Deep Neural Decision Trees

Yongxin Yang [1] **Irene Garcia Morillo** [1] **Timothy M. Hospedales** [1]

Abstract

Deep neural networks have been proven powerful at processing perceptual data, such as images and audio. However for tabular data, tree-based models are more popular. A nice property of tree-based models is their natural interpretability. In this work, we present Deep Neural Decision Trees (DNDT) – tree models realised by neural networks. A DNDT is intrinsically interpretable, as it is a tree. Yet as it is also a neural network (NN), it can be easily implemented in NN toolkits, and trained with gradient descent rather than greedy splitting. We evaluate DNDT on several tabular datasets, verify its efficacy, and investigate similarities and differences between DNDT and vanilla decision trees. Interestingly, DNDT self-prunes at both split and feature-level.

1. Introduction

The interpretability of predictive models is important, especially in cases where ethics are involved, such as law, medicine, and finance; and mission critical applications where we wish to manually verify the correctness of a model's reasoning. Deep neural networks (Lecun et al., 2015; Schmidhuber, 2015) have achieved excellent performance in many areas, such as computer vision, speech processing, and language modelling. However lack of interpretability prevents this family of black-box models from being used in applications for which we must know how the prediction is made in order to certify its decision process. Moreover, in some areas like Business Intelligence (BI), it is often more important to know how each factor contributes to the prediction rather than the conclusion itself. Decision tree (DT) based methods, such as C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984), have a clear advantage in this aspect, as one can easily follow the structure of the tree and check exactly how a prediction is made.

In this work, we propose a new model at the intersection of these two approaches – the Deep Neural Decision Tree (DNDT) – and explore its connections to each. DNDTs are neural networks with a special architecture, where any setting of DNDT weights corresponds to a specific decision tree, and is therefore interpretable[1]. However, as DNDT is realised by neural network (NN), it inherits several interesting properties different of conventional DTs: DNDT can be easily implemented in a few lines of code in any NN software framework; all parameters are simultaneously optimized with stochastic gradient descent rather than a more complex and potentially sub-optimal greedy splitting procedure; it is ready for large-scale processing with mini-batch-based learning and GPU acceleration out of the box, and it can be plugged into any larger NN model as a building block for end-to-end learning with back-propagation.

2. Related Work

Tree models Tree models are widely used in supervised learning, e.g., classification. They recursively partition the input space and assign a label/score to the final node. Well-known tree models include C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984). A key advantage of tree based models is that they are easy to interpret, since the predictions are given by a set of rules. It is also common to use an ensemble of multiple trees, such as Random Forest (Breiman, 2001) and XGBoost (Chen & Guestrin, 2016), to boost performance at the expense of interpretability. Such tree-based models are often competitive or better than neural networks at predictive tasks using tabular data.

Model interpretability With machine learning based predictions becoming ubiquitous and affecting many aspects of our daily lives, the focus of research moves beyond model performance (e.g., efficiency and accuracy), to other factors such as interpretability (Weller, 2017; Doshi-Velez, 2017). This is particularly so in applications where there are ethical (Bostrom & Yudkowsky, 2014) or safety concerns and models' predictions should be explainable in order to verify the correctness of their reasoning process or justify their decisions. There are now a number of attempts

[1]The University of Edinburgh. Correspondence to: Yongxin Yang <yongxin.yang@ed.ac.uk>.

2018 ICML Workshop on Human Interpretability in Machine Learning (WHI 2018), Stockholm, Sweden.

[1]The reverse is also true. For any DT, there is a corresponding DNDT that performs the same computation.

to make models explainable. Some are model-agnostic (Ribeiro et al., 2016), while most are associated with a certain type of model, e.g., rule-based classifiers (Dash et al., 2015; Malioutov et al., 2017), nearest neighbour models (Kim et al., 2016), and neural networks (Kim et al., 2017).

