Convolutional Soft Decision Trees

Alper Ahmetoğlu[1(✉)], Ozan İrsoy[2], and Ethem Alpaydın[1]

[1] Department of Computer Engineering, Boğaziçi University, Bebek, 34730 İstanbul, Turkey
{alper.ahmetoglu,alpaydin}@boun.edu.tr
[2] Bloomberg LP, 731 Lexington Ave, New York, NY 10022, USA
oirsoy@bloomberg.net

Abstract. Soft decision trees, aka hierarchical mixture of experts, are composed of soft multivariate decision nodes and output-predicting leaves. Previously, they have been shown to work successfully in supervised classification and regression tasks, as well as in training unsupervised autoencoders. This work has two contributions: First, we show that dropout and dropconnect on input units, previously proposed for deep multi-layer neural networks, can also be used with soft decision trees for regularization. Second, we propose a convolutional extension of the soft decision tree with local feature detectors in successive layers that are trained together with the other parameters of the soft decision tree. Our experiments on four image data sets, MNIST, Fashion-MNIST, CIFAR-10 and Imagenet32, indicate improvements due to both contributions.

Keywords: Soft decision trees · Convolutional neural networks

1 Introduction

Decision trees are hierarchical models that are composed of decision nodes and leaves. Decision nodes select among children nodes using a gating function that splits the input space, and the leaves contain output predictions, i.e., output labels in classification or real values in regression. In a hard decision tree, the decision nodes choose one of the children; with a soft decision tree, originally proposed as hierarchical mixture of experts [1], all the children are chosen but with different probabilities. In a hard tree, we follow a single path from the root to one leaf; in a soft tree, we traverse all the paths and reach all the leaves and we take a convex combination of leaves weighted by the probabilities on each path. This leads to a smoother fit and better generalization [2]. The parameters of the decision nodes and the leaves can be trained together by minimizing empirical error using gradient-descent.

Convolutional neural networks are shown to be successful in many applications, especially in computer vision [3]. The convolutional units have local receptive fields and learn basic primitives, which in successive layers are combined to learn more abstract features. The success of such networks imply that such a representation leads to better generalization. Building on this idea, we

V. Kůrková et al. (Eds.): ICANN 2018, LNCS 11139, pp. 134–141, 2018.
https://doi.org/10.1007/978-3-030-01418-6_14

incorporate convolutional layers to soft decision trees, so that the tree works not in the original input space but in the space learned by these convolutional layers. We show that as in deep neural networks, the convolutional layers can be trained together with the soft decision tree, leading to higher accuracy on four image recognition data sets, MNIST, Fashion-MNIST, CIFAR-10 and Imagenet32.

This paper is organized as follows. In Sect. 2, we review the soft decision tree architecture. In Sect. 3 we show how convolutional layers can be combined with a soft decision tree. We discuss the effect of input dropout on soft decision trees in Sect. 4. Our experimental results are given in Sect. 5; we conclude and discuss future work in Sect. 6.

2 Soft Decision Trees

Assume we have a K-class classification problem with d-dimensional input $\boldsymbol{x}$. The response of a binary decision tree node m is defined recursively as follows:

$$\boldsymbol{y}_m(\boldsymbol{x}) = \begin{cases} \boldsymbol{\rho}_m & \text{if } m \text{ is a leaf} \\ \boldsymbol{y}_m^L(\boldsymbol{x}) g_m(\boldsymbol{x}) + \boldsymbol{y}_m^R(\boldsymbol{x})(1 - g_m(\boldsymbol{x})) & \text{otherwise} \end{cases} \tag{1}$$

where $\boldsymbol{\rho}_m$ is a K-dimensional vector of predictions, $\boldsymbol{y}_m^L(\boldsymbol{x})$ and $\boldsymbol{y}_m^R(\boldsymbol{x})$ are responses of the left and the right children of node m respectively. In a hard decision node, the gating function $g_m(\boldsymbol{x})$ returns 0 or 1, and we choose either the left or the right subtree. In a soft decision tree, $g_m(\boldsymbol{x}) \in [0, 1]$, implements a soft split:

$$g_m(\boldsymbol{x}) = \frac{1}{1 + \exp[-(\boldsymbol{w}_m^T \boldsymbol{x} + b_m)]} \tag{2}$$

