

End-to-end Learning of Deterministic Decision Trees

Thomas Hehn Fred A. Hamprecht
HCI/IWR, Heidelberg University

{thomas.hehn, fred.hamprecht}@iwr.uni-heidelberg.de

Abstract

Conventional decision trees have a number of favorable properties, including interpretability, a small computational footprint 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, and to learn from scratch those features that best allow to solve a given supervised learning problem. Recent work (Kontschieder 2015) has 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 propose a model and Expectation-Maximization training scheme for decision trees that are fully probabilistic at train time, but after a deterministic annealing process become deterministic at test time. We also analyze the learned oblique split parameters on image datasets and show that Neural Networks can be trained at each split node. In summary, we present the first end-to-end learning scheme for deterministic decision trees and present results on par with or superior to published standard oblique decision tree algorithms.

1. Introduction

When selecting a supervised machine learning technique, we are led by multiple and often conflicting criteria. These include: how accurate is the resulting model? How much training data is needed to achieve a given level of accuracy? How interpretable is the model? How big is the computational effort at train time? And at test time? How well does the implementation map to the available hardware?

These days, neural networks have superseded all other approaches in terms of achievable accuracy of the predictions; but state of the art networks are not easy to interpret, are fairly hungry for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 rules out their use on small embedded devices. Decision trees achieve inferior accuracy, but are fundamentally more frugal.

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 being evaluated for every input; while in a 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.

It is this sparsity of the sample-dependent computational graph that piques our interest in decision trees; but we also hope to profit from their ability to learn their comparatively few parameters from a small training set, and their relative interpretability.

One hallmark of neural networks is their ability to learn a complex combination of many elementary decisions jointly, by end-to-end training using backpropagation. This is a feature that has so far been missing in deterministic decision trees, which are usually constructed greedily without subsequent tuning. We here propose a mechanism to remedy this deficit.

1.1. Contributions

- We propose a decision tree whose internal nodes are probabilistic and hence differentiable at train time. As a consequence, we are able to train the internal nodes jointly in an end-to-end fashion. This is true for linear nodes, but the property is maintained for more complex nodes, such as small Convolutional Neural Networks (CNNs) (section 3.4).
- We derive an expectation-maximization style algorithm for finding the optimal parameters in a split node (section 3.3). We develop a probabilistic split criterion that generalizes the long-established information gain [23]. 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 (section 3.2).
- We demonstrate good results by making the nodes deterministic at test time, sending each sample along a

unique path of only $\mathcal{O}(\log I)$ out of the I inner nodes in a tree. We evaluate the performance of the proposed method on the same datasets as used in related work [22] (section 4.1) and find steeper learning curves with respect to tree depth, as well as higher overall accuracy. We show the benefit of regularizing the spatial derivatives of learned features when samples are images or image patches (section 4.2). Finally, we report preliminary experiments with minimalistic trees with CNNs as split feature.

2. Related work

Decision trees and decision tree ensembles, such as random forests [1], are widely used for computer vision [4] and have proven effective on a variety of classification tasks [7]. In order to improve their performance for a specific task, it is common practice to engineer its features to a specific task [8, 14, 16]. Oblique linear and non-linear classifiers using more than one feature at a time have been benchmarked in [18], but the available algorithms are, in contrast to our approach, limited to binary classification problems.

There have been several attempts to train decision trees using gradient optimization techniques for more complex split functions. Similarly to our approach, [19] have successfully approximated information gain using a sigmoid function and a smoothness hyperparameter. However, that approach does not allow joint optimization of an entire tree.

In [22], the authors also 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. In order to find an initial tree structure, the work also relies on a greedy algorithm, which is based on the same upper bound approach [21]. Their method is restricted to linear splits and relies on the kernel trick to introduce higher order split features as opposed to our optimization, which allows more complex split features.

Other advances towards gradient-based decision tree optimization rely on either fuzzy or probabilistic split functions [28, 13, 10]. In contrast to our approach, the assignment of a single sample to the leaves remains fuzzy, respectively probabilistic, during prediction. Consequently, all leaves and paths need to be evaluated for every sample, which annihilates the computational benefits of trees.

We build on reference [13], which is closest to our work. The authors use sigmoid functions to model the probabilistic routes and employ the same log-likelihood objective. In contrast to their work, we derive the alternating optimization using the Expectation-Maximization approach as in [11] and aim for a deterministic decision tree for prediction. Also, they start from a random, but balanced tree, because their algorithm does not learn the structure of the

tree.

Finally, connections between neural networks and decision tree ensembles have been examined. In [27, 29] decision tree ensembles are cast 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 random forest. Subsequently, [25] cast stacked decision forests to convolutional neural networks and found an approximate mapping back. In [9, 17] several models of neural networks with separate, conditional data flows are discussed.

Our work builds on various ideas of previous work, however, none of these algorithms learn deterministic decision trees with arbitrary split functions in an end-to-end fashion.

3. Methods

Consider a classification problem with input space $\mathcal{X} \subset \mathbb{R}^p$ and output space $\mathcal{Y} = \{1, \dots, K\}$. The training set is defined as $\{\mathbf{x}_1, \dots, \mathbf{x}_N\} = \mathcal{X}_t \subset \mathcal{X}$ with corresponding classes $\{y_1, \dots, y_N\} = \mathcal{Y}_t \subset \mathcal{Y}$. We propose training a probabilistic decision tree model, which becomes deterministic at test time.