Neural Networks and Decision Trees Some studies have proposed to unify neural network and decision tree models. Bul & Kontschieder (2014) proposed Neural Decision Forests (NDF) as an ensemble of neural decision trees, where the split functions are realised by randomized multi-layer perceptrons. Deep-NDF (Kontschieder et al., 2015) exploited a stochastic and differentiable decision tree model, which jointly learns the representations (via CNNs) and the classification (via decision trees). Our proposed DNDT differs from those methods in many ways. First, we do not have an alternative optimisation procedure for structure learning (splitting) and parameter learning (score matrix). Instead, we learn them all via back-propagation in a single pass. Second, we do not restrict that the splits to be binary (left or right), as we apply a differentiable binning function that can split nodes into multiple (≥ 2) leaves. Finally, and most importantly, we design our model specifically for interpretability, especially for application to tabular data, where we can interpret every input feature. In contrast, the models in (Bul & Kontschieder, 2014; Kontschieder et al., 2015) are designed for prediction performance and applied to raw image data. Some design decisions make them not appealing to tabular data. E.g., in Kontschieder et al. (2015), they use a less flexible tree where the structure is fixed while the node split is learned.

Despite the similar name, our work is fundamentally different to Balestriero (2017) which developed a kind of 'oblique' decision tree realised by neural network. In contrast to conventional 'univariate' decision trees, each node in their oblique decision tree involves *all* features rather than a single feature, which renders the model uninterpretable.

Alternative Decision Tree Inducers Conventional DTs are learned by recursive greedy splitting of features (Quinlan, 1993; Breiman et al., 1984). This is efficient and has some benefits for feature selection, however such greedy search may be sub-optimal (Norouzi et al., 2015). Some recent work explores alternative approaches to training decision trees which aim to achieve better performance with less myopic optimization, for example with latent variable structured prediction (Norouzi et al., 2015), or training an RNN splitting controller using reinforcement learning (Xiong et al., 2017). In contrast, our DNDT is much simpler than these, but can still potentially find better solutions than conventional DT inducers by simultaneously searching the structure and parameters of the tree with SGD. Finally, we also note that while conventional DT inducers leverage only binary splits for simplicity, our DNDT model can equally easily work with splits of arbitrary cardinality, which can sometimes make for more interpretable trees.

3. Methodology

3.1. Soft binning function

The core module we implement here is a *soft* binning function (Dougherty et al., 1995) that we will use to make the split decisions in DNDT. Typically, a binning function takes as input a real scalar x and produces an index of the bins to which x belongs. Hard binning is non-differentiable, so we propose a differentiable approximation of this function.

Assuming we have a continuous variable x, that we want to bin into $n+1$ intervals. This leads to the need of n cut points, which are trainable variables in this context. We denote the cut points as $[\beta_1, \beta_2, \ldots, \beta_n]$ in a monotonically increasing manner[2], i.e., $\beta_1 < \beta_2 < \cdots < \beta_n$.

Now we construct a one-layer neural network with softmax as its activation function.

$$\pi = f_{w,b,\tau}(x) = \mathrm{softmax}((wx + b)/\tau) \tag{1}$$

Here w is a constant rather than a trainable variable, and its value is set as $w = [1, 2, \ldots, n+1]$. b is constructed as,

$$b = [0, -\beta_1, -\beta_1 - \beta_2, \ldots, -\beta_1 - \beta_2 - \cdots - \beta_n]. \tag{2}$$

and $\tau > 0$ is a temperature factor. As $\tau \to 0$ the output tends to a one-hot vector.

We can verify it by checking three consecutive logits o_{i-1}, o_i, o_{i+1}. When we have both $o_i > o_{i-1}$ (so $x > \beta_i$) and $o_i > o_{i+1}$ (so $x < \beta_{i+1}$), x must fall into the interval (β_i, β_{i+1}). Thus, the neural network in Eq. 1 will produce an *almost* one-hot encoding of the binned x, especially with lower temperature. Optionally, we can apply the slope annealing trick (Chung et al., 2017) that progressively reduces the temperature during training so that we can get a more deterministic model in the end.

If one prefers an *actual* one-hot vector, Straight-Through (ST) Gumbel-Softmax (Jang et al., 2017) can be applied: for the forward pass, we sample a one-hot vector using Gumbel-Max trick, while for the backward pass, we use Gumbel-Softmax to compute the gradient (see Bengio (2013) for a more detailed analysis).