If we consider a decision node with two leaves, this tree of depth one is equivalent to a *mixture of experts* with two experts [4]. If we replace both leaves with two such trees of depth one each, we get a tree of depth two, which is a *hierarchical mixture of experts* [1]. Because of the continuity due to the sigmoid gating function, the gating parameters, $(\boldsymbol{w}_m^T, b_m)$ over the whole tree and the leaf values $(\boldsymbol{\rho}_m)$ can be trained in a coupled manner using stochastic gradient-descent by back-propagating the empirical error from the root to leaves through chain rule. Each decision node effectively takes a convex combination of its left and right subtree children.

Note that $g_m(\boldsymbol{x})$ uses all d features of the input; that is we have a multivariate tree, as opposed to univariate trees that use one feature in each split. A multivariate node defines a split of arbitrary orientation whereas a univariate split is orthogonal to one of the axes. $\boldsymbol{\rho}_m$ stored at leaf m is a K-dimensional vector of values: For each class, we traverse the tree using the same gating values but the corresponding element of $\boldsymbol{\rho}_m$, we then softmax these to get posterior probabilities, and then minimize the cross-entropy during training. With constant leaves, we get a (smoothed) piecewise constant approximation. To get a (smoothed) piecewise linear approximation, we can have a linear model in each leaf, $\boldsymbol{\rho}_m = V_m \boldsymbol{x}$, where V_m is $K \times d$. In this setting, we learn V_m by adding one

more term, $\partial \boldsymbol{\rho}/\partial V_m$, to the chain rule when we back-propagate. Previously, we have shown that soft decision trees can be used successfully in regression and classification tasks [2], as well as for training unsupervised autoencoders [5].

3 Convolutional Soft Decision Trees

The performance of any learning algorithm directly depends on the quality of the input representation, and as such, any feature extraction step that returns better features helps accuracy. In applications where there is locality, having early layers with units that learn local convolutions lead to better generalization [3]. Indeed, all successful vision applications of deep learning use a number of convolution layers that start from the raw image and learn incrementally more abstract features in later layers. Once, we get to a representation that is abstract enough and is independent of location, dense fully-connected layers are used to learn to generate the correct output from that representation.

Using this same idea, one can incorporate convolutional layers to a soft decision tree too, where we learn the convolutional layers from scratch while also training the soft decision tree. This implies that all the gating models will include those convolutional layers and we back-propagate to update the parameters of these layers too. If our soft decision tree has linear leaves, those also have the convolution layers incorporated. In our implementation, we use weight sharing where all the gating models and the leaves share the same convolution layers. That is we back-propagate separately and then average them when we update.

4 Regularizing Soft Decision Trees

A full decision tree with depth P has $(2^P - 1) \cdot d$ parameters in the gating nodes and $2^P \cdot K$ parameters in constant leaves, or $2^P \cdot K \cdot d$ parameters if the leaves are linear. This makes a lot of parameters when P and/or d is high, and as with neural networks, L^2 or L^1 regularization can be used. In L^2 regularization, we add a penalty term $\alpha \|\boldsymbol{w}\|^2$ to the error function where $\boldsymbol{w}$ is the set of gating parameters of the model and α is a hyper-parameter to adjust the relaxation. This penalty term is $\alpha \|\boldsymbol{w}\|$ in L^1 regularization. Applying L^2 and L^1 regularization on soft decision trees are discussed in [6] and L^2 regularization is reported to work slightly better.

Another possibility is to use input dropout [7] where we set the elements of $\boldsymbol{w}_m$ to zero with some non-zero probability p, and scale $\boldsymbol{w}_m$ by $1/p$ for each gating function g_m. In doing this, there are two possibilities: We can do the dropout once and use the same non-dropped features in all gating models, or, we can do it independently in each, which corresponds to dropconnect [8]. Later in our experiments, we refer to the first one as input dropout and the second one as input dropconnect.

