Adaptive Neural Trees

Ryutaro Tanno [1] **Kai Arulkumaran** [2] **Daniel C. Alexander** [1] **Antonio Criminisi** [3] **Aditya Nori** [3]

Abstract

Deep neural networks and decision trees operate on largely separate paradigms; typically, the former performs representation learning with pre-specified architectures, while the latter is characterised by learning hierarchies over pre-specified features with data-driven architectures. We unite the two via *adaptive neural trees* (ANTs) that incorporates representation learning into edges, routing functions and leaf nodes of a decision tree, along with a backpropagation-based training algorithm that adaptively grows the architecture from primitive modules (e.g., convolutional layers). We demonstrate that, whilst achieving competitive performance on classification and regression datasets, ANTs benefit from (i) lightweight inference via conditional computation, (ii) hierarchical separation of features useful to the task e.g. learning meaningful class associations, such as separating natural vs. man-made objects, and (iii) a mechanism to adapt the architecture to the size and complexity of the training dataset.

1. Introduction

Neural networks (NNs) and decision trees (DTs) are both powerful classes of machine learning models with proven successes in academic and commercial applications. The two approaches, however, typically come with mutually exclusive benefits and limitations.

NNs are characterised by learning hierarchical representations of data through the composition of nonlinear transformations (Zeiler & Fergus, 2014; Bengio, 2013), which has alleviated the need for feature engineering, in contrast with many other machine learning models. In addition, NNs are trained with stochastic optimisers, such as stochastic gradient descent (SGD), allowing training to scale to large datasets. Consequently, with modern hardware, we can train NNs of many layers on large datasets, solving numerous problems ranging from object detection to speech recognition with unprecedented accuracy (LeCun et al., 2015). However, their architectures typically need to be designed by hand and fixed per task or dataset, requiring domain expertise (Zoph & Le, 2017). Inference can also be heavyweight for large models, as each sample engages every part of the network, i.e., increasing capacity causes a proportional increase in computation (Bengio et al., 2013).

Alternatively, DTs are characterised by learning hierarchical clusters of data (Criminisi & Shotton, 2013). A DT learns how to split the input space, so that in each subset, linear models suffice to explain the data. In contrast to standard NNs, the architectures of DTs are optimised based on training data, and are particularly advantageous in data-scarce scenarios. DTs also enjoy lightweight inference as only a single root-to-leaf path on the tree is used for each input sample. However, successful applications of DTs often require hand-engineered features of data. We can ascribe the limited expressivity of single DTs to the common use of simplistic routing functions, such as splitting on axis-aligned features. The loss function for optimising hard partitioning is non-differentiable, which hinders the use of gradient descent-based optimization and thus complex splitting functions. Current techniques for increasing capacity include ensemble methods such as random forests (RFs) (Breiman, 2001) and gradient-boosted trees (GBTs) (Friedman, 2001), which are known to achieve state-of-the-art performance in various tasks, including medical applications and financial forecasting (Sandulescu & Chiru, 2016; Kaggle.com, 2017; Le Folgoc et al., 2016; Volkovs et al., 2017).

The goal of this work is to combine NNs and DTs to gain the complementary benefits of both approaches. To this end, we propose *adaptive neural trees* (ANTs), which generalise previous work that attempted the same unification (Suárez & Lutsko, 1999; İrsoy et al., 2012; Laptev & Buhmann, 2014; Rota Bulo & Kontschieder, 2014; Kontschieder et al., 2015; Frosst & Hinton, 2017; Xiao, 2017) and address their limitations (see Tab. 1). ANTs represent routing decisions and root-to-leaf computational paths within the tree structures as NNs, which lets them benefit from hierarchical representation learning, rather than being restricted to partitioning the raw data space. On the other hand, unlike the fully distributed representaion of standard NN models,

Work done while Ryutaro and Kai were Research Interns at Microsoft Research Cambridge. [1]University College London, UK [2]Imperial College London, UK [3]Microsoft Research, Cambridge, UK. Correspondence to: Ryutaro Tanno <ryutaro.tanno.15@ucl.ac.uk>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

the tree topology of ANTs acts as a strong structural prior that enforces sparse structures by which features are shared and separated in a hierarchical fashion. In addition, we propose a backpropagation-based training algorithm to grow ANTs based on a series of decisions between making the ANT deeper—the central NN paradigm—or partitioning the data—the central DT paradigm (see Fig. 1 (Right)). This allows the architectures of ANTs to adapt to the data available. By our design, ANTs inherit the following desirable properties from both DTs and NNs:

- **Representation learning**: as each root-to-leaf path in an ANT is an NN, features can be learned end-to-end with gradient-based optimisation. Combined with the tree structure, an ANT can learn such features which are hierarchically shared and separated.
- **Architecture learning**: by progressively growing ANTs, the architecture adapts to the availability and complexity of data, embodying Occams razor. The growth procedure can be viewed as architecture search with a hard constraint over the model class.
- **Lightweight inference**: at inference time, ANTs perform conditional computation, selecting a single root-to-leaf path on the tree on a per-sample basis, activating only a subset of the parameters of the model.

We empirically validate these benefits for regression and classification through experiments on the SARCOS (Vijayakumar & Schaal, 2000), MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky & Hinton, 2009) datasets. The best performing methods on the SARCOS multivariate regression dataset are all tree-based, with ANTs achieving the lowest mean squared error. On the other hand, along with other forms of neural networks, ANTs far outperform state-of-the-art RF (Zhou & Feng, 2017) and GBT (Ponomareva et al., 2017) methods on image classification, with architectures achieving over 99% accuracy on MNIST and over 90% accuracy on CIFAR-10. Our ablations on all three datasets consistently show that the combination of feature learning and data partitioning are required for the best predictive performance of ANTs. In addition, we show that ANTs can learn meaningful hierarchical partitionings of data, e.g., grouping man-made and natural objects (see Fig. 2) useful to the end task. ANTs also have reduced time and memory requirements during inference, thanks to such hierarchical structure. In one case, we discover an architecture that achieves over 98% accuracy on MNIST using approximately the same number of parameters as a linear classifier on raw image pixels, showing the benefits of tree-shaped hierarchical sharing and separation of features in enhancing both computational and predictive performance. Finally, we demonstrate the benefits of architecture learning by training ANTs on subsets of CIFAR-10 of varying sizes. The method can construct architectures of adequate size, leading to better generalisation, particularly on small datasets.

2. Related work

Our work is primarily related to research into combining DTs and NNs. Here we explain how ANTs subsume a large body of such prior work as specific cases and address their limitations. We include additional reviews of work in conditional computation and neural architecture search in Sec.B in the supplementary material.

