The highlighted red connections show the tree structure when viewed from an output neuron while the dashed blue box shows a fully connected subnet in the first layer.

$$\vec{y} = \sigma_y(\mathbf{W_2}\vec{z^*} + \vec{b_y})$$

where $\mathbf{W_1}$ and $\mathbf{W_2}$ are block diagonal weight matrices, $\vec{b_z}$ and $\vec{b_y}$ are the bias vectors and $\sigma_z(\cdot)$ and $\sigma_y(\cdot)$ are the activation functions of the hidden and output layer, respectively. Since $\mathbf{W_1}$ and $\mathbf{W_2}$ are block diagonal weight matrices with non-overlapping blocks, the computation of TreeConnect network activations can efficiently be parallelized.

If $\sigma_z(\cdot)$ is chosen to be the identity function, the TreeConnect network is equivalent to a fully connected layer with linear dependencies between the weights, similar to low-rank matrix neural networks proposed in the literature [28], [29].

If one wants to emphasize the non-linear multi-layer structure, one can generalize the TreeConnect architecture to a deeper, l-layer architecture. An example 3-layer architecture can be seen in Figure 2. With a similar deduction as for the 2-layer version it can be shown that the trainable parameters of a general l-layer TreeConnect network scale as $\mathcal{O}(l \cdot n^{(2 \cdot l-1)/l})$, while a l-layer fully connected network needs $\mathcal{O}(l \cdot n^2)$. However, since the gain in parameter savings is most profound in the 2-layer version we focus the experiments in this paper on simple 2-layer TreeConnect networks. Note that for deeper architectures one can also concatenate multiple 2-layer TreeConnect networks.

IV. Experiments

To evaluate the proposed architecture, we implement several variations and test their training and generalization performance on the IMDB sentiment classification dataset [34], the Keras [35] version of the Reuters newswire text categorization dataset and the CIFAR-10 [36] image classification dataset.

In the TreeConnect network implementation, we grouped fully connected neurons together and used a glorot-uniform initialization [37] for all layers and architectures. We train all architectures with mini batches of 64 samples, using three different initial learning rates (0.001, 0.0001 and 0.00001) for the Adam optimizer [38] with all other hyper-parameters left at the default values. For the sake of visibility, we only report the training curve with the learning rate best suited for the specific architecture in terms of maximal achieved validation accuracy over the course of training. In all architectures we use

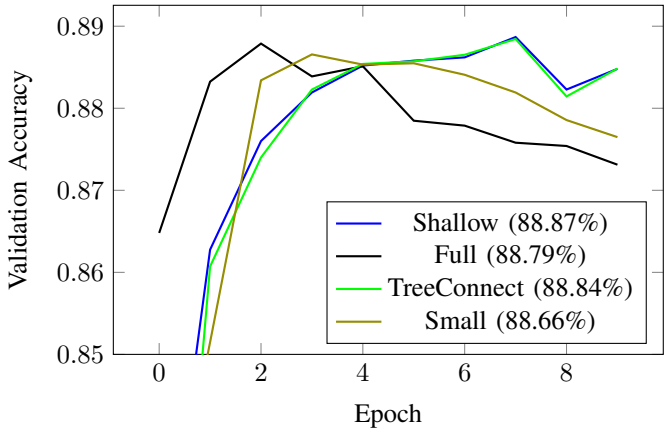

Fig. 3. IMDB sentiment classification validation set accuracy over the course of training. The number in brackets is the maximal achieved validation accuracy throughout the training.

ReLU [25] non-linearities as activation function on all hidden layers, if not stated otherwise. With our experiments we seek to answer the following questions:

1) How does a 2-layer TreeConnect network compare to a 1-layer fully connected network with equal input and output dimensions?
2) How does a 2-layer TreeConnect network compare to a 2-layer fully connected network with equal input, hidden and output dimensions?
3) How does a 2-layer TreeConnect network compare to a 2-layer fully connected network with a similar number of trainable parameters?