3.1. Standard decision tree and notation

In binary decision trees (figure 1c), split functions $s : \mathbb{R} \rightarrow [0, 1]$ 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 $i \in \{1, \dots, I\}$ computes a split feature from a sample, and sends that feature to into a split function. That function is a map $f_{\beta_i} : \mathbb{R}^p \rightarrow \mathbb{R}$ parametrized by β_i . For example, *oblique splits* are a linear combination of the input as in $f_{\beta_i}(\mathbf{x}) = (\mathbf{x}^T, 1) \cdot \beta_i$ with $\beta_i \in \mathbb{R}^{p+1}$. Similarly, an axis-aligned split perpendicular to axis a is represented by an oblique split whose only non-zero parameters are at index a and $p + 1$. We write $\theta_{\beta} = (\beta_1, \dots, \beta_I)$ to denote the collection of all split parameters in the tree.

Leaf nodes. Each leaf $\ell \in \{1, \dots, L\}$ stores a categorical distribution over classes $k \in \{1, \dots, K\}$ in a vector $\pi_{\ell} \in [0, 1]^K$. These vectors are normalized such that the probability of all classes in a leaf sum to $\sum_{k=1}^K (\pi_{\ell})_k = 1$. We define $\theta_{\pi} = (\pi_1, \dots, \pi_L)$ to include all leaf parameters in the tree.

Paths. For each leaf node there exists one unique set of split outcomes, called a path. We define the probability that a sample \mathbf{x} takes the path to leaf ℓ as

$$\mu_{\ell}(\mathbf{x}; s, \theta_{\beta}) = \prod_{r \in \mathcal{R}_{\ell}} s(f_{\beta_r}(\mathbf{x})) \prod_{l \in \mathcal{L}_{\ell}} (1 - s(f_{\beta_l}(\mathbf{x}))). \quad (1)$$

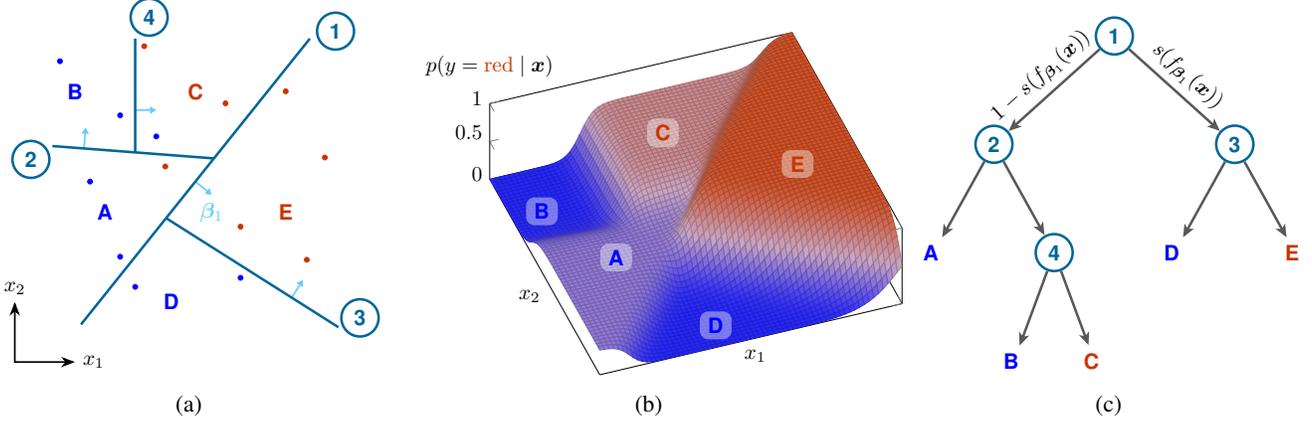


Figure 1: Probabilistic oblique decision trees. **a**) A feature space with a binary classification problem tessellated by an example oblique decision tree. The oblique splits (1-4) partition the feature space into five different leaves (A-E). **b**) The predicted $p(y = \text{red} | \mathbf{x})$ (eq. 2) of the oblique decision tree when a probabilistic split (eq. 3) is used. **c**) The corresponding tree diagram.

Here, $\mathcal{R}_\ell \subset \{1, \dots, I\}$ denotes the splits on the path which contain ℓ in the right subtree. Analogously, $\mathcal{L}_\ell \subset \{1, \dots, I\}$ denotes splits which contain ℓ in the left subtree. In figure 1c this means that $\mathcal{R}_B = \{2\}$ and $\mathcal{L}_B = \{1, 4\}$. Also note that in the following, we will omit the s dependency whenever we do not consider a specific function.

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

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

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

In standard deterministic decision trees as proposed in [2], the split function is a step function $s(x) = \Theta(x)$ with $\Theta(x) = 1$ if $x > 0$ and $\Theta(x) = 0$ otherwise.

3.2. Probabilistic decision tree

We now introduce a defining characteristic of the proposed method. Rather than sending a sample deterministically down the left or right subtree, depending on its features x , we send it left or right with a probability

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

This corresponds to regarding each split in the tree as a Bernoulli decision with mean $\sigma(f(x))$ and as a result equation 2 is the expected value over the possible outcomes. Figure 1b shows the prediction from equation 2 in the proba-

bilistic case for a class $y = \text{“red”}$ on the classification problem illustrated in figure 1a.

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

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

Importantly, while we propose to use a probabilistic decision tree for training, we use a deterministic decision tree for prediction. 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 [19]

$$s(f(x)) = \sigma_\gamma(f(x)) = \sigma(\gamma f(x)). \quad (5)$$

Note, for $\gamma \rightarrow \infty$ the model resembles a deterministic decision tree, since $\sigma_\infty(f(x)) = \Theta(f(x))$. During training, we iteratively increase γ , akin a temperature cooling schedule in deterministic annealing [26].