The very first soft decition tree (SDT) introduced by Suárez & Lutsko (1999) is a specific case where in our terminology the routers are axis-aligned features, the transformers are identity functions, and the routers are static distributions over classes or linear functions. The hierarchical mixture of experts (HMEs) proposed by Jordan & Jacobs (1994) is a variant of SDTs whose routers are linear classifiers and the tree structure is fixed; Léon & Denoyer (2015) recently proposed a more computationally efficient training method that is able to directly optimise hard-partitioning by differentiating through stochastic gradient estimators. More modern SDTs (Rota Bulo & Kontschieder, 2014; Laptev & Buhmann, 2014; Frosst & Hinton, 2017) have used multilayer perceptrons (MLPs) or convolutional layers in the routers to learn more complex partitionings of the input space. However, the simplicity of identity transformers used in these methods means that input data is never transformed and thus each path on the tree does not perform representation learning, limiting their performance.

More recent work suggested that integrating non-linear transformations of data into DTs would enhance model performance. The neural decision forest (NDF) (Kontschieder et al., 2015), which held cutting-edge performance on ImageNet (Deng et al., 2009) in 2015, is an ensemble of DTs, each of which is also an instance of ANTs where the whole GoogLeNet architecture (Szegedy et al., 2015) (except for the last linear layer) is used as the root transformer, prior to learning tree-structured classifiers with linear routers. Xiao (2017) employed a similar approach with a MLP at the root transformer, and is optimised to minimise a differentiable information gain loss. The conditional network proposed by Ioannou et al. (2016) sparsified CNN architectures by distributing computations on hierarchical structures based on directed acyclic graphs with MLP-based routers, and designed models with the same accuracy with reduced compute cost and number of parameters. However, in all cases, the model architectures are pre-specified and fixed.

In contrast, ANTs satisfy all criteria in Tab. 1; they provide a general framework for learning tree-structured models with the capacity of representation learning along each path and within routing functions, and a mechanism for learning its architecture.

Architecture growth is a key facet of DTs (Criminisi & Shotton, 2013), and typically performed in a greedy fashion with a termination criteria based on validation set error

Table 1: Comparison of tree-structured NNs. The first column denotes if each path on the tree is a NN, and the second column denotes if the routers learn features. The last column shows if the method grows an architecture, or uses a pre-specified one.

Method	Feature learning?		Grown?
	Path	Routers	
SDT (Suárez & Lutsko, 1999)	✗	✗	✓
SDT 2 / HME (Jordan & Jacobs, 1994)	✗	✓	✗
SDT 3 (İrsoy et al., 2012)	✗	✓	✓
SDT 4 (Frosst & Hinton, 2017)	✗	✓	✗
RDT (Léon & Denoyer, 2015)	✗	✓	✗
BT (İrsoy et al., 2014)	✗	✓	✓
Conv DT (Laptev & Buhmann, 2014)	✗	✓	✗
NDT (Rota Bulo & Kontschieder, 2014)	✗	✓	✓
NDT 2 (Xiao, 2017)	✓	✓	✗
NDF (Kontschieder et al., 2015)	✓	✓	✗
CNet (Ioannou et al., 2016)	✓	✓	✗
ANT (ours)	✓	✓	✓

(Suárez & Lutsko, 1999; İrsoy et al., 2012). Previous works in DT research have made attempts to improve upon this greedy growth strategy. Decision jungles (Shotton et al., 2013) employ a training mechanism to merge partitioned input spaces between different sub-trees, and thus to rectify suboptimal "splits" made due to the locality of optimisation. İrsoy et al. (2014) proposes budding trees, which are grown and pruned incrementally based on global optimisation of existing nodes. While our training algorithm, for simplicity, grows the architecture by greedily choosing the best option between going deeper and splitting the input space (see Fig. 1), it is certainly amenable to these advances.

3. Adaptive Neural Trees

We now formalise the definition of Adaptive Neural Trees (ANTs), which are a form of DTs enhanced with deep, learned representations. We focus on supervised learning, where the aim is to learn the conditional distribution $p(\mathbf{y}|\mathbf{x})$ from a set of N labelled samples $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), ..., (\mathbf{x}^{(N)}, \mathbf{y}^{(N)}) \in \mathcal{X} \times \mathcal{Y}$ as training data.

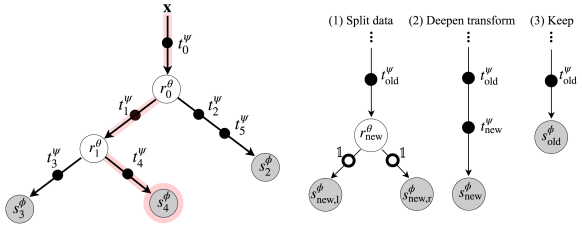


Figure 1: **(Left).** An example ANT. Data is passed through transformers (black circles on edges), routers (white circles on internal nodes), and solvers (gray circles on leaf nodes). The red shaded path shows routing of $\mathbf{x}$ to reach leaf node 4. Input $\mathbf{x}$ undergoes a series of selected transformations $\mathbf{x} \rightarrow \mathbf{x}_0^{\boldsymbol{\psi}} := t_0^{\boldsymbol{\psi}}(\mathbf{x}) \rightarrow \mathbf{x}_1^{\boldsymbol{\psi}} := t_1^{\boldsymbol{\psi}}(\mathbf{x}_0^{\boldsymbol{\psi}}) \rightarrow \mathbf{x}_4^{\boldsymbol{\psi}} := t_4^{\boldsymbol{\psi}}(\mathbf{x}_1^{\boldsymbol{\psi}})$ and the solver module yields the predictive distribution $p_4^{\boldsymbol{\phi},\boldsymbol{\psi}}(\mathbf{y}) := s_4^{\boldsymbol{\phi}}(\mathbf{x}_4^{\boldsymbol{\psi}})$. The probability of selecting this path is given by $\pi_2^{\boldsymbol{\psi},\boldsymbol{\theta}}(\mathbf{x}) := r_0^{\boldsymbol{\theta}}(\mathbf{x}_0^{\boldsymbol{\psi}}) \cdot (1 - r_1^{\boldsymbol{\theta}}(\mathbf{x}_1^{\boldsymbol{\psi}}))$. **(Right).** Three growth options at a given node: *split data, deepen transform & keep*. The small white circles on the edges denote identity transformers.