5 Experiments

5.1 Data Sets and Training Details

We have experimented on four datasets: MNIST, Fashion-MNIST, CIFAR-10 and Imagenet32. MNIST dataset contains handwritten digits by different writers where a sample is a 28×28 gray-scale image with pixel intensity in the range [0, 255]. There are 60,000 training and 10,000 test samples. Fashion-MNIST is a recently constructed dataset of fashion products with image sizes and the number of classes equal to those of MNIST. CIFAR-10 contains 60,000 32×32 colored images (RGB intensities) belonging to ten different classes. Downsampled Imagenet is developed as a more difficult replacement for CIFAR-10. It contains 1,281,167 32×32 colored images belonging to 1,000 different classes. There are 50,000 validation images which we used as the test set. In training, we made the training/validation split as 55K/5K for MNIST and Fashion-MNIST, 45K/5K for CIFAR-10. Imagenet is a very large data set on which we do a single run with recommended hyperparameters (as is frequently done) and thus did not need a validation set.

On all datasets, we divide the pixel intensity values by 255. We did not apply data augmentation for MNIST and Fashion-MNIST. For CIFAR-10 and Imagenet32, we subtracted the mean of the training set for each sample. We made random horizontal flips and shifted the image horizontal and/or vertical up to four pixels randomly. We use stochastic gradient-descent with a momentum factor of 0.9 in all our experiments. For MNIST and Fashion-MNIST, the learning rate is set to 0.01. For CIFAR-10, the initial learning rate is set to 0.1 and is divided by 10 at 32k and 48k iterations as in [9]. For Imagenet32, the initial learning rate is set to 0.01 and is divided by 5 at every 10 epochs. The batch size is 256 for MNIST and Fashion-MNIST, 128 for CIFAR-10 and Imagenet32. A weight decay of 0.0001 is used in CIFAR-10 training.

5.2 Regularization Experiments

We employ input dropout, dropconnect, and L^2 regularization with different coefficients for different models. On three data sets, we used trees with depths of one to five, and for each, we do five runs and report the average and standard deviation of test errors in Tables 1, 2 and 3—we could not do these experiments on Imagenet32 which is very large. Here, we only regularize the parameters of the gating functions. We see that dropout and dropconnect give better results than L^2 regularization. While the results of dropout and dropconnect are not significantly different, dropconnect seems to work slightly better with increasing tree depth. This makes sense because we drop weights of gatings independently of each other and we average over subtrees that use slightly different feature subsets (as in a random forest). Based on these results, we adopted input dropconnect with a keep probability of 0.5 as a regularizer in our later experiments.

Table 1. Effect of dropout for different keep probabilities.

