Building Sparse Deep Feedforward Networks using Tree Receptive Fields

Xiaopeng Li[∗]**, Zhourong Chen**[∗]**, Nevin L. Zhang**
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{xlibo, zchenbb, lzhang}@cse.ust.hk

Abstract

Sparse connectivity is an important factor behind the success of convolutional neural networks and recurrent neural networks. In this paper, we consider the problem of learning sparse connectivity for feedforward neural networks (FNNs). The key idea is that a unit should be connected to a small number of units at the next level below that are strongly correlated. We use Chow-Liu's algorithm to learn a tree-structured probabilistic model for the units at the current level, use the tree to identify subsets of units that are strongly correlated, and introduce a new unit with receptive field over the subsets. The procedure is repeated on the new units to build multiple layers of hidden units. The resulting model is called a TRF-net. Empirical results show that, when compared to dense FNNs, TRF-net achieves better or comparable classification performance with much fewer parameters and sparser structures. They are also more interpretable.

1 Introduction

It is common knowledge that architecture matters in deep learning. In general, convolutional neural networks (CNNs) are considered suitable for spatial data and recurrent neural networks (RNNs) are considered suitable for sequential data. However, it remains an art to design an appropriate architecture for a particular application. Typically, researchers need to evaluate a long list of candidate architectures before finding a satisfactory one, a process that is sometimes called graduate student descent. For this reason, there is growing interest in architecture learning [Baker *et al.*, 2017; Zoph and Le, 2017].

In addition to saving manual labor, architecture learning is interesting for two other reasons. First, it enables us to detect salient patterns in data and represent them using network structures. This can lead to better performing and more interpretable models. Second, architecture learning naturally would involve connectivity learning, which leads to sparse models. Model sparsity is helpful in avoiding overfitting and is desirable for applications on hand-held devices.
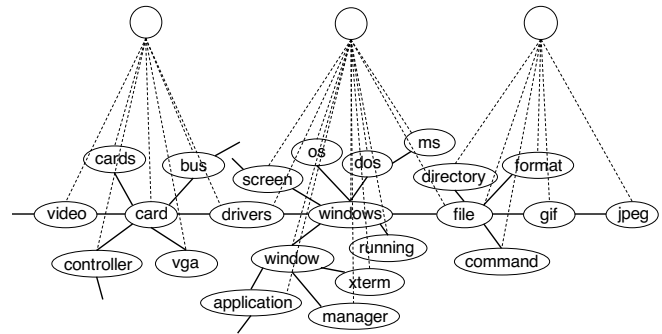

Figure 1: Example of local structure in 20newsgroup dataset. Three local neighborhoods are shown: video card, os, file format.

In this paper, we focus on feedforward neural networks (FNNs). While CNNs are designed for spatial data and RNNs for sequential data, FNNs are used for vector-form data that are neither spatial nor sequential. By definition, FNNs consists of fully-connected layers, where a neuron is connected to all neurons at the next level below. We call such models dense FNNs. Our objective is to learn sparse FNNs where a neuron is connected to only a small number of neurons at the next level below. One way to achieve this is to first learn a fully-connected model and then prune weak links [Han *et al.*, 2015; Guo *et al.*, 2016; Srinivas and Babu, 2015; Li *et al.*, 2017]. This approach does not solve the architecture learning problem because one needs to determine the structure for the fully-connected model to begin with. We aim at learning sparse FNN structures from scratch.

In CNNs, sparse layers are constructed by moving a small sliding window cross the spatial extent of the level below. A neuron is introduced for each position of the sliding window and it is connected to all units within the window at that position. The unspoken word for doing so is that the units in a small spatial region tend to be strongly correlated in their activations. While there is no spatial information in vector-form data, it is still possible to identify subsets of variables that are strongly correlated, and it is hence still possible to learn sparse architectures. An example is shown in Fig. 1, where the observed variables are words from the *20newsgroup* dataset. And it can be seen that three subsets of highly correlated words exist among the observed variables. The sta-

tistical structure, in place of spatial or sequential structure, is another promising way to achieve sparse connectivity.