3.1. Model Topology and Operations

In short, an ANT is a tree-structured model, characterized by a set of hierarchical partitions of the input space $\mathcal{X}$, a series of nonlinear transformations, and separate predictive models in the respective component regions. More formally, we define an ANT as a pair $(\mathbb{T}, \mathbb{O})$ where $\mathbb{T}$ defines the model topology, and $\mathbb{O}$ denotes the set of operations on it.

We restrict the model topology $\mathbb{T}$ to be instances of *binary trees*, defined as a set of graphs whose each node is either an internal node or a leaf, and is the child of exactly one parent node, except the root node at the top. We define the topology of a tree as $\mathbb{T} := \{\mathcal{N}, \mathcal{E}\}$ where $\mathcal{N}$ is the set of all nodes, and $\mathcal{E}$ is the set of edges between them. Nodes with no children are leaf nodes, $\mathcal{N}_{leaf}$, and all others are internal nodes, $\mathcal{N}_{int}$. Every internal node $j \in \mathcal{N}_{int}$ has exactly two children nodes, represented by $\text{left}(j)$ and $\text{right}(j)$. Unlike standard trees, $\mathcal{E}$ contains an edge which connects input data $\mathbf{x}$ with the root node, as shown in Fig.1 (Left).

Every node and edge is assigned with operations which acts on the allocated samples of data (Fig.1). Starting at the root, each sample gets transformed and traverses the tree according to the set of operations $\mathbb{O}$. An ANT is constructed based on three primitive modules of differentiable operations:

1. **Routers**, $\mathcal{R}$: each internal node $j \in \mathcal{N}_{int}$ holds a *router* module, $r_j^{\boldsymbol{\theta}} : \mathcal{X}_j \rightarrow [0, 1] \in \mathcal{R}$, parametrised by $\boldsymbol{\theta}$, which sends samples from the incoming edge to either the left or right child. Here $\mathcal{X}_j$ denotes the representation at node j. We use *stochastic routing*, where the decision (1 for the left and 0 for the right branch) is sampled from Bernoulli distribution with mean $r_j^{\boldsymbol{\theta}}(\mathbf{x}_j)$ for input $\mathbf{x}_j \in \mathcal{X}_j$. As an example, $r_j^{\boldsymbol{\theta}}$ can be defined as a small CNN.
2. **Transformers**, $\mathcal{T}$: every edge $e \in \mathcal{E}$ of the tree has one or a composition of multiple *transformer* module(s). Each transformer $t_e^{\boldsymbol{\psi}} \in \mathcal{T}$ is a nonlinear function, parametrised by $\boldsymbol{\psi}$, that transforms samples from the previous module and passes them to the next one. For example, $t_e^{\boldsymbol{\psi}}$ can be a single convolutional layer followed by ReLU (Nair & Hinton, 2010). Unlike in standard DTs, edges transform data and are allowed to "grow" by adding more operations (Sec. 4), learning "deeper" representations as needed.
3. **Solvers**, $\mathcal{S}$: each leaf node $l \in \mathcal{N}_{leaf}$ is assigned to a *solver* module, $s_l^{\boldsymbol{\phi}} : \mathcal{X}_l \rightarrow \mathcal{Y} \in \mathcal{S}$, parametrised by $\boldsymbol{\phi}$, which operates on the transformed input data and outputs an estimate for the conditional distribution $p(\mathbf{y}|\mathbf{x})$. For classification tasks, we can define, for example, $s^{\boldsymbol{\phi}}$ as a linear classifier on the feature space $\mathcal{X}_l$, which outputs a distribution over classes.

Defining operations on the graph $\mathbb{T}$ amounts to a specification of the triplet $\mathbb{O} = (\mathcal{R}, \mathcal{T}, \mathcal{S})$. For example, given image inputs, we would choose the operations of each module to be from the set of operations commonly used in CNNs (examples are given in Tab. 2). In this case, every computational path on the resultant ANT, as well as the set of

Table 2: Primitive module specifications for MNIST, CIFAR-10 and SARCOS datasets. "conv5-40" denotes a 2D convolution with 40 kernels of spatial size 5×5. "GAP", "FC", "LC" and "LR" stand for global-average-pooling, fully connected layer, linear classifier and linear regressor. "Downsample Freq" denotes the frequency at which 2×2 max-pooling is applied.

Model	Router, $\mathcal{R}$	Transformer, $\mathcal{T}$	Solver, $\mathcal{S}$	Downsample Freq.
ANT-SARCOS	$1 \times$ FC +Sigmoid	$1 \times$FC+ tanh	LR	0
ANT-MNIST-A	$1 \times$ conv5-40 + GAP + $2 \times$FC +Sigmoid	$1 \times$ conv5-40 + ReLU	LC	1
ANT-MNIST-B	$1 \times$ conv3-40 + GAP + $2 \times$FC +Sigmoid	$1 \times$ conv3-40 +ReLU	LC	2
ANT-MNIST-C	$1 \times$ conv5-5 + GAP + $2 \times$FC +Sigmoid	$1 \times$ conv5-5+ReLU	LC	2
ANT-CIFAR10-A	$2 \times$ conv3-128 + GAP + $1 \times$FC +Sigmoid	$2 \times$ conv3-128 +ReLU	GAP + LC	1
ANT-CIFAR10-B	$2 \times$ conv3-96 + GAP + $1 \times$FC +Sigmoid	$2 \times$ conv3-96 +ReLU	LC	1
ANT-CIFAR10-C	$2 \times$ conv3-48 + GAP + $1 \times$FC +Sigmoid	$2 \times$ conv3-96 +ReLU	GAP + LC	1

routers that guide inputs to one of these paths, are given by CNNs. Lastly, many existing tree-structured models (Suárez & Lutsko, 1999; İrsoy et al., 2012; Laptev & Buhmann, 2014; Rota Bulo & Kontschieder, 2014; Kontschieder et al., 2015; Frosst & Hinton, 2017; Xiao, 2017) are instantiations of ANTs with limitations which we will address with our model (see Sec. 2 for a more detailed discussion).

3.2. Probabilistic model and inference

An ANT models the conditional distribution $p(\mathbf{y}|\mathbf{x})$ as a hierarchical mixture of experts (HMEs) (Jordan & Jacobs, 1994), each of which is defined as an NN and is a root-to-leaf path in the tree. Standard HMEs are a special case of ANTs where transformers are the identity function. As a result, the representations within experts are hierarchically shared between similar experts, unlike the independent representations within experts in standard HMEs. In addition, ANTs come with a growth mechanism to determine the number of needed experts and their complexity, as discussed in Sec. 4.