3.3. Expectation-Maximization

For the optimization of the log-likelihood (equation 4), we propose a gradient-based, EM-style optimization strategy, which requires f and s to be differentiable with respect to the split parameters β_i . The derivation of the EM-algorithm for this model follows the spirit of [11]. We introduce additional latent random variables $z_{n,\ell}$, which indicate that leaf ℓ generated the class label of a given data point \mathbf{x}_n . Including these latent variables, the optimization objective (eq. 4) becomes the complete-data log-likelihood (including

latent variables)

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

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

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

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

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

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

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

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

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

M-Step. In the Maximization-Step of the EM-Algorithm, the expectation value computed in the E-Step (eq. 11) is maximized to find updated parameters

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

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 as

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

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

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

We use the first-order gradient-based stochastic optimization Adam [12] for optimization of the split parameters.

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

3.4. 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 (section 4.4).

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 (section 4.2). To encourage the learning of coherent spatial patterns at each split, we introduce a spatial regularization term

$$- \lambda \sum_{i=1}^I \beta_i^T M \beta_i \quad (15)$$

to the maximization objective of the split features (eq. 14) [5]. The 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 leads to stronger regularization.

3.5. Structure learning

The foregoing shows how to fit a decision tree to training data, given the tree topology (parameter learning). We now turn to the learning of the tree itself (structure learning). We recommend, and evaluate in section 4.1, a greedy strategy: Starting at the root, each split is considered and trained as a tree stump, consisting of one split and two leaf nodes.

Since there are only two leaves, the log-likelihood objective (eq. 4) then resembles an approximation of the widely popular information gain criterion [23, 24] (section 3.6). The previously found splits, in the more shallow levels of the tree, deterministically route data to the split currently being trained. In particular this means that, at first, the root split is trained on the entire training data. After training of the first split, both leaves are discarded and replaced by new

splits. According to the root split, the training data is deterministically divided into two subsets, which are now used to train the corresponding child nodes. This procedure is repeated until, some stopping criterion, *e.g.* maximum depth, maximum number of leaves or leaf purity, is reached. After this greedy structure learning, the nodes in the entire resulting tree can be finetuned jointly as described in section 3.3, this time with probabilistic routing of all training data.

3.6. 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 $\ell(\mathbf{x})$ that returns the index of the leaf sample \mathbf{x} reaches when the path is evaluated deterministically

$$\ell(\mathbf{x}) = \sum_{\ell=1}^L \ell \lim_{\gamma \rightarrow \infty} \mu_{\ell}(\mathbf{x}; \sigma_{\gamma}, \theta_{\beta}). \quad (16)$$

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

$$\max_{\theta} \sum_{n=1}^N \log(\pi_{\ell(x_n)})_{y_n} \quad (17)$$

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

$$\max_{\theta} \sum_{\ell=1}^L \sum_{k=1}^K N_{\ell,k} \log(\pi_{\ell})_k. \quad (18)$$

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

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

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

In conclusion, we have shown that for $\gamma \rightarrow \infty$, 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 [23] to an entire tree.

4. Experiments

We conduct experiments on data from very different domains: first, on the multivariate but unstructured datasets used in [22] (section 4.1). 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 (section 4.2). Then, we demonstrate the same property on a real-world biological image processing task (section 4.3). Finally, we deliver proof of principle 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 (section 4.4).

4.1. Performance of oblique decision trees

We compare the performance of our algorithm in terms of accuracy to all results reported in [22]. They only compare single, unpruned trees, since common ensemble methods such as bagging and boosting as well as pruning can be applied to all algorithms. In order to provide a fair comparison, we also refrain from pruning, ensembles and regularization.

Datasets. Reference [22] reports results on the following four datasets. The multi-class classification datasets *SensIT (combined)*, *Connect4*, *Protein* and *MNIST* are obtained from the LIBSVM repository [6]. 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. In a preprocessing step, we normalize the data to zero mean and unit variance of the training data.

Compared algorithms. The final model for prediction is always a deterministic decision tree with either oblique or axis-aligned splits. The following algorithms were evaluated in [22]. *Axis-aligned*: conventional axis-aligned splits based on information gain. *OCI*: oblique splits optimized with coordinate descent as proposed in [20]. *Random*: selected the best of randomly generated oblique splits based on information gain. *CO2*: greedy oblique tree algorithm based on structured learning [21]. *Non-greedy*: non-greedy oblique decision tree algorithm based on structured learning [22]. We compare the results of these algorithms with our proposed algorithms. Here, *Greedy* denotes a greedy initialization where each oblique split is computed using the EM optimization. For each depth, we apply the *Finetune* algorithm to the tree obtained from the *Greedy* algorithm at that depth.

Hyperparameters and initialization. We keep all hyperparameters fixed and conduct a grid search only 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.

