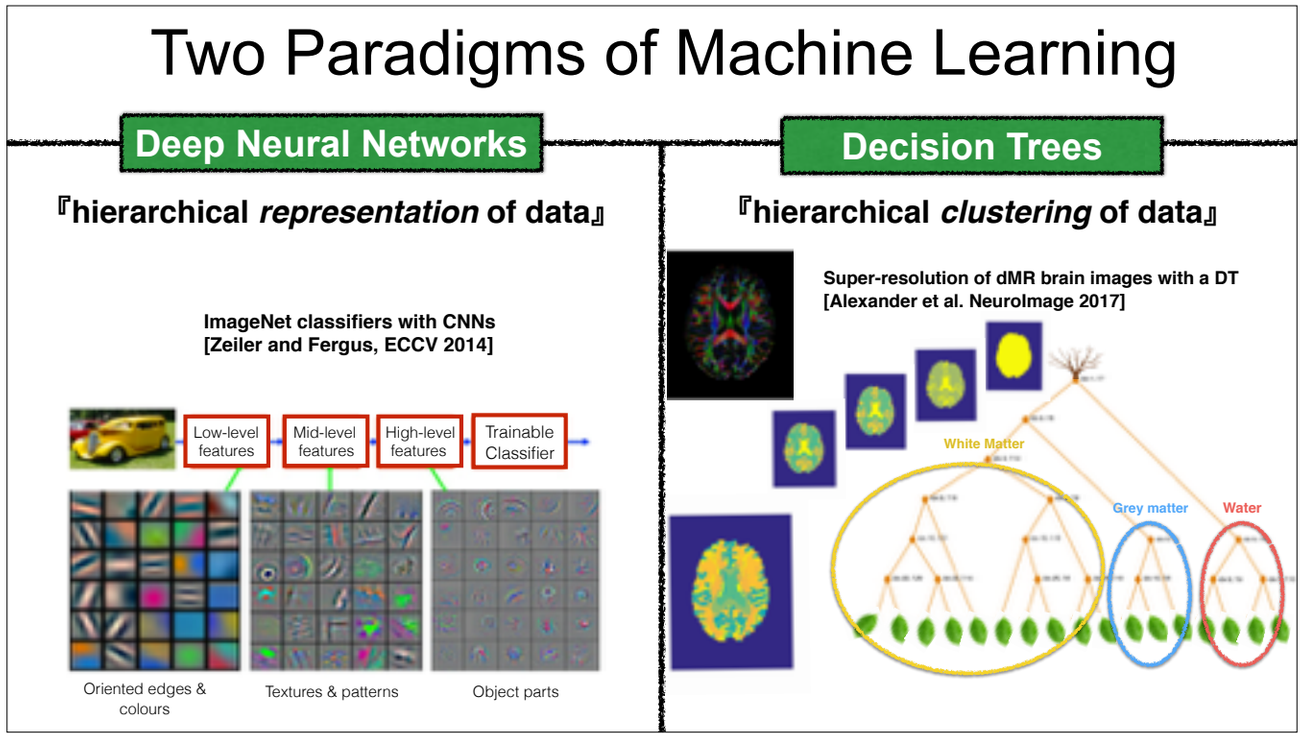
London Machine Learning Meetup
20 Feb 2019

How to Combine Neural Networks and Decision Trees

Ryutaro Tanno
University College London, UK

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

- In this presentation, I would like to talk about our recent attempt at combining DNNs and Decision Trees, in such a way that enjoys the benefits of the two approaches, while minimising their limitations.



First, I would like to start by illustrating the key difference between deep neural networks and decisions trees.
These two approaches operate on largely separate paradigms.

Typically, deep neural networks are characterised by learning hierarchical representation of data. In supervised learning setting, neural networks learn a complex non-linear transformations to apply to input data, so a simple linear model can adequately perform the predictive task. A commonly used illustrative example is the range of learned features of data at different layers in a CNN for image classification tasks, learning oriented edges/colours at the bottom, textures in the middle, and object parts at the top.

On the other hand, decision trees are characterised by hierarchical clustering of data. A decision tree learns how to split the input space so in each subset, a simple model such as linear model explains the data well. Here is an example from my previous work on an application in super-resolution of MR brain images. Here I assigned a colour to each subset of the partitioned input space. You can see that at the first split, brain tissues and water (CSF) are separated, and the next level, WM and GM are split.

Additional details:

- The next layer, we split the space into two by thresholding feature 7 (which is mean diffusivity) and the CSF is separated.
- Then, WM and GM are separated in the next layer by thresholding feature 6, isotropy. Fine tuning is done on the deeper layers.
- By looking at the last colour map, we can see that the tree managed to learn some meaningful structures within our brain for the task of regression.

Two Paradigms of Machine Learning

Deep Neural Networks

『hierarchical *representation* of data』

Decision Trees

『hierarchical *clustering* of data』

- These two approaches have their pros and cons

Two Paradigms of Machine Learning

Deep Neural Networks

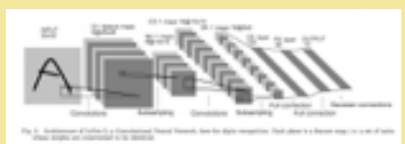
『hierarchical *representation* of data』

+ learn features of data

+ scalable learning with stochastic optimisation

- architectures are hand-designed

- heavy-weight inference, engaging every parameter of the model for each input



Decision Trees

『hierarchical *clustering* of data』

- A key advantage of neural networks is its ability to learn features or representation of raw input data. This means that unlike other machine learning algorithms, humans do not need to design good features for the algorithm to do well. Another advantage is scalable training with stochastic optimisation such as SGD. Consequently, with modern compute, you can train a very large network on an extremely large dataset, and this allows NNs to attain very good performance in many problems where you have large amount of data.

- **BUT!**
- architectures typically need to be designed by humans. Specifying a model at the right level of complexity could be a burden on practitioners and requires domain knowledge.
- also, inference can be heavy-weight for a very large model as each input data point engages every part of the network. In other words, increasing the capacity of the model requires proportional increase in computation.

Two Paradigms of Machine Learning

Deep Neural Networks

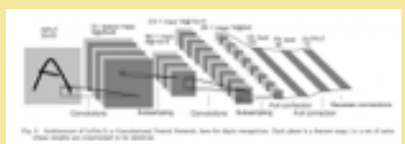
『hierarchical *representation* of data』

+ learn features of data

+ scalable learning with stochastic optimisation

- architectures are hand-designed

- heavy-weight inference, engaging every parameter of the model for each input



Decision Trees

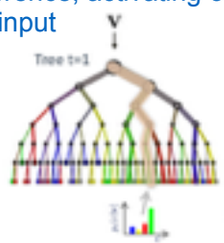
『hierarchical *clustering* of data』

- operate on hand-designed features

- limited expressivity with simple splitting functions

+ architectures are learned from data

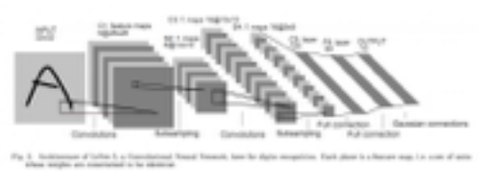
+ lightweight inference, activating only a fraction of the model per input



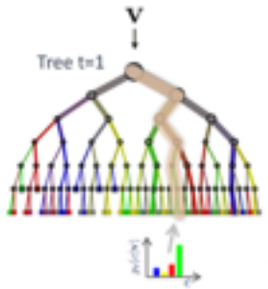
- On the other hand, the architectures of DTs are optimised based on training data, and display advantages particularly in data-scarce scenarios.
- Also, DTs often enjoy lightweight inference as only a single path on the tree is used for each data point.
- **BUT!**
- However, unlike DNNs, decision trees operate on hand-engineered features of data, which also require domain expertise. This is partly because the expressivity of the model is limited as the splitting functions often take the form of axis-aligned features. The cost function used for optimising the hard partioning of data is not differentiable, and this prevents us from using complex splitting functions in high-dimensional problems.

-Therefore, ensemble methods such random forests and gradient boosted trees are often used in more complicated problems, and are known to achieve state-of-the-art performance in many tasks.

Goal



+



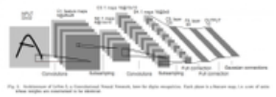
Our solution: ANTs (Adaptive Neural Trees)

The goal of this work to combine DNNs and DTs to design an algorithm that have the benefits of the two approaches and avoids their limitations.

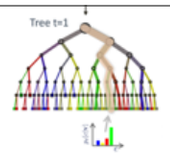
To this end, we propose Adaptive Neural Trees or ANTs in short.

Here, I would like to note that there is a large of body existing research that attempted the same unification. The main contribution of ANTs is to provide a simple yet general framework through which to think about the limitations of such previous works, and design something better.

Goal



+



- **ANTs enjoy the benefits of the two worlds:**

Deep Neural Networks	+ feature learning
	+ scalable optimisation with SGD
Decision Trees	+ architecture learning
	+ lightweight inference via conditional computation

- **ANTs for supervised learning & demonstrate on regression and classification.**

- **ANTs consist of two key designs:**

  (i). DTs which use NNs in every path and routing decisions.

  (ii). Back-propagation based training algorithm which grows the architecture.

