International Journal of Computer Vision (2020) 128:997–1011
https://doi.org/10.1007/s11263-019-01237-6

End-to-End Learning of Decision Trees and Forests

Thomas M. Hehn[1] · **Julian F. P. Kooij**[1] · **Fred A. Hamprecht**[2]

Received: 6 March 2019 / Accepted: 17 September 2019 / Published online: 9 October 2019

Abstract
Conventional decision trees have a number of favorable properties, including a small computational footprint, interpretability, and the ability to learn from little training data. However, they lack a key quality that has helped fuel the deep learning revolution: that of being end-to-end trainable. Kontschieder et al. (ICCV, 2015) have addressed this deficit, but at the cost of losing a main attractive trait of decision trees: the fact that each sample is routed along a small subset of tree nodes only. We here present an end-to-end learning scheme for deterministic decision trees and decision forests. Thanks to a new model and expectation–maximization training scheme, the trees are fully probabilistic at train time, but after an annealing process become deterministic at test time. In experiments we explore the effect of annealing visually and quantitatively, and find that our method performs on par or superior to standard learning algorithms for oblique decision trees and forests. We further demonstrate on image datasets that our approach can learn more complex split functions than common oblique ones, and facilitates interpretability through spatial regularization.

Keywords Decision forests · End-to-end learning · Efficient inference · Interpretability

1 Introduction

Neural networks are currently the dominant classifier in computer vision (Russakovsky et al. 2015; Cordts et al. 2016), whereas decision trees and decision forests have proven their worth when training data or computational resources are scarce (Barros et al. 2012; Criminisi and Shotton 2013). One can observe that both neural networks and decision trees are composed of basic computational units, the perceptrons and nodes, respectively. A crucial difference between the two is that in a standard neural network, *all* units are evaluated for every input, while in a reasonably balanced decision tree with I inner split nodes, only $\mathcal{O}(\log I)$ split nodes are visited. That is, in a decision tree, a sample is routed along a single path from the root to a leaf, with the path conditioned on the sample's features. Various works are now exploring the relation between both classification approaches (Ioannou et al. 2016; Wang et al. 2017), such as the Deep Neural Decision Forests (DNDFs) (Kontschieder et al. 2015). Similar to deep neural networks, DNDFs require evaluating all computational paths to all leaf nodes in a tree for each test sample, which results in high accuracy, but incurs large computational and memory costs especially as trees grow deeper.

Our work proposes an orthogonal approach. We seek to stick to traditional decision trees and forests as inference models for their advantages, while improving learning of such trees through end-to-end training with backpropagation, one of the hallmarks of neural networks. It is efficiency, induced by the sparsity of the sample-dependent computational graph, that piques our interest in decision trees. Further, we also hope to profit from their relative interpretability. End-to-end training allows optimizing all levels of a decision tree jointly. Furthermore, features can now be jointly learned through linear nodes, but also through more complex split functions such as small convolutional neural networks (CNNs). This is a feature that has so far been

Communicated by Mario Fritz.

The authors gratefully acknowledge financial support by DFG Grant HA 4364/10-1.

✉ Thomas M. Hehn
T.M.Hehn@tudelft.nl

Julian F. P. Kooij
J.F.P.Kooij@tudelft.nl

Fred A. Hamprecht
fred.hamprecht@iwr.uni-heidelberg.de

[1] Intelligent Vehicles Group, Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands

[2] HCI/IWR, Heidelberg University, 69120 Heidelberg, Germany

missing in deterministic decision trees, which are usually constructed greedily without subsequent tuning. We propose a mechanism to remedy this deficit.

1.1 Related Work

Random forests are ensembles of decision trees, and were introduced by Breiman (2001). In this section, we review use cases for trees and forests, choices for split functions, different optimization strategies, and the connection to deep neural networks.

Applications While neural networks have these days superseded all other approaches in terms of achievable accuracy on many benchmarks (Cardona et al. 2010; Lin et al. 2014; Russakovsky et al. 2015; Cordts et al. 2016), state-of-the-art networks are not easy to interpret, are fairly hungry for training data, often require weeks of GPU training and have a computational and memory footprint that limits their use on small embedded devices. Decision trees and decision tree ensembles, such as random forests, generally achieve lower accuracy on large datasets, but are fundamentally more frugal. They have shown their effectiveness on a variety of classification tasks (Barros et al. 2012; Fernández-Delgado et al. 2014) and also found wide application in computer vision, e.g. Viola and Jones (2001), Shotton et al. (2011), Criminisi and Shotton (2013), Dollár et al. (2014), Zhang et al. (2017), Cordts et al. (2017). They are well suited for tasks where computational resources are limited, e.g. real-time human pose estimation in the Microsoft Kinect (Shotton et al. 2011), or few and unbalanced training samples are available, e.g. during online object tracking (Zhang et al. 2017).

Decision trees are also found in expert systems, since they are widely recognized as interpretable models which divide a complex classification task in several simpler ones. For instance, Worachartcheewan et al. (2010), Pinhas-Hamiel et al. (2013), Huang et al. (2015) use their interpretability for diabetes research, and Guh et al. (2011) interpret their outcome prediction of in vitro fertilization. Likewise, De Ville (2006) explains the application of decision trees in business analytics.

Split functions There have been several attempts to train decision trees with more complex split functions. Menze et al. (2011) have benchmarked oblique random forests on various binary classification problems. These oblique random forests used linear and non-linear classifiers at each split in the decision trees and thereby combined more than one feature at a time. Montillo et al. (2013) have successfully approximated the information gain criterion using a sigmoid function and a smoothness hyperparameter. Expanding these ideas, Laptev and Buhmann (2014) have trained small convolutional neural networks (CNNs) at each split in a decision tree to perform binary segmentation. Rota Bulo and Kontschieder (2014) also apply a greedy strategy to learn neural networks for each split node and hence learn the structure of the tree. Notably, these approaches use gradient optimization techniques, but are lacking joint optimization of an entire tree, i.e. end-to-end learning of the entire model.

Optimization Hyafil and Rivest (1976) have shown that the problem of finding an optimal decision tree is NP-complete. As a consequence, the common approach is to find axis-aligned splits by exhaustive search, and learn a decision tree with a greedy level-by-level training procedure as proposed by Breiman et al. (1984). In order to improve their performance, it is common practice to engineer split features for a specific task (Lepetit et al. 2005; Gall and Lempitsky 2009; Kontschieder et al. 2013; Cordts et al. 2017). Evolutionary algorithms are another group of optimization methods which can potentially escape local optima, but are computationally expensive and heuristic, requiring to tune many parameters (cf. Barros et al. (2012) for a survey).

Norouzi et al. (2015b) propose an algorithm for optimization of an entire tree with a given structure. They show a connection between optimizing oblique splits and structured prediction with latent variables. As a result, they formulate a convex–concave upper bound on the tree's empirical loss. The same upper bound is used to find an initial tree structure in a greedy algorithm. Their method is restricted to linear splits and relies on the kernel trick to introduce higher order split features. Alternating Decision Forests (Schulter et al. 2013) instead include a global loss when growing the trees, thereby optimizing the whole forest jointly.

Some works have explored gradient-based optimization of a full decision tree model already. While Suárez and Lutsko (1999) focused on a fuzzy approach of decision tree, Jordan (1994) introduced hierarchical mixtures of experts. In the latter model the predictions of expert classifiers are weighted based on conditional path probabilities in a fully probabilistic tree.

Kontschieder et al. (2015) make use of gradient-based decision tree learning to learn a deep CNN and use it as a feature extractor for an entire ensemble of decision trees. They use sigmoid functions to model the probabilistic routes and employ a log-likelihood objective for training. However, their inference model is unlike a standard tree as it stays fuzzy or probabilistic after training. When predicting new samples, all leaves and paths need to be evaluated for every sample, which subverts the computational benefits of trees. Furthermore, they consider only balanced trees, so the number of evaluated split functions at test time grows exponentially with increased tree depth.

Connections to deep neural networks Various works explore the connections between neural networks and traditional

decision tree ensembles. Sethi (1990), Welbl (2014) cast decision tree ensembles to neural networks, which enables gradient descent training. As long as the structure of the trees is preserved, the optimized parameters of the neural network can also be mapped back to the decision forest. Subsequently, Richmond et al. (2016) map stacked decision forests to CNNs and found an approximate mapping back. Frosst and Hinton (2017) focus on using the learned predictions of neural networks as a training target for probabilistic decision trees.

A related research direction is to learn conditional computations in deep neural networks. In Ioannou et al. (2016), Bolukbasi et al. (2017), McGill and Perona (2017), Wang et al. (2018), Huang et al. (2018), several models of neural networks with separate, conditional data flows are discussed. Still, the structure of the resulting inference models is fixed a priori.

