Comparisons Between Variants of Differentiable Decision Tree Models

William Silva
silvaw2017@gmail.com
August 16, 2019

Introduction

The Differentiable Decision Tree, or DDT, is a machine learning classifier that I have researched over the past three months. In this report, I will give an overview of both the basis and intent of the research as well as the results that I found. Firstly, it is important to understand the two primary models that informed the creation of the DDT, namely the Decision Tree and the Neural Network.

Decision Tree

A decision tree is a model that, after fitting to given training data (for example a set of nuclei data inputs matched to their respective outputs, malignant or benign), generates a tree-like structure of nodes that either branch into other decision nodes or into leaves, which mark the end of the tree. This structure then takes a sample and performs a simple check of the data at each node, determining whether the sample meets the decision criteria, and routing the sample through the tree until it reaches a leaf, which contains the final label. This leaf is a single output label, such as “benign” or “malignant.” These checks can perfectly fit to the training data so that any input from the training data would move down the tree until it reaches the correct classification. This method for classification is very interpretable, meaning it is possible to determine which tests the model is performing in order to classify an input sample. Furthermore, because all inputs with similar features are moved to the same parts of the tree, the tree partitions the space of possible inputs in such a way that each node only sees a specific fraction of the data; this can be helpful for multiple reasons, for instance the creation of sub-trees which specialize in a particular set of inputs or samples. However, due to the simplicity of the model, the binary decision making that the tree makes often results in relatively low accuracy, especially when classifying complex data sets.

Neural Network

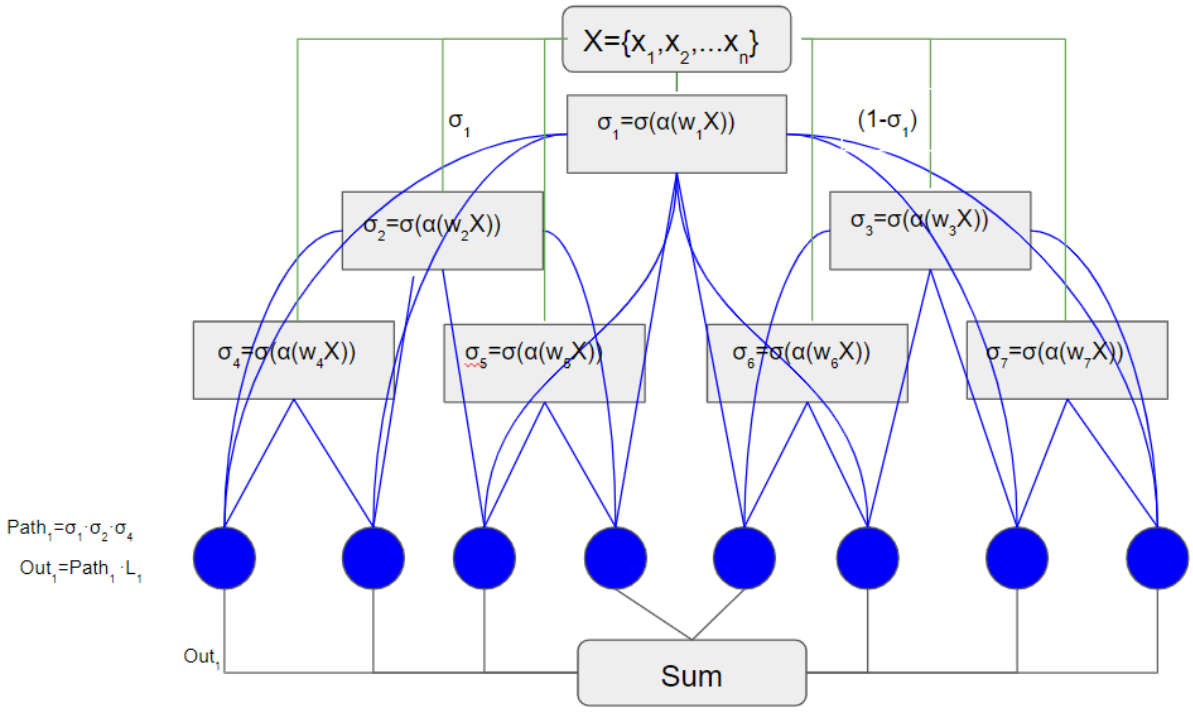
The second classifier, which in many ways contrasts the decision tree, is the neural network. Deep neural networks are machine learning models that can use multiple layers of trainable weights to create an embedded representation of the input. This representation is then used to generate a probability distribution for classification labels for a given sample. This means that each input is assigned a set of probabilities corresponding to each possible output, for example 0.1 for “malignant” and 0.9 for “benign,” the greatest probability of which is taken as the output