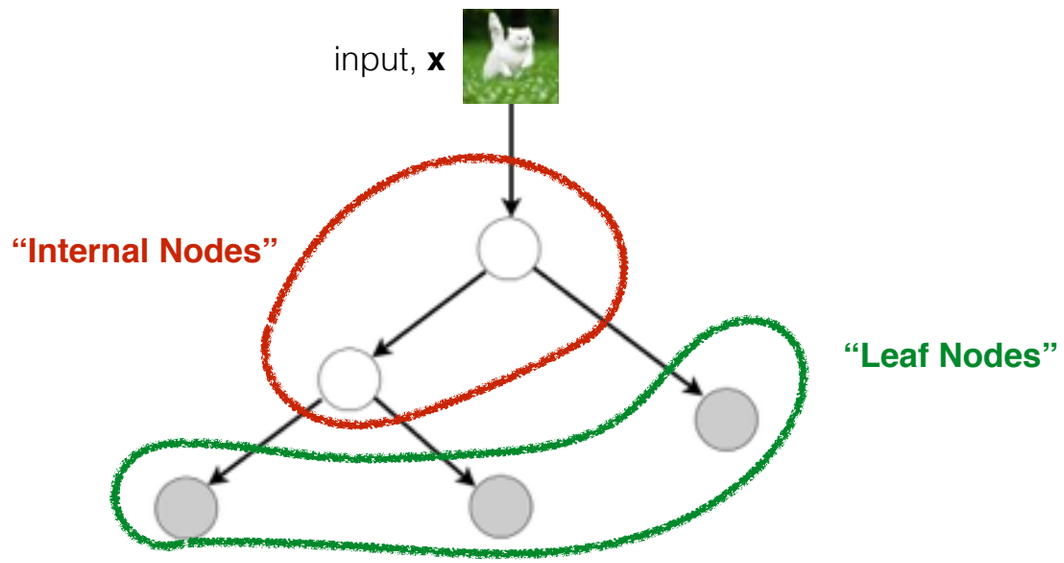
In particular, we would like to demonstrate that ANTs enjoy the following complementary benefits of the two approaches.
We formulate ANTs for supervised learning tasks and illustrate these properties on regression and image classification tasks.

We formulate ANTs for supervised learning tasks and illustrate these properties on regression and image classification tasks.

ANTs are comprised of two key features:
(1) Firstly, we propose a novel form of DTs, which use neural networks (NNs) in both its edges and nodes in the tree structure, combining the abilities of representation learning with hierarchical data partitioning.
(2) Secondly, we propose a back-propagation based training algorithm which grows the architecture in a progressive manner from simple building blocks.
We now describe each of these aspects in more detail.

Definition of ANTs

•*A **binary tree** with a set of operations*:



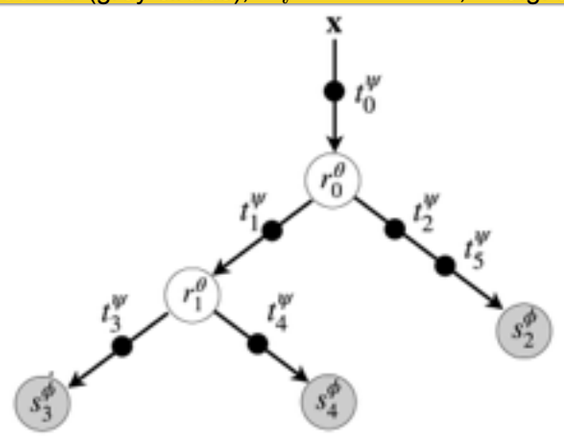
An ANT is defined as a form of binary trees with a set of NN operations defined on it.

Here is an example of a binary tree.
Every node is the child of exactly one parent except the root node at the top which has no parent. The nodes with no children at the end of the tree are called leaf nodes shown as grey circles, and the remaining white circles are called internal nodes, each of which has exactly two children.

Definition of ANTs

•*A **binary tree** with **3 types** of operations*:

internal node	= a ***router*** (white circles), $r_j^\theta : \mathcal{X}_j \to [0,1]$	e.g. a small CNN
edge	= one or multiple **transformer** (black circles) t^ψ,	e.g. 1 Conv. + ReLU
leaf node	= a ***solver*** (grey circles), $s_l^\phi : \mathcal{X}_l \to \mathcal{Y}$,	e.g. a linear classifier



Every node and edge in the tree is assigned with a operation. In particular, we consider the following 3 types of operations:
1. routers; 2. transformers; 3. solvers

Every internal node has a **router module,** which is a binary classifier which sends the allocated samples to the left or the right child with probability given by the value of the function. X_{j} denotes the feature space at node j.

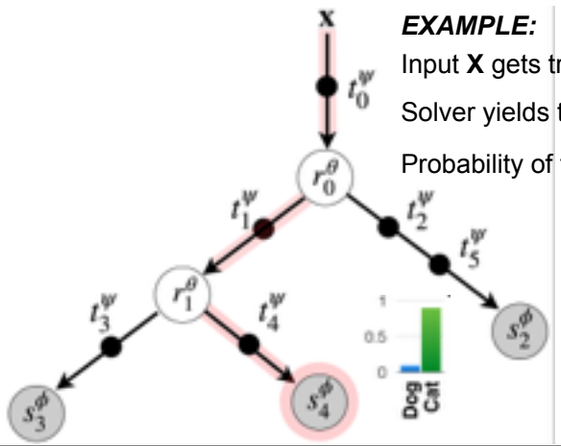
Edge of the tree has one or a composition of multiple so-called **transformer modules.** Each transformer is a non-linear function which transforms the incoming samples and pass them to the next operation.

Lastly, each leaf node is assigned with a **solver module** which takes the allocated transformed input and performs the predictive task of interest.

Definition of ANTs

•*A **binary tree** with **3 types** of operations*:

internal node	= a ***router*** (white circles),	$r_j^\theta : \mathcal{X}_j \to [0,1]$ e.g. a small CNN
edge	= one or multiple **transformer** (black circles)	t^ψ, e.g. 1 Conv. + ReLU
leaf node	= a ***solver*** (grey circles),	$s_l^\phi : \mathcal{X}_l \to \mathcal{Y}$, e.g. a linear classifier



EXAMPLE:

Input **X** gets transformed: $\mathbf{x} \to \mathbf{x}_0^\psi := t_0^\psi(\mathbf{x}) \to \mathbf{x}_1^\psi := t_1^\psi(\mathbf{x}_0^\psi) \to \mathbf{x}_4^\psi := t_4^\psi(\mathbf{x}_1^\psi)$

Solver yields the prediction: $p_4^{\phi,\psi}(\mathbf{y}) := s_4^\phi(\mathbf{x}_4^\psi)$

Probability of taking this path is: $\pi_4^{\psi,\boldsymbol{\theta}}(\mathbf{x}) := r_0^{\boldsymbol{\theta}}(\mathbf{x}) \cdot (1 - r_1^{\boldsymbol{\theta}}(\mathbf{x}_1^\psi))$

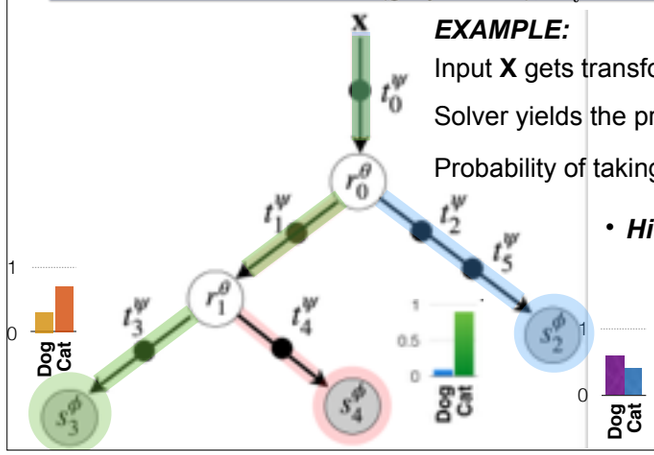
To give you an example, the red shaded path shows the routing of this cat image to reach leaf node 4 where the class probabilities are predicted.

As the input x traverses the path, it undergoes a sequence of non-linear transformations on the edges en route, and solver module at the destination leaf node takes this transformed input and generates a predictive distribution. The probability of selecting this particular path is given by the product of the respective routing probabilities.

Definition of ANTs

- *A **binary tree** with **3 types** of operations*:

internal node	= a ***router*** (white circles), $r_j^\theta : \mathcal{X}_j \to [0,1]$	e.g. a small CNN
edge	= one or multiple **transformer** (black circles) t^ψ,	e.g. 1 Conv. + ReLU
leaf node	= a ***solver*** (grey circles), $s_l^\phi : \mathcal{X}_l \to \mathcal{Y}$,	e.g. a linear classifier



EXAMPLE:

Input **X** gets transformed: $\mathbf{x} \to \mathbf{x}_0^\psi := t_0^\psi(\mathbf{x}) \to \mathbf{x}_1^\psi := t_1^\psi(\mathbf{x}_0^\psi) \to \mathbf{x}_4^\psi := t_4^\psi(\mathbf{x}_1^\psi)$

Solver yields the prediction: $p_4^{\phi,\psi}(\mathbf{y}) := s_4^\phi(\mathbf{x}_4^\psi)$

Probability of taking this path is: $\pi_4^{\psi,\theta}(\mathbf{x}) := r_0^\theta(\mathbf{x}) \cdot (1 - r_1^\theta(\mathbf{x}_1^\psi))$

- ***Hierarchical Mixture of Experts** (Jordan 1994)*

$$p(\mathbf{y}|\mathbf{x},\theta,\phi,\psi) = \sum_{l\in\{2,3,4\}} \pi_l^{\psi,\theta}(\mathbf{x}) \cdot p_l^{\phi,\theta}(\mathbf{x})$$

Full predictive distribution | **Reaching probability** | **Leaf prediction**

Each ANT effectively models the conditional distribution p(y|x) over output given input as a type of hierarchical mixture of expert NNs.

The routing is stochastic in this case, and there is a non-zero probability of the input x to reach other leaf nodes such as leaf 2 or leaf 3.

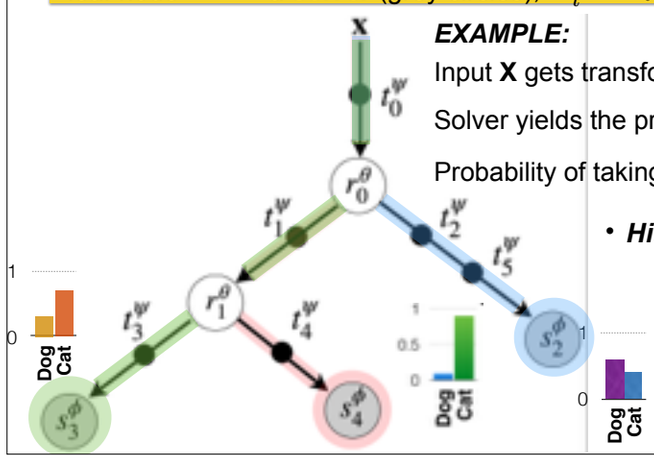
And the full-predictive distribution is given as the average of respective leaf predictions weighted by the probability of reaching the leaf node.