1.2 Contributions

This work extends our previous conference contribution (Hehn and Hamprecht 2018) where we introduced end-to-end learning for decision trees. Here, we add an extension to decision forests, which we compare to state-of-the-art methods for training forests, and provide additional results on interpretability and the effect of the steepness parameter. Compared to existing research, our work provides the following contributions:

- We propose to learn deterministic decision trees and forests in an end-to-end fashion. Unlike related end-to-end approaches (Kontschieder et al. 2015), we obtain trees with deterministic nodes at test time. This results in efficient inference as each sample is only routed along one unique path of only $\mathcal{O}(\log I)$ out of the I inner nodes in a tree. To reduce variance, we can also combine multiple trees in a decision forest ensemble. Furthermore, an end-to-end trainable tree can provide interpretable classifiers on learned visual features, similar to how decision trees are used in financial or medical expert systems on handcrafted features. In this context, we show the benefit of regularizing the spatial derivatives of learned features when samples are images or image patches.
- To enable end-to-end training of a decision tree, we propose to use differentiable probabilistic nodes at *train time only*. We develop a new probabilistic split criterion that generalizes the long-established information gain (Quinlan 1990). A key aspect of this new tree formulation is the introduction of a steepness parameter for the decision (Montillo et al. 2013). The proposed criterion is asymptotically identical to information gain in the limit of very steep non-linearities, but allows to better model class overlap in the vicinity of a split decision boundary.
- A matching optimization procedure is proposed. During training, the probabilistic trees are optimized using the Expectation–Maximization algorithm (Jordan and Jacobs 1994). Importantly, the steepness parameter is incrementally adjusted in an annealing scheme to make decisions ever more deterministic, and bias the model towards crispness. The proposed procedure also constructs the decision trees level-by-level, hence trees will not grow branches any further than necessary. Compared to initialization with balanced trees (Kontschieder et al. 2015) our approach reduces the expected depth of the tree, which further improves efficiency.

Section 2 formalizes the new optimization procedure and probabilistic decision tree formulation. In Sect. 3, we compare the performance of the proposed method to that of related work on decision trees and decision forests. We also evaluate the benefits of the annealing scheme when training decision forests for a given number of epochs, demonstrate improved interpretability by regularizing the spatial derivatives of learned features on images or image patches, and show the use of CNNs as split features. Finally, Sect. 4 presents the conclusions and suggests future work.

2 Methods

Consider a classification problem with input space $\mathcal{X} \subset \mathbb{R}^p$ and output space $\mathcal{Y} = \{1, \ldots, K\}$. The training set is defined as $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} = \mathcal{X}_t \subset \mathcal{X}$ with corresponding classes $\{y_1, \ldots, y_N\} = \mathcal{Y}_t \subset \mathcal{Y}$.

We propose training a probabilistic decision tree model to enable end-to-end learning. Nevertheless, we train a model which is deterministic at test time. To account for this discrepancy, we introduce a steepness parameter to gradually enforce more deterministic splits during training. This addition is further motivated by the connection of our learning objective to the information gain criterion (see Sect. 2.6).

2.1 Standard Decision Tree and Notation

In binary decision trees (Fig. 1c), split functions $s : \mathbb{R} \to [0, 1]$ determine the routing of a sample through the tree, conditioned on that sample's features. The split function controls whether the splits are deterministic or probabilistic. The prediction is made by the leaf node that is reached by the sample.

Split nodes Each split node $i \in \{1, \ldots, I\}$ computes a split feature from a sample, and sends that feature into a split function. That function is a map $f_{\boldsymbol{\beta}_i} : \mathbb{R}^p \to \mathbb{R}$ parametrized by $\boldsymbol{\beta}_i$. For example, *oblique splits* are a linear combination of the input, i.e. $f_{\boldsymbol{\beta}_i}(\boldsymbol{x}) = (\boldsymbol{x}^T, 1) \cdot \boldsymbol{\beta}_i$ with $\boldsymbol{\beta}_i \in \mathbb{R}^{p+1}$. Similarly, an axis-aligned split perpendicular to axis a is represented

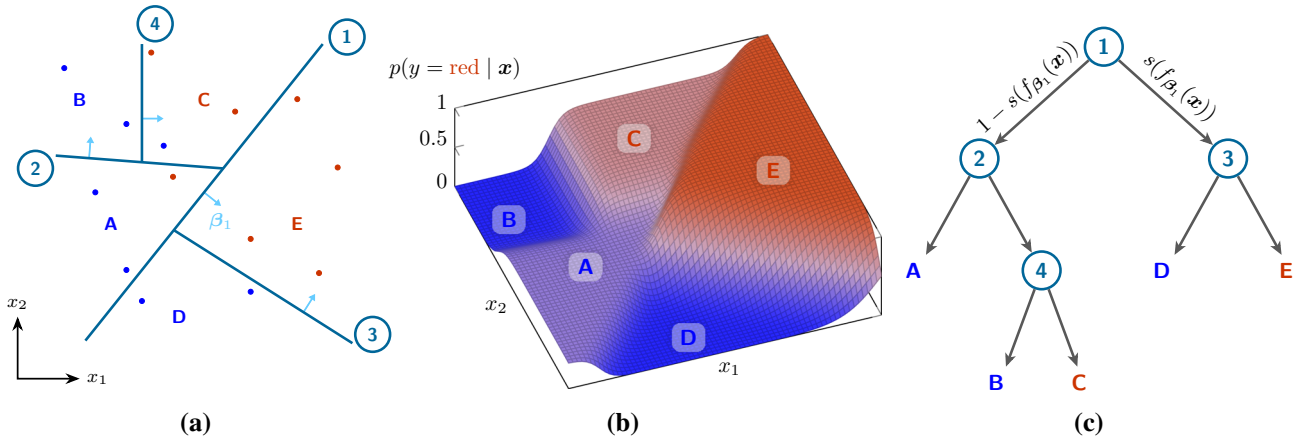

Fig. 1 Probabilistic oblique decision trees. **a** A feature space with a binary classification problem tessellated by an example oblique decision tree. The oblique splits (1–4) partition the feature space into five different leaves (A–E). **b** The predicted $p(y = \text{red} \mid \boldsymbol{x})$ (Eq. 2) of the oblique decision tree when a probabilistic split (Eq. 3) is used. **c** The corresponding tree diagram (Color figure online)

by an oblique split whose only non-zero parameters are at index a and $p + 1$. We write $\boldsymbol{\theta}_{\boldsymbol{\beta}} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_I)$ to denote the collection of all split parameters in the tree.

Leaf nodes Each leaf $\ell \in \{1, \ldots, L\}$ stores the parameters of a categorical distribution over classes $k \in \{1, \ldots, K\}$ in a vector $\boldsymbol{\pi}_\ell \in [0, 1]^K$. These vectors are normalized such that the probability of all classes in a leaf sum to $\sum_{k=1}^{K} (\boldsymbol{\pi}_\ell)_k = 1$. We define $\boldsymbol{\theta}_{\boldsymbol{\pi}} = (\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_L)$ to include all leaf parameters in the tree.

Paths Each leaf is reached by precisely one unique set of split outcomes, called a path. We define the probability that a sample $\boldsymbol{x}$ takes the path to leaf ℓ as

$$\mu_\ell(\boldsymbol{x}; s, \boldsymbol{\theta}_{\boldsymbol{\beta}}) = \prod_{r \in \mathcal{R}_\ell} s(f_{\boldsymbol{\beta}_r}(\boldsymbol{x})) \prod_{l \in \mathcal{L}_\ell} \big(1 - s(f_{\boldsymbol{\beta}_l}(\boldsymbol{x}))\big). \tag{1}$$

Here, $\mathcal{R}_\ell \subset \{1, \ldots, I\}$ denotes the splits on the path which contain ℓ in the right subtree. Analogously, $\mathcal{L}_\ell \subset \{1, \ldots, I\}$ denotes splits which contain ℓ in the left subtree. In Fig. 1c this means that $\mathcal{R}_\text{B} = \{2\}$ and $\mathcal{L}_\text{B} = \{1, 4\}$. Also note that in the following, we will omit the dependency on s, except when we consider a specific function s.

The prediction of the entire decision tree is given by multiplying the path probability with the corresponding leaf prediction:

$$p(y|\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{\ell=1}^{L} (\boldsymbol{\pi}_\ell)_y \mu_\ell(\boldsymbol{x}; \boldsymbol{\theta}_{\boldsymbol{\beta}}). \tag{2}$$

Here, $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\boldsymbol{\beta}}, \boldsymbol{\theta}_{\boldsymbol{\pi}})$ comprises all parameters in the tree. This representation of a decision tree allows choosing between different split features and different split functions, by varying the functions f and s, respectively.

In standard deterministic decision trees as proposed in Breiman et al. (1984), the split function is a step function $s(x) = \Theta(x)$ with $\Theta(x) = 1$ if $x > 0$ and $\Theta(x) = 0$ otherwise.

2.2 Probabilistic Decision Tree

A defining characteristic of our method is that the decision tree is probabilistic during training, similar to Kontschieder et al. (2015). Rather than sending a sample deterministically down the right (or left) subtree depending on its features x, we send it right with a probability

$$s(f(x)) = \sigma(f(x)) = \frac{1}{1 + e^{-f(x)}}. \tag{3}$$

This corresponds to regarding each split in the tree as a Bernoulli decision with mean $\sigma(f(x))$. As a result, Eq. 2 is the expected value over the possible outcomes. Figure 1b shows the prediction from Eq. 2 in the probabilistic case for a class $y =$ "red" on the classification problem illustrated in Fig. 1a.

To train our probabilistic decision trees, we choose the empirical log-likelihood of the training data as the maximization objective:

$$\max_{\boldsymbol{\theta}} \mathfrak{L}(\boldsymbol{\theta}; \mathcal{X}_t, \mathcal{Y}_t) = \max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log p(y_n|\boldsymbol{x}_n; \boldsymbol{\theta}). \tag{4}$$

Importantly, while we propose to use a *probabilistic* decision tree for training, we use a *deterministic* decision tree for prediction on test samples. To better match the models used at train and test time, we introduce a hyperparameter γ, which steers the steepness of the split function by scaling the split feature (Montillo et al. 2013)

$$s(f(x)) = \sigma_\gamma(f(x)) = \sigma(\gamma f(x)). \tag{5}$$

Note, for $\gamma \to \infty$ the model resembles a deterministic decision tree, since $\sigma_\infty(f(x)) = \Theta(f(x))$. During training, we iteratively increase γ, akin to a temperature cooling schedule in deterministic annealing (Rose et al. 1990).