We propose the following method for learn sparse FNNs: (1) Run the Chow-Liu algorithm [Chow and Liu, 1968] to learn a tree-structured probabilistic graphical models over the observed features; (2) use the tree to identify subsets of correlated features; introduce a new neuron for each subset; convert the newly introduced neurons into observed variables; and repeat (1) and (2) to build multiple hidden layers. To capture global patterns, we allow a small number of neurons to be connected to all units at the level below. We call deep models built by our method Tree-Receptive-Field networks (*TRF-nets*). Here are the contributions that we make in this papers:

- We propose a novel algorithm for learning sparse FNNs.

- We have conducted extensive experiments to compare TRF-nets with dense FNNs. The results show that the TRF-nets achieve better or comparable classification performance with much fewer parameters and sparser structures, and they are more interpretable.

- We have also empirically compared TRF-nets with sparse FNNs obtained by pruning dense FNNs and models obtained by regularization-based methods. Overall, the TRF-nets outperform other models and have higher interpretability.

2 Related Works

The primary goal in structure learning is to find a model with optimal or close to optimal generalization performance. Bruce-force search is not feasible because the search space is large and evaluating each model is costly as it necessitates training. Early works in the 1980's and 1990's have focused on what we call the *micro expansion* approach where one starts with a small network and gradually add new neurons to the network until a stopping criterion is met [Kwok and Yeung, 1997b; 1997a; Bello, 1992; Ash, 1989; Fahlman and Lebiere, 1990]. The word "micro" is used here because at each step only one (or a few) neurons are added. This makes learning a large model computationally difficult as reaching a large model would require many steps and model evaluation is needed at each step. In addition, those early methods typically do not produce layered structures that are commonly used today. Note that, in another line of work, micro expansion methods have been developed for sum-product networks [Hsu *et al.*, 2017].

Recent efforts have concentrated on what we call the *contraction approach* where one starts with a larger-than-necessary structure and reduce it to the desired size. Contraction can be done either by repeatedly pruning neurons and/or connections [LeCun *et al.*, 1990; Hassibi *et al.*, 1993; Han *et al.*, 2015; Guo *et al.*, 2016; Srinivas and Babu, 2015; Li *et al.*, 2017], or by using regularization to force some of the weights to zero [Collins and Kohli, 2014; Wen *et al.*, 2016]. From the perspective of structure learning, the contraction approach is not ideal because it requires a complex model as input. A key motivation for a user to consider structure learning is to avoid building models manually.

A third approach is to explore the model space stochastically. One way is to place a prior over the space of all possible structures and carry out MCMC sampling to obtain a collection of models with high posterior probabilities [Adams *et al.*, 2010]. Another way is to encode a model structure as a sequence of numbers, use a reinforcement meta model to explore the space of such sequences, learn a good meta policy from the sequences explored, and use the policy to generate model structures [Baker *et al.*, 2017; Zoph and Le, 2017]. An obvious drawback of such *stochastic exploration* method is that they are computationally very expensive.

What we propose in this paper is a *macro expansion* method where we start from scratch and repeatedly add layers of hidden units until a threshold is met. Our method is computationally cheaper than the micro expansion approach because it evaluates one two-layer model for each layer of the final model. It also produces layered models. Note that, while parameters are trained layer by layer in deep belief network and deep Boltzmann machines, we learn model structures layer by layer. The learned model is trained as a whole via backpropagation.

Another macro expansion method has recently been proposed by Liu et [Liu *et al.*, 2017]. It learns a new layer by solving a difficult multi-objective optimization problem. In contrast, we run Chow-Liu's algorithm to build a tree among the units on the current layer, which gives a "spatial structure" among the units, and we build the next layer by applying the concept of receptive field from CNNs on the tree.

3 Method

In this section, we present our method for learning sparse FNNs.

3.1 Learning the Tree Structures

We start by learning a tree-structured probabilistic graphical model for the observed variables. Fig. 2(a) illustrates one such model. Each node in the model represents an observed variable. We assume the observed variables are discrete. In the case of real-valued data, we would discretize the data for the purpose of this subsection and use the original data elsewhere.