The key difference with traditional HMEs is that the input is not only routed but also transformed within the tree hierarchy

Definition of ANTs

•*A **binary tree** with **3 types** of operations*:

internal node	= a ***router*** (white circles), $r_j^\theta : \mathcal{X}_j \to [0,1]$	e.g. a small CNN
edge	= one or multiple **transformer** (black circles) t^ψ,	e.g. 1 Conv. + ReLU
leaf node	= a ***solver*** (grey circles), $s_l^\phi : \mathcal{X}_l \to \mathcal{Y}$,	e.g. a linear classifier



EXAMPLE:

Input **X** gets transformed: $\mathbf{x} \to \mathbf{x}_0^\psi := t_0^\psi(\mathbf{x}) \to \mathbf{x}_1^\psi := t_1^\psi(\mathbf{x}_0^\psi) \to \mathbf{x}_4^\psi := t_4^\psi(\mathbf{x}_1^\psi)$

Solver yields the prediction: $p_4^{\phi,\psi}(\mathbf{y}) := s_4^\phi(\mathbf{x}_4^\psi)$

Probability of taking this path is: $\pi_4^{\psi,\theta}(\mathbf{x}) := r_0^\theta(\mathbf{x}) \cdot (1 - r_1^\theta(\mathbf{x}_1^\psi))$

- ***Hierarchical Mixture of Experts*** *(Jordan 1994)*

$$p(\mathbf{y}|\mathbf{x},\theta,\phi,\psi) = \sum_{l\in\{2,3,4\}} \pi_l^{\psi,\theta}(\mathbf{x}) \cdot p_l^{\phi,\theta}(\mathbf{x})$$

Full predictive distribution | **Reaching probability** | **Leaf prediction**

- ***Two inference schemes:***
  (1) Multi-path & (2) Single-path

- We consider **two inference schemes**, based on a trade-off between accuracy and computation.
- **Multi-path inference** uses the full predictive distribution as the final estimate. However, computing this quantity re- quires averaging the distributions over all the leaves involving computing all operations at all nodes and edges of the tree.
- On the other hand, **single-path inference** only uses the predictive distribution from a single chosen leaf node. The single path inference selects the leaf by greedily traversing the tree in the directions of highest confidence of the routers.
- In other words, you compute the probability of router's output, and send it to the right if its below 0.5, otherwise to the left.
- This approximation constrains computations to a single path, allowing for more memory- and time-efficient inference.

Empirically, as the routers decisions tend to be confident, the single-path inference approximates the multi-path inference well and the accuracy is only marginally compromised.

Optimisation of ANTs

- **Training of an ANT proceeds in two phases:**
  1). *Growth*: grow the architecture based on *local optimisation*.
  2) *Refinement*: fine-tunes based on *global optimisation*.

Now, we describe how each ANT is grown and optimised.

- Training of an ANT proceeds in two phases, namely growth phase and fine-tuning phase.
- The growth phase progressively grows the architecture of the model based on local optimisation.
- The refinement phase fine-tunes the parameters of the grown model, based on global-optimisation.

Optimisation of ANTs

- **Training of an ANT proceeds in two phases:**
  1). *Growth*: grow the architecture based on *local optimisation*.
  2) *Refinement*: fine-tunes based on *global optimisation*.
- **Objective function:** negative log likelihood (NLL)

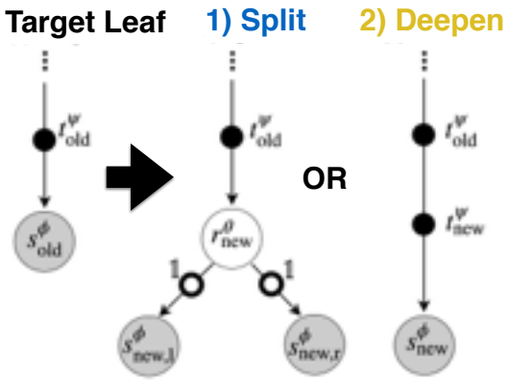
$$-\log\, p(\mathbf{Y}|\mathbf{X},\boldsymbol{\theta},\boldsymbol{\psi},\boldsymbol{\phi}) = -\sum_{n=1}^{N}\log\Big(\sum_{l=1}^{L}\pi_l^{\boldsymbol{\theta},\boldsymbol{\psi}}(\mathbf{x}^{(n)})\cdot p_l^{\boldsymbol{\phi},\boldsymbol{\psi}}(\mathbf{y}^{(n)}|\mathbf{x}^{(n)})\Big)$$

For both phases, we use negative log-likelihood as the common objective. In this case, we assume that all the component modules e.g. routers, transformers and solvers are differentiable, we can use gradient-based optimisation. As is typically the case with NNs, we use back propagation for gradient computation and use gradient descent to minimise the NLL.

The second phase, “refinement phase” takes the ANT of architecture learned from the growth phase, and tunes the parameters of the all component modules by minimising this loss function.

Now, how does the growth phase work?

Optimisation of ANTs



- **Training of an ANT proceeds in two phases:**
  1). *Growth*: grow the architecture based on *local optimisation*.
  2) *Refinement*: fine-tunes based on *global optimisation*.

- **Objective function:** negative log likelihood (NLL)

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

- **Details of growth phase:** at each leaf node,
  (I). construct **2** models: one with a new router (**Split**) and the other with a new transformer (**Deepen**)
  (II). optimise new modules (locally) by minimising NLL
  (III). pick the best choice

Repeat until no more **"Split"** or **"Deepen"** pass the test.

- Growth phase repeatedly applies the following sequence of 3 steps until some termination criteria are met.
- Starting from the root node, we choose one of the leaf-nodes in breadth first order and we construct two new models.
- 1st model extends the current model by splitting the node by adding a router, and we refer to this extension as "Split".
- 2nd model increase the depth of the incoming edge by adding a new transformer, which we refer to as "Deepen".
- Step (II) optimises the parameters of the newly added modules, while fixing the parameters of the remaining part of the graph.
- Step (III) executes the option with lowest validation loss. If the loss is reduced by neither extensions, we keep the original leaf node as it is.
- This process is repeated to all new nodes level-by-level until no more splits or edge extensions pass the validation test.
- The rationale behind this scheme is to give the model a freedom to choose the most effective option between "going deeper" or partioning the data space.
- It is also worth noting that local optimisation is both space- and time-efficient. This is because gradient computations are constrained to the new peripheral part of the tree and thus forward activations except the target leaf nodes do not need to be stored in memory.

(Refer to the diagram on the left), splitting a node is equivalent to a soft partioning 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. Edge extension on the other hand does not change the number of leaf nodes, but instead seeks to learn richer representation via an extra non-linear transformation, and replaces the old solver with a new one.

Also, for each of the constructed new models be the test option either split or extend, we perform this local optimisation for relatively small number of epochs (e.g. up to 25 on MNIST and CIFAR10). This is motivated by the observation that the performance of a network early in training provides a meaningful indication of performance after convergence. We observed that this approach is not only more efficient but also achieves a more accurate model than the standard greedy protocol used in decision tree training where local optimisation is performed until convergence.

Generalisation of previous work

	Feature learning?		Adaptive architecture?
	Routers	Path	
Soft DT, (Suarez & Lutsko, 1999)			
Soft DT 2 / HMEs (Jordan & Jacobs, 1994)			
Soft DT 3, (Frosst & Hinton, 2017)			
Convolutional DT, (Laptev & Buhman, 2014)			
Neural Decision Tree (Kontschieder et al., 2014)			
Neural Decision Forest (Kontschieder et al., 2015)			
Conditonal Net (Ioanou et al., 2016)			
ANT (Ours)			

- This will be the last slide before moving on to the experiments and results.
- As I mentioned at the start, there is a large body of previous work which attempted combining the decision trees and neural networks.
- Here I am going to give a quick survey of such works, and explain how ANTs provide a simple generalisation and address their limitations.
- The table on the slide compares ANTs against the previously introduced tree-structured models. The first column denotes if the routers learn features from data, and the second column indicates if each root-to-leaf path on the tree is a NN. The last column indicates if the method grows an architecture, or uses a pre-specified one.

Generalisation of previous work

	Feature learning?		Adaptive architecture?
	Routers	Path	
Soft DT, (Suarez & Lutsko, 1999)	✗	✗	✓
Soft DT 2 / HMEs (Jordan & Jacobs, 1994)			
Soft DT 3, (Frosst & Hinton, 2017)			
Convolutional DT, (Laptev & Buhman, 2014)			
Neural Decision Tree (Kontschieder et al., 2014)			
Neural Decision Forest (Kontschieder et al., 2015)			
Conditonal Net (Ioanou et al., 2016)			
ANT (Ours)			

Routers: axis-aligned features
Transformers: Identity
No representation learning ✗

(Show Grey box): The very fist soft decision tree introduced in 1999 (Suarez & Lutsko, 1999) is a specific case where in our terminology the routers are thresholding on hand-engineered features (axis-aligned features), the transformers are identity function, and the routers are static distributions over classes or linear functions.

Generalisation of previous work

	Feature learning?		Adaptive architecture?
	Routers	Path	
Soft DT, (Suarez & Lutsko, 1999)	✗	✗	✓
Soft DT 2 / HMEs (Jordan & Jacobs, 1994)	✓	✗	✗
Soft DT 3, (Frosst & Hinton, 2017)	✓	✗	✗
Convolutional DT, (Laptev & Buhman, 2014)	✓	✗	✗
Neural Decision Tree (Kontschieder et al., 2014)	✓	✗	✓
Neural Decision Forest (Kontschieder et al., 2015)			
Conditonal Net (Ioanou et al., 2016)			
ANT (Ours)			

Routers: axis-aligned features
Transformers: Identity
No representation learning ✗

Routers: Linear classifier, MLPs, Conv. + Linear classifier
Transformers: Identity
No representation learning ✗

(Show Blue box): More modern DTs used linear classifier, MLPs or convolution layer in the routing functions to learn more complex partitioning of the input space. However, the simplicity of identity transformers means that each path on the tree does not perform representation learning, limiting their performance.