We base our implementation on Keras [35] layers and leave a run time optimized implementation for future work.[1] Note also that our goal was not to achieve state of the art performance, but rather to give an empirical comparison of different architectures. In experiments with low variance between training runs we report a typical run, otherwise we report average and standard deviation of 5 training runs.

In the following we use the shorthand notations FCn to denote a fully connected layer with n output neurons and P(k) or P(k, k) to denote an average pooling operation, 1- or 2-dimensional, with a kernel of size k or $k \times k$, respectively. Further, we use TC((c, a - b) - m) to denote a 2-layer TreeConnect network with c channels in the first layer, each with a input neurons and b output neurons in the hidden layer, and m output neurons. As an example, the network in Figure 1 would be described as TC((3, 3 - 2) - 4).

A. IMDB Sentiment Classification

The IMDB movie review sentiment classification dataset [34] contains 50,000 movie reviews, split equally into test and validation set. The goal is a binary classification into positive and negative reviews. We preprocess the dataset by discarding all but the 5,000 most frequent words and

[1] An implementation of our final CIFAR10 experiment is available at https://github.com/OliverRichter/TreeConnect.

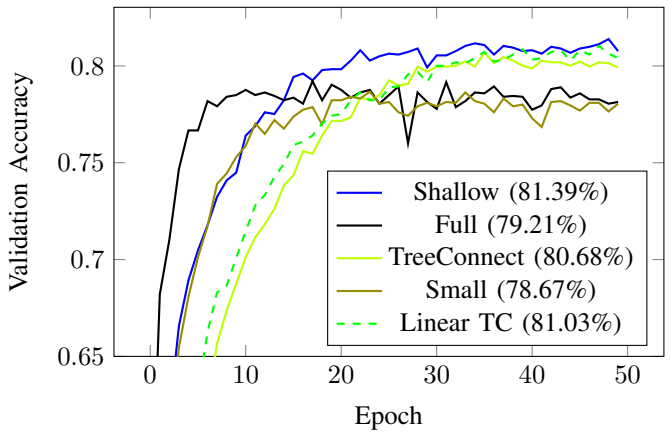

Fig. 4. Reuters news categorization validation set accuracy. The number in brackets is the maximal achieved validation accuracy throughout the training. The dashed line's architecture is the same as the TreeConnect architecture, but has no ReLU activation on the hidden layer.

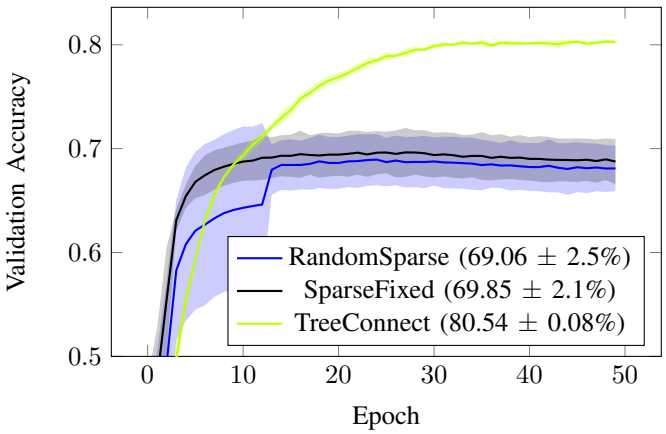

Fig. 5. Reuters news categorization average validation accuracy of 5 training runs for each architecture: a random sparse architecture with the architecture re-sampled for every run ("RandomSparse"), a random sparse architecture with the architecture fixed across training runs ("SparseFixed") and the TreeConnect architecture. The shaded area indicates the standard deviation across training runs while the numbers in brackets show the average and standard deviation of the maximal achieved validation accuracy.

truncate/pad all reviews to a length of 512 words. To each word we assign a trainable 256-dimensional embedding and average the 512 embeddings to get a 256-dimensional input to the network. This simple averaging across the entire review gives an independence to (relative) word location and yielded better performance than recurrent neural networks or CNN based architectures we tried. We use a sigmoid activation on the output neuron and a binary cross-entropy loss.