2.3 Expectation–Maximization

To optimize the log-likelihood of Eq. 4, we propose a gradient-based, EM-style optimization strategy, which requires f and s to be differentiable with respect to the split parameters $\boldsymbol{\beta}_i$. The derivation of the EM-algorithm for this model follows the spirit of Jordan and Jacobs (1994). We introduce additional latent random variables $z_{n,\ell}$, which indicate that leaf ℓ generated the class label of a given data point $\boldsymbol{x}_n$. Including these latent variables, the optimization objective now becomes the complete data log-likelihood

$$\mathfrak{L}(\boldsymbol{\theta}; \mathcal{X}_t, \mathcal{Y}_t, \mathcal{Z}_t) = \sum_{n=1}^{N} \sum_{\ell=1}^{L} z_{n,\ell} \log\left((\boldsymbol{\pi}_\ell)_{y_n} \mu_\ell(\boldsymbol{x}_n; \boldsymbol{\theta}_{\boldsymbol{\beta}})\right). \tag{6}$$

E-Step In the Expectation-Step, the expected value of the complete-data log-likelihood over the latent variables given the previous parameters $\boldsymbol{\theta}'$ is computed

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}') = E_{\mathcal{Z}_t|\mathcal{X}_t,\mathcal{Y}_t;\boldsymbol{\theta}'}[\mathfrak{L}(\boldsymbol{\theta}; \mathcal{X}_t, \mathcal{Y}_t, \mathcal{Z}_t)]. \tag{7}$$

For this purpose, it is necessary to compute the probability that $z_{n,\ell} = 1$ for each training sample n:

$$h_{n,\ell} := p(z_{n,\ell} = 1 \mid \boldsymbol{x}_n, y_n; \boldsymbol{\theta}') \tag{8}$$

$$= \frac{p(y_n \mid z_{n,\ell} = 1, \boldsymbol{x}_n; \boldsymbol{\theta}') p(z_{n,\ell} = 1 \mid \boldsymbol{x}_n; \boldsymbol{\theta}')}{p(y_n \mid \boldsymbol{x}_n; \boldsymbol{\theta}')} \tag{9}$$

$$= \frac{(\boldsymbol{\pi}'_\ell)_{y_n} \mu_\ell(\boldsymbol{x}_n; \boldsymbol{\theta}'_{\boldsymbol{\beta}})}{\sum_{\ell'=1}^{L} (\boldsymbol{\pi}'_{\ell'})_{y_n} \mu_{\ell'}(\boldsymbol{x}_n; \boldsymbol{\theta}'_{\boldsymbol{\beta}})}. \tag{10}$$

Thus, the expectation value of the complete-data log-likelihood yields

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}') = \sum_{n=1}^{N} \sum_{\ell=1}^{L} h_{n,\ell} \log\left((\boldsymbol{\pi}_\ell)_{y_n} \mu_\ell(\boldsymbol{x}_n; \boldsymbol{\theta}_{\boldsymbol{\beta}})\right). \tag{11}$$

M-Step In the Maximization-Step, the expectation value computed in the E-Step (Eq. 11) is maximized to find the updated parameters $\boldsymbol{\theta}$,

$$\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}'). \tag{12}$$

Due to the latent variables we introduced, it is now possible to separate the parameter dependencies in the logarithm into a sum. As a result, the leaf predictions and split parameters are optimized separately.

The optimization of the leaf predictions including the normalization constraint can be computed directly,

$$(\boldsymbol{\pi}_\ell)_k = \frac{\sum_{n=1}^{N} \mathbb{1}(y_n = k) h_{n,\ell}}{\sum_{n=1}^{N} h_{n,\ell}}. \tag{13}$$

Here, the indicator function $\mathbb{1}(y_n = k)$ equals 1 if $y_n = k$ and 0 otherwise.

The optimization of the split parameters in the M-Step is performed using gradient based optimization. The separated objective for the split parameters without the leaf predictions is

$$\max_{\boldsymbol{\theta}_{\boldsymbol{\beta}}} \sum_{n=1}^{N} \sum_{\ell=1}^{L} h_{n,\ell} \log \mu_\ell(\boldsymbol{x}_n; \boldsymbol{\theta}_{\boldsymbol{\beta}}). \tag{14}$$

In practice we use the first-order gradient-based stochastic optimization Adam (Kingma and Ba 2015) to optimize this objective.

In summary, each iteration of the algorithm requires evaluation of Eqs. 10 and 13, as well as at least one update of the split parameters based on Eq. 14. This iterative algorithm can be applied to a binary decision tree of any given structure.

2.4 Complex Splits and Spatial Regularization

The proposed optimization procedure only requires the split features f to be differentiable with respect to the split parameters. As a result, it is possible to implement more complex splits than axis-aligned or oblique splits. For example, it is possible to use a small convolutional neural network (CNN) as split feature extractor for f and learn its parameters (Sect. 3.4).

Furthermore, the optimization objective can also include regularization constraints on the parameters. This is useful to avoid overfitting and learn more robust patterns. When the inputs are from images, spatial regularization also reveals more discernible spatial structures in the learned parameters without sacrificing accuracy (Sect. 3.3). To encourage the learning of coherent spatial patterns at each split, we add a spatial regularization term (Eilers and Marx 1996)

$$-\lambda \sum_{i=1}^{I} \boldsymbol{\beta}_i^T M \boldsymbol{\beta}_i \tag{15}$$

to the maximization objective of the split features of Eq. 14. Here, matrix M denotes the Laplacian matrix when interpreting the image as a grid graph. For a single pixel,

corresponding to weight β_i, the diagonal element M_{ii} contains the number of neighboring pixels. If pixels i and j are neighboring pixels, then $M_{ij} = M_{ji} = -1$. All remaining elements in M are 0. This regularization term penalizes spatial finite differences, encouraging similar parameters for neighboring pixels. The hyperparameter λ controls the regularization strength, with higher λ leading to stronger regularization.

2.5 Decision Tree Construction

The previous sections outlined how to fit a decision tree to training data, given a fixed tree topology (parameter learning). Additionally to this deterministic decision tree *Finetuning*, we propose a *Greedy* algorithm to construct a tree by successively splitting nodes and optimizing them on subsets of the training data.

As a stopping criterion for training of a single split, we limit the number of training epochs. The size of the tree can be limited either by a maximum number of leaf nodes or the depth of the tree. Furthermore, in cases with a very small subset of the training data, it may happen that training of a split fails and all training samples are passed to a single child of that node. For these cases we set a maximum number of attempts to fit a split function. Step-by-step, the *Greedy* algorithm works as follows:

1. Initialize the decision tree with a single candidate node as the tree root.
2. Split the training data into subsets. Starting from the root node with the entire training dataset, the data is successively decomposed using deterministic routing. As a result, non-overlapping subsets are assigned to the candidate nodes.
3. For each candidate node:
   (a) If the training data subset of the candidate node is pure or the maximum number of attemps has been reached, skip steps 3b to 3d for this node and fix it as a leaf node.
   (b) Replace node with a new tree stump, i.e. one split node and two leaf nodes.
   (c) Optimize only the tree stump using the *Finetune* algorithm (see Sect. 2.3) on the assigned training data subset for the specified number of epochs.
   (d) If training the stump failed, then try training a new stump by repeating from 3a.
4. Find leaf node candidates that may be split according to the specified tree limits.
5. If candidate nodes are found, repeat from 2. Otherwise, stop, the decision tree construction is finished.

The main intent for this greedy algorithm is that trees are only grown further when necessary and thereby reduce the amount of computations and parameters in the model (see Sect. 3.6). The distribution of the training data in step 2 means that each node only shares training data with its ancestors. In particular this means that, at first, the root split is trained on the entire training data. At some point the training data is pure, i.e. all samples are of same class, and this node can be fixed as a leaf node.

During training of a tree stump, only one split node and two leaf nodes are optimized. As a result, the log-likelihood objective (Eq. 4) then resembles an approximation of the widely used information gain criterion Quinlan (1990, 1993) (Sect. 2.6).

After this greedy structure learning, the nodes in the entire resulting tree can be finetuned jointly as described in Sect. 2.3, this time with probabilistic routing of all training data.

2.6 Relation to Information Gain and Leaf Entropies

We now show that maximization of the log-likelihood of the probabilistic decision tree model approximately minimizes the weighted entropies in the leaves. The steeper the splits become, the better the approximation.