Generalisation of previous work

	Feature learning?		Adaptive architecture?
	Routers	Path	
Soft DT, (Suarez & Lutsko, 1999)	✗	✗	✓
Soft DT 2 / HMEs (Jordan & Jacobs, 1994)	✓	✗	✗
Soft DT 3, (Frosst & Hinton, 2017)	✓	✗	✗
Convolutional DT, (Laptev & Buhman, 2014)	✓	✗	✗
Neural Decision Tree (Kontschieder et al., 2014)	✓	✗	✓
Neural Decision Forest (Kontschieder et al., 2015)	✓	✓	✗
Conditonal Net (Ioanou et al., 2016)			
ANT (Ours)			

Routers: axis-aligned features
Transformers: Identity
No representation learning ✗

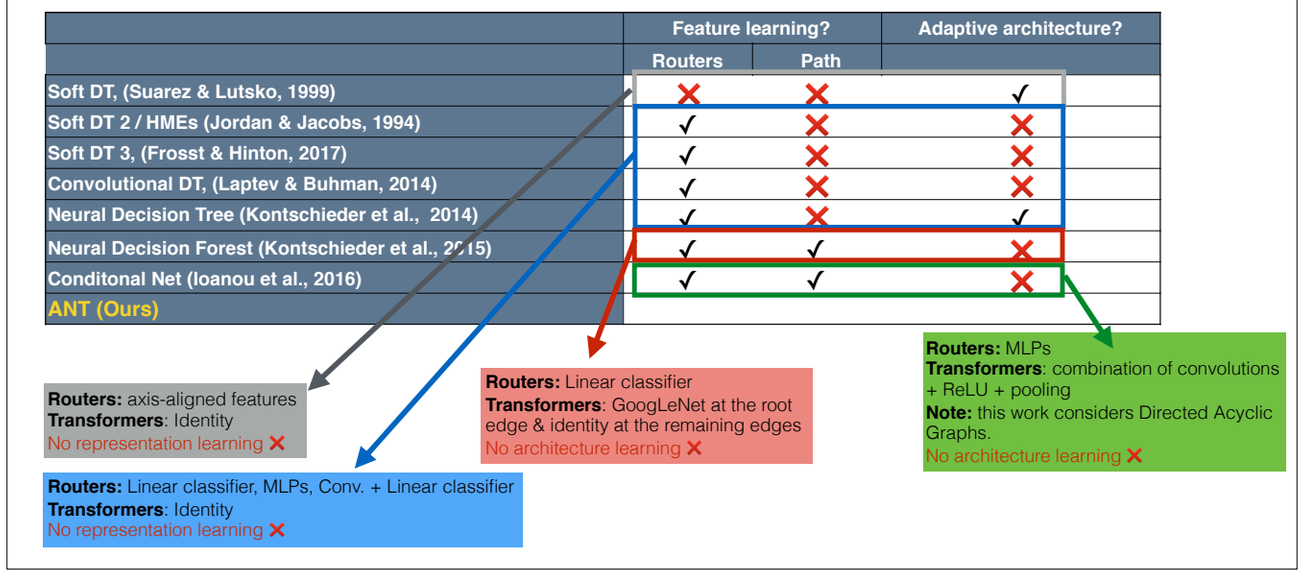
Routers: Linear classifier
Transformers: GoogLeNet at the root edge & identity at the remaining edges
No architecture learning ✗

Routers: Linear classifier, MLPs, Conv. + Linear classifier
Transformers: Identity
No representation learning ✗

Previous work has also suggested that integrating non-linear transformations of data into DTs enhance model performance.

(Show Red box): the neural decision forest (NDF), which held cutting-edge performance on ImageNet in 2015, is an ensemble of DTs, each of which is also an instance of ANTs where the whole GoogLeNet architecture (except for the last linear layer) is used as the root transformer, prior to learning tree-structured classifiers with linear routers.

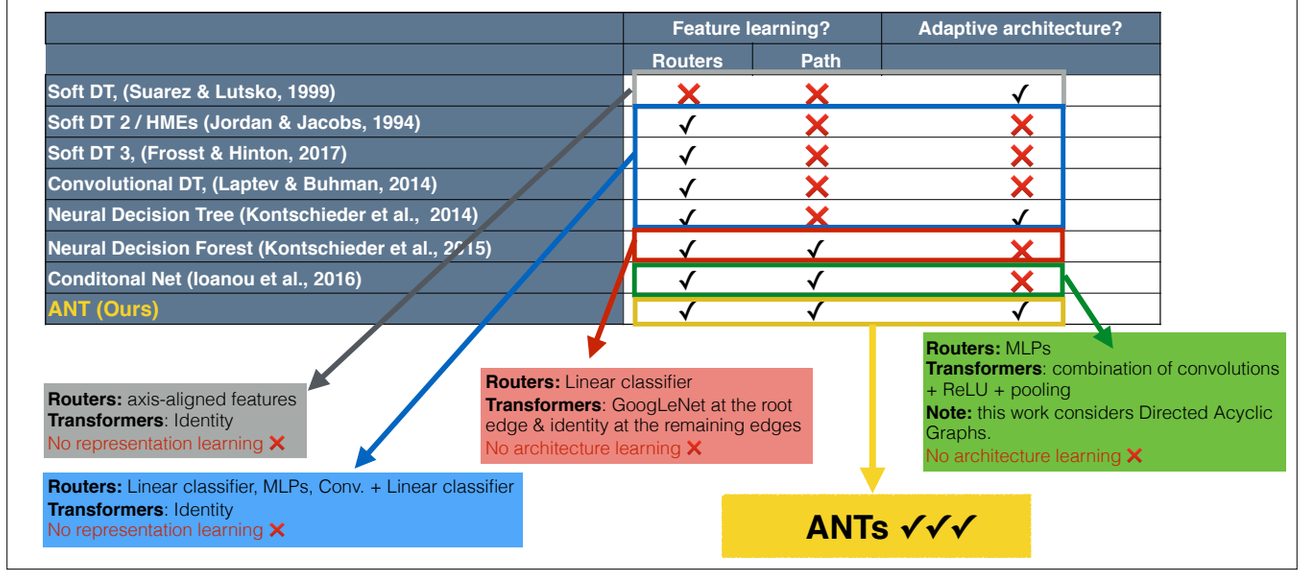
Generalisation of previous work



(Show Green box): the conditional network proposed in (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 both cases the model architectures are pre-specified!

Generalisation of previous work



(Show Yellow box): In contrast, ANTs provides a simple general framework for learning a tree-structured model with the capacity of representation learning along each path and routing decisions, and of optimising its architecture.

———————

ANTs provides a very simple abstraction which elucidates the limitations of these works.

TODO (?): should we consider making connections to neural architecture search and conditional computation?

Experiments & Results

Experiments

- **Data:** SARCOS (regression) & MNIST + CIFAR-10 (classification)

- Now we'd like to evaluate the performance of ANTs on regression and classification benchmarks.
- **SARCOS** is a regression dataset, and the task is inverse dynamics problem where you want to predict the torques of a robotic arm from its position, velocity and acceleration.
- **MNIST** is a classification dataset consisting of hand-written digits from 0 to 9.
- **CIFAR-10** is a object recognition dataset which include natural images of different animals and automobiles.

Experiments

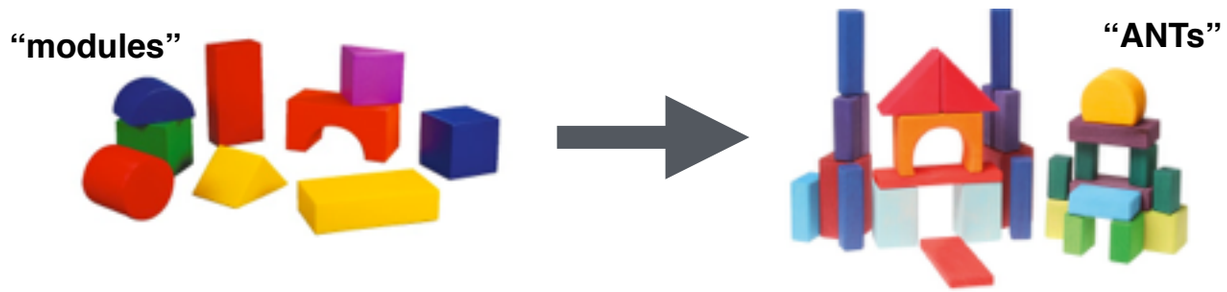
- **Data:** SARCOS (regression) & MNIST + CIFAR-10 (classification)
- **Analysis:**
    1. Comparison with relevant DT and NN models
    2. Learned hierarchical structures
    3. Adaptive model complexity
    4. Effect of global refinement

- In summary, we perform the following experiments.
- Firstly, we compare the performance of ANTs to relevant DT and NN models on these datasets.
- Secondly, we illustrate the ability of ANTs to learn hierarchical structures in data with an example.
- Thirdly, we examine how our proposed training procedure adapts the model size to varying amounts of training data.
- And lastly, we look into the effects of global refinement step on ANTs and show that it can automatically prune the tree.