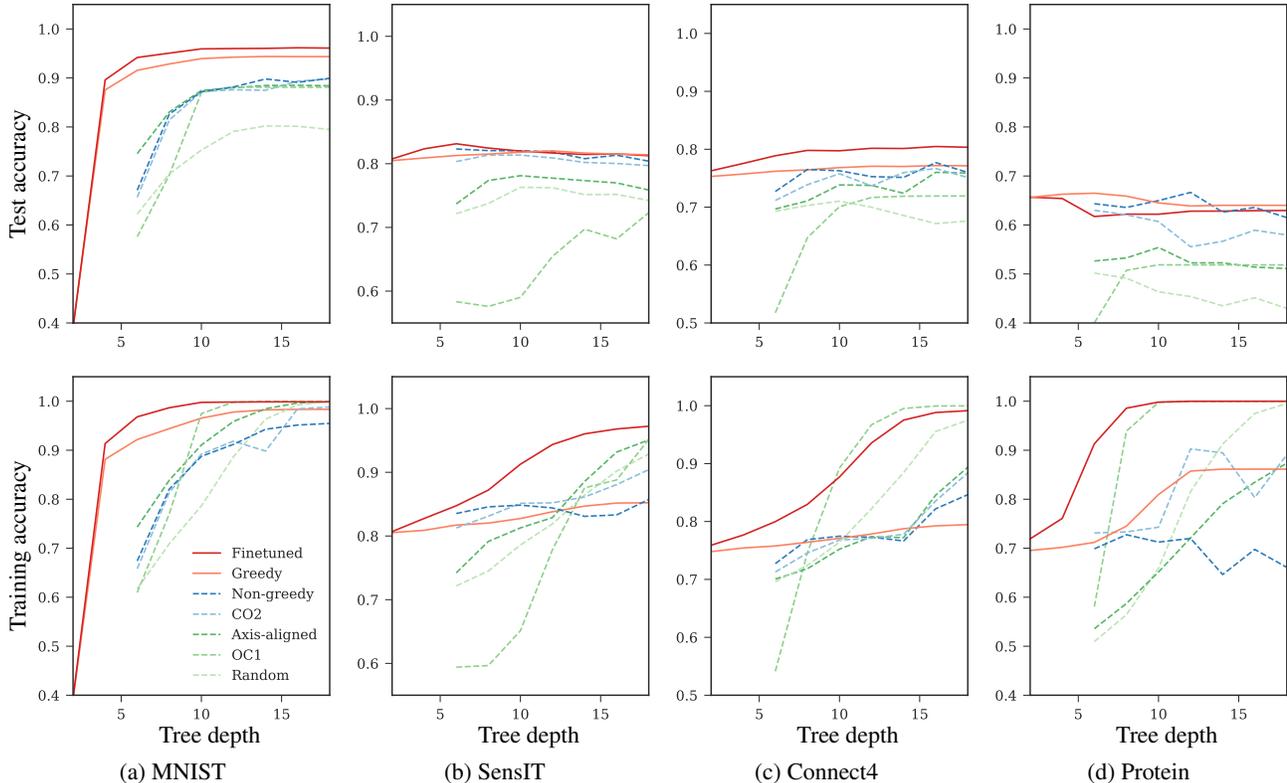


Figure 2: Performance of deterministic oblique decision trees. Accuracy of *Greedy* trained (solid, light red line) and *Fine-tuned* oblique decision trees (solid, dark red line) on test and training sets is compared against other algorithms. The maximum tree depth varies from 2 to 18 with stepsize 2. Dashed lines represent results reported in [22].

For gradient-based split parameter optimization, we use the Adam optimizer [12] 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. 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 θ_β updates of all training batches as well as the update of the leaf predictions θ_π).

Initial split directions are sampled from the unit sphere and the categorical leaf predictions are initialized uniformly random.

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 solid lines show the result of our proposed algorithm, the dashed lines represent results from [22].

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.

Likewise, on *Connect4* our approach performs better at all depths. Notably, a decision tree of depth 2 is sufficient to exceed previous approaches.

On *SensIT* and *Protein* we perform better than or on par with the *Non-greedy* approach proposed in [22]. However, experiments with regularization of leaf features have shown that with more hyperparameter tuning overfitting may be reduced, *e.g.* on the *Protein* dataset and thus the results may be improved. We did not include this here, as we aimed to provide a fair comparison and show the performance given very little fine-tuning.

Generally, our algorithm also trains more accurate oblique decision trees on of the depth complexity on the training data.

In conclusion, our experiments show that our proposed algorithm is able to learn more accurate deterministic oblique decision trees than previous approaches. Furthermore, we refrained from hyperparameter tuning to show that the approach even works well with default parameters.

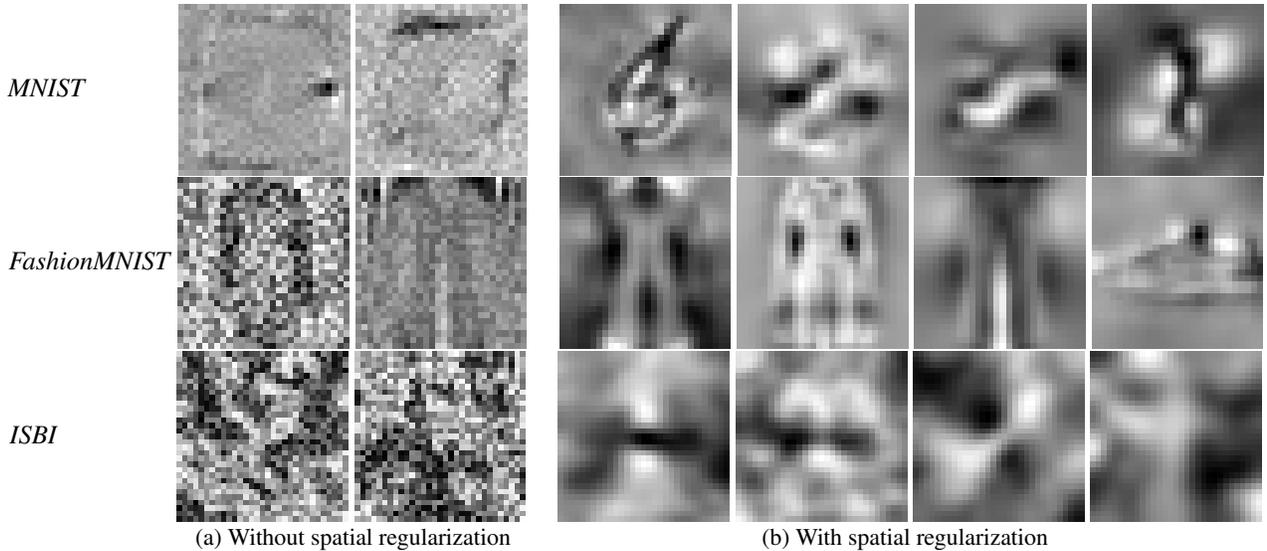


Figure 3: Visualizations of oblique split parameters learned with and without spatial regularization (section 3.4). The parameters are learned on different datasets, viz. *MNIST* [15] (top row), *FashionMNIST* [30] (center row) and *ISBI* [3] (bottom row). Parameters trained with spatial regularization show visible structures and patterns, whereas parameters learned without regularization appear noisy. In both cases, we selected the parameters that show the best visible structures.