To establish this connection we use hyperparameter γ to control the steepness of the probabilistic split function (Eq. 5). We introduce the function $\ell(\boldsymbol{x})$ that returns the index of the leaf that sample $\boldsymbol{x}$ reaches when the path is evaluated deterministically

$$\ell(\boldsymbol{x}) = \sum_{\ell=1}^{L} \ell \lim_{\gamma \to \infty} \mu_\ell(\boldsymbol{x}; \sigma_\gamma, \boldsymbol{\theta}_\beta). \tag{16}$$

This simplifies the log-likelihood objective (Eq. 4) to

$$\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log(\boldsymbol{\pi}_{\ell(\boldsymbol{x}_n)})_{y_n} \tag{17}$$

because each sample reaches only one leaf. Let $N_{\ell,k}$ be the number of training samples in leaf ℓ with class k and $N_\ell = \sum_{k=1}^{K} N_{\ell,k}$ denote all training samples in leaf ℓ. Since training samples with the same class and in the same leaf contribute the same term, the equations may be rearranged to

$$\max_{\boldsymbol{\theta}} \sum_{\ell=1}^{L} \sum_{k=1}^{K} N_{\ell,k} \log(\boldsymbol{\pi}_\ell)_k. \tag{18}$$

With $\gamma \to \infty$, the optimal leaf predictions are the same as in a standard, deterministic decision tree, i.e. $(\boldsymbol{\pi}_\ell)_k = \frac{N_{\ell,k}}{N_\ell}$. Accordingly, the objective can be rewritten as

$$\max_{\boldsymbol{\theta}} \lim_{\gamma \to \infty} \pounds(\boldsymbol{\theta}; \mathcal{X}_t, \mathcal{Y}_t) = \min_{\boldsymbol{\theta}} \sum_{\ell=1}^{L} \frac{N_\ell}{N} H_\ell. \tag{19}$$

Here, $H_\ell = -\sum_{k=1}^{K} (\boldsymbol{\pi}_\ell)_k \log(\boldsymbol{\pi}_\ell)_k$ denotes the entropy in leaf ℓ.

In conclusion, we have shown that for $\gamma \to \infty$, maximizing the log-likelihood objective minimizes a weighted sum of leaf entropies. For the special case of a single split with two leaves, this is the same as maximizing the information gain. Consequently, the log-likelihood objective (Eq. 4) can be regarded as a generalization of the information gain criterion (Quinlan 1990) to an entire tree.

2.7 Decision Forest

Following the ideas introduced by Breiman (2001), we combine decision trees to a decision forest. Specifically, each decision tree is constructed with our *Greedy* algorithm on the full dataset. Afterwards, using our *Finetune* algorithm, each tree is optimized end-to-end. Note that the result is a *decision forest* rather than a *random forest*, since each tree is trained independently on all train data rather instead of on random subsets.

In order to reduce correlation between the decision tree predictions, we train each split function only on a subset of the available features. For each split, this feature subset is sampled from a uniform distribution or, in the case of 2D images, will consist of connected 2D patches of the image.

Let $\boldsymbol{\theta}_t$ denote all parameters of a single tree t out of T trees in the ensemble $\mathcal{T}$ of learned trees. The final prediction of the decision forest for a single sample $\boldsymbol{x}$ is computed as the mean prediction of the single trees (Eq. 2):

$$p(y|\boldsymbol{x}; \mathcal{T}) = \frac{1}{T} \sum_{t \in \mathcal{T}} p(y|\boldsymbol{x}; \boldsymbol{\theta}_t). \tag{20}$$

3 Experiments

We conduct experiments on data from various domains. For quantitative comparison of our end-to-end learned oblique decision trees (*E2EDT*), we evaluate the performance on the multivariate but unstructured datasets used in Norouzi et al. (2015b) (Sect. 3.1). In order to understand the learning process of the probabilistic training and deterministic inference model, we visually examine the models on an image segmentation dataset (Sect. 3.2). Next, we show that the proposed algorithm can learn meaningful spatial features on *MNIST*, *FashionMNIST* and *ISBI*, as has previously been demonstrated in neural networks but not in decision trees (Sect. 3.3). We also demonstrate that a deterministic decision tree with complex split nodes can be trained end-to-end, by using a small neural network in each split node (Sect. 3.4). Further, we quantitatively evaluate the effect of the steepness annealing in an end-to-end learned decision forest (*E2EDF*, Sect. 3.5) and compare the trade-off between computational load and accuracy to state-of-the-art decision forests (Sect. 3.6).

For gradient-based split parameter optimization, we use the Adam optimizer (Kingma and Ba 2015) with default parameters ($\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$) and a batch size of 1000 with shuffled batches. All data is normalized to zero mean and unit variance based on the training data. Unless stated otherwise, we use this setup for all our experiments. The source code of our implementation using PyTorch (Paszke et al. 2017) is available online[1].

3.1 Performance of Oblique Decision Trees

We compare the performance of our algorithm in terms of accuracy to all results reported in Norouzi et al. (2015b). In order to provide a fair comparison, we refrain from using pruning, ensembles and regularization.

Datasets Norouzi et al. (2015b) reports results on the following four datasets: *MNIST* (LeCun et al. 1998), *SensIT* (Duarte and Hu 2004), *Connect4* (Dua and Graff 2017) and *Protein* (Wang 2002). The multi-class classification datasets are obtained from the LIBSVM repository (Fan and Lin 2011). When a separate test set is not provided, we randomly split the data into a training set with 80% of the data and use 20% for testing. Likewise, when no validation set is provided, we randomly extract 20% of the training set as validation set.

Compared algorithms We compare algorithms that use a deterministic decision tree for prediction, with either oblique or axis-aligned splits. The following baselines were evaluated in Norouzi et al. (2015b): *Axis-aligned*: conventional axis-aligned splits based on information gain; *OC1*: oblique splits optimized with coordinate descent as proposed in Murthy (1996); *Random*: selected the best of randomly generated oblique splits based on information gain; *CO2*: greedy oblique tree algorithm based on structured learning (Norouzi et al. 2015a); *Non-greedy*: non-greedy oblique decision tree algorithm based on structured learning (Norouzi et al. 2015b).

We compare the results of these algorithms with two variants of our proposed method. Here, *Greedy E2EDT* denotes a greedy initialization where each oblique split is computed using the EM optimization. For each depth, we apply the *Finetune E2EDT* algorithm to the tree obtained from the

[1] Code is available at http://www.github.com/tomsal/endtoenddecisiontrees

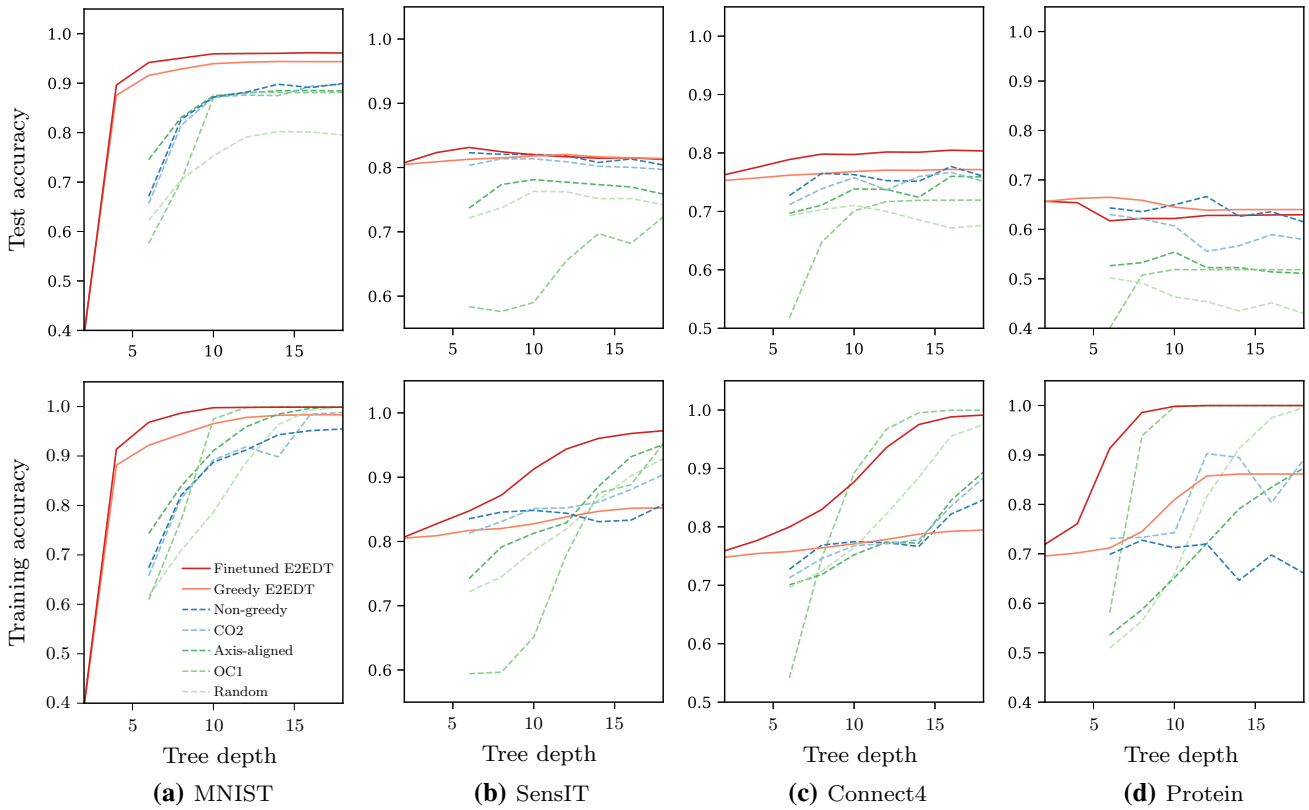

Fig. 2 Accuracy on test and training sets of various optimization methods for deterministic oblique decision trees. For of our proposed approach, we show results with *Greedy* initialization only (solid, light red line), and after being *Finetuned* (solid, dark red line). The results of the baseline algorithms (dashed lines) were reported in Norouzi et al. (2015b), see text for more details. The maximum tree depth varies from 2 to 18 with stepsize 2 (Color figure online)

Greedy E2EDT algorithm at that depth. In the following we refer to them as *Greedy* and *Finetune*.