Each input $\mathbf{x}$ to the ANT stochastically traverses the tree based on decisions of routers and undergoes a sequence of transformations until it reaches a leaf node where the corresponding solver predicts the label $\mathbf{y}$. Suppose we have L leaf nodes, the full predictive distribution, with parameters $\Theta = (\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\phi})$, is given by

$$p(\mathbf{y}|\mathbf{x}, \Theta) = \sum_{l=1}^{L} \underbrace{p(z_l = 1|\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\psi})}_{\text{Leaf-assignment prob. } \pi_l^{\boldsymbol{\theta},\boldsymbol{\psi}}} \underbrace{p(\mathbf{y}|\mathbf{x}, z_l = 1, \boldsymbol{\phi}, \boldsymbol{\psi})}_{\text{Leaf-specific prediction. } p_l^{\boldsymbol{\phi},\boldsymbol{\psi}}} \tag{1}$$

where $\mathbf{z} \in \{0, 1\}^L$ is an L-dimensional binary latent variable such that $\sum_{l=1}^{L} z_l = 1$, which describes the choice of leaf node (e.g. $z_l = 1$ means that leaf l is used). Here $\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\phi}$ summarise the parameters of router, transformer and solver modules in the tree. The mixing coefficient $\pi_l^{\boldsymbol{\theta},\boldsymbol{\psi}}(\mathbf{x}) := p(z_l = 1|\mathbf{x}, \boldsymbol{\psi}, \boldsymbol{\theta})$ quantifies the probability that $\mathbf{x}$ is assigned to leaf l and is given by a product of decision probabilities over all router modules on the unique path $\mathcal{P}_l$ from the root to leaf node l:

$$\pi_l^{\boldsymbol{\psi},\boldsymbol{\theta}}(\mathbf{x}) = \prod_{r_j^{\boldsymbol{\theta}} \in \mathcal{P}_l} r_j^{\boldsymbol{\theta}}(\mathbf{x}_j^{\boldsymbol{\psi}})^{\mathbb{1}[l \swarrow j]} \cdot \left(1 - r_j^{\boldsymbol{\theta}}(\mathbf{x}_j^{\boldsymbol{\psi}})\right)^{1 - \mathbb{1}[l \swarrow j]}$$

where $l \swarrow j$ is a binary relation and is only true if leaf l is in the left subtree of internal node j, and $\mathbf{x}_j^{\boldsymbol{\psi}}$ is the feature representation of $\mathbf{x}$ at node j. Let $\mathcal{T}_j = \{t_{e_1}^{\boldsymbol{\psi}}, ..., t_{e_n}^{\boldsymbol{\psi}}\}$ denote the ordered set of the n transformer modules on the path from the root to node j, the feature vector $\mathbf{x}_j^{\boldsymbol{\psi}}$ is given by

$$\mathbf{x}_j^{\boldsymbol{\psi}} := \left(t_{e_n}^{\boldsymbol{\psi}} \circ ... \circ t_{e_2}^{\boldsymbol{\psi}} \circ t_{e_1}^{\boldsymbol{\psi}}\right)(\mathbf{x}).$$

On the other hand, the leaf-specific conditional distribution $p_l^{\boldsymbol{\phi},\boldsymbol{\psi}}(\mathbf{y}) := p(\mathbf{y}|\mathbf{x}, z_l = 1, \boldsymbol{\phi}, \boldsymbol{\psi})$ in (1) yields an estimate for the distribution over target $\mathbf{y}$ for leaf node l and is given by its solver's output $s_l^{\boldsymbol{\phi}}(\mathbf{x}_{\text{parent}(l)}^{\boldsymbol{\psi}})$.

We consider two inference schemes based on a trade-off between accuracy and computation, which we refer to as *multi-path* and *single-path* inference. The multi-path inference uses the *full predictive distribution* given in eq. (1). However, computing this quantity requires averaging the distributions over all the leaves involving computing all operations at all nodes and edges of the tree, which is expensive for a large ANT. On the other hand, the single-path inference scheme only uses the predictive distribution at the leaf node chosen by greedily traversing the tree in the directions of highest confidence of the routers. This approximation constrains computations to a single path, allowing for more memory- and time-efficient inference.

4. Optimisation

Training of an ANT proceeds in two stages: 1) *growth phase* during which the model architecture is learned based on *local* optimisation, and 2) *refinement phase* which further tunes the parameters of the model discovered in the first phase based on *global* optimisation. We include a pseudocode of the training algorithm in Supp. Sec. A.

4.1. Loss function: optimising parameters of $\mathbb{O}$

For both phases, we use the negative log-likelihood (NLL) as the common objective function to minimise:

$$-\log p(\mathbf{Y}|\mathbf{X}, \Theta) = -\sum_{n=1}^{N} \log \left(\sum_{l=1}^{L} \pi_l^{\boldsymbol{\theta},\boldsymbol{\psi}}(\mathbf{x}^{(n)})\, p_l^{\boldsymbol{\phi},\boldsymbol{\psi}}(\mathbf{y}^{(n)})\right)$$

where $\mathbf{X} = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}\}$, $\mathbf{Y} = \{\mathbf{y}^{(1)}, ..., \mathbf{y}^{(N)}\}$ denote the training inputs and targets. As all component modules (routers, transformers and solvers) are differentiable with respect to their parameters $\Theta = (\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\phi})$, we can use gradient-based optimisation. Given an ANT with fixed topology $\mathbb{T}$, we use backpropagation (Rumelhart et al., 1986) for gradient computation and use gradient descent to minimise the NLL for learning the parameters.

4.2. Growth phase: learning architecture $\mathbb{T}$

We next describe our proposed method for growing the tree $\mathbb{T}$ to an architecture of adequate complexity for the given training data. Starting from the root, we choose one of the leaf nodes in breadth-first order and incrementally modify the architecture by adding computational modules to it. In particular, we evaluate 3 choices (Fig. 1 (Right)) at each leaf node; (1)"split data" extends the current model by splitting the node with an addition of a new router; (2) "deepen transform" increases the depth of the incoming edge by adding a new transformer; (3) "keep" retains the current model. We then locally optimise the parameters of the newly added modules in the architectures of (1) and (2) by minimising NLL via gradient descent, while fixing the parameters of the previous part of the computational graph. Lastly, we select the model with the lowest validation NLL if it improves on the previously observed lowest NLL, otherwise we execute (3). This process is repeated to all new nodes level-by-level until no more "split data" or "deepen transform" operations pass the validation test.