Fig. 1 demonstrates a concrete example where we have a scalar x in the range of $[0, 1]$ and two cut points at 0.33

[2]During training, the order of β's may be shuffled up after updating, so we have to sort them first in every forward pass. However, this will not affect the differentiability because sort just swaps the positions of β's.

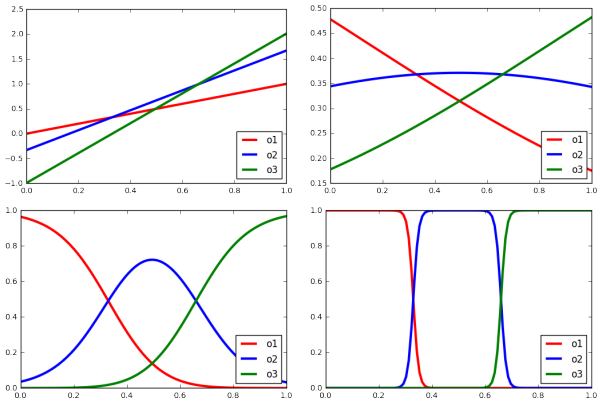

Figure 1. A concrete example of our soft binning function using cut-points at 0.33 and 0.66. x-axis is the value of a continuous input variable $x \in [0, 1]$. Top-left: the original values of logits; Top-right: values after applying softmax function with $\tau = 1$; Bottom-left: $\tau = 0.1$; Bottom-right: $\tau = 0.01$.

and 0.66 respectively. Based on Eq. 1 and Based on Eq. 2, we have the three logits $o_1 = x$, $o_2 = 2x - 0.33$, $o_3 = 3x - 0.99$.

3.2. Making Predictions

Given our binning function, the key idea is to construct the decision tree via Kronecker product $\otimes$. Assume we have an input instance $x \in \mathcal{R}^D$ with D features. Binning each feature x_d by its own neural network $f_d(x_d)$, we can exhaustively find all final nodes by,

$$z = f_1(x_1) \otimes f_2(x_2) \otimes \cdots \otimes f_D(x_D). \qquad (3)$$

Here z is now also an almost one-hot vector that indicates the index of the leaf node where instance x arrives. Finally, we assume a linear classifier at each leaf z classifies instances arriving there. DNDT is illustrated in Fig. 2.

3.3. Learning the Tree

With the method described so far we can route input instances to leaf nodes and classify them. Thus training a decision tree now becomes a matter of training the bin cut points and leaf classifiers. Since all steps of our forward pass are differentiable, all parameters (Fig. 2, red) can now be straightforwardly and simultaneously trained with SGD.

Discussion DNDT scales well with number of instances due to neural network style mini-batch training. However a key drawback of the design so far is that, due to the use of Kronecker product, it is not scalable with respect to the number of features. In our current implementation, we avoid this issue with 'wide' datasets by training a forest with random subspace (Ho, 1998) – at the expense of our interpretibility. That is, introducing multiple trees, each trained on a random subset of features. A better solution that does not require an uninterpretable forest is to exploit the sparsity of the final binning during learning: the number of non-empty leaves grows much slower than the total number of leaves. But this somewhat complicates the otherwise simple implementation of DNDT.

