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.
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. Typically, a binning function (Dougherty et al., 1995) that we will use to make the split decisions in DNDT. 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.

\[
\pi = f_{w, b, \tau}(x) = \text{softmax}(w x + b) / \tau
\]  

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,

\[
b = [0, -\beta_1, -\beta_1 - \beta_2, \ldots, -\beta_1 - \beta_2 - \cdots - \beta_n].
\]

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 \( (\beta_i, \beta_{i+1}) \). 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 \( \beta \)'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 \( \beta \)'s.
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 interpretability. 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.

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-
Deep Neural Decision Trees

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                      | #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 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.

| 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 (K)     | 66.9  | **74.7** | 64.9 |
| Gime-Me-Some-Credit (K)      | 98.6  | 92.2 | **100.0** |
| Poker Hand                   | 50.0  | **65.1** | 50.0 |
| Flight Delay                 | **78.4** | 67.1 | 78.3 |
| HR Evaluation (K)            | 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

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 hyperparameters 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 trees 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.

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.

\(^3\)https://github.com/wOOL/DNDT
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 feature selection over multiple runs: The more times a feature is ignored, the less important it is likely to be.

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

| Dataset     | Feature 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 |

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 feature importance through Gini used in decision tree (Fig. 5) with our selection rate metric (Table 3).

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...
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 Titan GPU. Average over 5 runs.](image)

### 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, Martin, 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, Jouppi, Pranav, Keutzer, Kael, 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, Xiaolong. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/.

- Balestrieri, 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 Kotschieder, 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 *Transpar-ent 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., Cour, napeau, 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.