There are multiple ways to parameterize the model. In this paper, we turn the tree into a rooted tree by arbitrarily picking one of the nodes as the root. The parameters of the model then includes the conditional probability $p(x_t|pa(x_t))$ of each variable x_t given its parent $pa(x_t)$. In other words, we deal with a tree-structured Bayesian network here. The model defines a joint distribution over all the observed variables:

$$p(\mathbf{x}|\mathcal{T}, \theta) = \prod_{t \in \mathcal{V}} p(x_t|pa(x_t)), \quad (1)$$

where $\mathcal{T}$ denote the tree structure and θ denotes the model parameters.

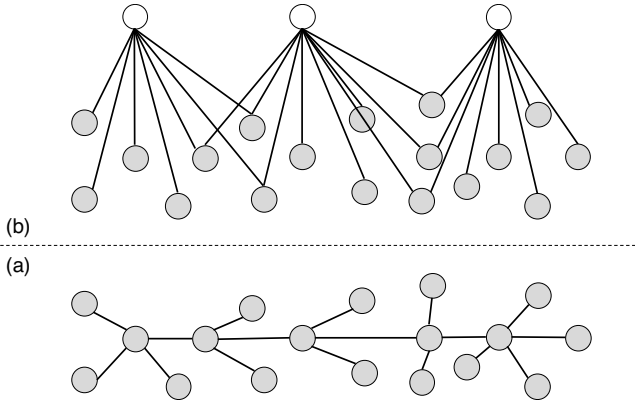Given a dataset $\mathcal{D}$, the log-likelihood of the parameters θ

Figure 2: Model construction by learning the local structure of observation variables via Chow-liu Tree.

Figure 3: A two-layer TRF-net with TRF and global neurons

and the model structure $\mathcal{T}$ is

$$
\begin{aligned}
l(\theta, \mathcal{T}|\mathcal{D}) &= \log p(\mathcal{D}|\theta, \mathcal{T}) \\
&= \sum_t \sum_k N_{tk} \log p(x_t = k|\theta) \\
&+ \sum_{s,t} \sum_{j,k} N_{stjk} \log \frac{p(x_s = j, x_t = k|\theta)}{p(x_s = j|\theta)p(x_t = k|\theta)},
\end{aligned}
\tag{2}
$$

where N_{stjk} is the number of times node s is in state j and node t is in state k, and N_{tk} is the number of times node t is in state k. The maximized log-likelihood of the model structure $\mathcal{T}$ is

$$
\begin{aligned}
l^*(\mathcal{T}|\mathcal{D}) &= \max_\theta l(\theta, \mathcal{T}|\mathcal{D}) \\
&= \sum_{t \in \mathcal{V}} \sum_k \hat{p}(x_t = k) \log \hat{p}(x_t = k) \\
&+ \sum_{(s,t) \in \mathcal{E}(\mathcal{T})} \hat{I}(x_s, x_t|\mathcal{D}),
\end{aligned}
\tag{3}
$$

where $\hat{p}(x_t = k) = N_{tk}/N$ is the emperical distribution and $\hat{I}(x_s, x_t|\mathcal{D})$ is the emperical mutual information between x_s and x_t, which is given by the following formula:

$$
\hat{I}(x_s, x_t|\mathcal{D}) = \sum_j \sum_k \hat{p}(x_s = j, x_t = k) \log \frac{\hat{p}(x_s = j, x_t = k)}{\hat{p}(x_s = j)\hat{p}(x_t = k)}.
\tag{4}
$$

Thus the tree topology that maximize the likelihood can be found by computing the maximum weight spanning tree, where the edge weights are the emperical pairwise mutual information. This is also known as the Chow-Liu algorithm [Chow and Liu, 1968]. We use *ChowLiu(𝒟)* to denote a subroutine that discretizes data and builds a Chow-Liu tree.

3.2 Building Two-Layer Structures

After obtaining a tree structure using Chow-Liu's algorithm, we next build a two-layer structure such as the one shown in Fig. 2(b). The tree captures correlation strengths among variables. As a matter of fact, a variable is more strongly correlated with its neighbors than with other variables. We therefore use local neighborhoods in the tree as receptive fields,
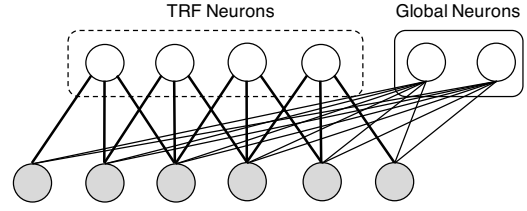
and introduce a latent node for each of them. A latent node is connected to all the nodes its receptive field. After latent node introduction, edges between observed nodes are removed.