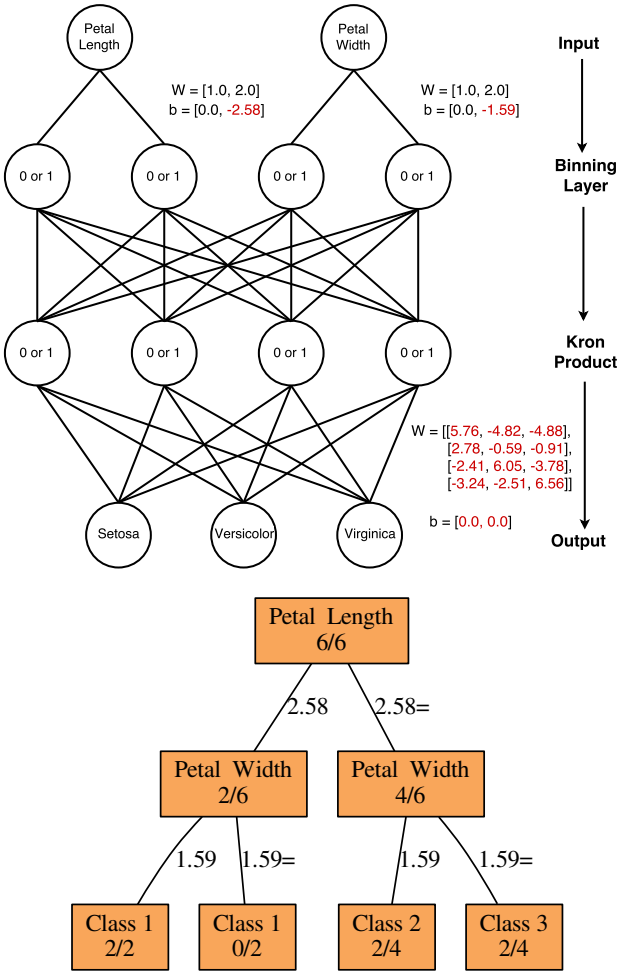

Figure 2. A learned DNDT for the *Iris* dataset (reduced two-feature version). Top: DNDT-view where red fonts indicate trainable variables, and black indicates constants. Below: DT-view. The same network rendered as a conventional decision tree. The fractions indicate the route of a randomly chosen 6 instances being classified.

4. Experiments

4.1. Implementation

DNDT is conceptually simple and easy to implement with $\approx$ 20 lines code in TensorFlow (Abadi et al., 2015) or Py-

Dataset	#inst.	#feat.	#cl.
Iris	150	4	3
Haberman's Survival	306	3	2
Car Evaluation	1728	6	4
Titanic (K)	714	10	2
Breast Cancer Wisconsin	683	9	2
Pima Indian Diabetes (K)	768	8	2
Gime-Me-Some-Credit (K)	201669	10	2
Poker Hand	1025010	11	9
Flight Delay	1100000	9	2
HR Evaluation (K)	14999	9	2
German Credit Data	1000	20	2
Connect-4	67557	42	2
Image Segmentation	2310	19	7
Covertype	581012	54	7

Table 1. Collection of 14 datasets from Kaggle (indicated with (K)) and UCI: number of instances (#inst.), number of features (#feat.), and number of classes (#cl.)

Dataset	DNDT	DT	NN
Iris	**100.0**	**100.0**	**100.0**
Haberman's Survival	**70.9**	66.1	**70.9**
Car Evaluation	95.1	**96.5**	91.6
Titanic	**80.4**	79.0	76.9
Breast Cancer Wisconsin	94.9	91.9	**95.6**
Pima Indian Diabetes	66.9	**74.7**	64.9
Gime-Me-Some-Credit	98.6	92.2	**100.0**
Poker Hand	50.0	**65.1**	50.0
Flight Delay	**78.4**	67.1	78.3
HR Evaluation	92.1	**97.9**	76.1
German Credit Data (*)	**70.5**	66.5	**70.5**
Connect-4 (*)	66.9	**77.7**	75.7
Image Segmentation (*)	70.6	**96.1**	48.05
Covertype (*)	49.0	**93.9**	49.0
# of wins	5	**7**	5
Mean Reciprocal Rank	0.65	**0.73**	0.61

Table 2. Test set accuracy of each model: DT: Decision tree. NN: neural network. DNDT: Our deep neural decision tree, where (*) indicates that the ensemble version is used.

Torch (Paszke et al., 2017)[3]. Because it is implemented as a neural network, DNDT supports 'out of the box' GPU acceleration and mini-batch based learning of datasets that do not fit in memory, thanks to modern deep learning frameworks.

4.2. Datasets and Competitors

We compare DNDT against neural networks (implemented by TensorFlow (Abadi et al., 2015)) and decision tree (from Scikit-learn (Pedregosa et al., 2011)) on 14 datasets collected from Kaggle and UCI (see Tab. 1 for dataset details).

For decision tree (DT) baseline we set two of the key hyper-parameters *criterion* as 'gini' and *splitter* as 'best'. For neural network (NN), we use an architecture of two hidden layers with 50 neurons each for all datasets. DNDT also has a hyper-parameter, the number of cut points for each feature (branching factor), which we set to 1 for all features and datasets. A detailed analysis of the effect of this hyper-parameter can be found in Sec. 4.4. For datasets with more than 12 features, we use an ensemble of DNDT, where each tree picks 10 features randomly, and we have 10 tress in total. The final prediction is given by majority voting.