The rationale for evaluating the two choices is to give the model a freedom to choose the most effective option between "going deeper" or splitting the data space. Splitting a node is equivalent to a soft partitioning of the feature space of incoming data, and gives birth to two new leaf nodes (left and right children solvers). In this case, the added transformer modules on the two branches are identity functions. Deepening an edge on the other hand seeks to learn richer representation via an extra nonlinear transformation, and replaces the old solver with a new one. Local optimisation is efficient in time and space; gradients only need to be computed for the parameters of the new parts of the architecture, reducing computation, while forward activations prior to the new parts do not need to be stored in memory, saving space.

4.3. Refinement phase: global tuning of $\mathbb{O}$

Once the model topology is determined in the growth phase, we finish by performing global optimisation to refine the parameters of the model, now with a fixed architecture. This time, we perform gradient descent on the NLL with respect to the parameters of all modules in the graph, jointly optimising the hierarchical grouping of data to paths on the tree and the associated expert NNs. The refinement phase can correct suboptimal decisions made during the local optimisation of the growth phase, and empirically improves the generalisation error (see Sec. 5.3).

5. Experiments

We evaluate ANTs using the SARCOS multivariate regression dataset (Vijayakumar & Schaal, 2000), and the MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky & Hinton, 2009) classification datasets. We run ablation studies to show that our different components are vital for the best performance. We then assess the ability of ANTs to automatically learn meaningful hierarchical structures in data. Next, we examine the effects of refinement phase on ANTs, and show that it can automatically prune the tree. Finally, we demonstrate that our proposed training procedure adapts the model size appropriately under varying amounts of labelled data. All of our models are implemented in PyTorch (Paszke et al., 2017)[1]. Full training details, including training times on a single GPU, are provided in Supp. Sec. C and D.

5.1. Model performance

We compare the performance of ANTs (Tab. 2) against a range of DT and NN models (Tab. 3), where notably the relative performance of these two classes of models differs between datasets. ANTs inherit from both and achieve the lowest error on SARCOS, and perform favourably on MNIST and CIFAR-10. In general, DT methods without feature learning, such as RFs (Breiman, 2001; Zhou & Feng, 2017) and GBTs (Ponomareva et al., 2017), perform poorly on image classification tasks (Krizhevsky & Hinton, 2009). In comparison with CNNs without shortcut connections (LeCun et al., 1998; Goodfellow et al., 2013; Lin et al., 2014; Springenberg et al., 2015), different ANTs balance between stronger performance with comparable numbers of trainable parameters, and comparable performance with smaller amount of parameters. At the other end of the spectrum, state-of-the-art NNs (Sabour et al., 2017; Huang et al., 2017) contain significantly more parameters.

Conditional computation: Tab.3 compares the errors and number of parameters of different ANTs for both multi-path and single-path inference schemes. While reducing the number of parameters (from Params (multi-path) to Params (single-path)) across all ANT models, we observe only a small difference in error (between Error (multi-path) and Error (single-path)), with the largest deviations being 0.06% for classification and 0.158 for regression. In addition, Supp. Sec. H shows that the single-path inference reduces FLOPS. This means that single-path inference gives an accurate approximation of the multi-path inference, while being more efficient to compute. This close approximation comes from the confident splitting probabilities of routers, being close to 0 or 1 (see blue histograms in Fig. 2(b)).

Ablation study: we compare the predictive errors of different variants of ANTs in cases where the options for adding transformer or router modules are disabled (see Tab. 4). In the first case, the resulting models are equivalent to SDTs (Suárez & Lutsko, 1999) or HMEs (Jordan & Jacobs, 1994) with locally grown architectures, while the second case is equivalent to standard CNNs, grown adaptively layer by layer. We observe that either ablation consistently leads to higher errors across different module configurations on all three datasets, justifying the combination of feature learning and hierarchical partitioning in ANTs.

[1]Codes: https://github.com/rtanno21609/AdaptiveNeuralTrees

Table 3: Comparison of performance of different models on SARCOS, MNIST and CIFAR-10. The columns "Error (multi-path)" and "Error (single-path)" indicate the classification (%) or regression (MSE) errors of predictions based on the multi-path and the single-path inference. The columns "Params. (multi-path)" and "Params. (single-path)" respectively show the total number of parameters in the model and the average number of parameters used during single-path inference. "Ensemble Size" indicates the size of ensemble used. An entry of "–" indicates that no value was reported. Methods marked with † are from our implementations trained in the same experimental setup. * indicates that the parameters are initialised with a pre-trained CNN.

	Method	Error (multi-path)	Error (single-path)	Params. (multi-path)	Params. (single-path)	Ensemble Size
SARCOS	Linear regression	10.693	N/A	154	N/A	1
	MLP with 2 hidden layers (Zhao et al., 2017)	5.111	N/A	31,804	N/A	1
	Decision tree	3.708	3.708	319,591	25	1
	MLP with 1 hidden layer	2.835	N/A	7,431	N/A	1
	Gradient boosted trees	2.661	2.661	391,324	2,083	7×30
	MLP with 5 hidden layers	2.657	N/A	270,599	N/A	1
	Random forest	2.426	2.426	40,436,840	4,791	200
	Random forest	2.394	2.394	141,540,436	16,771	700
	MLP with 3 hidden layers	2.129	N/A	139,015	N/A	1
	SDT (with MLP routers)	2.118	2.246	28,045	10,167	1
	Gradient boosted trees	1.444	1.444	988,256	6,808	7×100
	ANT-SARCOS	1.384	1.542	103,823	61,640	1
	ANT-SARCOS (ensemble)	1.226	1.372	598,280	360,766	8
MNIST	Linear classifier	7.91	N/A	7,840	N/A	1
	RDT (Léon & Denoyer, 2015)	5.41	–	–	–	1
	Random Forests (Breiman, 2001)	3.21	3.21	–	–	200
	Compact Multi-Class Boosted Trees (Ponomareva et al., 2017)	2.88	–	–	–	100
	Alternating Decision Forest (Schulter et al., 2013)	2.71	2.71	–	–	20
	Neural Decision Tree (Xiao, 2017)	2.10	–	1,773,130	502,170	1
	ANT-MNIST-C	1.62	1.68	39,670	7,956	1
	MLP with 2 hidden layers (Simard et al., 2003)	1.40	N/A	1,275,200	N/A	1
	LeNet-5† (LeCun et al., 1998)	0.82	N/A	431,000	N/A	1
	gcForest (Zhou & Feng, 2017)	0.74	0.74	–	–	500
	ANT-MNIST-B	0.72	0.73	76,703	50,653	1
	Neural Decision Forest (Kontschieder et al., 2015)	0.70	–	544,600	463,180	10
	ANT-MNIST-A	0.64	0.69	100,596	84,935	1
	ANT-MNIST-A (ensemble)	0.29	0.30	850,775	655,449	8
	CapsNet (Sabour et al., 2017)	0.25	–	8.2M	N/A	1
CIFAR-10	Compact Multi-Class Boosted Trees (Ponomareva et al., 2017)	52.31	–	–	–	100
	Random Forests (Breiman, 2001)	50.17	50.17	–	–	2000
	gcForest (Zhou & Feng, 2017)	38.22	38.22	–	–	500
	MaxOut (Goodfellow et al., 2013)	9.38	N/A	6M	N/A	1
	ANT-CIFAR10-C	9.31	9.34	0.7M	0.5M	1
	ANT-CIFAR10-B	9.15	9.18	0.9M	0.6M	1
	Network in Network (Lin et al., 2014)	8.81	N/A	1M	N/A	1
	All-CNN† (Springenberg et al., 2015)	8.71	N/A	1.4M	N/A	1
	ANT-CIFAR10-A	8.31	8.32	1.4M	1.0M	1
	ANT-CIFAR10-A (ensemble)	7.71	7.79	8.7M	7.4M	8
	ANT-CIFAR10-A*	6.72	6.74	1.3M	0.8M	1
	ResNet-110 (He et al., 2016)	6.43	N/A	1.7M	N/A	1
	DenseNet-BC (k=24) (Huang et al., 2017)	3.74	N/A	27.2M	N/A	1