Hyperparameters and initialization We keep all hyperparameters fixed and conduct a grid search over the number of training epochs in $\{20, 35, 50, 65\}$, using a train/validation split. The test data is only used to report the final performance.

The split steepness hyperparameter is set to $\gamma = 1.0$ initially and increased by 0.1 after each epoch (one epoch consists of the split parameter $\boldsymbol{\theta}_{\boldsymbol{\beta}}$ updates of all training batches as well as the update of the leaf predictions $\boldsymbol{\theta}_{\boldsymbol{\pi}}$). Initial split directions are sampled from the unit sphere and the categorical leaf predictions are initialized uniformly.

Results Figure 2 shows the test and training statistical accuracy of the different decision tree learning algorithms. The accuracy of a classifier is defined as the ratio of correctly classified samples in the respective set. It was evaluated for a single tree at various maximum depths. The red solidlines show the result of our proposed algorithm, the dashed lines represent results reported by Norouzi et al. (2015b).

Our algorithms achieve higher test accuracy than previous work, especially in extremely shallow trees. The highest increase in test accuracy is observed on the *MNIST* data set. Here, we significantly outperform previous approaches for oblique decision trees at all depths. In particular, an oblique decision tree of depth 4 is already sufficient to surpass all competitors.

On *SensIT* and *Protein* we perform better than or on par with the *Non-greedy* approach proposed in Norouzi et al. (2015b). Note that further hyperparameter tuning may reduce overfitting, e.g. on the *Protein* dataset, and thus the results may improve. We did not include this here, as we aimed to provide a fair comparison and show the performance given very little parameter-tuning.

In conclusion, our proposed (*E2EDT*) algorithm is able to learn more accurate deterministic oblique decision trees than the previous approaches.

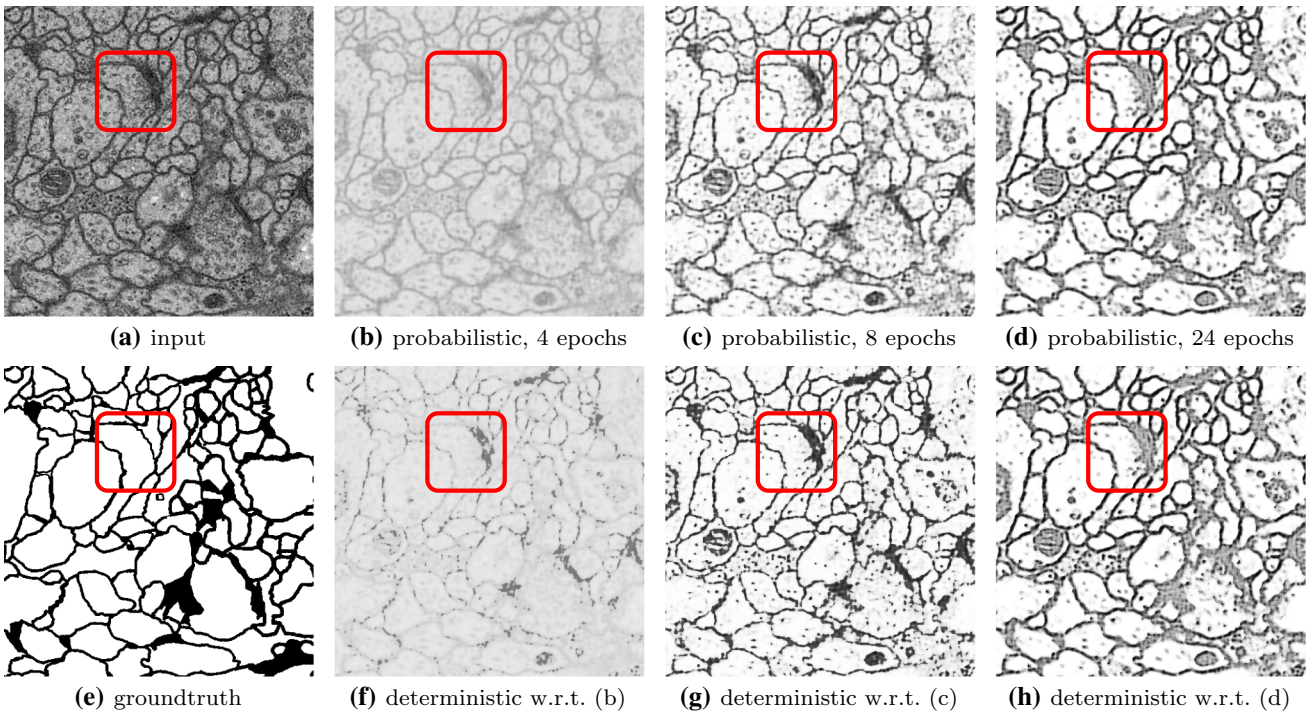

Fig. 3 Posterior probability (Eq. 2) after various epochs of learning a single oblique decision tree on the ISBI binary segmentation dataset. **a** Shows the input image and **e** the corresponding groundtruth labels. **b–d** Illustrate the posterior of the probabilistic tree as γ increases at various stages during training. **f–h** illustrate the posterior of the deterministic equivalents at those stages after setting $\gamma \to \infty$. Except for (**a**), darker means higher probability for class "membrane". Note how discrepancies between the predictions of the probabilistic tree and its deterministic counterpart disappear as steepness γ increases. The highlighted region is discussed in the text

3.2 Visual Convergence of Training and Inference Model

During training, we gradually steer the probabilistic training model towards a deterministic model by increasing the steepness γ. We now visually examine the difference between the probabilistic training model and the deterministic inference model. For this purpose, we train an oblique decision tree for a binary image segmentation task on the ISBI challenge dataset (Cardona et al. 2010). This challenging image segmentation benchmark comprises serial section Transmission Electron Microscopy images (Fig. 3a) and binary annotations of neurons and membranes (Fig. 3e). For every pixel, we take a 9×9 window around the current pixel as input features to an oblique decision tree. Consequently, the learned parameters at each split node can be regarded as a spatial kernel. We initialize a balanced oblique decision tree of depth 6 and use the *Finetune* algorithm to optimize the entire tree. We use the default steepness increase of $\Delta\gamma = 0.1$ per epoch.

Results Figure 3a shows a sample image of the input and Fig. 3e the correspondinggroundtruth labels. Figure 3b–d illustrate the posterior probability (Eq. 2) predicted by the probabilistic training model at different training stages. The posterior probabilities of the corresponding inference models are shown below, in Fig. 3f–h. The visualization of the prediction shows pixels more likely to be of class "membrane" with darker color.

The gradual convergence of the probabilistic training model and the deterministic inference model is well visible. After 4 epochs, the probabilistic model (Fig. 3b) already reflects the structure of the segmentation problem. The corresponding deterministic model is more fragmented and also exhibits stronger confidence values, i.e. darker pixels. This is especially clear to see in the highlighted red squares in Fig. 3. The discrepancy between the probabilistic and deterministic model is reduced after 8 epochs (Fig. 3c, g). However, the deterministic posterior estimate reveals high confidence even in misclassified areas (higher contrast). This effect is dampend after 24 epochs. The corresponding leaf probabilities now learned to estimate confidence and thus show more gray areas in difficult regions. The differences between the probabilistic (Fig. 3d) and the deterministic (Fig. 3h) predictions are hardly visible anymore.

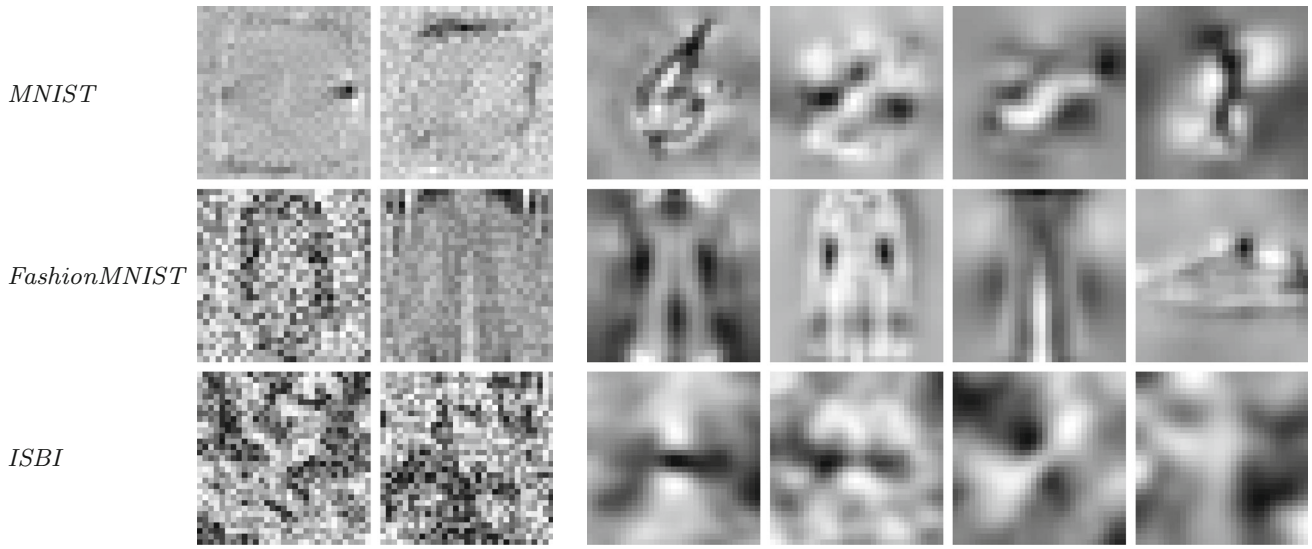

Fig. 4 Visualizations of oblique split parameters learned with and without spatial regularization (Sect. 2.4). Each row shows a selection of parameters which were learned on a different dataset, namely *MNIST* (LeCun et al. 1998), *FashionMNIST* (Xiao et al. 2017), and *ISBI* (Cardona et al. 2010). Parameters trained with spatial regularization show visible structures and patterns, whereas parameters learned without regularization appear noisy

3.3 Interpretation of Spatially Regularized Parameters

We now investigate the effects of spatial regularization (Sect. 2.4) on the parameters of oblique decision trees learned with our algorithm. Recall that regularization penalizes differences in adjacent parameters. For this purpose, we train oblique decision trees on the *MNIST* digit dataset (LeCun et al. 1998), the *FashionMNIST* fashion product dataset (Xiao et al. 2017) and the *ISBI* image segmentation dataset (Cardona et al. 2010). For *MNIST* and *FashionMNIST*, the training images consist of 28×28 images. For the segmentation task on *ISBI*, a sliding window of size 31×31 is used as input features for each pixel in the center of the window.