A receptive field over a tree is defined by a field center x, which is one of the nodes, and a field radius r. It includes x and all nodes reachable from x in r hops. To cover a tree with receptive fields, we need another parameter, stride s, which is the number of hops between the centers of two neighboring receptive fields. Given r and s, we create the first receptive field by randomly picking a node as its center. At each step after that, we pick the center for the next receptive field by identifying a node that is s hops away one of the existing centers. The process terminates when such a node no long exists.

Latent nodes introduced by the method described so far are meant to model local interactions among variables. There could be long-range interactions that they do not capture. Such interactions are indirectly modelled by multiple layers latent nodes. In this paper, we also consider modeling long-range interactions directly. Specifically, we introduce a small number of global neurons and have them connected to all observed nodes. As such, a two-layer structure built by our method looks like what is shown in Fig. 3. We use *Build2LayerStructure* to denote the procedure described in this subsection.

3.3 Training Two-Layer TRF-nets

After obtaining a two-layer structure, we next turn it into a neural network model and train it as denoising autoencoder.

Let $\mathbf{x} = [x_i]$ and $\mathbf{h} = [h_j]$ be the vectors of visible and hidden units respectively. And let $\mathbf{A} = [a_{ij}]$ be the connectivity matrix where $a_{ij} = 1$ if unit x_i is connected to unit h_j and 0 otherwise. Let $\mathbf{W}$ and $\mathbf{b}$ be the weight and bias parameters of the neural network.

The model defines a conditional distribution $p(\mathbf{h}|\mathbf{x})$ of the hidden units given the observed units:

$$
p(\mathbf{h} = \mathbf{1}|\mathbf{x}) = s((\mathbf{A} \circ \mathbf{W})\mathbf{x} + \mathbf{b}),
\tag{5}
$$

where s is the probability function, e.g. sigmoid, and $\circ$ denotes element-wise product. The model also defines a conditional distribution $p(\mathbf{x}|\mathbf{h})$ of the observed units given the hidden units:

$$
\begin{aligned}
p(\mathbf{x}|\mathbf{h}) &= Ber(\sigma((\mathbf{A} \circ \mathbf{W})'\mathbf{h} + \mathbf{b})) \quad \text{if } \mathbf{x} \text{ is binary} \\
p(\mathbf{x}|\mathbf{h}) &= \mathcal{N}((\mathbf{A} \circ \mathbf{W})'\mathbf{h} + \mathbf{b}, \lambda\mathbf{I}) \quad \text{if } \mathbf{x} \text{ is real-value},
\end{aligned}
\tag{6}
$$

where $(\mathbf{A} \circ \mathbf{W})'$ is the transpose of $\mathbf{A} \circ \mathbf{W}$.

The model weights are determined using gradient descent and the denoising criteria [Vincent *et al.*, 2008] is used as the

---

Algorithm 1: TRF-NET($\mathcal{D}, r, s, d$)

Input: $\mathcal{D}$—a collection of data, k—kernel size,
$\quad\quad\quad$ s—stride, d—depth
Output: m—TRF-net

1   $\mathcal{D}_1 \leftarrow \mathcal{D}, m \leftarrow null$
2   **repeat**
3      $\mathcal{T} \leftarrow$ CHOWLIU($\mathcal{D}_1$)
4      $\mathcal{S} \leftarrow$ BUILD2LAYERSTRUCTURE($\mathcal{T}, r, s$)
5      $m_1 \leftarrow$ TRAINDENOISING($\mathcal{S}, \mathcal{D}_1$)
6      $\mathcal{D}_1 \leftarrow$ PROJECDATA($m_1, \mathcal{D}_1$)
      /* End current layer   */
7      $m \leftarrow$ STACKMODELS(m, m_1)
8   **until** *the number of layers in m reaches d*
9   **return** m