4.2. Spatially regularized parameters

We now investigate the effects of spatial regularization (section 3.4) on the parameters of oblique decision trees learned with our algorithm. For this purpose, we train oblique decision trees on the *MNIST* digit dataset [15], the *FashionMNIST* fashion product dataset [30] and the *ISBI* image partitioning dataset [3] comprising serial section Transmission Electron Microscopy images. In figure 3, we visualized selected parameters of the oblique splits at various depths with and without regularization. In both cases, we selected the parameters that displayed the best visible structures. For *MNIST* and *FashionMNIST*, the parameters were reshaped to 28×28 images, such that each parameter pixel corresponds to the respective pixel in the training images. To solve the segmentation task on *ISBI*, we provide a sliding window of size 31×31 as features for each pixel in the center of the window. Moreover, we linearly normalized the parameters to the full grayscale range.

Results. Regularization penalizes differences in adjacent parameters. The parameters without regularization appear very noisy and it is difficult for the human eye to identify structures. In contrast, in the experiments with regularization the algorithm learns smoother parameter patterns, without decreasing the accuracy of the decision trees. The regularized parameters display structures and recognizable patterns. 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.

4.3. Image segmentation

We test the applicability of the proposed decision tree algorithm for image segmentation on the *ISBI* challenge dataset [3]. This image partitioning benchmark comprises serial section Transmission Electron Microscopy images and binary annotations of neurons and membranes (figure 4a).

For every pixel, we provide a sliding window around the current pixel as input features to the oblique decision tree. Consequently, the learned parameters at each split node can be regarded as a spatial kernel. We learn an oblique decision tree of depth 8 with a maximum of 256 leaves with our greedy EM algorithm. We use the default parameters as described in section 4.1 and train each split for 40 epochs.

Results. Figure 4 shows a sample image of the input, the groundtruth labels, the predicted probability of our oblique decision tree and the color-coded leaf affiliation. The visualization of the prediction shows pixels more likely to be of class “membrane” in darker color. In the color-coded leaf affiliation, each grayscale value represents a leaf in the oblique decision tree. Darker pixels have reached a leaf further on the left side of the decision tree.

In the prediction most of the membranes are correctly identified. However, many mitochondria are falsely classified

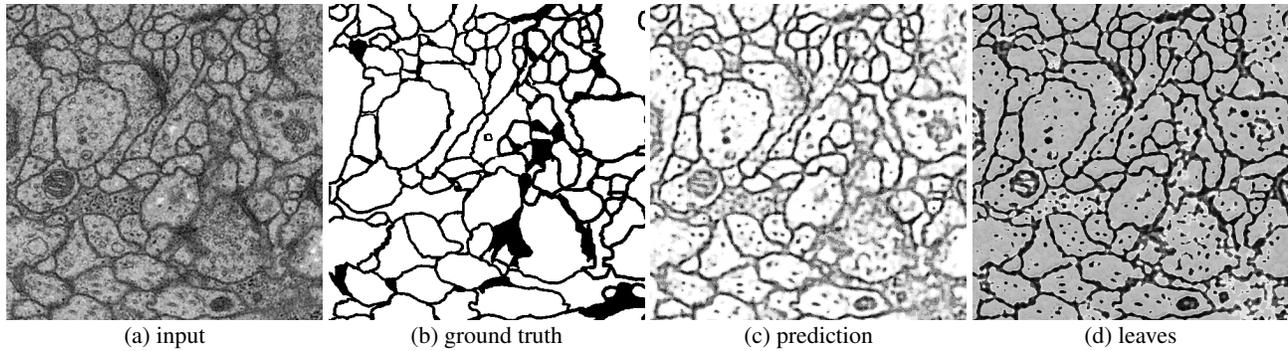


Figure 4: Visualization of results of oblique decision trees on the ISBI binary segmentation dataset. Column **a** shows the input image and column **b** the corresponding groundtruth labels. Column **c** illustrates the probability estimate predicted by the oblique decision tree. Darker means higher probability for class “membrane”. Column **d** show the leaf affiliation in the oblique decision tree. The leaves from left to right are equidistantly assigned to grayscale values from black to white.

as membrane. Interestingly, the leaf affiliation is fairly regular, implying that most adjacent pixels are routed to the same leaf, in spite of some variation in appearance. The leaf affiliation could also be provided as additional feature in order to stack classifiers.

4.4. CNN split features

In a preliminary experiment, we test the effectiveness of Convolutional Neural Networks as split features 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.

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, we achieve a test accuracy of 0.982 ± 0.003 deterministic evaluation of nodes. This model provides 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 nec-

essary to evaluate $K(K-1)/2$ and respectively K different classifiers at test time.

5. Conclusion

We have presented a new approach to train deterministic decision trees with gradient-based optimization in an end-to-end manner. We show that this approach outperforms previous algorithms for oblique decision trees. The approach is not restricted in the complexity of the split features and we have provided preliminary evidence of the effectiveness of more complex split features, such as convolutional neural networks. Moreover, our approach allows imposing additional regularization constraints on the learned split features. We have demonstrated these capabilities by visualizing spatially regularized parameters on image processing datasets. The overall approach provides high flexibility and the potential for accurate models that maintain interpretability and efficiency due to the conditional data flow.