Tree depth	0.25	0.4	0.5	0.6	0.75
MNIST					
1	5.04 ± 0.07	$\mathbf{4.89 \pm 0.15}$	4.93 ± 0.14	4.93 ± 0.06	4.93 ± 0.16
2	3.58 ± 0.09	3.58 ± 0.16	3.42 ± 0.09	3.48 ± 0.18	$\mathbf{3.41 \pm 0.18}$
3	3.15 ± 0.18	2.88 ± 0.17	$\mathbf{2.87 \pm 0.08}$	2.91 ± 0.16	2.94 ± 0.11
4	2.68 ± 0.07	2.67 ± 0.20	$\mathbf{2.48 \pm 0.15}$	2.64 ± 0.26	2.70 ± 0.06
5	2.55 ± 0.14	$\mathbf{2.44 \pm 0.26}$	2.47 ± 0.12	2.51 ± 0.25	2.57 ± 0.12
Fashion-MNIST					
1	13.78 ± 0.10	13.59 ± 0.13	13.58 ± 0.11	$\mathbf{13.42 \pm 0.09}$	13.48 ± 0.06
2	12.66 ± 0.07	12.55 ± 0.26	12.49 ± 0.12	12.56 ± 0.24	$\mathbf{12.47 \pm 0.23}$
3	12.21 ± 0.10	12.09 ± 0.06	$\mathbf{12.01 \pm 0.27}$	12.03 ± 0.32	12.10 ± 0.09
4	12.05 ± 0.27	11.87 ± 0.14	11.95 ± 0.16	$\mathbf{11.73 \pm 0.16}$	11.86 ± 0.23
5	11.96 ± 0.23	11.72 ± 0.21	11.65 ± 0.17	$\mathbf{11.49 \pm 0.10}$	11.55 ± 0.25
CIFAR-10					
1	59.10 ± 0.34	57.16 ± 0.57	57.47 ± 0.84	57.14 ± 0.60	$\mathbf{56.96 \pm 0.30}$
2	56.28 ± 0.79	54.96 ± 0.35	54.39 ± 0.79	54.20 ± 0.38	$\mathbf{53.51 \pm 0.57}$
3	53.69 ± 0.40	52.67 ± 0.75	51.78 ± 0.92	52.29 ± 0.37	$\mathbf{51.26 \pm 0.40}$
4	52.55 ± 0.47	51.05 ± 0.49	49.83 ± 0.48	50.98 ± 0.22	$\mathbf{49.76 \pm 0.47}$
5	51.02 ± 0.53	50.16 ± 0.35	$\mathbf{48.77 \pm 0.39}$	50.09 ± 0.87	48.84 ± 0.38

5.3 Convolutional Tree Experiments

On MNIST and Fashion-MNIST, the convolutional network structure we used consists of two blocks, each of which has two consecutive convolutional layers followed by a max-pooling layer. The number of filters are 8, 16 and 16, 32 for the two blocks. Filter sizes are 3×3 with a stride of 1 for all convolutional layers, and 2×2 with a stride of 2 for max pooling layers. After the second max pooling layer, the data is projected onto a $\boldsymbol{z}$-dimensional space by a fully-connected layer where z is the hyper-parameter we finetune; it is the dimensionality of the input fed to the tree for classification. For CIFAR-10 and Imagenet32, we use the wide residual network (WRN) architecture previously successfully used on these data sets [9,10]. The WRN architecture we use consists of 4 residual blocks each of which has 6 convolutional layers. There is an additional convolutional layer between residual blocks, which leads to a total of 28 convolutional layers. In WRN-28-1, the number of filters are 16, 32 and 64 for residual blocks, and in WRN-28-2 these numbers are doubled. We add another fully-connected layer to the WRN to map to a specific $\boldsymbol{z}$ dimension which is the input to the tree.

We compare three different models. SDT-k is a convolutional soft decision tree of depth k that takes the output of the convolutional network as its input. SDT-Lk, is the same as the first model except that the leaves contain linear

Table 2. Effect of dropconnnect for different keep probabilities.