Results In Fig. 4 we visualized selected parameters of the oblique splits at various depths with and without regularization. The learned parameter vectors are reshaped to the respective training image dimensions, and linearly normalized to the full grayscale range. In both cases, we select parameter vectors that display interesting visible structures.

The parameters without regularization appear very noisy. In contrast, with regularization the algorithm learns smoother parameter patterns, without decreasing the accuracy of the decision trees. The patterns learned on the *MNIST* show visible sigmoidal shapes and even recognizable digits. On the *FashionMNIST* dataset, the regularized parameters display the silhouettes of coats, pants and sneakers. Likewise, our algorithm is able to learn the structures of membranes on the real-world biological electron microscopy images from the *ISBI* dataset.

Figure 5 visualizes half of a learned tree for the *FashionMNIST* dataset. One can see that at deeper splits, more distinctive features of various fashion products are tested. The split parameters of the first split at depth 3 show bright trousers and its right child predicts the class "trousers". The same holds for the second split at depth 3, showing a bright silhouette of a dress and its right child predicts "dress". The parameters of the third split at depth 3 reveal some kind of upper body clothes, but it is difficult to determine the kind. Yet, these parameters separate samples of class "pullover" and "shirt" (left child) from class "coat" (right child). Such decision tree illustrations thus reveal important features that drive the internal decisions made towards the final prediction. This provides a useful tool to interpret our model, akin to the use of decision trees in expert systems of other domains (De Ville 2006; Huang et al. 2015).

3.4 CNN Split Features

We test the effectiveness of CNNs as split features in a decision tree on *MNIST*. At each split, we trained a very simple CNN of the following architecture: Convolution 5×5 kernel @ 3 output channels $\rightarrow$ Max Pool $2 \times 2 \rightarrow$ ReLU $\rightarrow$ Convolution 5×5 @ $6 \rightarrow$ Max Pool $2 \times 2 \rightarrow$ ReLU $\rightarrow$ Fully connected layer $96 \times 50 \rightarrow$ ReLU $\rightarrow$ Fully connected layer 50×1. The final scalar output is the split feature, which is the input to the split function.

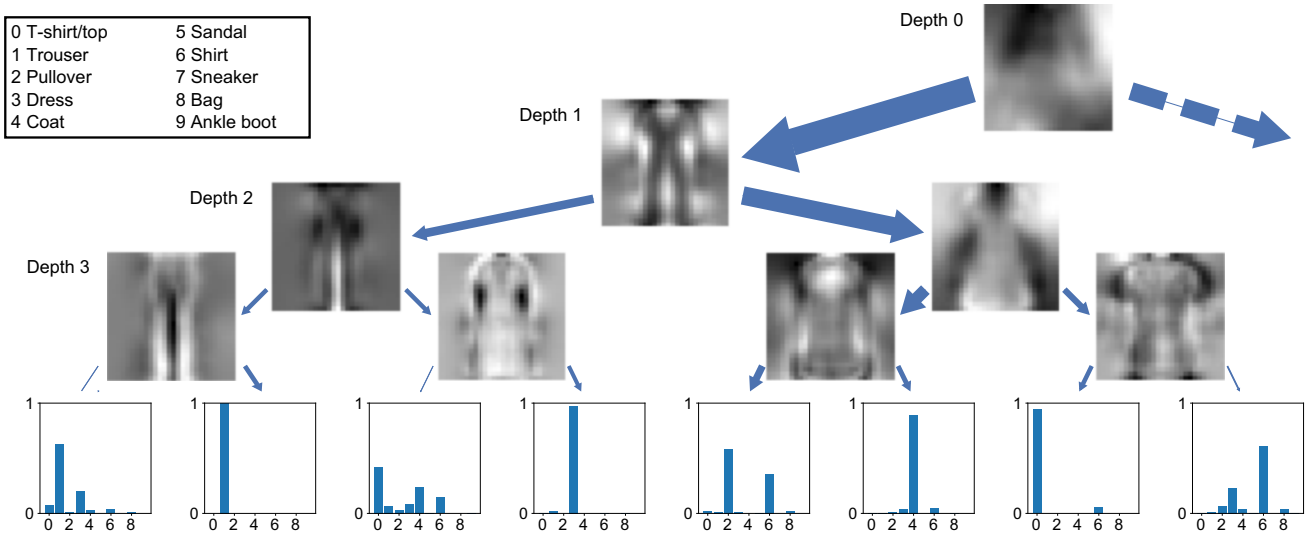

Fig. 5 Visualization of an oblique decision tree learned on FashionMNIST (Xiao et al. 2017) with spatial regularization. The right branch of the root (dashed arrow) is hidden for clarity. The split parameters are visualized as in Fig. 4. Intuitively, the better the input image matches the parameter image, the more likely the sample will go to the right child. If the input image better resembles the negative parameter image, the sample will go to the left. The thickness of an arrow indicates the number of training samples following the path when the decision tree is evaluated deterministically. Distributions at the leaves are visualized as bar plots. The x-axis denotes classes (see legend in the top-left), and the y-axis corresponds to the class probability

Again, we train greedily to initialize the tree, however we split nodes in a best-first manner, based on highest information gain. As a result, the trees can be fairly unbalanced despite impure leaves. We now choose to stop at a maximum of 10 leaves, as we aim to increase interpretability and efficiency by having one expert leaf per class.

Results In this setting, a single decision tree achieves a test accuracy of $98.2\% \pm 0.3\%$ deterministic evaluation of nodes. For comparison, a standard random forest ensemble with 100 trees only reaches $96.79\% \pm 0.07\%$.

Such decision tree models provide interesting benefits in interpretability and efficiency, which are the main advantages of decision trees. When a sample was misclassified it is straightforward to find the split node that is responsible for the error. This offers interpretability as well as the possibility to improve the overall model. Other methods, such as *OneVsOne* or *OneVsRest* multi-class approaches, provide similar interpretability, however at a much higher cost at test time. This is due to the fact that in a binary decision tree with K leaves, i.e. a leaf for each class, it is sufficient to evaluate $\mathcal{O}(\log K)$ split nodes. In *OneVsOne* and *OneVsAll* it is necessary to evaluate $K(K-1)/2$ and K different classifiers at test time, respectively.

3.5 Steepness Annealing Analysis

In Sect. 2, we motivate the introduction and annealing of the steepness hyperparameter based on two observations. Firstly, steeper decisions, although hindering end-to-end learning, reflect our final inference model more closely. Secondly, in the limit of steep decisions, our learning objective approximates the information gain (see Sect. 2.6), which is well established for decision tree learning.

In this experiment, we investigate the effectiveness of annealing the steepness hyperparameter. For this purpose, we train decision forest ensembles of oblique deterministic decision trees (see Sect. 2.7). We use different annealing schemes for the steepness to study the impact on the performance. The steepness is always initialized as $\gamma = 1.0$ and $\Delta\gamma > 0$ denotes the enforced increase in steepness after each epoch. Thus, $\Delta\gamma = 0$ effectively ignores the hyperparameter as it will stay constant during training. We perform this comparison for three different settings of the number of epochs (15, 30, 45). This means that during the *Greedy* tree construction each split is trained for exactly this number of epochs. Afterwards, each tree is optimized end-to-end based on our *Finetune* algorithm for three times as many epochs as during the construction phase (e.g. 30 epochs *Greedy* and 90 epochs *Finetune* training). This choice is motivated by validation experiments which showed the importance of the *Finetune* algorithm in the decision forest and do not affect the comparison of different $\Delta\gamma$.

Datasets We follow the procedure described in section 5.1 of Kontschieder et al. (2015), and use the same datasets, number of features, and number of trees as they do. These datasets are *Letter* (Frey and Slate 1991), *USPS* (Hull 1994) and *MNIST* (LeCun et al. 1998). The features are randomly chosen for each split separately. For completeness, details on the dataset and specific settings are listed in Table 2 in the Appendix.