References

- [1] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. **2**
- [2] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Chapman & Hall/CRC, 1984. **3**
- [3] A. Cardona, S. Saalfeld, S. Preibisch, B. Schmid, A. Cheng, J. Pulokas, P. Tomancak, and V. Hartenstein. An integrated micro- and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy. *PLOS Biology*, 8(10):1–17, 10 2010. **7**
- [4] A. Criminisi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer, 2013. **2**
- [5] P. H. C. Eilers and B. D. Marx. Flexible smoothing with B-splines and penalties. *Statistical Science*, 11:89–121, 1996. **4**

- [6] R.-E. Fan and C.-J. Lin. Libsvm data: Classification, regression and multi-label. <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>, 2011. 5
- [7] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014. 2
- [8] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1022–1029, June 2009. 2
- [9] Y. Ioannou, D. Robertson, D. Zikic, P. Kotschieder, J. Shotton, M. Brown, and A. Criminisi. Decision forests, convolutional networks and the models in-between. In *arXiv:1603.01250*, March 2016. 2
- [10] M. I. Jordan. A statistical approach to decision tree modeling. In *Proceedings of the Seventh Annual Conference on Computational Learning Theory, COLT '94*, pages 13–20, New York, NY, USA, 1994. 2
- [11] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Comput.*, 6(2):181–214, Mar. 1994. 2, 3
- [12] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 4, 6
- [13] P. Kotschieder, M. Fiterau, A. Criminisi, and S. Rota Bulò. Deep neural decision forests. In *ICCV*, 2015. 2
- [14] P. Kotschieder, P. Kohli, J. Shotton, and A. Criminisi. Geof: Geodesic forests for learning coupled predictors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013. 2
- [15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. 7
- [16] V. Lepetit, P. Lagler, and P. Fua. Randomized trees for real-time keypoint recognition. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 775–781 vol. 2, June 2005. 2
- [17] M. McGill and P. Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2363–2372, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. 2
- [18] B. H. Menze, B. M. Kelm, D. N. Splitthoff, U. Koethe, and F. A. Hamprecht. *On Oblique Random Forests*, pages 453–469. Springer, 2011. 2
- [19] A. Montillo, J. Tu, J. Shotton, J. Winn, J. Iglesias, D. Metaxas, and A. Criminisi. Entanglement and differentiable information gain maximization. In *Decision Forests for Computer Vision and Medical Image Analysis*, chapter 19, pages 273–293. Springer, January 2013. 2, 3
- [20] K. V. S. Murthy. *On Growing Better Decision Trees from Data*. PhD thesis, The Johns Hopkins University, 1996. 5
- [21] M. Norouzi, M. D. Collins, D. J. Fleet, and P. Kohli. Co2 forest: Improved random forest by continuous optimization of oblique splits. *arXiv:1506.06155*, 2015. 2, 5
- [22] M. Norouzi, M. D. Collins, M. Johnson, D. J. Fleet, and P. Kohli. Efficient non-greedy optimization of decision trees. In *NIPS*, December 2015. 2, 5, 6
- [23] J. R. Quinlan. Induction of decision trees. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986. 1, 4, 5
- [24] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. 4
- [25] D. Richmond, D. Kainmueller, M. Yang, E. Myers, and C. Rother. Mapping auto-context decision forests to deep convnets for semantic segmentation. In E. R. H. Richard C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 144.1–144.12. BMVA Press, September 2016. 2
- [26] K. Rose, E. Gurewitz, and G. C. Fox. Statistical mechanics and phase transitions in clustering. *Phys. Rev. Lett.*, 65:945–948, Aug 1990. 3
- [27] I. K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, Oct 1990. 2
- [28] A. Suárez and J. F. Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(12):1297–1311, Dec. 1999. 2
- [29] J. Welbl. Casting random forests as artificial neural networks (and profiting from it). In *GCPR*, 2014. 2
- [30] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. In *arXiv:1708.07747*, 2017. 7

Supplementary Material to End-to-end Learning of Deterministic Decision Trees

Thomas Hehn Fred A. Hamprecht
HCI/IWR, Heidelberg University
{thomas.hehn, fred.hamprecht}@iwr.uni-heidelberg.de

1. Alternating optimization

Similarly to the EM-algorithm presented in section 3.3 of the main paper, the alternating strategy aims to optimize the split parameters and the leaf prediction parameters separately. However, instead of computing the exact leaf predictions, we approximate them using a deterministic decision tree.

Let $N_{\ell,k}$ be the number of training samples in leaf ℓ with class k and $N_\ell = \sum_{k=1}^K N_{\ell,k}$ denotes the sum of all training samples in leaf ℓ . The optimal leaf predictions are

$$(\boldsymbol{\pi}_\ell)_k = \frac{N_{\ell,k}}{N_\ell}. \quad (1)$$

Analogously to the EM-algorithm in the main paper, we use Adam [1] to optimize the split parameters. The log-likelihood objective given the current estimate of the leaf predictions is:

$$\max_{\boldsymbol{\theta}_\beta} \sum_{n=1}^N \log \left(\sum_{\ell=1}^L (\boldsymbol{\pi}_\ell)_y \mu_\ell(\mathbf{x}; \sigma_\gamma, \boldsymbol{\theta}_\beta) \right). \quad (2)$$

This algorithm is applicable to optimize an entire tree, as well as to optimize tree stumps as required for the greedy structure learning.

1.1. Experiments

We compare this algorithm to the results of our EM algorithm for oblique decision trees reported in section 4.1 of the main paper. The hyperparameter tuning and the experimental setup are done in the same way as for the EM algorithm. Figure 1 illustrates the results of the experiments. The results show that both algorithms perform equally well in terms of accuracy.