of the network. In the case of a fully-connected neural network, the neurons, or sets of weights that perform linear calculations based on the input, are arranged in layers and connected so that each neuron of the network connects to every neuron of the previous and subsequent layer. It is this relationship between neurons that creates the embedded data representation and the depth of a deep neural network. Unlike the decision tree, the neural network does not simply fit to a training set and perfectly cover the data. Instead, the model trains on the data, actively adjusting its neurons' weights in order to decrease loss and increase accuracy. The increased complexity of the fully connected neural network means that it is generally more accurate than a decision tree, especially when tasked with classifying inputs of complex data sets and in the classification of images. However, due to the creation of new data representations at each layer, it is difficult to interpret the decision making process of the network.

Differentiable Decision Tree Variants

Although it is important for classifiers to be accurate, it is often equally important for users to understand the method by which the model is classifying inputs. The DDT is an attempt to allow for a model to have the interpretability provided by a tree-like structure while maintaining a higher level of accuracy made possible by the actively trainable weights of a neural network. There are two main variations of the DDT, the Flat DDT and the Deep DDT.

Flat DDT



The first of the two DDT models is the Flat DDT. The organization of nodes shown in the image of a Flat DDT (of depth 3) above displays the model's similarities with the decision tree: each node splits twice until the leaves are reached. This property of the model is what creates the

space partitioning qualities of a decision tree; however, the most significant difference between the Flat DDT and a decision tree is in how the input is passed down the tree. The decision tree uses specific tests of features at each node in order to create a binary split: any given input only goes to the left or to the right at each node with 100% certainty until it reaches a leaf, which is a single label. In contrast, the Flat DDT calculates at each node some value between 0 and 1 that represents the probability that the input is sent to the right. Specifically, the model implements the linear calculation of a linear neural network at each node, given by the equation $\sigma_n=\sigma(\alpha(w_nX + b_n))$ (for simplicity, the image uses the equation $\sigma(\alpha(w_nX)$; in this case, b_n is implied) where σ_n is the probability of an input going to the right at the next layer of depth ($1-\sigma_n$ is therefore the probability of a left turn), w_n and b_n are the weight and bias vectors for the node n, α is a temperature value or measure of confidence for the network, and σ is the sigmoid activation function, which places all inputs between 0 and 1 with very high positive inputs near 1, very negative inputs near 0, and inputs close to 0 near 0.5. Much like a neural network, all weights and biases used for calculation of the output are randomly initialized and trained as the model's parameters. Accordingly, the Flat DDT is iteratively trained via gradient descent, rather than initialized all at once like a decision tree.

In the image above, the blue lines indicate the leaves that each split probability leads to; if the leaf is to the right of one of its connected nodes, the probability value that is sent is the σ_n, whereas leaves to the left of a connected node receive the node's $1-\sigma_n$ value. In the Flat DDT, a leaf does not represent a single possible label for the data. Instead, much like a neural network, each leaf contains a probability distribution that represents the probability that the output is of a certain label, for example the 0.1 "malignant" and 0.9 "benign" scenario mentioned before. These leaf probability distributions are parameters of the model, so they are also updated via gradient descent as the model fits to training data. For any given input, the probability of a leaf being the model's best output is the product of all of the probabilities nodes along the path to that leaf (where the probabilities are produced by the sigmoid activation functions at each decision node). To produce the final output of the network, each leaf's probability distribution weights are multiplied by their respective path probabilities, and then corresponding labels of all of the leaves are summed together to create the final probability distribution of the model.