4.3. Accuracy

We evaluate the performance of DNDT, decision tree, and neural network models on each of the datasets in Tab. 1. The test set accuracies are presented in Tab. 2.

Overall the best performing model is the DT. DT's good performance is not surprising because these datasets are mainly tabular and the feature dimension is relatively low. Conventionally, neural networks do not have a clear advantage on this kind of data. However, DNDT is slightly better than the vanilla neural network, as it is closer to decision tree by design. Of course this is only an indicative result, as all of these models have tuneable hyperparameters. Nevertheless, it's interesting that no model has a dominant advantage. This is reminiscent of *no free lunch theorems* (Wolpert, 1996).

4.4. Analysis of active cut-points

In DNDT the number of cut points per feature is the model complexity parameter. We do not bound the cut points' values, which means it is possible that some of them are inactive. E.g., they are either smaller than the minimal x_d or greater than the maximal x_d.

In this section, we investigate how many of cut points are *actually* used after DNDT learning. A cut point is active when at least one instance from the dataset falls on each side of it. For four datasets, Car Evaluation, Pima, Iris, and Haberman's, we set the number of cut points per feature from 1 to 5, and calculate the percentage of active cut points, as shown in Fig. 3. We can see that as the number of cut points increases, their utilisation generally decreases. This implies that DNDT is somewhat self-regularising: it does not make use of all the parameters available to it.

We can further investigate how the number of available cut points affects performance on these datasets. As we can see in Fig. 4, performance initially increases with more cut points, before stabilising after a certain value. This is reassuring because it means that large DNDTs do not over-fit

[3] https://github.com/wOOL/DNDT

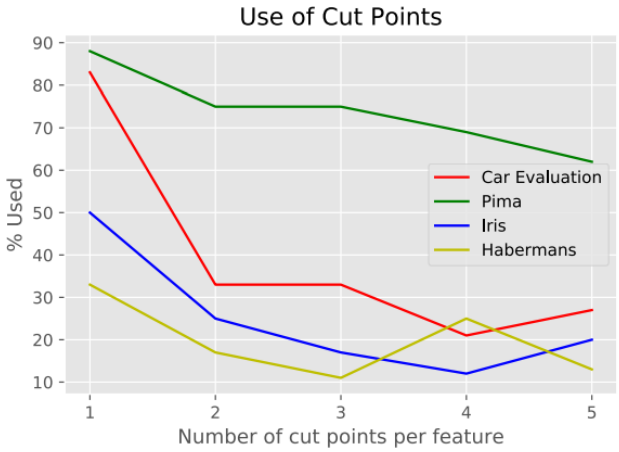

Figure 3. Percentage (%) of active cut points used by DNDT.

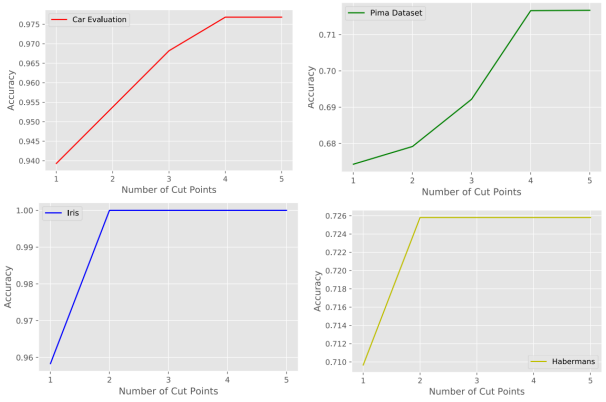

Figure 4. Test accuracy of DNDT for increasing number of cut points (model complexity).

the training data, even without explicit regularisation.

4.5. Analysis of active features

In DNDT learning, it is also possible that for a certain feature all cut points are inactive. This corresponds to disabling the feature, so that it does not impact prediction. It is analogous to a conventional DT learner never selecting a given feature to make a split anywhere in the tree. In this section we analyse how DNDT rules out features in this way. We run DNDT 10 times, and record the number of times a given feature is excluded because all its cut points are inactive.