2. Visualization of oblique decision trees

Figures 2 and 3 show visualizations of entire oblique decision trees of depth 4 trained on *MNIST* [2] and *FashionMNIST* [3]. Both trees were trained with the EM

algorithm and spatial regularization to obtain smooth visualizations. Leaf predictions are visualized as bar plots. The split parameters are visualized as in section 4.2 of the main paper. Intuitively, the better the input image matches the parameter image, the more likely it is to go to the right. Likewise, the better the input images resembles the negative parameter image, the more likely it is to go the left child. How many training samples follow a certain path is indicated by the thickness of the arrows. Thicker arrows mean more training samples follow the path when the decision tree is deterministic.

Some split parameters of the decision tree trained on *MNIST* (figure 2) reveal interesting structures. At depth 3 (row 4), the fifth split node from the left shows a dark silhouette of the number “6”. Accordingly, the left child predicts class “6”. In the same manner, the parameters of the sixth split node at depth 3 expose a dark stroke in the center surrounded by brighter pixels. This is used to distinguish class “1”, which is predicted by the left child.

The decision tree in figure 3 was trained on the *FashionMNIST* dataset [3], a dataset comprising 28×28 grayscale images of fashion products. These fashion products can be discovered in the learned split parameters. The split parameters of the first split at depth 3 (row 4) 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 which 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). Similarly, the fifth split node also reveals only a slight silhouette of a shoe. Still, its right child is certain predicting “sneaker”, whereas the left child is rather indecisively predicting “sandals”, “sneaker” and “ankle boots”.

The decision tree illustrations reveal the internal decisions being made to reach a final prediction and provide a useful tool to interpret our model.

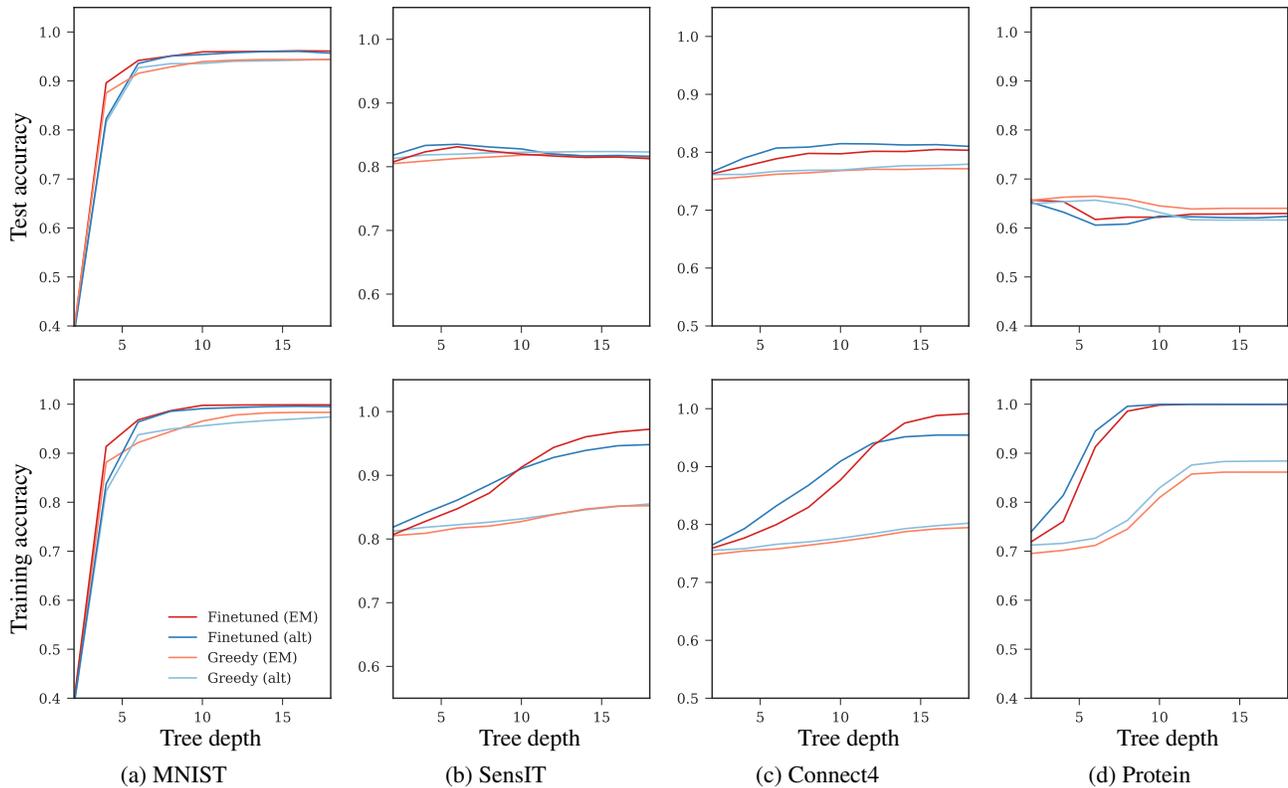


Figure 1: Performance of alternating optimization (*alt*) compared to the EM optimization (*EM*) presented in the main paper. Accuracy of oblique decision trees at different maximum depth on test and training sets is compared. Maximum tree depth is varied from 2 to 18 with step size 2. At each depth, both algorithms (*alt* and *EM*) are used to learn respectively an initial *greedy* tree and *finetune* the initial tree.