Tree depth	0.25	0.4	0.5	0.6	0.75
MNIST					
1	4.93 ± 0.06	4.93 ± 0.17	4.92 ± 0.08	$\mathbf{4.83 \pm 0.02}$	4.84 ± 0.06
2	3.74 ± 0.10	3.43 ± 0.07	$\mathbf{3.42 \pm 0.17}$	3.45 ± 0.17	3.43 ± 0.13
3	2.87 ± 0.14	$\mathbf{2.67 \pm 0.15}$	2.74 ± 0.10	2.77 ± 0.11	2.95 ± 0.37
4	2.43 ± 0.12	$\mathbf{2.41 \pm 0.09}$	2.55 ± 0.15	2.60 ± 0.08	2.58 ± 0.09
5	2.33 ± 0.17	$\mathbf{2.24 \pm 0.11}$	2.31 ± 0.11	2.48 ± 0.12	2.44 ± 0.17
Fashion-MNIST					
1	13.74 ± 0.09	13.71 ± 0.09	13.61 ± 0.17	13.62 ± 0.13	$\mathbf{13.53 \pm 0.15}$
2	12.70 ± 0.05	12.71 ± 0.23	12.60 ± 0.17	12.55 ± 0.16	$\mathbf{12.46 \pm 0.28}$
3	12.08 ± 0.20	11.98 ± 0.21	12.06 ± 0.18	$\mathbf{11.84 \pm 0.20}$	11.94 ± 0.21
4	11.83 ± 0.18	11.65 ± 0.22	$\mathbf{11.53 \pm 0.20}$	11.82 ± 0.19	11.59 ± 0.32
5	11.59 ± 0.12	11.64 ± 0.25	$\mathbf{11.46 \pm 0.19}$	11.54 ± 0.17	11.54 ± 0.25
CIFAR-10					
1	57.33 ± 0.22	57.24 ± 0.51	56.97 ± 0.40	57.49 ± 0.34	$\mathbf{56.88 \pm 0.38}$
2	55.50 ± 0.78	55.23 ± 0.34	54.48 ± 0.23	$\mathbf{54.33 \pm 0.43}$	54.71 ± 0.29
3	53.60 ± 0.29	53.05 ± 0.35	52.84 ± 0.46	52.70 ± 0.45	$\mathbf{52.49 \pm 0.51}$
4	51.42 ± 0.59	51.12 ± 0.67	50.89 ± 0.44	51.09 ± 0.41	$\mathbf{50.67 \pm 0.71}$
5	50.01 ± 0.27	50.26 ± 0.33	49.96 ± 0.31	50.18 ± 0.60	$\mathbf{49.77 \pm 0.53}$

projectors instead of constant vectors. In MLP-k we follow the convolution layers with a hidden layer of k units and then a layer of softmax units; both layers are fully connected and all are trained together, and we take k to be comparable with the number of gating units on SDT.

In Table 4, results are given on MNIST, Fashion-MNIST and CIFAR-10. For MNIST and Fashion-MNIST, we also give results where we use the original input without any convolutional layers—we did not do this for CIFAR-10 and Imagenet32 because one cannot get any decent accuracy on them without any convolutional layers. First we see that convolutional layers help significantly, both with SDT and MLP. We also see that SDT-L works generally better than SDT. On MNIST, Fashion-MNIST and CIFAR-10, we see that SDT-L is almost as accurate as MLP with an equivalent number of hidden units; sometimes it is slightly better, sometimes it is slightly worse.

On Imagenet32 where we could not run many models, we compare SDT-L and the base model which is an MLP variant, and we see in Table 5 that again SDT is almost as accurate. These experiments also indicate that deep convolutional layers can be trained with the error signal that is back-propagated through the soft decision tree without any problem.

Table 3. Effect of L^2 regularization for different α coefficients.

Tree depth	$\alpha = 1 \times 10^{-4}$	$\alpha = 5 \times 10^{-5}$	$\alpha = 1 \times 10^{-5}$
MNIST			
1	**5.00 ± 0.03**	5.07 ± 0.17	5.13 ± 0.11
2	4.15 ± 0.30	3.84 ± 0.22	**3.74 ± 0.24**
3	3.60 ± 0.18	3.52 ± 0.20	**3.22 ± 0.17**
4	3.55 ± 0.13	3.19 ± 0.15	**3.06 ± 0.24**
5	3.53 ± 0.19	3.28 ± 0.17	**3.03 ± 0.12**
Fashion-MNIST			
1	13.81 ± 0.12	**13.78 ± 0.14**	13.79 ± 0.15
2	13.19 ± 0.26	12.92 ± 0.15	**12.88 ± 0.13**
3	12.74 ± 0.20	12.70 ± 0.24	**12.41 ± 0.22**
4	12.71 ± 0.16	12.38 ± 0.22	**12.07 ± 0.17**
5	12.65 ± 0.12	12.48 ± 0.06	**11.84 ± 0.12**
CIFAR-10			
1	60.77 ± 0.28	**57.51 ± 0.31**	61.13 ± 0.77
2	57.45 ± 0.43	**54.64 ± 0.47**	57.36 ± 0.43
3	54.04 ± 0.73	**51.84 ± 0.77**	54.15 ± 0.23
4	52.29 ± 0.46	**50.64 ± 0.33**	51.99 ± 0.40
5	50.90 ± 0.28	**50.09 ± 0.30**	50.43 ± 0.23

Table 4. Error percentages on the test sets.