---

objective function. That is, we first introduce a corruption process $C(\tilde{\mathbf{x}}|\mathbf{x})$ which represents a conditional distribution over corrupted samples $\tilde{\mathbf{x}}$, given a data sample $\mathbf{x}$. The neural network takes a corrupted sample $\tilde{\mathbf{x}}$, maps it to a hidden representation $\mathbf{h} = f(\tilde{\mathbf{x}}) = s((\mathbf{A} \circ \mathbf{W})\tilde{\mathbf{x}} + b)$, and then tries to reconstruct the original $\mathbf{x}$, i.e. $\mathbf{x} \rightarrow \tilde{\mathbf{x}} \rightarrow \mathbf{h} \rightarrow \mathbf{x}$. The objective of the denoising training process is

$$\mathcal{L}_{\text{denoising}} = -\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\tilde{\mathbf{x}} \sim C(\tilde{\mathbf{x}}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{h} = f(\tilde{\mathbf{x}})). \quad (7)$$

With Gaussian distribution, the loss corresponds to reconstruction loss with squared error. With Bernoulli distribution, the loss corresponds to cross-entropy loss. We use *TrainDenoising* to denote the subroutine for training the two-layer network.

3.4 Learning a Deep Structure

After training the two-layer model, we turn the hidden units into observed variables, and repeat the process to learn multiple layers of hidden units.More specifically, with a learned two-layer model, the observed variables can be mapped to hidden representations through inference: $p(\mathbf{h}|\mathbf{x})$. Another layer of hidden units can be built on top of the hidden representations. When learning the Chow-Liu tree structure, the hidden representations can be discretized to binary according to whether it activates or does not activate. When learning the weights, the probability values are taken directly as the hidden representations. The resulting layer is then stacked on top of the previous layers. And the procedure is performed recursively.

The overall algorithm of learning TRF-net is shown in Algorithm 1. The resulting sparse FNN is then trained using backpropagation for upstream tasks.

4 Experiments

We have carried out experiments to determine whether our method can produce high quality model structures. The TRF-nets produced by our method are compared with FNNs obtained by manual grid search, and sparse FNNs obtained by weight pruning and weight regularization [Han *et al.*, 2015; Collins and Kohli, 2014]. The micro expansion and stochastic exploration methods are not included in the comparisons

because they do not produce layered FNNs and they are extremely expensive computationally. The macro expansion method by [Liu et al 2017] would be a good baseline. Unfortunately, we were able to obtain its implementation.

4.1 Datasets

Six publically available datasets were use in the experiments. One of them is about chemical compounds, and the other five are text data.

- **Tox21 challenge dataset.**[1] There are about 12,000 environmental chemical compounds in the dataset. The tasks are to predict 12 different toxic effects. We treat them as 12 binary classification tasks. We filter out sparse features which are present in fewer than 5% of the compounds, and rescale the remaining 1,644 features to zero mean and unit variance. We report the average AUC as the results for this dataset.

- **Text classification datasets.**[2] The five text datasets from [Zhang *et al.*, 2015]: *AG's News*, *DBPedia*, *Yelp Review*, *Yahoo!Answer* and *Sogou News*. The number of classes ranges from 2 to 14 and the dataset sizes are around 130,000~1,400,000. Stop words were removed during preprocessing. The top 10,000 most frequent words are selected as the vocabulary, and each document is represented as a bag-of-words.

4.2 Experiment Setup

We compare TRF-nets with standard fully-connected feedforward neural networks (FNNs), sparse neural networks obtained by weight pruning (Pruned FNNs) [Han *et al.*, 2015], and FNNs with L1 weight regularization (Reg FNNs) [Collins and Kohli, 2014].

When running the TRF-net algorithm on the Tox21 dataset, we used receptive fields with radius $r = 5$. Two different values, 1 and 2, were used for the stride s. When $s = 1$, the number of units on the next layer is the same as the number of units on the current layer. When $s = 2$, the number of units on the next layer is reduced by half. For text data, there are 10,000 input units. So, we set $r = 6$ and $s = 5$ for the first layer, $r = 5$ and $s = 4$ for the second layer to quickly reduce the number of units. The number of global neurons introduced at each layer is 10% of the number of TRF neurons on the same layer.

In the case of FNNs, the number of layers and the number of hidden units were determined by manual grid search. We followed [Klambauer *et al.*, 2017] and chose the number of neurons for each layer from {512, 1024, 2048}, and tested both rectangular and conic structures. We considered up to 4 hidden layers. For Pruned FNNs, we took the best FNN as the initial model and performed weight pruning as described in [Han *et al.*, 2015]. The pruned model is then retrained to obtain the final model. For Reg FNNs, we added L1 norm term to the training loss with the regularization strength found through validation (around 1e-5).