Given randomness from both weight initialisation and mini-batch sampling, we observe that some features (e.g., index 0 feature in iris) are consistently ignored by DNDT (See Tab. 3 for all results). This suggests that DNDT does some implicit feature selection by pushing cut points out of the data boundary for unimportant features. As a side product, we can obtain a measure of feature importance from

Dataset \ Feat. Idx	0	1	2	3	4	5	6	7	8	9
Haberman's	**100**	**100**	0	-	-	-	-	-	-	-
Iris	**100**	90	50	10	-	-	-	-	-	-
Pima	10	0	0	0	20	0	0	**100**	-	-
Titanic	0	0	0	0	0	10	20	10	20	40

Table 3. Percentage (%) of times that DNDT ignores each feature.

feature selection over multiple runs: The more times a feature is ignored, the less important it is likely to be.

4.6. Comparison to decision tree

Using the techniques developed in Sec. 4.5, we investigate whether DNDT and DT favour similar features. We compare the the feature importance through Gini used in decision tree (Fig. 5) with our selection rate metric (Table 3).

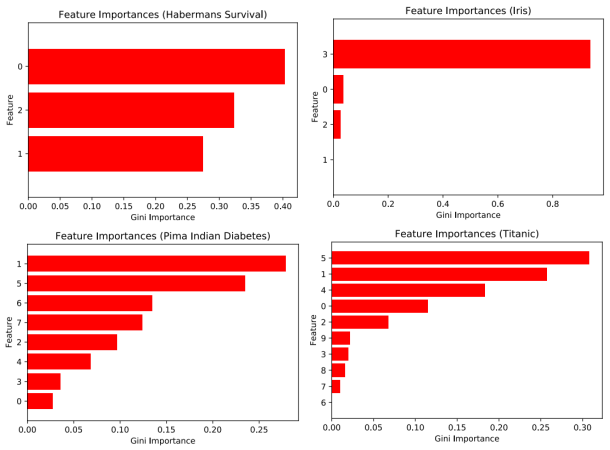

Figure 5. Feature importance ranking produced by DT (Gini).

Comparing these results we see that sometimes DNDT and DT share a feature selection preference. E.g., for Iris, they both rank feature 3 as the most important. But it happens that they can also have different views, e.g., for Haberman's, DT picked feature 0 as the most important, whereas DNDT completely ignored it. In fact, DNDT only makes use of feature 2 for prediction, which is ranked second by DT. However, this kind of disagreement may not necessarily lead to significantly different performance. As we can see in Tab. 2, for Haberman's, the test accuracies of DNDT and DT are 70.9% and 66.1% respectively.

Finally, we quantify the similarity between DNDT feature ranking and DT feature ranking by calculating Kendall's Tau of two ranking lists. The results in Tab. 4 suggest a moderate correlation overall.

4.7. GPU Acceleration

Finally we verify the ease of accelerating DNDT learning of DTs by GPU processing – a capability not common

Dataset	Titanic	Iris	Pima	Habermans
Kendall's Tau	0.4	0.33	0.32	0.0

Table 4. Kendall's Tau of DNDT's and DT's feature ranking: larger values mean 'more similar'

or straightforward for conventional DT learners. By increasing the number of cut points for each feature, we can get larger models, for which GPU mode has significantly shorter running time (see Fig. 6).

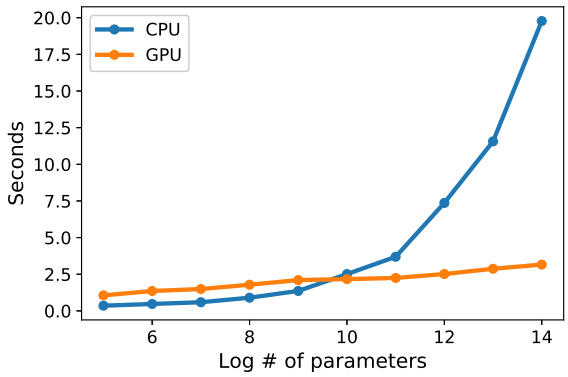

Figure 6. GPU Acceleration illustration: DNDT training time on 3.6GHz CPU vs GTX Titian GPU. Average over 5 runs.