Specifically, we compare the following four architectures:

- **Shallow:** P(512) - FC1
  Parameters in the network: 257
- **Full:** P(512) - FC16 - FC1
  Parameters in the network: 8,225
- **TreeConnect:** P(512) - TC((16, 16 - 1) - 1)
  Parameters in the network: 289
- **Small:** P(512) - FC2 - FC1
  Parameters in the network: 517

Note that the "Small" architecture is the smallest possible fully connected 2-layer architecture that is not too similar to the "Shallow" architecture.

We train each architecture for 10 epochs and find that all architectures performed best, in terms of maximal achieved validation accuracy throughout the training, for the highest learning rate of 0.001. We show the corresponding validation accuracies in Figure 3. Even though the training curves differ, the maximal achieved validation accuracy is quite similar for all architectures. We also tried deeper architectures, but did not find them to improve validation accuracy for this problem.

B. Reuters Text Categories

The Reuters newswire dataset consists of 11,228 newswires from 46 topics, split into 8,982 training and 2,246 validation samples. We use the same preprocessing as for the reviews before, but use a larger word embedding size of 1,024 dimensions and train for 50 epochs. We use a softmax activation on the output layer and train the network with a cross-entropy loss between the network output and the one-hot encoded topic labels. We show the results of following four architectures:

- **Shallow:** P(512) - FC46
  Parameters in the network: 47,150
- **Full:** P(512) - FC736 - FC46
  Parameters in the network: 788,302
- **TreeConnect:** P(512) - TC((32, 32 - 23) - 46)
  Parameters in the network: 25,806
- **Small:** P(512) - FC24 - FC46
  Parameters in the network: 25,750

Also in this setup we find that all architectures prefer the highest learning rate (0.001). As with the IMDB dataset, we did not find deeper architectures to improve performance. In contrast, the results presented in Figure 4 suggest that the problem is close to linearly separable for large enough trainable embeddings, since linear models ("Shallow" and "Linear TC") outperform all others. The results also suggest that a TreeConnect network is, in terms of validation performance, closer to a single fully connected layer than to a two layer network. This similarity becomes even more pronounced when the non-linearity in the hidden layer of the TreeConnect network is removed (shown as dashed line in Figure 4).

To test our claim that TreeConnect is more resilient to initialization than other sparse architectures, we take the "Shallow" architecture and randomly mask 45.3125% of the weights to 0 such that the trainable parameters are in the same range as the TreeConnect architecture's. We train this random sparse architecture 5 times with 5 different weight mask and 5 times with a single weight mask kept across training runs, always with a learning rate of 0.001. We compare the validation accuracy achieved against 5 training runs of the TreeConnect architecture in Figure 5. Not only does the TreeConnect architecture outperform the random sparse architectures, it also shows a smaller standard deviation across

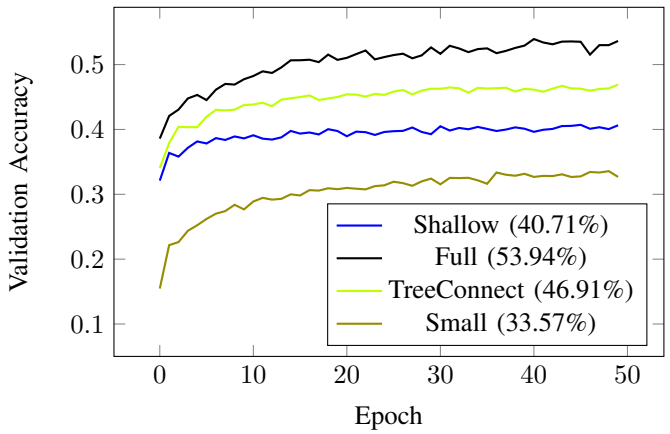

Fig. 6. Validation accuracy on CIFAR-10 when directly connecting input to output through shallow architectures. The number in brackets is the maximal achieved validation accuracy throughout the training.