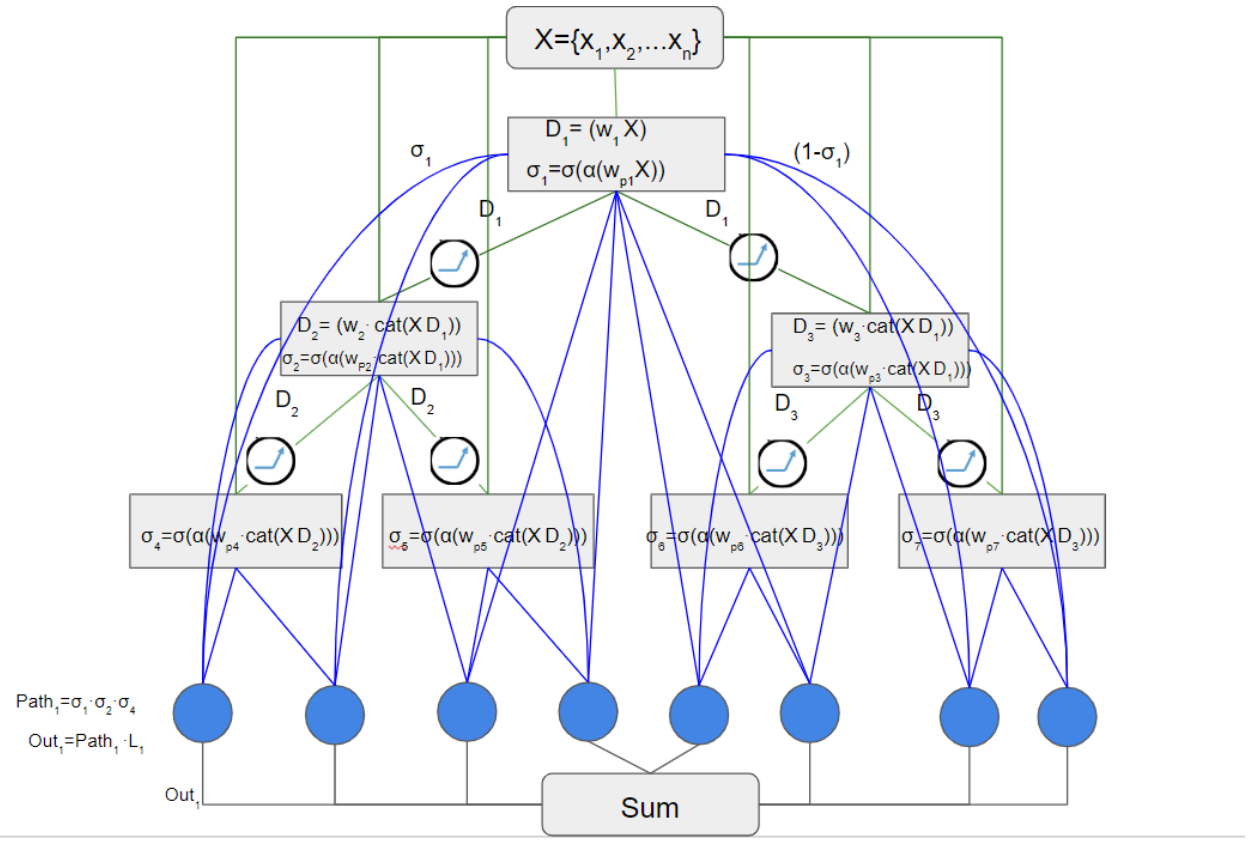
It is important to note that while the entire tree structure (all individual nodes) is used to calculate the final output for every input, this does not mean that the model no longer partitions space to certain inputs like the decision tree is able to do. This is because lower probability paths are less significant in determining the classification of an input; because of this, the nodes and leaves that have lower probabilities will not be as affected by the network's corrections during training. If, for instance, an input had a path probability of 0 for a certain leaf, the weights of that leaf would be almost unaffected by any loss generated by the result (though the weights that generated the 0 probability might change). Because of this, certain parts of the tree are more affected by certain inputs, and a space-partitioning structure is maintained.