Table 1 Comparison of the validation accuracy of our end-to-end learned deterministic decision forests for different values of the gradual steepness increase $\Delta\gamma$ on various datasets (see Appendix, Table 2)

Dataset	$\Delta\gamma$	15 Epochs	30 Epochs	45 Epochs
Letter	0.0	76.7	81.1	84.7
Letter	0.01	81.8	89.2	92.5
Letter	0.1	**92.6**	**94.5**	**95.5**
USPS	0.0	88.5	92.3	94.1
USPS	0.01	91.5	95.3	96.3
USPS	0.1	**96.1**	**96.6**	**96.7**
MNIST	0.0	**97.9**	**98.1**	**98.1**
MNIST	0.01	**97.9**	**98.1**	**98.1**
MNIST	0.1	97.7	97.8	97.4

Best results per dataset and number of training epochs are highlighted bold

Results Table 1 shows the accuracy of our oblique decision forest when trained with different steepness increase. It is important to note that $\Delta\gamma = 0$ does not mean that the steepness is fixed throughout training. The model may still learn to have steeper decisions by increasing the L2-norm of the split parameters $\boldsymbol{\theta}_\beta$. After training for same number of epochs, a steepness increase of $\Delta\gamma = 0.1$ per epoch consistently outperforms the trained models without annealing $\Delta\gamma = 0.0$ on the *Letter* and *USPS* datasets. On these two datasets, final deterministic model improves by a large margin when trained with steepness increase. Interestingly, on *MNIST*, the decision forests without steepness increase perform equally well as the corresponding models with $\Delta\gamma = 0.01$. A possible interpretation is that larger datasets do not benefit from this. Compared to the other datasets and considering the simplicity of the task, *MNIST* can be considered a large dataset. Conversely, further tuning of γ may show different results. Generally, our default steepness annealing choice of $\Delta\gamma = 0.1$ per epoch performs well.

3.6 Trade-off Between Computational Load and Accuracy

Due to the conditional data flow, deterministic decision forests only evaluate a fraction of the entire model during prediction and thus require significantly less computations than a probabilistic forest model. We now quantify the trade-off in computational load and accuracy of our end-to-end learned deterministic decision forests compared to the state-of-the-art probabilistic *shallow Neural Decision Forests* (sNDF) by Kontschieder et al. (2015). For this purpose, we evaluate our decision forest (*E2EDF*) on the same datasets which were used in their evaluation (see Sect. 3.5). Both models, *E2EDF* and *sNDF*, are based on oblique splits and we use the same maximum depth per tree and the same number of trees in a forest as the *sNDF*.

We additionally compare our results to other deterministic tree ensemble methods: the standard random forest (*RF*), boosted trees (*BT*) and alternating decision forests (*ADF*). The corresponding results were reported by Schulter et al. (2013) and are always based on 100 trees in the ensemble with maximum depth of either 10, 15 or 25. Since their work only lists ranges of explored parameter settings, we will base the estimated computational load (i.e. number of split evaluations) on the most favorable parameter settings.

Since *BT*, *RF* and *ADF* are in practice limited to linear split functions, we restrict our *E2EDF* models to oblique splits as well in this comparison. To train our *E2EDF* models, we use our default steepness increase of $\Delta\gamma = 0.1$ per epoch. On *USPS* as well as *Letter*, the models are trained for 45 epochs, whereas on *MNIST*, training is done only for 15 epochs due to the larger amount of training data. Note that, as in Sect. 3.5, we *Finetune* the final tree for three times as many epochs as during the *Greedy* training (e.g. for *USPS*: 45 epochs *Greedy* and 135 epochs Finetune). Training is done on the full training data, i.e. including validation data, and evaluate on the provided test data. The reported accuracy is averaged over three runs.

Results The trade-off in terms of computational load and accuracy of the different decision forest models is shown in Fig. 6. We find that deterministic *E2EDF* achieves higher average accuracy than *RF* and *BT* on all datasets, and outperforms all other methods on *MNIST*. Compared to *ADF*, the results of *E2EDF* are competitive, although relative performance varies between datasets. A possible explanation is that the *ADF* results were obtained using different hyperparameters that allow more and deeper trees, which can lead to significant differences as shown in Schulter et al. (2013).

On *Letter* and *USPS*, *sNDF* achieves higher accuracy but at several orders of magnitude higher computational cost as it lacks the conditional data flow property. In fact, a single tree in the *sNDF* requires a total of 1023 split evaluations, which is more than for our entire forest models, namely up to 1000 evaluations on USPS. A complete overview of the number split function evaluations per algorithm is given in Table 3 in the Appendix.

Figure 6 further presents the impact of using fewer decision trees in our forest model by illustrating the performance of small ensembles ($T \in \{1, 3, 5, 10\}$). On *MNIST* and *USPS* we observe that even smaller *E2EDF* ensembles with only $T = 10$ trees already obtains competitive accuracy.

Finally, we note that due to our greedy initialization of trees, the actual number of splits is less than the maximum depth would allow. The trained *E2EDF* trees only required on average 758 ± 1 (*Letter*), 372 ± 2 (*USPS*) and 938 ± 2 (*MNIST*) split functions, while the maximum number of split

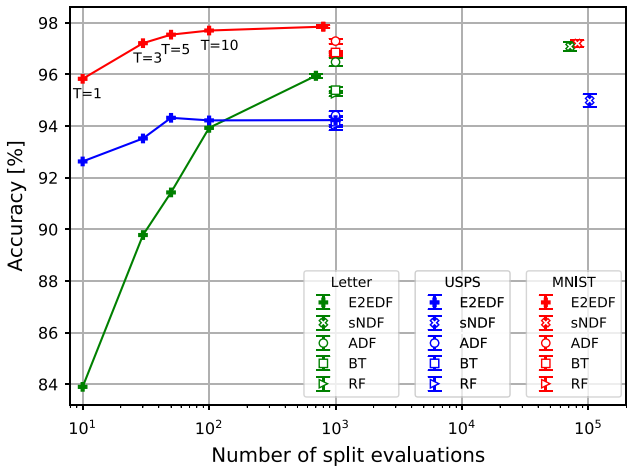

Fig. 6 Trade-off between computational load and accuracy of oblique decision forest models. The computational load is represented by the number split function evaluations (x-axis, log scale) in the forest required for a single sample at test time. For *RF*, *BT* and *ADF*, the split function evaluations are estimated in favor of those methods. For our *E2EDF* models, we show results with the same number of trees as used by *sNDF*, and additionally include results with fewer trees in the forest, namely $T \in \{1, 3, 5, 10\}$ (Color figure online)

decisions for a tree of depth 10 is $2^{10} - 1 = 1023$. Overall, having fewer trees and fewer decisions in the forest reduces the required number of split evaluations at test time, and thus enables even more efficient inference.

4 Conclusion

We presented a new approach to train deterministic decision trees with gradient-based optimization in an end-to-end manner, *E2EDT*. The approach uses a probabilistic tree formulation during training to facilitate back-propagation and optimize all splits of a tree jointly.

We found that by adjusting the steepness of the decision boundaries in an annealing scheme, the method learns increasingly more crisp trees that capture uncertainty as distributions at the leaf nodes, rather than as distributions over multiple paths. The resulting optimized trees are therefore deterministic rather than probabilistic, and run efficiently at test time as only a single path through the tree is evaluated. This approach outperforms previous training algorithms for oblique decision trees. In a forest ensemble, our method shows competitive or superior results to the state-of-the-art *sNDF*, even though our trees only evaluate a fraction of the split functions at test time. Unlike *ADF*, we are not restricted to only use oblique split functions, thanks to the gradient-based optimization. We show that it is straightforward to include more complex split features, such as convolutional neural networks, or to add spatial regularization constraints. Another demonstrated benefit is that the learned decision tree can also help interpret how the decision of a visual classification tasks is constructed from a sequence of simpler tests on visual features.

Future work can proceed in various directions. First, alternatives for the annealing scheme could be explored, e.g. the changes in the steepness of tree splits might be adjusted dynamically rather than in a fixed schedule. Second, we have so far only optimized each tree independently, but potentially optimizing and refining the whole forest jointly could yield further improvements, similar to *ADF* and *sNDF*.

Overall, the presented approach provides high flexibility and the potential for accurate models that maintain interpretability and efficiency due to the conditional data flow.

Appendix

Table 2 lists the dataset specific parameters used in the experiments of Sects. 3.5 and 3.6. These parameters are the same as in Kontschieder et al. (2015).

Table 2 Properties of the datasets used in our decision forest experiments, respectively published by Frey and Slate (1991), Hull (1994), LeCun et al. (1998)

Dataset	Letter	USPS	MNIST
Features	16	16×16	28×28
Classes	26	10	10
No. training samples	16,000	7291	60,000
No. test samples	4000	2007	10,000
No. validation samples	3000	1500	10,000
Features per split	8	10×10	15×15
Trees	70	100	80

The lower part describes our processing of the data. We also list the size of our validation data that was randomly taken from the training data. Features per split indicate the number of features that are randomly sampled to find a split in a tree. In the case of 2D inputs, these random features are 2D patches located randomly at different positions of the input. The final row lists the number of trees in our forest on each dataset, which are taken equal to the number of trees used in Kontschieder et al. (2015)

Table 3 Comparison of the computational load of the models evaluated in the experiments

	Letter	USPS	MNIST
sNDF (single tree)	1023	1023	1023
sNDF (forest)	71,610	102,300	81,840
RF, BT, ADF (forest)	≤ 1000	≤ 1000	≤ 1000
E2EDF (single tree)	≤ 10	≤ 10	≤ 10
E2EDF (forest)	≤ 700	≤ 1000	≤ 800

We show the number of oblique splits that need to be evaluated in each model per prediction of a single sample. Probabilistic trees in *sNDF* evaluate every split function in the tree and thus requires $2^{D_{\max}} - 1$ dot products per sample and tree. In deterministic trees (*RF*, *BT*, *ADF* and *E2EDF*), the number of split function evaluations grows linearly with increasing depth. Due to the tree construction, trees may not always reach the maximum depth. Here we report the worst case, but assuming the most favorable maximum tree depth for *RF*, *BT* and *ADF*

Table 3 lists the number of the split function evaluations for the methods discussed in Sect. 3.6.