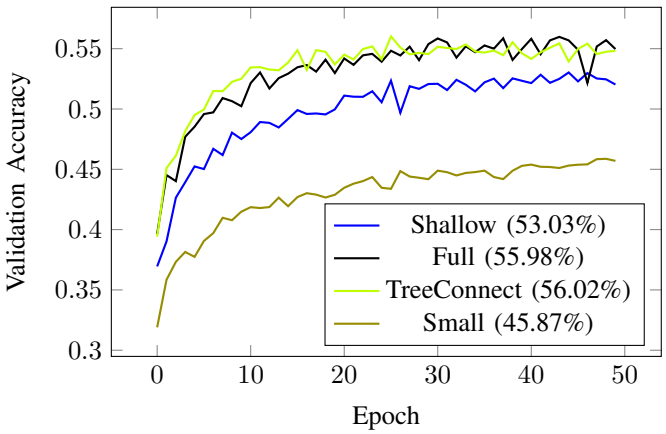

Fig. 7. Validation accuracy on CIFAR-10 when adding a fully connected layer to all architectures before the output. The number in brackets is the maximal achieved validation accuracy throughout the training.

training runs and thereby more resilience to initialization. Note that even with the weight mask (and thereby the architecture) fixed, the random sparse architecture still experiences a large variation across training runs due to the random initialization of trainable parameters.

C. CIFAR-10 Image Classification

Next, we turn our attention to a more difficult task. The CIFAR-10 dataset [36] consists of 60,000 32x32 RGB images from 10 different classes, split into 50,000 training and 10,000 validation samples. We chose this image classification dataset since it allows for fast experimentation while the classification task is complex enough to easily verify performance differences. Again, we use a standard softmax-cross-entropy loss and train all networks described hereafter for 50 epochs.

1) Application to Pixel Space: First we focus on directly applying the following four simple architectures between the 3,072 dimensional input and the 10 dimensional output:

- **Shallow:** FC10
  Parameters in the network: 30,730
- **Full:** FC320 - FC10
  Parameters in the network: 986,570
- **TreeConnect:** TC((64, 48 - 5) - 10)
  Parameters in the network: 16,330
- **Small:** FC5 - FC10
  Parameters in the network: 15,425

For this setup, all but the TreeConnect architecture preferred the medium learning rate of 0.0001, while the TreeConnect network performed best for the highest learning rate (0.001). As can be seen from the validation accuracies in Figure 6, this problem requires a non-linear function approximator for reasonable performance. Also, a bottleneck in the hidden layer (as in the "Small" architecture) hinders training severely. However, the multiple paths from input to output in the "Full" architecture allow for redundancy in the learned hidden representation, which helps the training. We therefore turn our attention to deeper architectures. More specifically:

- **Shallow:** FC256 - FC10
  Parameters in the network: 789,258
- **Full:** FC1024 - FC256 - FC10
  Parameters in the network: 3,411,722
- **TreeConnect:** TC((64, 48 - 16) - 256) - FC10
  Parameters in the network: 69,386
- **Small:** FC22 - FC54 - FC10
  Parameters in the network: 69,398

Again, the "Shallow", "Full" and "Small" architecture prefer the medium learning rate (0.0001), while the "TreeConnect" architecture requires the highest learning rate (0.001). Looking at the validation accuracies in Figure 7 we can see that the TreeConnect network followed by a fully connected layer performs similar to a 3-layer fully connected network of equal hidden layer dimensions, while requiring 49x less parameters.