SARCOS multivariate regression: Tab. 3 shows that ANT-SARCOS outperforms all other methods in mean squared error (MSE) with the full set of parameters. With the single-path inference, GBTs performs slightly better than a single ANT while requiring fewer parameters. We note that the top 3 methods are all tree-based, with the third best method being an SDT (with MLP routers). On the other hand, ANT and GBTs outperform the best standard NN model with less than a half of the parameter count. This highlights the value of hierarchical clustering for predictive performance and inference speed. Meanwhile, we still reap the benefits of representation learning, as shown by both ANT-SARCOS and the SDT (which is a specific form of ANT with identity transformers) requiring fewer parameters than the best-performing GBT configuration. Finally, we note that deeper NNs (5 vs. 3 hidden layers) can overfit on this small dataset, which makes the adaptive growth procedure of tree-based methods ideal for finding a model that exhibits good generalisation.

MNIST digit classification: we observe that ANT-MNIST-A outperforms state-of-the-art GBT (Ponomareva et al., 2017) and RF (Zhou & Feng, 2017) methods in accuracy. This performance is attained despite the use of a single tree, while RF methods operate with ensembles of classifiers (the size shown in Tab. 2). In particular, the NDF (Kontschieder et al., 2015) has a pre-specified architecture where LeNet-5 (LeCun et al., 1998) is used as the root transformer module, and 10 trees of fixed depth 5 are built on this base features. On the other hand, ANT-MNIST-A is constructed in a data-driven manner from primitive modules, and displays an improvement over the NDF both in terms of accuracy and number of parameters. In addition, reducing the size of convolution kernels (ANT-MNIST-B) reduces the total number of parameters by 25% and the path-wise average by almost 40% while only increasing the error by $< 0.1\%$.

We also compare against the LeNet-5 CNN (LeCun et al., 1998), comprised of the same types of operations used in our primitive modules (i.e. convolutional, max-pooling and

Table 4: Ablation study on regression (MSE) and classification (%) errors. "CNN" refers to the case where the ANT is grown without routers while "SDT/HME" refers to the case where transformer modules on the edges are disabled.

Model	Error (multi-path)			Error (single-path)		
	ANT (default)	CNN (no $\mathcal{R}$)	HME (no $\mathcal{T}$)	ANT (default)	CNN (no $\mathcal{R}$)	HME (no $\mathcal{T}$)
SARCOS	1.38	2.51	2.12	1.54	2.51	2.25
MNIST-A	0.64	0.74	3.18	0.69	0.74	4.19
MNIST-B	0.72	0.80	4.63	0.73	0.80	3.62
MNIST-C	1.62	3.71	5.70	1.68	3.71	6.96
CIFAR10-A	8.31	9.29	39.29	8.32	9.29	40.33
CIFAR10-B	9.15	11.08	43.09	9.18	11.08	44.25
CIFAR10-C	9.31	11.61	48.59	9.34	11.61	50.02

FC layers). For a fair comparison, the network is trained with the same protocol as that of the ANT refinement phase, achieving an error rate of 0.82%. Both ANT-MNIST-A and ANT-MNIST-B attain better accuracy with a smaller number of parameters than LeNet-5. The current state-of-the-art, capsule networks (CapsNets) (Sabour et al., 2017), have more parameters than ANT-MNIST-A by almost two orders of magnitude.[2] By ensembling ANTs, we can reach similar performance (0.29% versus 0.25%) with an order of magnitude less parameters (see Supp. Sec. I).

Lastly, we highlight the observation that ANT-MNIST-C, with the simplest primitive modules, achieves an error rate of 1.68% with single-path inference, which is significantly better than that of the linear classifier (7.91%), while engaging almost the same number of parameters ($7,956$ vs. $7,840$) on average. To isolate the benefit of convolutions, we took one of the root-to-path CNNs on ANT-MNIST-C and increased the number of kernels to adjust the number of parameters to the same value. We observe a higher error rate of 3.55%, which indicates that while convolutions are beneficial, data partitioning has additional benefits in improving accuracy. This result demonstrates the potential of ANT growth protocol for constructing performant models with lightweight inference. See Sec. G in the supplementary materials for the architecture of ANT-MNIST-C.

CIFAR-10 object recognition: we see that ANTs largely outperform the state-of-the-art DT method, gcForest (Zhou & Feng, 2017), achieving over 90% accuracy, demonstrating the benefit of representation learning in tree-structured models. Secondly, with fewer number of parameters in single-path inference, ANT-CIFAR-A achieves higher accuracy than CNN models without shortcut connections (Goodfellow et al., 2013; Lin et al., 2014; Springenberg et al., 2015) that held the state-of-the-art performance in respective years. With simpler primitive modules we learn more compact models (ANT-MNIST-B and -C) with a marginal compromise in accuracy. In addition, initialising the parameters of transformers and routers from a pre-trained single-path CNN further reduced the error rate of ANT-MNIST-A by 20% (see ANT-MNIST-A* in Tab. 3), indicating room for improvement in our proposed optimisation method.