In the case of the Flat DDT, because all nodes take raw data as an input for a probability calculation, the weights of each node can give insight into the process by which each leaf's path probability is determined, granting the model greater interpretability.

The most significant limitation of the Flat DDT lies in its depth.The Flat DDT lacks the true depth that gives the deep neural network many of its benefits. Although each leaf can be

connected to multiple nodes, these nodes do not interact with each other in any way: they simply send probabilities to a shared leaf. This is different from a deep neural network, which uses linear calculations to create a data representation of the input and sends that representation to the next layer of the network. In order to promote interpretability, the Flat DDT creates an output based entirely off of raw input data, which could be coming at the expense of accuracy.

Deep DDT



In order to fix this possible issue and further observe the effect of real depth on a DDT, I experimented with the Deep DDT, a variation on the Flat DDT with some key differences. The Deep DDT maintains the Flat DDT's organization of nodes and leaves into a decision tree structure. Furthermore, the properties of the leaves and the equation for probability calculations are identical.The key difference between the Deep DDT and the Flat DDT is in the interaction between nodes. In the Flat DDT, all nodes' split probabilities are determined based off a linear calculation with the model's raw input data as an input. In the Deep DDT, however, each parent node creates a data representation that it sends to its two children nodes; the children nodes use this data representation along with raw input in their probability calculations as well as the creation of their own data representations. In order to perform these calculations, each decision node has an extra set of weights which it uses to transform its input into a data representation that could be more useful to the model. These weights, as in a neural network, are parameters

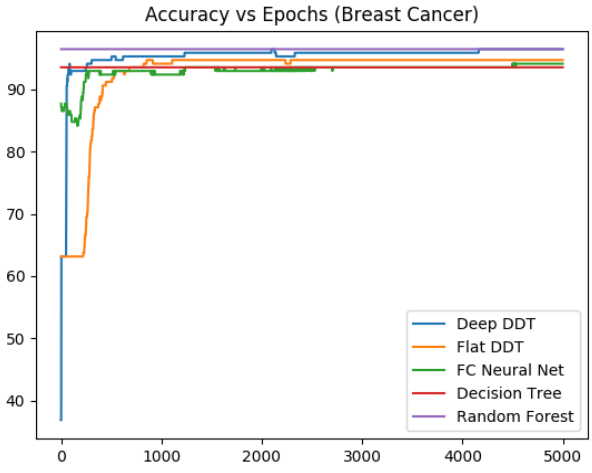
of the model and are updated via gradient descent. Additionally before each node's data representation is sent to subsequent nodes, it is placed into an activation function, in this case ReLU.

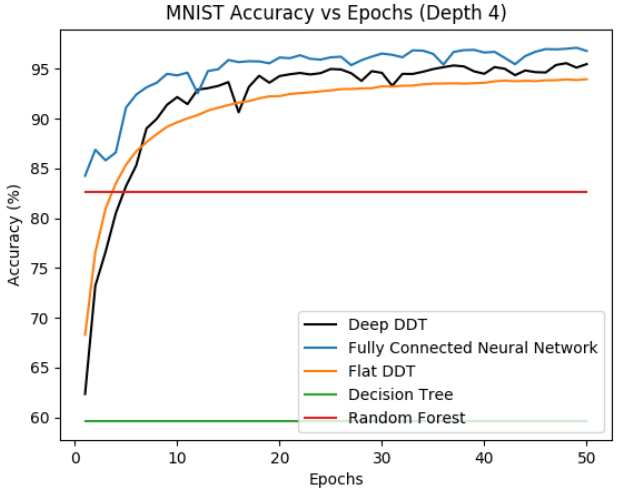
The Deep DDT, because of the creation of embedded data representations at each node, is more similar to a neural network than the Flat DDT. It can be helpful to think of each path to a leaf of a Deep DDT as similar to its own linear neural network that changes the data at each node in order to more effectively determine if the leaf's weight vector represents the correct classification. In extension, the model can be thought of as two separate sets of calculations that work off of the same inputs: multiple paths of neural networks that create data representations to fit to leaves' weights and probabilities that chose which neural network's output is most correct.