To see whether the same performance can be achieved by a one or two layer randomly sparse architecture, we take the "Shallow" architecture and randomly mask 91.536% of the weights in the first layer to 0. Further we take the "Full" architecture and randomly mask 98.4375% of the weights in the first layer and 93.75% of the weights in the second layer such that the trainable parameters are in the same range as the TreeConnect architecture's. An initial run showed that both new architectures perform best for the highest learning rate of 0.001. We run each architecture for 5 training runs, sampling the weight masks for each run individually. As can be seen in Figure 8, the variance across training runs is not as significant anymore. This is because a path from most inputs to most outputs is given with high probability in this setting. However, the balanced TreeConnect architecture still outperforms the two other sparse architectures.

2) Fully Connected Feature Extractor: Since the TreeConnect architecture might introduce a small architectural prior by grouping close pixels into channels, we next investigate architectures with a fully connected layer to the first hidden representation. This first layer decouples the local relations in the input, such that the first hidden layer is unstructured.

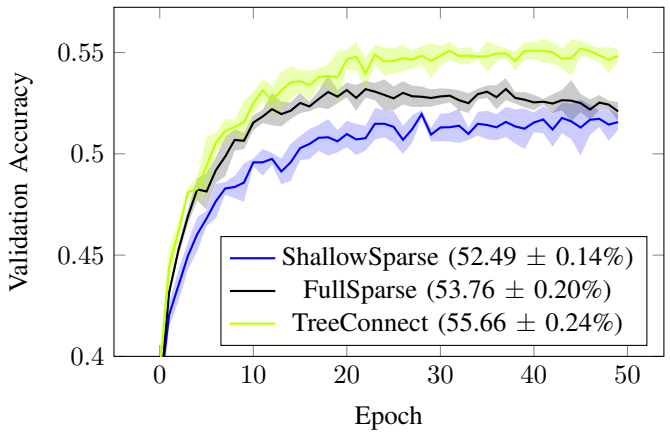

Fig. 8. Average validation accuracy on CIFAR-10 of 5 training runs for each of the following architectures: a 1-layer random sparse architecture with the architecture re-sampled for every run ("ShallowSparse"), a 2-layer random sparse architecture with the architecture re-sampled for every run ("FullSparse") and the TreeConnect architecture as in Figure 7. The shaded area indicates the standard deviation across training runs while the numbers in brackets show the average and standard deviation of the maximal achieved validation accuracy.

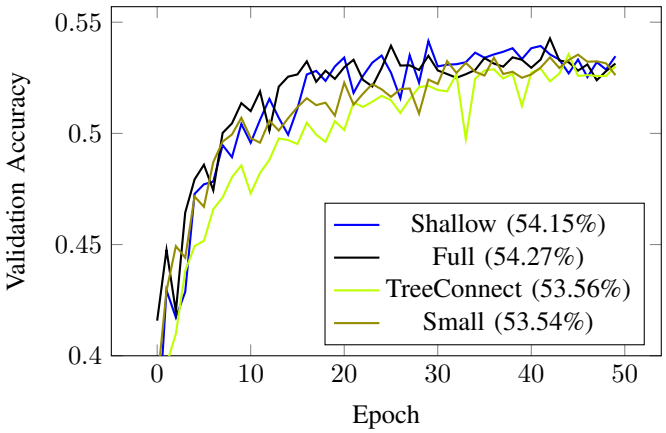

Fig. 9. Validation accuracy over the course of training of the four small architectures that start and end with a fully connected layer. The number in brackets is the maximal achieved validation accuracy throughout the training.