References

Barros, R. C., Basgalupp, M. P., De Carvalho, A. C., & Freitas, A. A. (2012). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(3), 291–312.

Bolukbasi, T., Wang, J., Dekel, O., & Saligrama, V. (2017). Adaptive neural networks for fast test-time prediction. arXiv:1702.07811

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324.

Breiman, L., Friedman, J., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Boca Raton: Chapman & Hall/CRC.

Cardona, A., Saalfeld, S., Preibisch, S., Schmid, B., Cheng, A., Pulokas, J., et al. (2010). An integrated micro- and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy. *PLOS Biology*, *8*(10), 1–17. https://doi.org/10.1371/journal.pbio.1000502.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE computer society conference on computer vision and pattern recognition*.

Cordts, M., Rehfeld, T., Enzweiler, M., Franke, U., & Roth, S. (2017). Tree-structured models for efficient multi-cue scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(7), 1444–1454.

Criminisi, A., & Shotton, J. (2013). *Decision forests for computer vision and medical image analysis*. Berlin: Springer.

De Ville, B. (2006). *Decision trees for business intelligence and data mining: Using SAS enterprise miner*. Cary: SAS Institute.

Dollár, P., Appel, R., Belongie, S., & Perona, P. (2014). Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(8), 1532–1545.

Dua, D., & Graff, C. (2017). UCI machine learning repository. Retrieved February 18, 2019 from http://archive.ics.uci.edu/ml.

Duarte, M. F., & Hu, Y. H. (2004). Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, *64*(7), 826–838.

Eilers, P. H. C., & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. *Statistical Science*, *11*, 89–121.

Fan, R. E., & Lin, C. J. (2011). Libsvm data: Classification, regression and multi-labe. Retrieved May 30, 2017 from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

Fernández-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, *15*, 3133–3181.

Frey, P. W., & Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, *6*(2), 161–182.

Frosst, N., & Hinton, G. (2017). Distilling a neural network into a soft decision tree. arXiv:1711.09784.

Gall, J., & Lempitsky, V. (2009). Class-specific hough forests for object detection. In *2009 IEEE computer society conference on computer vision and pattern recognition* (pp. 1022–1029). https://doi.org/10.1109/CVPR.2009.5206740.

Guh, R. S., Wu, T. C. J., & Weng, S. P. (2011). Integrating genetic algorithm and decision tree learning for assistance in predicting in vitro fertilization outcomes. *Expert Systems with Applications*, *38*(4), 4437–4449. https://doi.org/10.1016/j.eswa.2010.09.112.

Hehn, T. M., & Hamprecht, F. A. (2018). End-to-end learning of deterministic decision trees. In *German conference on pattern recognition* (pp. 612–627). Berlin, Springer.

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., & Weinberger, K. (2018). Multi-scale dense networks for resource efficient image classification. In *International conference on learning representations (ICLR)*.

Huang, G. M., Huang, K. Y., Lee, T. Y., & Weng, J. T. Y. (2015). An interpretable rule-based diagnostic classification of diabetic nephropathy among type 2 diabetes patients. *BMC Bioinformatics*, *16*(1), S5.

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *16*(5), 550–554.

Hyafil, L., & Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, *5*(1), 15–17.

Ioannou, Y., Robertson, D., Zikic, D., Kontschieder, P., Shotton, J., Brown, M., & Criminisi, A. (2016). Decision forests, convolutional networks and the models in-between. arXiv:1603.01250.

Jordan, M. I. (1994). A statistical approach to decision tree modeling. In *Proceedings of the seventh annual conference on computational learning theory*, New York, NY, USA, COLT '94 (pp. 13–20).

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*(2), 181–214. https://doi.org/10.1162/neco.1994.6.2.181.

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Kontschieder, P., Fiterau, M., Criminisi, A., & Rota Bulò S. (2015). Deep neural decision forests. In *ICCV*.

Kontschieder, P., Kohli, P., Shotton, J., & Criminisi, A. (2013). Geof: Geodesic forests for learning coupled predictors. In *2013 IEEE computer society conference on computer vision and pattern recognition*.

Laptev, D., & Buhmann, J. M. (2014). Convolutional decision trees for feature learning and segmentation. In *German Conference on Pattern Recognition* (pp. 95–106). Springer, Berlin.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lepetit, V., Lagger, P., & Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *2005 IEEE computer society conference on computer vision and pattern recognition* (vol. 2, pp. 775–781 vol. 2). https://doi.org/10.1109/CVPR.2005.288.

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. arXiv:1405.0312.

McGill, M., & Perona, P. (2017). Deciding how to decide: Dynamic routing in artificial neural networks. In Precup, D., & Teh, Y.W. (Eds.) *Proceedings of the 34th international conference on machine learning, PMLR, International Convention Centre, Sydney, Australia, Proceedings of Machine Learning Research* (vol. 70, pp. 2363–2372).

Menze, B. H., Kelm, B. M., Splitthoff, D. N., Koethe, U., & Hamprecht, F. A. (2011). On oblique random forests. Springer (pp. 453–469).

Montillo, A., Tu, J., Shotton, J., Winn, J., Iglesias, J., Metaxas, D., & Criminisi, A. (2013). Entanglement and differentiable information gain maximization. In *Decision forests for computer vision and medical image analysis*, Chapter 19 (pp. 273–293). Springer.

Murthy, K. V. S. (1996). *On growing better decision trees from data*. Ph.D. thesis, The Johns Hopkins University.

Norouzi, M., Collins, M. D., Fleet, D. J., & Kohli, P. (2015a). Co2 forest: Improved random forest by continuous optimization of oblique splits. arXiv:1506.06155.

Norouzi, M., Collins, M. D., Johnson, M., Fleet, D. J., & Kohli, P. (2015b). Efficient non-greedy optimization of decision trees. In *NIPS*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Pinhas-Hamiel, O., Hamiel, U., Greenfield, Y., Boyko, V., Graph-Barel, C., Rachmiel, M., et al. (2013). Detecting intentional insulin omission for weight loss in girls with type 1 diabetes mellitus. *International Journal of Eating Disorders*, *46*(8), 819–825. https://doi.org/10.1002/eat.22138.

Quinlan, J. R. (1990). Induction of decision trees. In Shavlik, J. W., Dietterich, T. G. (Eds.), *Readings in machine learning, Morgan Kaufmann, originally published in Machine Learning* 1:81–106, 1986.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Richmond, D., Kainmueller, D., Yang, M., Myers, E., & Rother, C. (2016). Mapping auto-context decision forests to deep convnets for semantic segmentation. In Richard C Wilson, E. R. H., Smith, W. A. P. (Eds.), *Proceedings of the British machine vision conference (BMVC)*, BMVA Press (pp. 144.1–144.12). https://doi.org/10.5244/C.30.144.

Rose, K., Gurewitz, E., & Fox, G. C. (1990). Statistical mechanics and phase transitions in clustering. *Physics Review Letters*, *65*, 945–948. https://doi.org/10.1103/PhysRevLett.65.945.

Rota Bulo, S., & Kontschieder, P. (2014). Neural decision forests for semantic image labelling. In *2014 IEEE computer society conference on computer vision and pattern recognition*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y.

Schulter, S., Wohlhart, P., Leistner, C., Saffari, A., Roth, P. M., & Bischof, H. (2013). Alternating decision forests. In *2013 IEEE computer society conference on computer vision and pattern recognition* (pp. 508–515). https://doi.org/10.1109/CVPR.2013.72.

Sethi, I. K. (1990). Entropy nets: From decision trees to neural networks. *Proceedings of the IEEE*, *78*(10), 1605–1613.

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., & Blake, A. (2011). Real-time human pose recognition in parts from single depth images. In *2011 IEEE computer society conference on computer vision and pattern recognition* (pp. 1297–1304). https://doi.org/10.1109/cvpr.2011.5995316.

Suárez, A., & Lutsko, J. F. (1999). Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis Machine Intelligence*, *21*(12), 1297–1311.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *2001 ieee computer society conference on computer vision and pattern recognition* (p. 511). IEEE.

Wang, J. Y. (2002). *Application of support vector machines in bioinformatics*. Master's thesis, National Taiwan University, Department of Computer Science and Information Engineering.

Wang, S., Aggarwal, C., & Liu, H. (2017). Using a random forest to inspire a neural network and improving on it. In *Proceedings of the 2017 SIAM international conference on data mining* (pp. 1–9). SIAM.

Wang, X., Yu, F., Dou, Z. Y., Darrell, T., & Gonzalez, J. E. (2018). Skipnet: Learning dynamic routing in convolutional networks. In *The European conference on computer vision (ECCV)*.

Welbl, J. (2014). Casting random forests as artificial neural networks (and profiting from it). In *GCPR*.

Worachartcheewan, A., Nantasenamat, C., Isarankura-Na-Ayudhya, C., Pidetcha, P., & Prachayasittikul, V. (2010). Identification of metabolic syndrome using decision tree analysis. *Diabetes Research and Clinical Practice*, *90*(1), e15–e18.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747.

Zhang, L., Varadarajan, J., Nagaratnam Suganthan, P., Ahuja, N., & Moulin, P. (2017). Robust visual tracking using oblique random forests. In *2017 IEEE computer society conference on computer vision and pattern recognition* (pp. 5589–5598). IEEE.