In all cases, we used ReLUs [Nair and Hinton, 2010] as the non-linear activation functions, and Adam [Kingma and Ba,

---

[1] https://github.com/bioinf-jku/SNNs

[2] https://github.com/zhangxiangxiao/Crepe

	TRF-net		FNN	Pruned FNN		Reg FNN
Task	Accuracy	Parameter # / Sparsity	Accuracy	Accuracy	Sparsity	Accuracy
Tox21 Average	**0.8135± 0.0038**	0.22M / 10.33%	0.8010±0.0017	0.7998±0.0034	32%	0.8038±0.0015
AG's News	**91.80%±0.05%**	1.62M / 6.93%	91.61%±0.01%	91.49%±0.09%	6%	91.54%±0.02%
DBPedia	**98.02%±0.01%**	3.50M / 13.90%	97.99%±0.04%	97.95%±0.02%	25%	97.91%±0.03%
Yelp Review Full	**59.14%±0.06%**	3.23M / 13.85%	59.13%±0.14%	58.83%±0.01%	24%	59.00%±0.11%
Yahoo!Answer	71.40%±0.01%	2.27M / 10.61%	**71.84%±0.07%**	71.74%±0.05%	32%	71.15%±0.05%
Sogou News	**96.29%±0.07%**	2.27M / 10.10%	96.11%±0.06%	96.20%±0.06%	31%	95.56%±0.04%

Table 1: Comparison between TRF-nets, FNNs, Pruned FNNs and Reg FNNs on 6 classification datasets. The structures of FNNs are chosen by using validation data. Each experiment is run for three times.

	TRF-net	TRF only	
Task	Accuracy	Accuracy	Sparsity w.r.t FNNs
Tox21 Average	0.8135	**0.8196**	2.26%
AG's News	**91.80%**	91.00%	2.48%
DBPedia	**98.02%**	97.75%	9.94%
Yelp Review Full	**59.14%**	58.69%	9.49%
Yahoo!Answer	**71.40%**	70.43%	5.69%
Sogou News	**96.29%**	96.03%	5.42%

Table 2: Comparison between TRF-nets with global neurons and TRF-nets with TRF neurons only

2015] as the optimizer, and we applied dropout [Srivastava *et al.*, 2014] with rate 0.5. We ran all the experiments for three times and report the average results.

4.3 Results

Classification Performances

Table 1 shows the classification performances of TRF-nets, FNNs, Pruned FNNs and Reg FNNs on different datasets. It is clear that TRF-nets achieved better AUC scores than FNNs on the Tox21 dataset and better or comparable accuracy on 5 text classification datasets. They did so with significantly fewer parameters and sparser structures. In comparison with FNNs, the sparsity of TRF-nets (# of connections w.r.t that of fully-connected ones) is around 10% for all datasets. This confirms that our method is able to achieve high generalization performance with much fewer parameters and sparser structures. Compared with Pruned FNNs and Reg FNNs, TRF-nets also achieved better or comparable classification accuracy. Prunned FNNs have comparable amounts of model parameters as TRF-ntes. Reg FNNs are not sparse models. Their weights are pushed toward 0 by regularization, but few actually reached 0. With weights of absolute values less than 0.001 removed, their sparsity is between 20% and 30% in comparison with FNNs. Note that pruned FNNs and Reg FNNs were obtained from FNNs that took much manual search to construct. In contrast, TRF-nets were automatically and directly learned from data.

Contribution of the Tree Receptive Fields

To validate our assumption that the TRF neurons in TRF-nets capture most of the patterns in data, we performed another set of experiments with all the global neurons removed. Table 2

shows the results. As we can see, the classification performances degraded only slightly, while model sparsity is futher improved significantly. The results not only show the importance of the TRF neurons in TRF-nets, but also shows that our structure learning method is effective.