Shortcut connections (Fahlman & Lebiere, 1990) have recently lead to leaps in performance in deep CNNs (He et al., 2016; Huang et al., 2017). We observe that our best network, ANT-MNIST-A*, has a comparable error rate and half the parameter count (with single-path inference) to the best-performing residual network, ResNet-110 (He et al., 2016). Densely connected networks have better accuracy, but with an order of magnitude more parameters (Huang et al., 2017). We expect shortcut connections to improve ANT performance, and leave integrating them to future work.

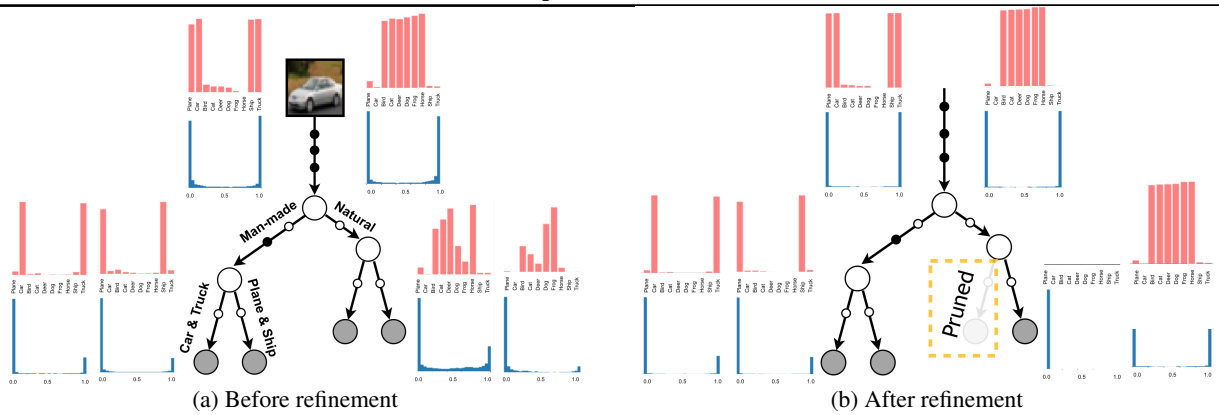
5.2. Interpretability

The growth procedure of ANTs is capable of discovering hierarchical structures in the data that are useful to the end task. Without any regularization imposed on routers, the learned hierarchies often display strong specialisation of paths to certain classes or categories of data on both the MNIST and CIFAR-10 datasets. Fig. 2 (a) displays an example with particularly "human-interpretable" partitions e.g. man-made versus natural objects, and road vehicles versus other types of vehicles. It should, however, be noted that human intuitions on relevant hierarchical structures do not necessarily equate to optimal representations, particularly as datasets may not necessarily have an underlying hierarchical structure, e.g., MNIST. Rather, what needs to be highlighted is the ability of ANTs to learn when to share or separate the representation of data to optimise end-task performance, which gives rise to automatically discovering such hierarchies. To further attest that the model learns a meaningful routing strategy, we also present the test accuracy of the predictions from the leaf node with the smallest reaching probability in Supp. Sec. F. We observe that using the least likely "expert" leads to a substantial drop in classification accuracy. In addition, most learned trees are unbalanced. This property of adaptive computation is plausible since certain types of images may be easier to classify than others, as seen in prior work (Figurnov et al., 2017).

5.3. Effect of refinement phase

We observe that global refinement phase improves the generalisation error. Fig. 3 (Right) shows the generalisation error of various ANT models on CIFAR-10, with vertical dotted lines indicating the epoch when the models enter the refinement phase. As we switch from optimising parts of the ANT in isolation to optimising all parameters, we shift the optimisation landscape, resulting in an initial drop in performance. However, they all consistently converge to higher test accuracy than the best value attained during the growth phase. This provides evidence that refinement phase remedies sub-optimal decisions made during the locally-optimised growth phase. In many cases, we observed that global optimisation polarises the decision probability of routers, which occa-

[2] Notably, CapsNets also feature a routing mechanism, but with a significantly different mechanism and motivation.



(a) Before refinement (b) After refinement

Figure 2: Visualisation of class distributions (red) and path probabilities (blue) computed over the whole test set at respective nodes of an example ANT (a) before and (b) after the refinement phase. (a) shows that the model captures an interpretable hierarchy, grouping semantically similar images on the same branches. (b) shows that the refinement phase polarises path probabilities, pruning a branch.

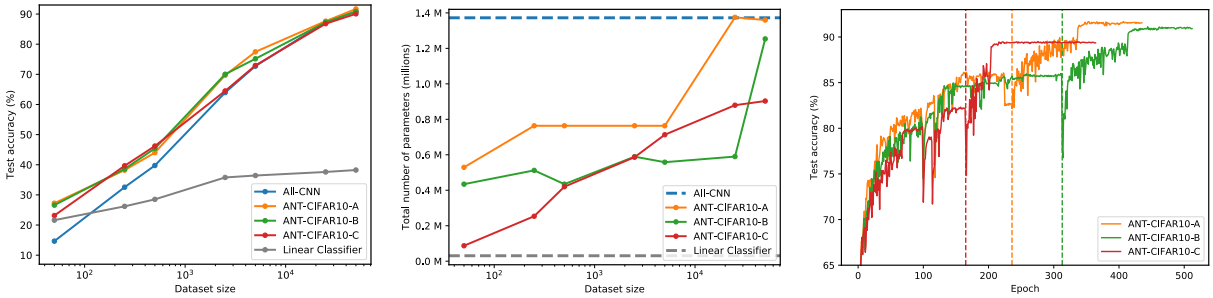


Figure 3: (Left). Test accuracy on CIFAR-10 of ANTs for varying amounts of training data. (Middle) The complexity of the grown ANTs increases with dataset size. (Right) Refinement improves generalisation; the dotted lines show where the refinement phase starts.

sionally leads to the effective "pruning" of some branches. For example, in the case of the tree shown in Fig. 2(b), we observe that the decision probability of routers are more concentrated near 0 or 1 after global refinement, and as a result, the empirical probability of visiting one of the leaf nodes, calculated over the validation set, reduces to 0.09%—meaning that the corresponding branch could be pruned without a negligible change in the network's accuracy. The resultant model attains lower generalisation error, showing the pruning has resolved a suboptimal partioning of data.