5. Conclusion

We introduced a neural network based tree model DNDT. It has better performance than NNs for certain tabular datasets, while providing an interpretable decision tree. Meanwhile compared to conventional DTs, DNDT is simpler to implement, simultaneously searches tree structure and parameters with SGD, and is easily GPU accelerated.

There are many avenues for future work. We want to investigate the source of self-regularisation that we observed; explore plugging in DNDT as a module connected to a conventional CNN feature learner for end-to-end learning; find out whether DNDT's whole-tree SGD-based learning can be used as postprocessing to fine-tune conventional greedily trained DTs and improve their performance; and find out whether the many NN-based approaches to transfer learning can be leveraged to enable transfer learning for DTs.

Acknowledgements This work was supported by the EPSRC grant EP/R026173/1.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dandelion, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/.

Balestriero, R. Neural Decision Trees. *ArXiv e-prints*, 2017.

Bengio, Yoshua. Estimating or propagating gradients through stochastic neurons. *CoRR*, abs/1305.2982, 2013.

Bostrom, Nick and Yudkowsky, Eliezer. *The ethics of artificial intelligence*, pp. 316334. Cambridge University Press, 2014.

Breiman, L., H. Friedman, J., A. Olshen, R., and J. Stone, C. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

Breiman, Leo. Random forests. *Machine Learning*, 45(1): 5–32, October 2001.

Bul, S. and Kontschieder, P. Neural decision forests for semantic image labelling. In *CVPR*, 2014.

Chen, Tianqi and Guestrin, Carlos. Xgboost: A scalable tree boosting system. In *KDD*, 2016.

Chung, J., Ahn, S., and Bengio, Y. Hierarchical Multiscale Recurrent Neural Networks. In *ICLR*, 2017.

Dash, S., Malioutov, D. M., and Varshney, K. R. Learning interpretable classification rules using sequential rowsampling. In *ICASSP*, 2015.

Doshi-Velez, Finale; Kim, Been. Towards a rigorous science of interpretable machine learning. *ArXiv e-prints*, 2017.

Dougherty, James, Kohavi, Ron, and Sahami, Mehran. Supervised and unsupervised discretization of continuous features. In *ICML*, 1995.

Ho, Tin Kam. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

Jang, E., Gu, S., and Poole, B. Categorical Reparameterization with Gumbel-Softmax. In *ICLR*, 2017.

Kim, B., Gilmer, J., Viegas, F., Erlingsson, U., and Wattenberg, M. TCAV: Relative concept importance testing with Linear Concept Activation Vectors. *ArXiv e-prints*, 2017.

Kim, Been, Khanna, Rajiv, and Koyejo, Sanmi. Examples are not enough, learn to criticize! Criticism for interpretability. In *NIPS*, 2016.

Kontschieder, P., Fiterau, M., Criminisi, A., and Bul, S. R. Deep neural decision forests. In *ICCV*, 2015.

Lecun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521(7553):436–444, 5 2015.

Malioutov, Dmitry M., Varshney, Kush R., Emad, Amin, and Dash, Sanjeeb. Learning interpretable classification rules with boolean compressed sensing. In *Transparent Data Mining for Big and Small Data*, pp. 95–121. Springer International Publishing, 2017.

Norouzi, Mohammad, Collins, Maxwell D., Johnson, Matthew, Fleet, David J., and Kohli, Pushmeet. Efficient non-greedy optimization of decision trees. In *NIPS*, 2015.

Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in pytorch. In *NIPS Workshop on Autodiff*, 2017.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Quinlan, J. Ross. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

Ribeiro, Marco Tulio, Singh, Sameer, and Guestrin, Carlos. "why should i trust you?": Explaining the predictions of any classifier. In *KDD*, 2016.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

Weller, Adrian. Challenges for transparency. In *ICML Workshop on Human Interpretability in Machine Learning*, pp. 55–62, 2017.

Wolpert, David H. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

Xiong, Zheng, Zhang, Wenpeng, and Zhu, Wenwu. Learning decision trees with reinforcement learning. In *NIPS Workshop on Meta-Learning*, 2017.