Similar to before, we also end all architectures with a fully connected layer to the 10 output classes. This setup allows us to test the proposed architecture between two unstructured hidden representations. Further, since we do not want the first layer to dominate the total number of parameters in the network, we down sample the input by average pooling (P) the images with a 4x4 kernel. The first architectures we try with this setup are the following small architectures:

- **Shallow:**
  P(4,4) - FC256 - FC64 - FC10
  Parameters in the network: 66,506
- **Full:**
  P(4,4) - FC256 - FC128 - FC64 - FC10
  Parameters in the network: 91,210
- **TreeConnect:**
  P(4,4) - FC256 - TC((16, 16 - 8) - 64) - FC10
  Parameters in the network: 53,322
- **Small:**
  P(4,4) - FC176 - FC88 - FC44 - FC10
  Parameters in the network: 53,910

We find that all architectures perform best, in terms of maximal achieved validation accuracy, for the highest learning rate (0.001). We report the validation accuracy over the course of training in Figure 9. As can be seen, all architectures perform very similar, which might be due to the fact that even with average pooling most network parameters (49,408) are concentrated in the first fully connected layer.

To overcome the concentration of network parameters to the first layer, we increase the hidden layer sizes. More precisely, we compare the following 4 larger architectures:

- **Shallow:**
  P(4,4) - FC16384 - FC1024 - FC10
  Parameters in the network: 19,950,602
- **Full:**
  P(4,4) - FC16384 - FC4096 - FC1024 - FC10
  Parameters in the network: 74,480,650
- **TreeConnect:**
  P(4,4) - FC16384 - TC((128, 128 - 32) - 1024) - FC10
  Parameters in the network: 3,832,842
- **Small:**
  P(4,4) - FC3456 - FC864 - FC216 - FC10
  Parameters in the network: 3,842,866

Here we find that all but the "Full" architecture require the medium learning rate (0.0001) for the best performance, while the "Full" architecture requires the lowest learning rate (0.00001). The corresponding validation performances can be seen in Figure 10. As can be seen, the TreeConnect architecture performs slightly worse than all other architectures. We believe this is due to the fact that most parameters (3,162,112 out of 3,832,842) in the TreeConnect architecture are still concentrated in the first fully connected layer. The sparse connections from this large layer to the output might not yield diverse enough gradients to usefully diverge all parameters from the random initialization fast enough. We therefore turn our attention to a sparser, more powerful feature extractor.

3) Convolutional Feature Extractor: Since CNNs [3] are good feature extractors that require only few parameters, we deploy a simple CNN as first part of our network, such that we can test our architecture on the extracted features. More precisely, we pass the full images through 6 convolutional layers of the following structure: 64 3x3 filters, stride 1 - 64 3x3 filters, stride 1 - 128 4x4 filters, stride 2 - 128 3x3 filters, stride 1 - 128 3x3 filters, stride 1 - 256 4x4 filters, stride 2. This results in a total number of 842,048 parameters in the CNN and a output (hidden) layer dimension of 16,384. Note that compared to fully connected layers, CNNs keep some of the spacial structure of the input. We use the shorthand CNN16384 to describe this 6 layer feature extractor and compare the following architectures:

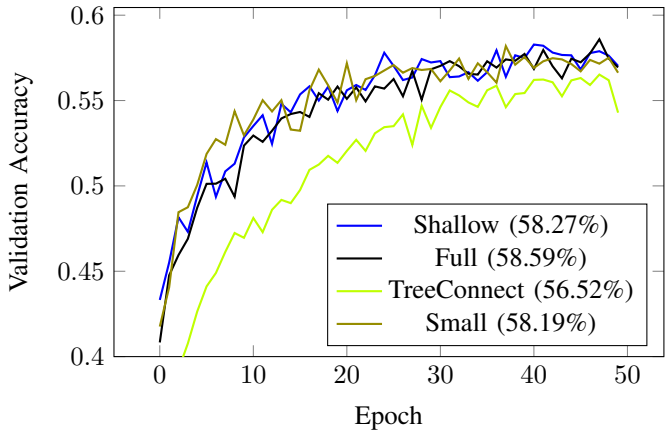

Fig. 10. Validation accuracy over the course of training of the four larger architectures that start and end with a fully connected layer. The number in brackets is the maximal achieved validation accuracy throughout the training. The worse performance of TreeConnect is due to the unbalanced weight distribution as explained in the main text.