Experiments

- **Models:** choices of *building blocks*

Model	Router, $\mathcal{R}$	Transformer, $\mathcal{T}$	Solver, $\mathcal{S}$	Downsample Freq.
ANT-MNIST-A	1 × conv5-40 + GAP + 2×FC	1 × conv5-40	LC	1
ANT-MNIST-B	1 × conv3-40 + GAP + 2×FC	1 × conv3-40	LC	2
ANT-MNIST-C	1 × conv5-5 + GAP + 2×FC	1 × conv5-5	LC	2
ANT-CIFAR10-A	2 × conv3-128 + GAP + 1×FC	2 × conv3-128	LC	1
ANT-CIFAR10-B	2 × conv3-96 + GAP + 1×FC	2 × conv3-96	LC	1
ANT-CIFAR10-C	2 × conv3-72 + GAP + 1×FC	2 × conv3-72	GAP + LC	1
ANT-SARCOS	1× MLP ($h = 128$)	1× FC ($h = 128$) + tanh	LR	0



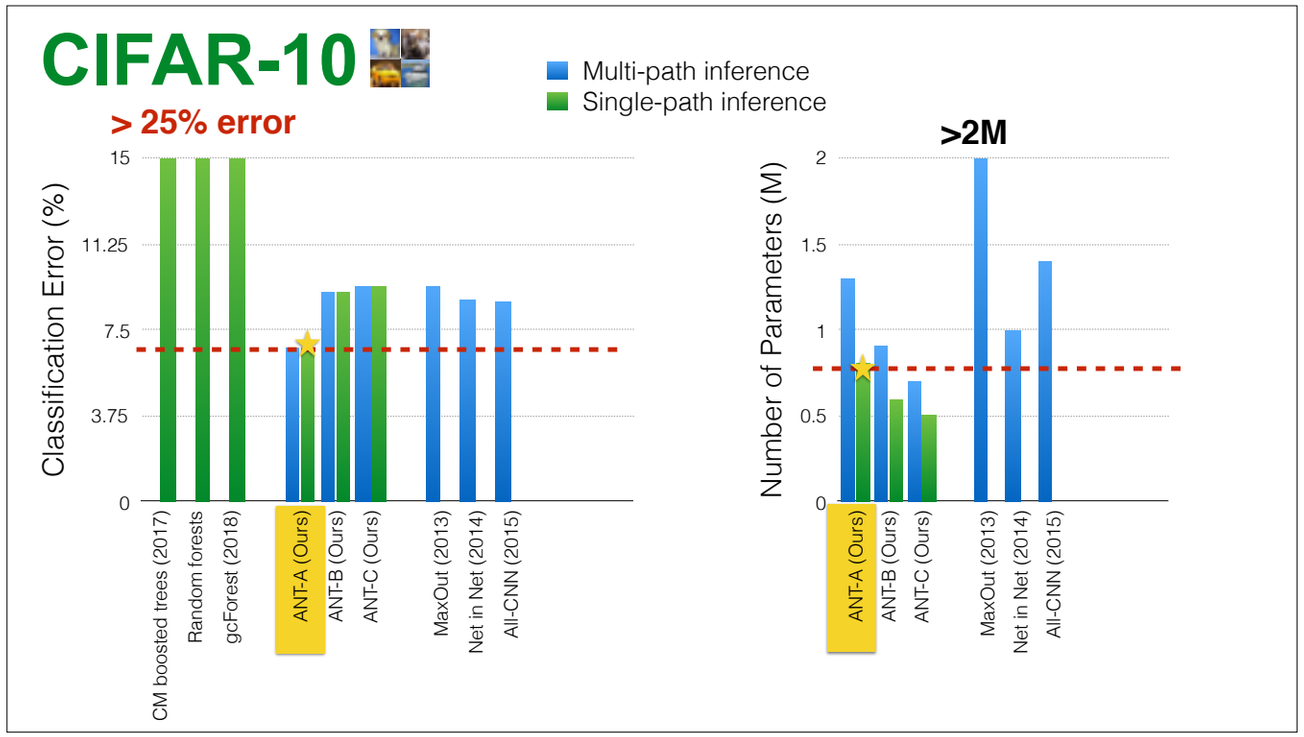
- We trained variants of ANTs with different primitive module specifications for each dataset.
- For simplicity, we define primitive modules based on three types of NN layers: convolutional, global-average-pooling (GAP) and fully-connected (FC).
- Solvers are always defined as linear models.
- Routers are binary classifiers with a sigmoid output.
- (Optional) For image classification, we also downsample features with the specified frequency by using max-pooling. For example, in the case of MNIST, we apply pooling after every two transformer modules.

- For the training of all ANTs, we used ADAM.
- Before going into the numerical results on these datasets, I am going show different properties of ANTs …

- Now, I will talk about the quantitative results on 3 datasets, which I skipped past earlier.
- I will start with the SARCOS dataset, where the task is to predict 7 dimensional torques from 21 dimensional inputs.
- in this bar chart, the blue bars are showing the errors for multi-path inference while the green ones show the corresponding results from the single-path inference.
- Firstly, on SARCOS dataset, the top 3 models are all tree-based, with NDT, GBTs and ANT achieving the lowest MSE.

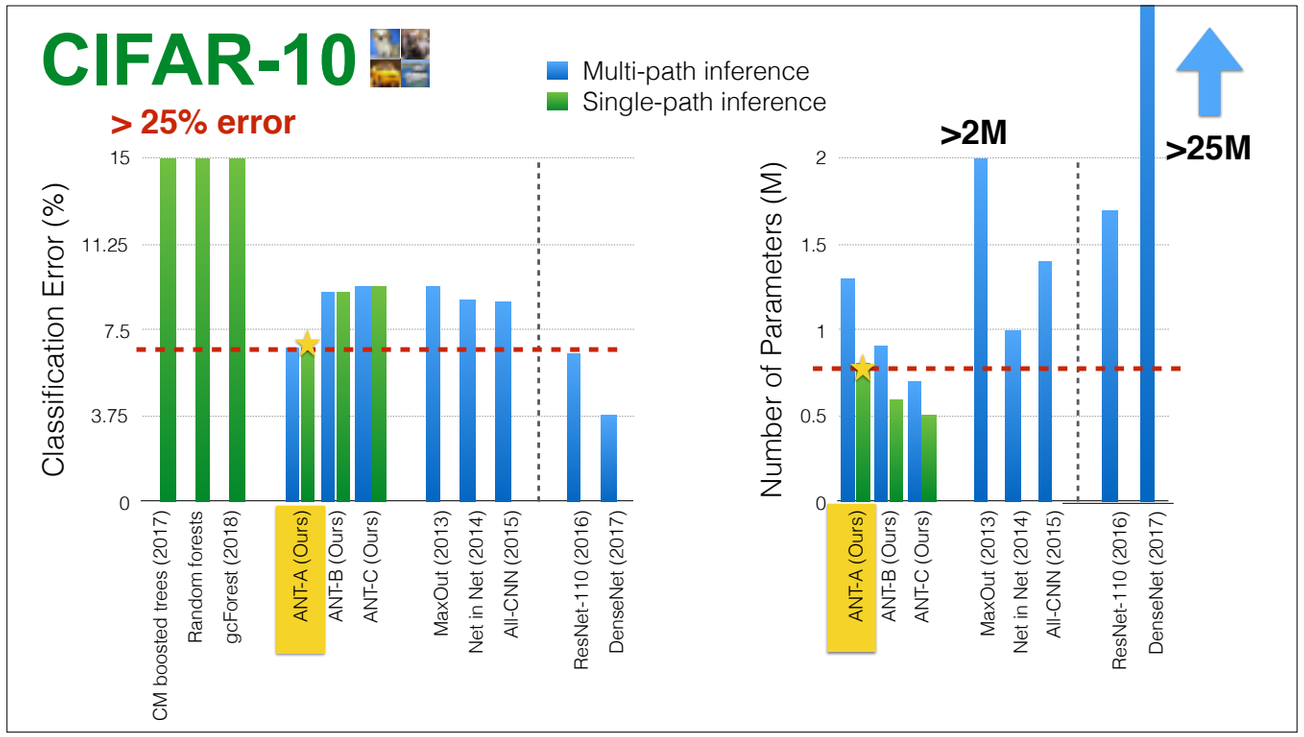
- This bar chart on the right shows the corresponding number of parameters on logarithmic scale, and the red-dotted line shows the number of parameters of ANT for single-path inference.
- ANT requires less than a half of what's required for the best standard NN model.
- On the other hand, Gradient Boosted trees only need a magnitude less parameters.
- We should however note that the total number of parameters for standard trees models are significantly larger than that of an ANT.

- All in all, ANT achieves the best accuracy on SARCOS beating both standard NNs and standard tree-based methods. Secondly, in terms of computational efficiency, ANTs is somewhere between NNs and Gradient Boosted Trees.
- This highlights the value of hierarchical clustering of the input space for both predictive accuracy and speed.