References

- [1] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 1
- [2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. 1, 3
- [3] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. In *arXiv:1708.07747*, 2017. 1, 4

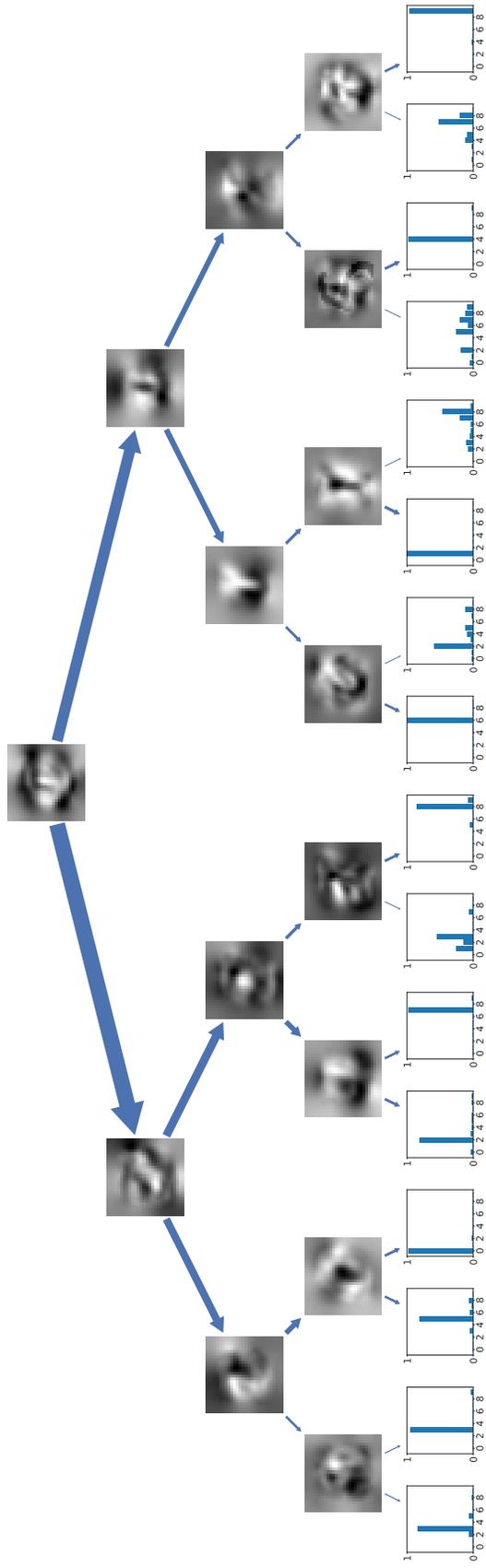


Figure 2: Visualization of an oblique decision tree learned on *MNIST* [2] with spatial regularization. The split parameters are visualized as in section 4.2 of the main paper. As a rule of thumb: The better the input image matches the parameter image, the more likely the sample will go to the right 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. Leaf predictions are visualized as bar plots, where the x -axis denotes classes 0 to 9 and the y -axis corresponds to the probability of each class. Axis labels were omitted due to space restrictions.

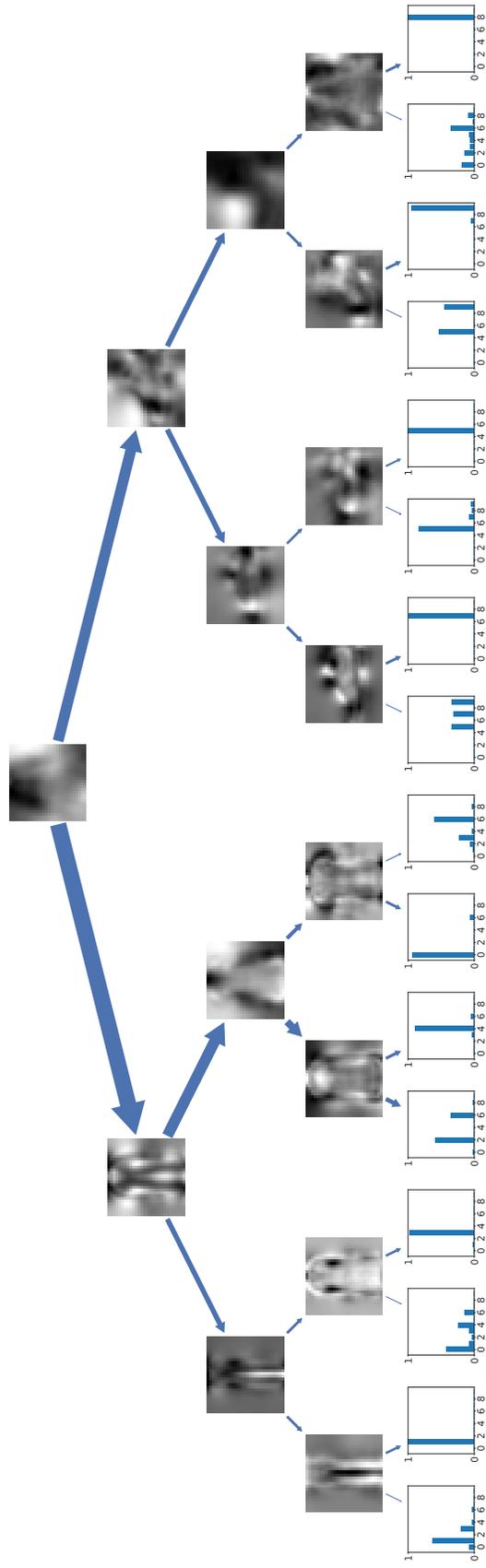


Figure 3: Visualization of an oblique decision tree learned on *FashionMNIST* [3] with spatial regularization. The split parameters are visualized as in section 4.2 of the main paper. As a rule of thumb: The better the input image matches the parameter image, the more likely the sample will go to the 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. Leaf predictions are visualized as bar plots. The x -axis denotes classes (0: T-shirt/top, 1: Trouser, 2: Pullover, 3: Dress, 4: Coat, 5: Sandal, 6: Shirt, 7: Sneaker, 8: Bag, 9: Ankle boot) and the y -axis corresponds to the probability of each class. Axis labels were omitted due to space restrictions.