5.4. Adaptive model complexity

Overparametrised models, trained without regularization, are vulnerable to overfitting on small datasets. Here we assess the ability of our proposed ANT training method to adapt the model complexity to varying amounts of labelled data. We run classfication experiments on CIFAR-10 and train three variants of ANTs, All-CNN (Springenberg et al., 2015) and linear classifier on subsets of the dataset of sizes 50, 250, 500, 2.5k, 5k, 25k and 45k (the full training set). Here we choose All-CNN as the baseline as it has similar number of parameters when trained on the full dataset and is the closest in terms of constituent operations (convolutional, GAP and FC layers). Fig.3 (Left) shows the corresponding test performances. The best model is picked based on the performance on the same validation set of 5k examples as before. As the dataset gets smaller, the margin between the test accuracy of the ANT models and All-CNN/linear classifier increases (up to 13%). Fig. 3 (Middle) shows the model size of discovered ANTs as the dataset size varies. For different settings of primitive modules, the number of parameters generally increases as a function of the dataset size. All-CNN has a fixed number of parameters, consistently larger than the discovered ANTs, and suffers from overfitting, particularly on small datasets. The linear classifier, on the other hand, underfits to the data. Our method constructs models of adequate complexity, leading to better generalisation. This shows the value of our tree-building algorithm over using models of fixed-size structures.

6. Conclusion

We introduced Adaptive Neural Trees (ANTs), a holistic way to marry the architecture learning, conditional computation and hierarchical clustering of decision trees (DTs) with the hierarchical representation learning and gradient descent optimization of deep neural networks (DNNs). Our proposed training algorithm optimises both the parameters and architectures of ANTs through progressive growth, tuning them to the size and complexity of the training dataset. Together, these properties make ANTs a generalisation of previous work attempting to unite NNs and DTs. Finally, we validated the claimed benefits of ANTs for regression (SARCOS dataset) and classification (MNIST & CIFAR10 datasets), whilst still achieving high performance.

Acknowledgements

We would like to thank Konstantinos Kamnitsas, Sebastian Tschiatschek, Jan Stühmer, Katja Hofmann and Danielle Belgrave for their insightful discussions. Daniel C. Alexander is supported by the NIHR UCLH Biomedical Research Centre and EPSRC grants M020533, R006032, and R014019. Kai Arulkumaran is supported by a Phd scholaship from the Department of Bioengineering, Imperial College London. Ryutaro Tanno is supported by a Microsoft scholarship. Lastly, we would like to thank the anonymous reviewers for their valuable suggestions.

References

Bengio, Y. Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, pp. 1–37. Springer, 2013.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, 2013.

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Criminisi, A. and Shotton, J. *Decision forests for computer vision and medical image analysis*. Springer, 2013.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Fahlman, S. E. and Lebiere, C. The cascade-correlation learning architecture. In *Advances in neural information processing systems*, pp. 524–532, 1990.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D. P., and Salakhutdinov, R. Spatially adaptive computation time for residual networks. *CVPR*, pp. 1790–1799, 2017.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Frosst, N. and Hinton, G. E. Distilling a neural network into a soft decision tree. *CoRR*, 2017.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. In *ICML*, 2013.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. Densely connected convolutional networks. In *CVPR*, 2017.

Ioannou, Y., Robertson, D., Zikic, D., Kontschieder, P., Shotton, J., Brown, M., and Criminisi, A. Decision forests, convolutional networks and the models in-between. *CoRR*, 2016.

İrsoy, O., Yıldız, O. T., and Alpaydın, E. Soft decision trees. In *ICPR*, pp. 1819–1822. IEEE, 2012.

İrsoy, O., Yildiz, O. T., and Alpaydın, E. Budding trees. In *ICPR*, pp. 3582–3587. IEEE, 2014.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 1994.

Kaggle.com. Two sigma financial modeling challenge, 2017. URL https://www.kaggle.com/c/two-sigma-financial-modeling.

Kontschieder, P., Fiterau, M., Criminisi, A., and Rota Bulo, S. Deep neural decision forests. In *ICCV*, pp. 1467–1475, 2015.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

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

Le Folgoc, L., Nori, A. V., Ancha, S., and Criminisi, A. Lifted auto-context forests for brain tumour segmentation. In *International Workshop on Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pp. 171–183. Springer, 2016.

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

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436, 2015.

Léon, A. and Denoyer, L. Policy-gradient methods for decision trees. In *ESANN*, 2015.

Lin, M., Chen, Q., and Yan, S. Network in network. In *ICLR*, 2014.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *ICML*, pp. 807–814, 2010.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS Autodiff Workshop*, 2017.

Ponomareva, N., Colthurst, T., Hendry, G., Haykal, S., and Radpour, S. Compact multi-class boosted trees. In *International Conference on Big Data*, pp. 47–56, 2017.

Rota Bulo, S. and Kontschieder, P. Neural decision forests for semantic image labelling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 81–88, 2014.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pp. 3856–3866, 2017.

Sandulescu, V. and Chiru, M. Predicting the future relevance of research institutions-the winning solution of the kdd cup 2016. *CoRR*, 2016.

Schulter, S., Wohlhart, P., Leistner, C., Saffari, A., Roth, P. M., and Bischof, H. Alternating decision forests. In *CVPR, 2013*, 2013.

Shotton, J., Sharp, T., Kohli, P., Nowozin, S., Winn, J., and Criminisi, A. Decision jungles: Compact and rich models for classification. In *Advances in Neural Information Processing Systems*, pp. 234–242, 2013.

Simard, P. Y., Steinkraus, D., Platt, J. C., et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pp. 958–962, 2003.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for simplicity: The all convolutional net. *CoRR*, 2015.

Suárez, A. and Lutsko, J. F. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions. PAMI*, 21(12):1297–1311, 1999.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., et al. Going deeper with convolutions. In *CVPR*, 2015.

Vijayakumar, S. and Schaal, S. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *ICML*, volume 1, pp. 288–293, 2000.

Volkovs, M., Yu, G. W., and Poutanen, T. Content-based neighbor models for cold start in recommender systems. In *Proceedings of the Recommender Systems Challenge 2017*, pp. 7. ACM, 2017.

Xiao, H. Ndt: Neual decision tree towards fully functioned neural graph. *arXiv preprint arXiv:1712.05934*, 2017.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *ECCV*, pp. 818–833. Springer, 2014.

Zhao, H., Stretcu, O., Negrinho, R., Smola, A., and Gordon, G. Efficient multi-task feature and relationship learning. *arXiv preprint arXiv:1702.04423*, 2017.

Zhou, Z.-H. and Feng, J. Deep forest: Towards an alternative to deep neural networks. In *IJCAI*, 2017.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *ICLR*, 2017.