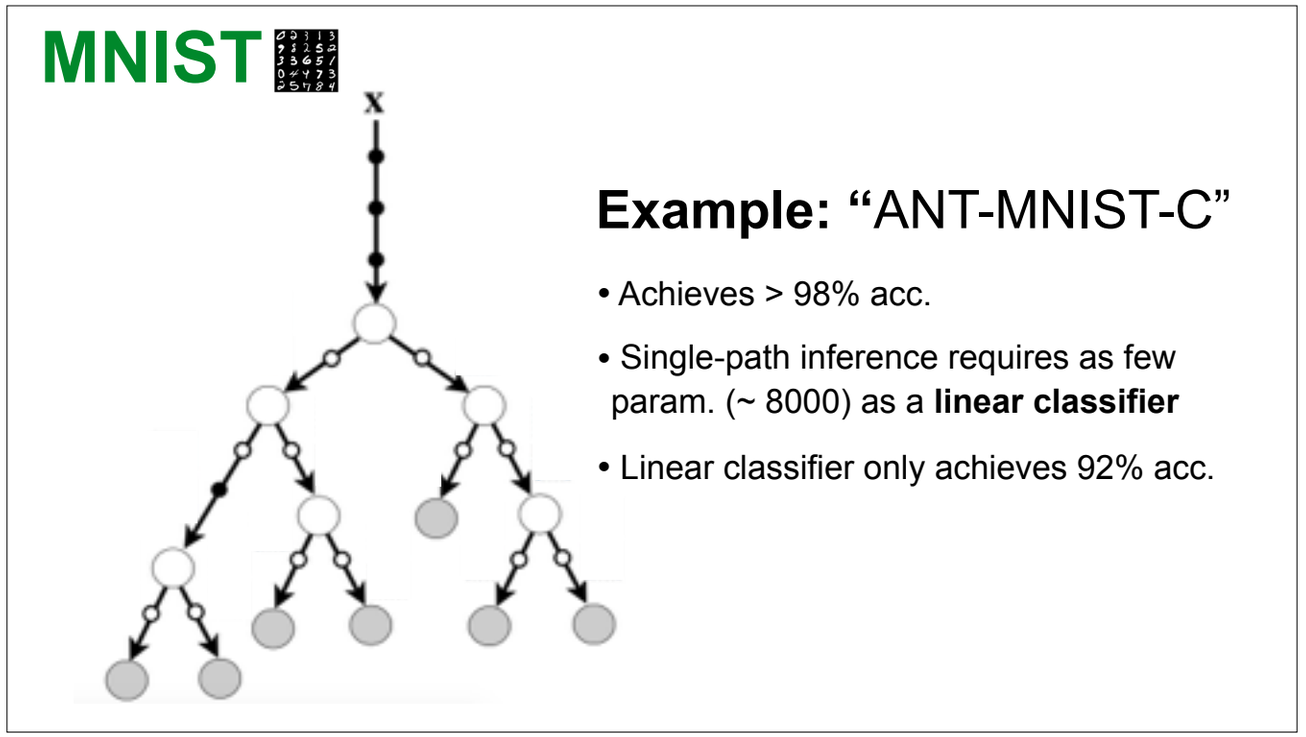


- Firstly, ANTs far outperform recent DT methods by a large margin, achieving over 90% accuracy, demonstrating the benefit of representation learning in tree-structured models.
- We did a similar comparison for CIFAR-10 object recognition dataset
- Secondly, ANTs show competitive performance wrt relatively recent CNN models, such as MaxOut, Net in Net, All-CNN, all of which held the state-of-the-art performance at the time of publications.
- In particular, as shown by the red dotted line, ANT-A archives the lowest classification error of 6.5%.

- Also, **ANT-A** requires fewer parameters during single-path inference than these CNN classifiers, showing improved efficiency.
- Here I also want to note these CNN models have linear structures with up to 15 layers, with similar set of operations to the components of ANT modules (e.g. convolutions, GAP and FC layers).

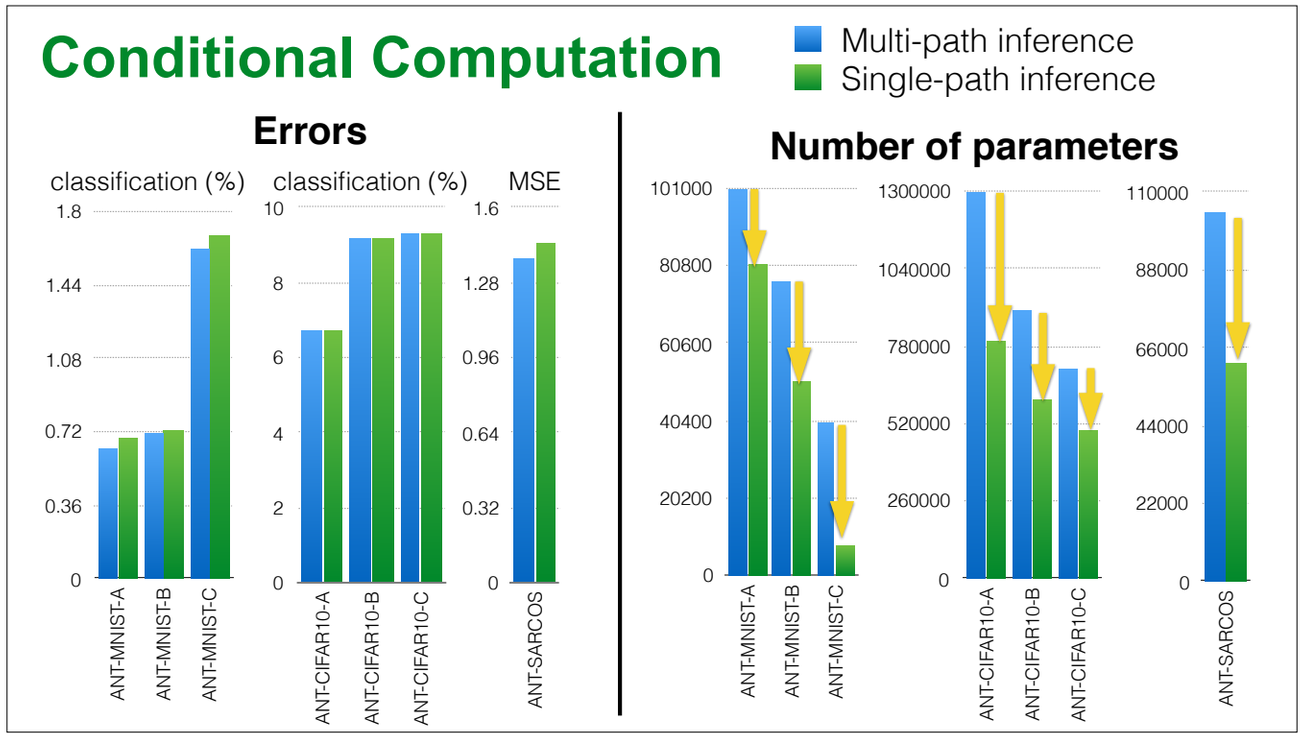


- However, shortcut connections such as residual learning and dense connections achieve even better classification accuracy.
- However, both of these architectures use significantly more parameters.
- Certainty ANTs would benefit from these architectural advances in deep learning research, and integrating them into ANT modules remains a future work.



- On MNIST dataset, similar results were obtained, but instead of showing another set of bar charts, I would like to show a nice learned example architecture, which comes with many of the desirable properties of ANTs.
- ANT-MNIST-C, which has the simplest primitive modules, achieves over 98% accuracy, with as few parameters as required for a linear classifier with the single-path inference. On the other hand, linear classifier on raw pixels only attains 92% accuracy.
- This epitomises the power of combining the hierarchical feature learning and hierarchical partitioning.

- Although precluded from the talk, we have actually done an ablation study and discovered we need both routers and transformers i.e. hierarchical clustering and feature learning to attain the best performance of ANTs on all 3 datasets for all configurations of ANTs. If you are curious, it’s in the paper or I can show you the results.



- Lastly, we tested the benefits of conditional computation by comparing the two inference schemes.
- In particular, we considered multi-path inference which uses all the leaf-nodes on the tree, which is more computational expensive, and the single-path inference which only uses a single dominant leaf node.

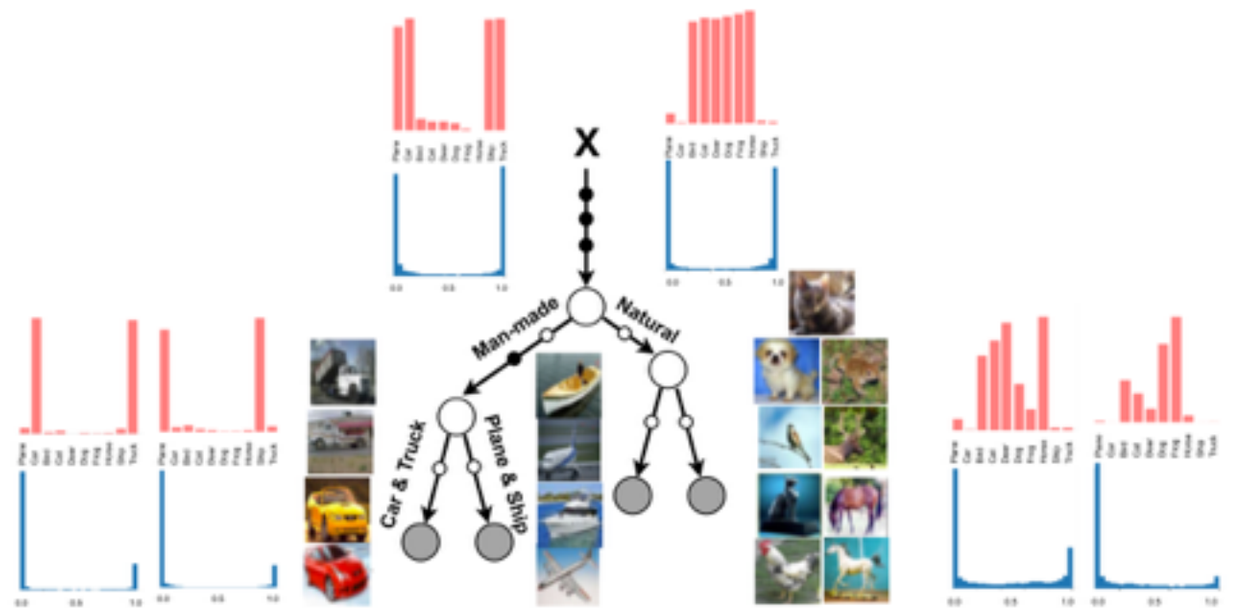
- The set of bar charts on the right compares the errors for different variants of ANTs on all 3 datasets, where blue shows the error for multi-path inference and blue shows the error for single-path.
- You can see that the errors only marginally increase with single-path inference.

- On the other hand, with single-path inference the number of parameters during inference significantly **drops,** showing that single-path inference is cheaper.
- This shows that ANTs indeed benefits from faster inference via conditional computation, without compromising accuracy.
- (Optional) This close approximation comes from the confident splitting probabilities in the routers, being close to 0 or 1

———

This means that single-path inference gives an accurate approximation of the multi-path inference, while being more efficient to compute.