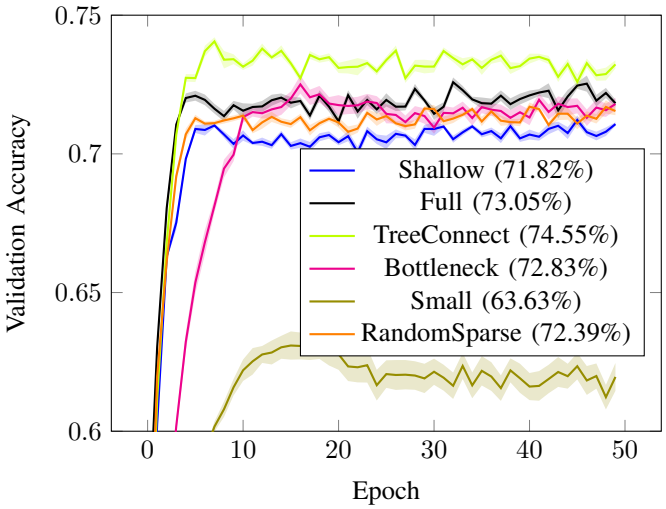

Fig. 11. Validation accuracy over the course of training of the six architecture with a convolutional feature extractor. Each line indicates the average while the shaded area indicates 0.2 times the standard deviation across the 5 training runs. The number in brackets is the maximal achieved validation accuracy throughout the training, averaged across the 5 training runs.

- **Shallow:**
  CNN16384 - FC256 - FC10
  Parameters in the network: 5,186,762
- **Full:**
  CNN16384 - FC2048 - FC256 - FC10
  Parameters in the network: 35,073,226
- **TreeConnect:**
  CNN16384 - TC((128, 128 - 16) - 256) - FC10
  Parameters in the network: 1,289,418

For the "Small" architecture we implement two versions: One that uses the same CNN but due to the parameter limitation suffers from a bottle neck, and one that uses less filters in the CNN, more precisely 8, 8, 16, 16, 16 and 32 filters. This results in the architectures:

- **Bottleneck:**
  CNN16384 - FC18 - FC167 - FC10
  Parameters in the network: 1,289,415
- **Small:**
  CNN2048 - FC552 - FC256 - FC10
  Parameters in the network: 1,290,922

Further, we test a RandomSparse architecture which is equivalent to the "Full" architecture, but in the first and second fully connected layer 99.22% and 93.75% of the weights are masked to 0 such that the architecture has a similar number of trainable weights as in the TreeConnect network. The weight masks are re-sampled for each training run. A first run showed that all architectures except the TreeConnect and RandomSparse network perform best for the medium learning rate of 0.0001, while the TreeConnect and RandomSparse architecture perform best for the highest learning rate (0.001). We train each architecture 5 times with the corresponding learning rate and report the validation accuracy over the course of training in Figure 11. Note that the TreeConnect network outperforms all other architectures. We believe that this is due to the following reasons:

1) The small amount of connections within the network regularizes the function approximation in an Occam's Razor [39] fashion while maintaining a high dimensional hidden representation and a full connectivity.
2) In this setup, TreeConnect introduces a slight architectural prior in that it first reduces the dimensionality of the feature vectors extracted at each spacial location before mixing the feature vectors of different spacial locations.

We also find that removing filters in the convolutional layers leads to worse performance than introducing a bottleneck.

V. Conclusion

We presented TreeConnect, a simple sparse architecture that guarantees a connection between all input and output neurons. Compared to a fully connected layer, the parameters and computations required scale as $\mathcal{O}(n^{1.5})$ as opposed to $\mathcal{O}(n^2)$, n being the number of input neurons. For large hidden representations it therefore easily outperforms its fully connected counterpart in terms of storage and computation requirements. In our experiments we showed that it also performs similar if not superior to its fully connected counterpart in terms of generalization capability. We showed that in a shallow network the full connectivity between input and output makes it more resilient to initialization than randomly sparsifying the network a priori. If compared after a convolutional feature extractor on the CIFAR-10 [36] image classification task, TreeConnect can even outperform a 2-layer fully connected network. We further found in our experiments that sparse and small architectures require a larger learning rate than networks with more parameters, therefore a hyper-parameter search should be done when comparing different architectures. We leave it to future work to integrate the idea into recurrent and convolutional neural network structures.