Influence of the Model Depth

The results reported in Tables 1 and 2 were obtained using TRF-nets with 3 hidden layers. The decision to use 3 hidden layers was influenced by the literature. Typically authors use only a small number of hidden layers when it comes to FNNs. Nonetheless, it would be interesting to see how model depth influences the performance TRF-nets. So, we performed a set of experiments with varying model depths. The results are shown in Table 3. It can be seen that our method continues to work when the network become deeper. For example, for the *AG's News* dataset, the best performance was achieved with a 5-layer TRF-net. In contrast, deep architecture might lead to severe overfitting for FNNs.

Computational Time

A comparison of the computational time for TRF-nets and Pruned FNNs is given in Table 4. As it can be seen, the structure learning phase of TRF-nets costs almost as much time as the finetuning phase. The overall computational time is comparable with Pruned FNNs without structure searching. Note that FNNs, Pruned FNNs and Reg FNNs need to determine the best-performing FNN structure by grid searching in the pre-defined structure space, as described in the experiment setup. Pruned FNNs further prune and finetune the best structure. Although for TRF-nets there are also hyperparameters, i.e. receptive field and stride, it is significantly easier to determine those controlling hyperparameters than to determine the actual structures. Therefore, in practice the overall computational time including grid searching over the structure space for FNNs, Pruned FNNs and Reg FNNs is significantly larger than that for TRF-nets. All neural networks are trained on a Tesla K20 GPU, while the structure learning is run on a CPU.

Interpretability

Next we compare TRF-nets with FNNs and Pruned FNNs in terms of the interpretability of their hidden unit. We characterize the "meaning" of a hidden unit using the top 10 words that have the strongest correlations with the unit. Following [Chen *et al.*, 2017], we measure the "interpretability" of a hidden unit by considering how similar pairs of words in the top-10 list are using a word2vec model [Mikolov *et al.*, 2013a;

Layers	Tox21 Average	AG's News	DBPedia	Yelp Review Full	Yahoo!Answer	Sogou News
1	**0.8139**±**0.0026**	91.65%±0.14%	**98.09%**±**0.01%**	59.00%±0.12%	71.43%±0.02%	96.24%±0.04%
2	0.7992±0.0020	91.82%±0.22%	98.02%±0.01%	58.89%±0.09%	71.42%±0.03%	96.18%±0.02%
3	0.8135±0.0038	91.80%±0.05%	98.04%±0.02%	**59.14%**±**0.06%**	71.40%±0.01%	**96.29%**±**0.07%**
4	0.8025±0.0091	91.88%±0.14%	98.04%±0.01%	59.09%±0.04%	71.39%±0.07%	96.17%±0.04%
5	0.8081±0.0063	**92.02%**±**0.06%**	98.04%±0.04%	59.08%±0.11%	**71.51%**±**0.04%**	96.20%±0.01%
6	0.7991±0.0048	92.00%±0.14%	98.02%±0.03%	59.01%±0.04%	71.48%±p.02%	96.17%±0.02%
7	0.8055±0.0101	91.88%±0.04%	97.99%±0.03%	58.98%±0.05%	71.38%±0.03%	96.21%±0.03%

Table 3: Precision of TRF-nets with different depths on classification datasets. For each task, best result is **bolded**.

	TRF-net		Pruned FNN
			w/o structure
Task	structure	finetune	searching
Tox21 Average	145	27	43
AG's News	3,388	1,670	3,956
DBPedia	4,082	2,829	6,562
Yelp Review Full	5,072	3,135	3,618
Yahoo!Answer	7,215	5,870	12,440
Sogou News	4,311	3,948	7,340

Table 4: Comparison of computational time (s) between TRF-nets and Pruned FNNs on 6 classification datasets.

Task	TRF-nets	FNNs	Pruned FNNs
Yelp Review Full	**0.1176**	0.1117	0.1002
DBPedia	**0.0577**	0.0497	0.0553
Yahoo!Answer	**0.1754**	0.1632	0.1553
AG's News	0.0527	**0.0595**	0.0561

Table 5: Interpretability scores of TRF-nets, FNNs and Pruned FNNs on different datasets

Task	Characterization Words
Yelp Review Full	tasteless unseasoned flavorless paprika panko crusts unagi crumb vindaloo tortas spicey wink drapes
DBPedia	album songwriting chet saxophone hurling backstroke badminton skier journalists hardcover editors
Yahoo!Answer	xp desktop adobe peut cpu pagans atheist mormons apostle tylenol diabetic acids kidneys
AG's News	mozilla mainframe microprocessors republicans prosecutor argument jfk noted furious harsh concessions

Table 6: Qualitative interpretability results of hidden units in TRF-nets. Each line corresponds to one hidden unit.

resulting TRF-nets have shown to achieve better or comparable classification performance in all kinds of tasks compared with standard FNNs, while containing significantly fewer parameters. In addition, TRF-nets have also shown to be more interpretable than FNNs, which is interesting because interpretability is an important issue in deep learning.