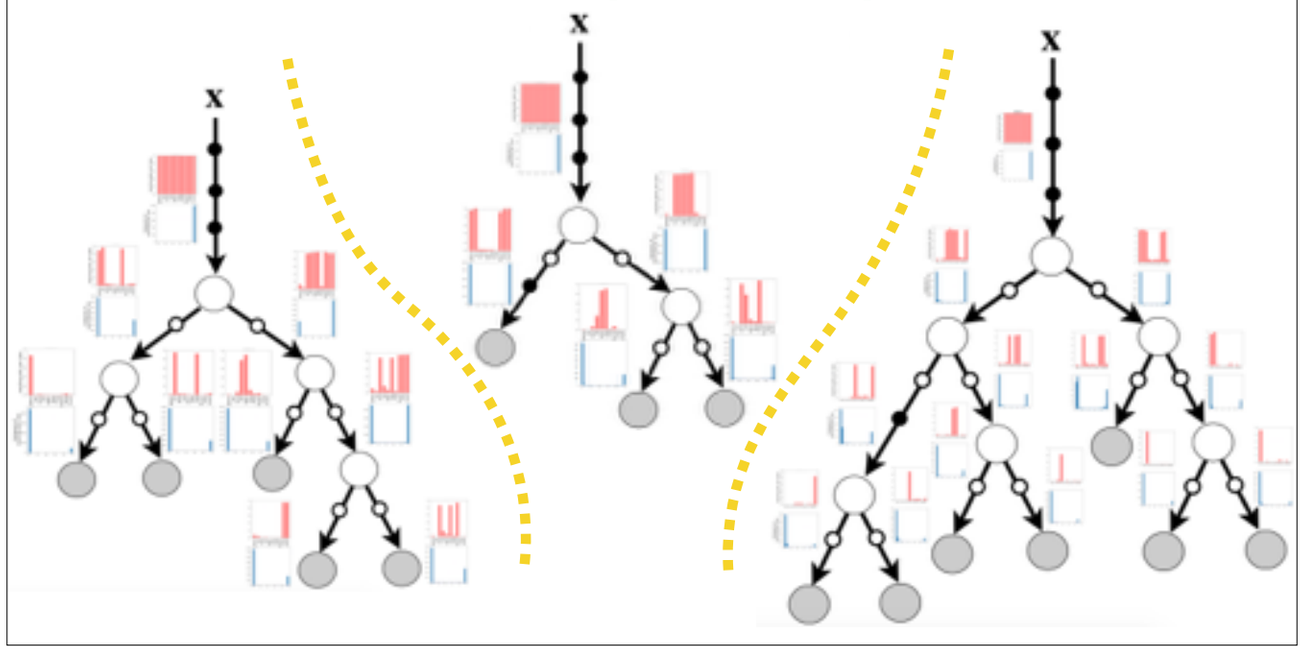
Learned Hierarchy on CIFAR-10



- First, I would like illustrate the ability of ANTs to discover hierarchical structures in the data that are useful to the end task.
- Learned hierarchies often show strong specialisation of paths to certain classes or categories of data on both the MNIST and CIFAR10.
- Here is a cherry-picked example of a learned ANT architecture with particularly “human-interpretable” structures.
- You can see that man-made and natural objects are separated at the first router, and then the man-made objects are subsequently split into road vehicles and non-road vehicles.
- The histogram in red visualises the distribution of test examples allocated to respective branches, and the histogram in blue shows the distributions of router’s decision probabilities.

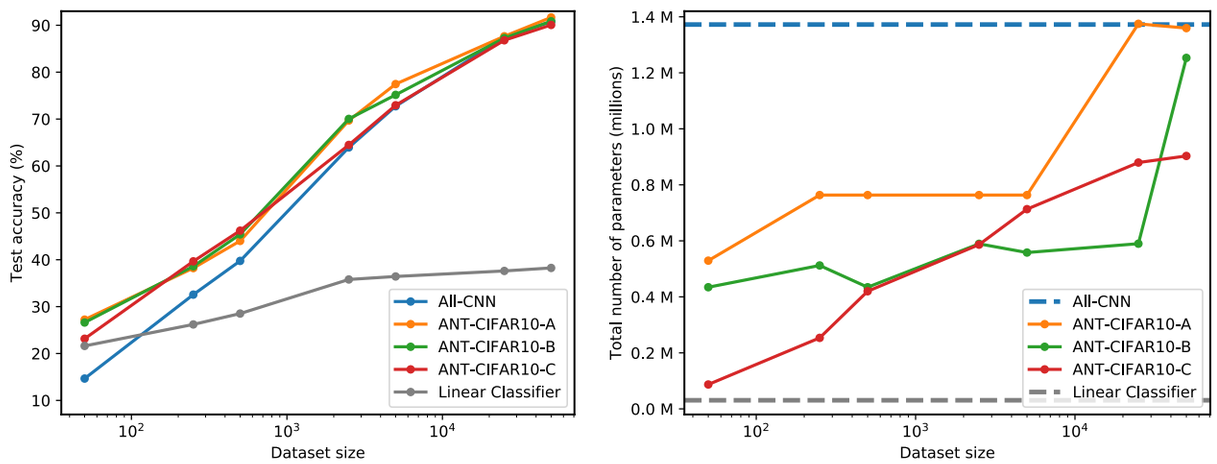
Here I want to emphasise that human intuitions on relevant hierarchical structures do not necessarily equate to optimal clustering of data. Rather I would like to emphasise that that these hierarchies are automatically learned to optimise the end-task performance, and this is possible and visible to users thanks to the freedom of ANTs to learn when to separate and share representations.

Unbalanced Trees: Adaptive Computation



We observe that most learned trees are unbalanced. This means that different categories of data undergo different amount of processing. This property of adaptive computation is plausible since certain types of images may be easier to classify than others.

Adaptive Model Complexity

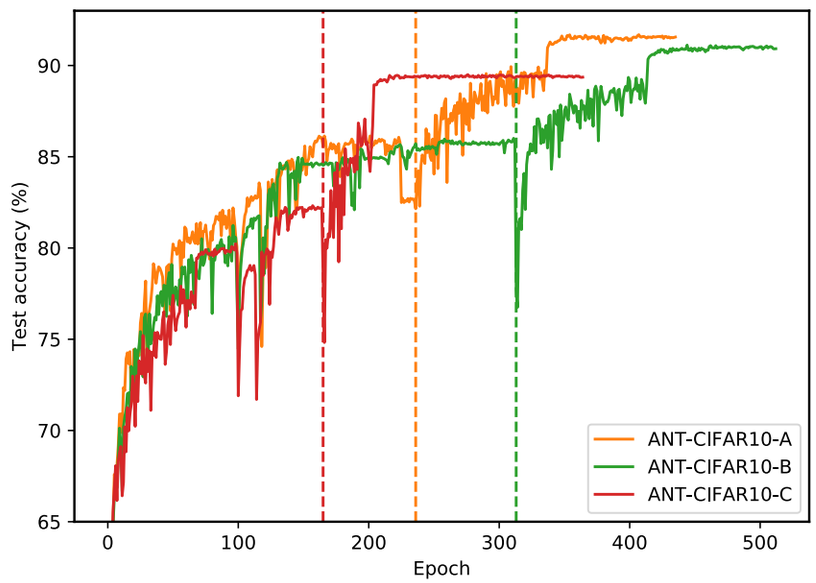


Models are trained on subsets of size 50, 250, 500, 2.5k, 5k, 25k, 45k examples.

- Next, we assess the property of adaptive model complexity. One strength of ANTs is its ability to adapt the model structure to the given availability of data and the complexity of the task. On the other hand, oversized models without regularization, are vulnerable to overfitting on small datasets, and undersizes models would lead to underfitting.
- In this experiment, we trained three variants of ANTs on subsets of CIFAR-10 of varying size 50, 250, 500, 2.5k, 5k, 25k to 45k (the full training set) examples. For comparison, we also trained all-CNN and linear classifier on these datasets.
- Figure on the left shows the corresponding test performance of respective models as a function of training set size, while the figure on the right plots the total number of parameters.
- We see that the variants of ANT achieve comparable or better performance than the baseline All-CN consistently, and in particular, the smaller the training set, the larger the improvement we observe, where the difference can get up to 13%. Figure on the right shows that the model size of ANTs generally increases as a function of the dataset size.
- On the other hand, the baseline All-CNN has a fixed of parameters, consistently larger than the learnt ANTs, and suffers from overfitting, particularly on small datasets. The linear classifier underfits to the data, with consistently poor training/validation performance.

These results support the ability of ANTs to construct models of adequate complexity, embodying Occam's razor leading better generalisation.

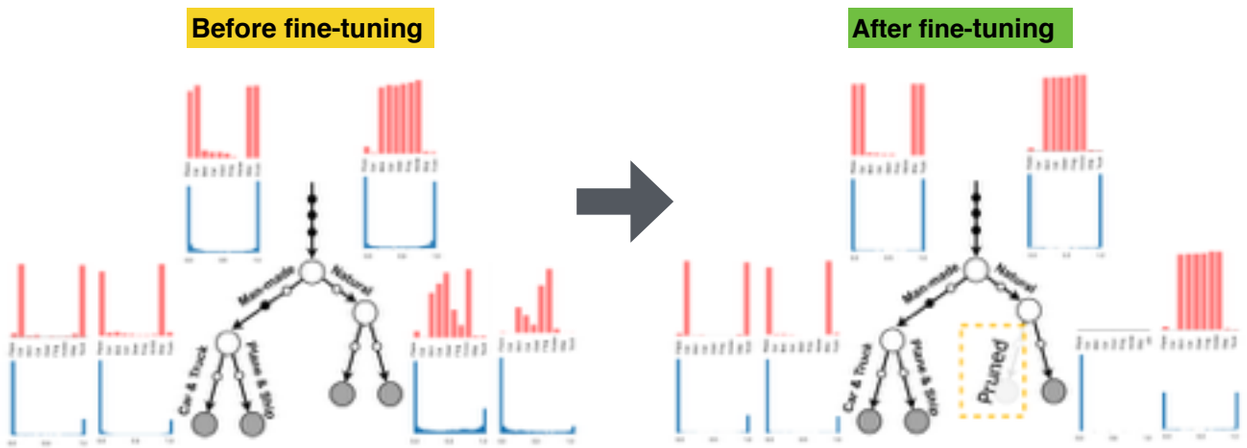
“Pruning” via Global Refinement



Lastly, we look into the effect of global refinement phase on ANTs. This figure plots the test accuracy as a function of training epochs, where dotted lines show the epoch at which the model entered the refinement phase.