VI. Acknowledgements

We would like to thank Dario Fuoli for the first experiments conducted in the direction of developing a sparser architecture as well as Gino Brunner for the helpful discussions on the topic and his ideas on implementing random sparse architectures.

References

[1] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[3] Y. LeCun, "Generalization and network design strategies," in *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Eds. Zurich, Switzerland: Elsevier, 1989, an extended version was published as a technical report of the University of Toronto.

[4] Y. LeCun and Y. Bengio, "The handbook of brain theory and neural networks," M. A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1998, ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. [Online]. Available: http://dl.acm.org/citation.cfm?id=303568.303704

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[6] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013. [Online]. Available: http://arxiv.org/abs/1312.4400

[7] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," *CoRR*, vol. abs/1412.6806, 2014. [Online]. Available: http://arxiv.org/abs/1412.6806

[8] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[11] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," *CoRR*, vol. abs/1603.08029, 2016. [Online]. Available: http://arxiv.org/abs/1603.08029

[12] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, vol. 4, 2017, p. 12.

[13] M. Mavrovouniotis and S. Chang, "Hierarchical neural networks," *Computers & chemical engineering*, vol. 16, no. 4, pp. 347–369, 1992.

[14] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 44–51.

[15] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, 2017, pp. 3859–3869.

[16] G. Hinton, N. Frosst, and S. Sabour, "Matrix capsules with em routing," 2018.

[17] A. Bourely, J. P. Boueri, and K. Choromonski, "Sparse neural networks topologies," *CoRR*, vol. abs/1706.05683, 2017. [Online]. Available: http://arxiv.org/abs/1706.05683

[18] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," *CoRR*, vol. abs/1306.0543, 2013. [Online]. Available: http://arxiv.org/abs/1306.0543

[19] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE transactions on neural networks*, vol. 1, no. 2, pp. 239–242, 1990.

[20] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE transactions on Neural networks*, vol. 8, no. 3, pp. 519–531, 1997.

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[22] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1135–1143. [Online]. Available: http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf

[23] C. Louizos, M. Welling, and D. P. Kingma, "Learning Sparse Neural Networks through L_0 Regularization," *ArXiv e-prints*, Dec. 2017.

[24] M. D. Collins and P. Kohli, "Memory bounded deep convolutional networks," *CoRR*, vol. abs/1412.1442, 2014. [Online]. Available: http://arxiv.org/abs/1412.1442

[25] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudk, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html

[26] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[27] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1058–1066. [Online]. Available: http://proceedings.mlr.press/v28/wan13.html

[28] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6655–6659.

[29] C. Tai, T. Xiao, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization," *CoRR*, vol. abs/1511.06067, 2015. [Online]. Available: http://arxiv.org/abs/1511.06067

[30] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, "Compressing deep convolutional networks using vector quantization," *CoRR*, vol. abs/1412.6115, 2014. [Online]. Available: http://arxiv.org/abs/1412.6115

[31] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition." in *Interspeech*, 2013, pp. 2365–2369.

[32] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," *CoRR*, vol. abs/1509.06569, 2015. [Online]. Available: http://arxiv.org/abs/1509.06569

[33] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 3088–3096.

[34] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[35] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[36] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html

[37] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[39] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Inf. Process. Lett.*, vol. 24, no. 6, pp. 377–380, Apr. 1987. [Online]. Available: http://dx.doi.org/10.1016/0020-0190(87)90114-1