$dim(\boldsymbol{z})$	SDT-3	SDT-4	SDT-5	SDT-L3	SDT-L4	SDT-L5	MLP-8	MLP-16	MLP-32
MNIST									
Orig. $\boldsymbol{x}$	11.96	7.99	7.51	2.67	2.57	**2.30**	7.76	4.74	3.16
50	1.37	1.08	0.76	0.72	0.71	0.63	0.56	0.54	**0.52**
100	1.02	0.96	0.98	0.66	0.67	0.74	**0.59**	0.61	0.59
200	1.11	0.84	0.95	0.76	0.76	0.62	0.68	**0.55**	0.57
Fashion-MNIST									
Orig. $\boldsymbol{x}$	20.95	29.80	20.83	11.94	11.50	**11.35**	16.66	14.50	13.47
50	10.46	10.24	10.56	7.36	**7.28**	8.08	8.02	7.55	7.73
100	10.12	10.40	9.76	7.89	**7.36**	8.05	8.16	7.67	7.56
200	12.28	9.14	10.37	7.55	7.18	**7.08**	7.59	7.51	7.81
CIFAR-10									
50	9.38	9.52	9.18	8.85	8.76	**8.64**	8.94	8.66	8.99
100	9.71	9.27	9.67	8.83	8.72	8.96	9.02	**8.69**	9.07
200	11.83	10.90	9.95	8.91	9.60	9.75	9.16	9.01	**8.85**

Table 5. Error percentages on Imagenet32 validation set.

	Top-1 error		Top-5 error	
	Base	SDT-L5	Base	SDT-L5
WRN-28-1	67.10	**66.86**	42.33	**41.91**
WRN-28-2	56.40	**56.33**	**31.14**	31.34

6 Conclusions

We show that input dropout and dropconnect can be used with soft decision trees as alternatives to L^2 regularization; of the two, input dropout seems the more interesting. On four image data sets, we see that convolutional layers can be incorporated into a decision tree and the whole can be trained in a coupled manner. The resulting architecture is as accurate as a deep MLP with the added advantage of interpretability. The depth of a tree has a different interpretation than in an MLP: In the former it corresponds to levels of granularity or resolution, whereas in the latter it corresponds to levels of abstraction.

Acknowledgements. The numerical calculations are performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

References

1. Jordan, M.I., Jacobs, R.A.: Hierarchical mixtures of experts and the EM algorithm. Neural Comput. **6**(2), 181–214 (1994)
2. İrsoy, O., Yıldız, O.T., Alpaydın E.: Soft decision trees. In: Proceedings of the International Conference on Pattern Recognition, Tsukuba, Japan, pp. 1819–1822 (2012)
3. LeCun, Y., et al.: Handwritten digit recognition with a back-propagation network. In: Advances in Neural Information Processing Systems, vol. 2, pp. 396–404 (1990)
4. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. Neural Comput. **3**(1), 79–87 (1991)
5. İrsoy, O., Alpaydın E.: Autoencoder trees. In: Asian Conference on Machine Learning, Hong Kong, China, pp. 378–390 (2015)
6. Yıldız, O.T., Alpaydın E.: Regularizing soft decision trees. In: International Symposium on Computer and Information Sciences, Paris, France (2013)
7. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
8. Wan, L., Zeiler, M., Zhang, S., LeCun, Y., Fergus, R.: Regularization of neural networks using DropConnect. In: International Conference on Machine Learning, Atlanta, GA, pp. 1058–1066 (2013)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. https://arxiv.org/abs/1512.03385 (2015)
10. Zagoruyko, S., Komodakis, N.: Wide residual networks. https://arxiv.org/abs/1605.07146 (2016)