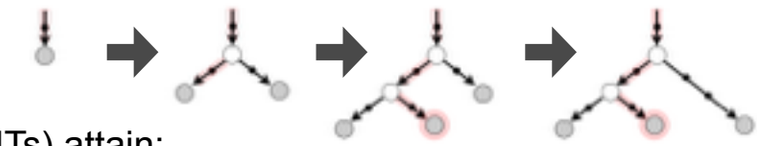
You can see that while all three models initially undergo a steep drop in accuracy, they all converge to higher test accuracy than the best value attained during the growth phase. This is presumably because the global optimisation remedies suboptimal decision made during the locally optimised growth phase. To test this hypothesis, we looked into the dynamics of ANTs before and after the refinement phase.

“Pruning” via Global Refinement



In many cases, we have observed that the global-optimisation polarise the decision probability of routers, which occasionally lead to pruning of some branches. In this example, you can see that the decision probability of routers are more concentrated near 0 or 1 after fine-tuning, and as a result one of the branches is pruned. The resultant model attains better generalisation error, displaying resolvement of suboptimal partioning of data.

Summary



Adaptive Neural Trees (ANTs) attain:

- **Freedom to decide when to share & separate representation:**
  - achieves high accuracy with lightweight inference
  - learns hierarchical structures in data useful to the end-task
- **Adaptive model complexity**:
  - grows the architecture, tuning to the size and complexity of data

- We introduced Adaptive Neural Trees, a form of decision trees (DTs) which use neural networks (NNs) in both its edges and nodes, combining representation learning with hierarchical data partitioning.
- As a result, ANTs attain two key properties:
- Firstly, freedom to decide when to share and separate representation. This allows ANTs to learn a model which achieves high accuracy in classification tasks with lightweight inference via conditional computation. In addition, as a byproduct, ANTs automatically learn hierarchical structures in data useful to the end-task.
- Secondly, we have seen that ANTs benefits from adaptive model complexity. The training algorithm grows the architecture progressively from simple building blocks, adapting to the size and complexity of the training data.

—————

(Optional) Moreover, this method for growing ANTs is both time- and space-efficient because the local nature of optimisation means that gradient computation is constrained to the newly added part of the computational graph, and as a result, most activations do not need to be kept in memory.

Future Challenges

- **Scale up to larger, higher dimensional datasets**
  - more *effective* optimisation (non-local?)
  - more *efficient* optimisation (stochastic?)
- **Evaluate "interpretability" (algorithmic transparency)**
- **Extend to other problems:**
  - structured prediction, generative models, continual learning

- Although we strived for simplicity in the current design of ANTs, as a result, there is a large room for improvement.

(First challenge)
- In the future, I would be interested in scaling up ANTs to larger, higher dimensional datasets, and I think this requires improving the optimisation method to be able to train wider and deeper ANTs.
- Firstly the current growth procedure is greedy and suboptimal. The sub-optimality of the local decisions only gets worse as the tree gets more complex. In the future, we will look into different ways to alleviate this issue such as (1) more global optimisation during growth phase, (2) merging of branches or (3) training in entangled settings as already done in decision tree research.
- Secondly, for large trees, the global refinement phase would be computationally expensive. One potential solution to this problem would be to employ a MC approximation of the training by stochastically traversing the tree and using gradient estimators for parameter updates.

(Second challenge)
- Another challenge is to explore the value of the learned hierarchical structure of ANTs in terms of "interpretability" (or algorithmic transparency)?.
- I think that the tree-shaped hierarchy provides a new means to understand its internal decision making process.
- For instance, given an image where the ANT fail to classify, you could compare against a large number of correctly predicted examples, and potentially localise a point of routing failure where the ANT makes the wrong decision.
- On the other hand, such localisation of failures is more difficult in conventional CNNs with a single fully distributed representation.
- In addition, such hierarchical and decomposed interpretations are complementary to existing visualisation techniques.
- For example, providing saliency maps of the series of routing decisions may potentially provide a layer of transparency into the decisions made in the raw feature space.
- (Optional) although interpretability of ANTs is limited by its own fully distributed representation in the routers and edges of the tree, I think it would still be interesting to study its potential utility.

(3rd challenge)
- Extending to other tasks or learning scenarios such as structured prediction (e.g. segmentation or super-resolution), or continual learning or generative models would also be interesting.

Acknowledgements

Kai Arulkumaran

Daniel C. Alexander

Antonio Criminisi

Aditya Nori

I would also like to thank:
Konstantinos Kamnitsas, Jan Stuehmer, Danielle Belgrave, Sebastian Tschiatschek

cmic
Centre for Medical Image Computing

Microsoft Research Cambridge

I would like to thank main contributors to this work, my intern supervisor Aditya, and my Phd supervisor, Antonio, and Kai who was also an intern at MSR last year. I would like to thank Jan, Danielle, Sebastien and Kostas for the valuable discussions I had with them.

Training Time

Tab. 5 summarises the time taken on a single Titan X GPU for the growth phase and refinement phase of various ANTs, and compares against the training time of All-CNN (Springenberg et al., 2015). Local optimisation during the growth phase means that the gradient computation is constrained to the newly added component of the graph, allowing us to grow a good candidate model under 2 hours on a single GPU.

Table 5: Training time comparison. Time and number of epochs taken for the growth and refinement phase are shown. along with the time required to train the baseline, All-CNN (Springenberg et al., 2015).

	Growth		**Fine-tune**	
Model	Time	Epochs	Time	Epochs
All-CNN (baseline)	–	–	1.1 (hr)	200
ANT-CIFAR10-A	1.3 (hr)	236	1.5 (hr)	200
ANT-CIFAR10-B	0.8 (hr)	313	0.9 (hr)	200
ANT-CIFAR10-C	0.7 (hr)	285	0.8 (hr)	200

Ablation Study

Module Spec.	Error % (Full)			Error % (Path)		
	ANT (default)	CNN (no routers)	SDT/HME (no transformers)	ANT (default)	CNN (no routers)	STD/HME (no transformers)
ANT-MNIST-A	0.64	0.74	3.18	0.69	0.74	4.19
ANT-MNIST-B	0.72	0.80	4.63	0.73	0.80	3.62
ANT-MNIST-C	1.62	3.71	5.70	1.68	3.71	6.96
ANT-CIFAR10-A	8.31	9.29	39.29	8.32	9.29	40.33
ANT-CIFAR10-B	9.15	11.08	43.09	9.18	11.08	44.25
ANT-CIFAR10-C	9.31	11.61	48.59	9.34	11.61	50.02

Module Spec.	Error (Full)			Error (Path)		
	ANT (default)	NN (no routers)	SDT/HME (no transformers)	ANT (default)	NN (no routers)	SDT/HME (no transformers)
ANT-SARCOS	1.384	2.511	2.118	1.542	2.511	2.246

- No routers = standard CNNs
- No transformers = SDTs / HMEs
- Ablation of transformers/routers leads to higher classification errors.

We lastly compare against 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 (Sua´rez & 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 classification errors across different module configurations

Learned Hierarchical Structures

Module Spec.	Error % (Selected path)	Error % (Least likely path)
ANT-MNIST-A	0.69	86.18
ANT-MNIST-B	0.73	81.98
ANT-MNIST-C	1.68	98.84
ANT-CIFAR10-A	8.32	74.28
ANT-CIFAR10-B	9.18	89.74
ANT-CIFAR10-C	9.34	97.52

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 the table. We observe that using the least likely “expert” leads to a substantial drop in classification accuracy.

Benefits of ensembling

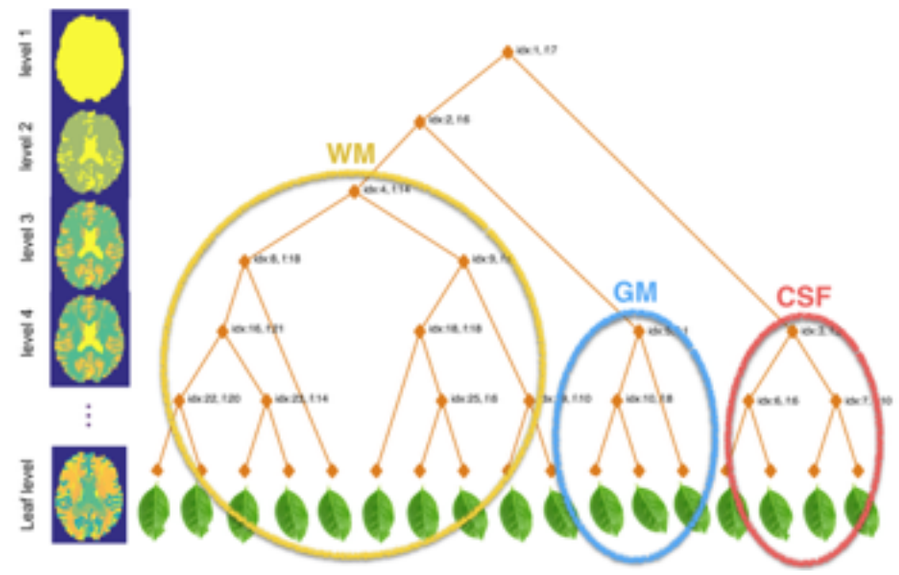
Table 9: Comparison of prediction errors of a single ANT versus an ensemble of 8, with predictions averaged over all ANTs in the ensemble.

	MNIST (Class Error %)		**CIFAR-10** (Class Error %)		**SARCOS** (MSE)	
	Error (Full)	Error (Path)	Error (Full)	Error (Path)	Error (Full)	Error (Path)
Single model	0.64	0.69	8.31	8.32	1.384	1.542
Ensemble	0.29	0.30	7.76	7.79	1.226	1.372

Table 10: Parameter counts for a single ANT versus an ensemble of 8.

	MNIST		**CIFAR-10**		**SARCOS**	
	Params. (Full)	Params. (Path)	Params. (Full)	Params. (Path)	Params. (Full)	Params. (Path)
Single model	100,596	84,935	1.4M	1.0M	103,823	61,640
Ensemble	850,775	655,449	8.7M	7.4M	598,280	360,766

Debugging a Decision Tree



Debugging a Decision Tree