Acknowledgments

2013b] trained on part of the Google News datasets. The interpretability score of a model is defined as the average of interpretability scores of all top-layer units. The results are reported in Table 5. There are no results for the *Sogounews* dataset because its vocabulary are Chinese pingyin characters and most of them do not appear in the Google News word2vec model. For the fairness of comparison, all models have approximately the same number of top-layer hidden units. As it can be seen, TRFT-nets outperformed FNNs in most cases. In contrast, Pruned FNNs achieved lower scores than FNNs in most cases. This indicates that the retraining after weight pruning reduces the interpretability of hidden units. As concrete examples, Table 6 shows the top 3 words that were used to characterize some of the hidden units. Those word groups are clearly meaningful. For example, the first group for the *Yahoo!Answer* dataset are about computers, while the third group is about medicine.

5 Conclusions

Structure learning for deep neural network is a challenging and interesting research problem. We have proposed an unsupervised structure learning method, which first learns a tree-structured probabilistic graphical model and then constructs hidden neurons with local receptive field over the tree. The

References

[Adams *et al.*, 2010] Ryan Adams, Hanna Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 1–8, 2010.

[Ash, 1989] Timur Ash. Dynamic node creation in back-propagation networks. *Connection science*, 1(4):365–375, 1989.

[Baker *et al.*, 2017] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[Bello, 1992] Martin G Bello. Enhanced training algorithms, and integrated training/architecture selection for multi-

layer perceptron networks. *IEEE Transactions on Neural networks*, 3(6):864–875, 1992.

[Chen *et al.*, 2017] Zhourong Chen, Nevin L Zhang, Dit-Yan Yeung, and Peixian Chen. Sparse boltzmann machines with structure learning as applied to text analysis. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.

[Chow and Liu, 1968] C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[Collins and Kohli, 2014] Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[Fahlman and Lebiere, 1990] Scott E Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, pages 524–532, 1990.

[Guo *et al.*, 2016] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems*, 2016.

[Han *et al.*, 2015] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 2015.

[Hassibi *et al.*, 1993] Babak Hassibi, David G. Stork, and Stork Crc. Ricoh. Com. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, 1993.

[Hsu *et al.*, 2017] Wilson Hsu, Agastya Kalra, and Pascal Poupart. Online structure learning for sum-product networks with gaussian leaves. *arXiv preprint arXiv:1701.05265*, 2017.

[Kingma and Ba, 2015] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[Klambauer *et al.*, 2017] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, 2017.

[Kwok and Yeung, 1997a] Tin-Yan Kwok and Dit-Yan Yeung. Objective functions for training new hidden units in constructive neural networks. *IEEE Transactions on neural networks*, 8(5):1131–1148, 1997.

[Kwok and Yeung, 1997b] Tin-Yau Kwok and Dit-Yan Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Transactions on Neural Networks*, 8(3):630–645, 1997.

[LeCun *et al.*, 1990] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1990.

[Li *et al.*, 2017] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[Liu *et al.*, 2017] Jia Liu, Maoguo Gong, Qiguang Miao, Xiaogang Wang, and Hao Li. Structure learning for deep neural networks based on multiobjective optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[Mikolov *et al.*, 2013a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations Workshops*, 2013.

[Mikolov *et al.*, 2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning*, 2010.

[Srinivas and Babu, 2015] Suraj Srinivas and R. Venkatesh Babu. Data-free parameter pruning for deep neural networks. In *Proceedings of the British Machine Vision Conference*, 2015.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[Vincent *et al.*, 2008] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International conference on Machine learning*, pages 1096–1103. ACM, 2008.

[Wen *et al.*, 2016] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[Zhang *et al.*, 2015] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, 2015.

[Zoph and Le, 2017] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.