The Deep DDT sacrifices the interpretability gained from using only raw data in exchange for learned features and true depth. This is not to say that the Deep DDT is as uninterpretable as a neural network, however. The Deep DDT shares the ability to partition the input space, so although one cannot observe the weights of each node to understand the decision making process of the model, it is possible to tell what parts of the tree favor certain inputs. For instance, I created a specific batch of Breast Cancer dataset inputs, half with malignant classifications and the other half with benign classifications, and printed the node probabilities of each input. From this, I was able to determine that as the DDT (of depth 2) trained, certain nodes (but not all) would develop very high right turn probabilities for one label and very low probabilities for the other, displaying that the tree was not simply acting as a neural network but was also learning to partition its space to different inputs.

Performance

Overall, as one might expect, in exchange for higher interpretability, the Flat DDT classifies data sets with lower accuracy than the Deep DDT. In both cases, the hand-picked feature-set Breast Cancer and the 28x28 pixel black and white image set MNIST, the DDTs were able to outperform most other baseline models of the same depth while performing comparably to the



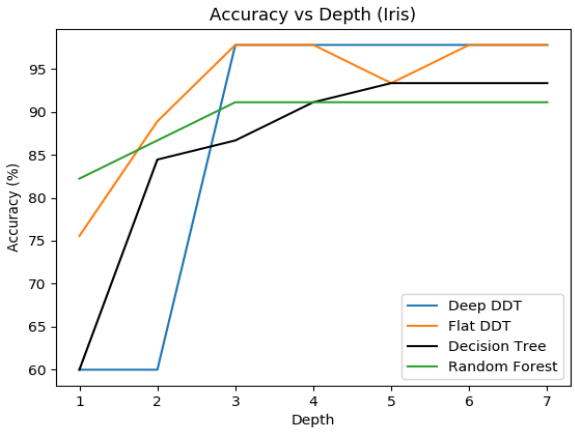


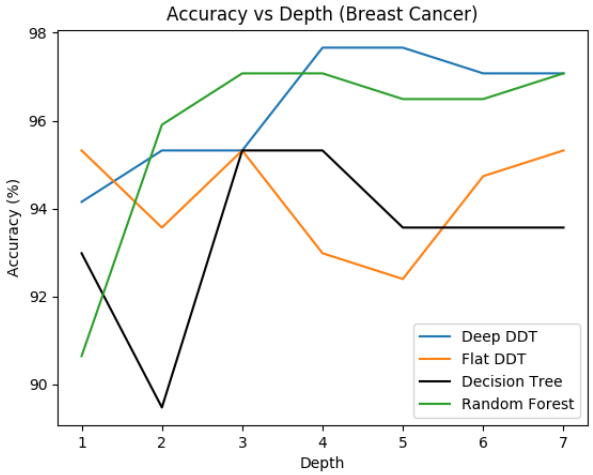
most accurate for each dataset (note that the y axes are different between the two images). Generally (on more than these two sets) the Deep DDT required fewer epochs to approach its maximum accuracy than the Flat DDT.

Hyperparameters

Depth

There are two main hyperparameters that can greatly affect the performance of both DDT models. Firstly, the depth of a DDT has different meanings based on which DDT is being used. In a Flat DDT, varying depth varies the number of nodes connected to each leaf as well as the total number of leaves. In a Deep DDT, increasing depth does the same but also increases the number of data representations that are created in each network. As such, depth has a greater impact on the accuracy of the Deep DDT, which is indicated by the larger shift in accuracy as depth increased on both datasets (Breast Cancer and Iris, a very simple hand selected feature set). It is possible, additionally, that too high a depth on a Deep DDT can produce results similar to having too many layers for a certain data set on a deep neural network; the number of epochs over which the model trained was held constant when comparing depth, and it is therefore possible that the highest depth of Deep DDT required more training to reach its maximum because of the exponential increase in nodes.

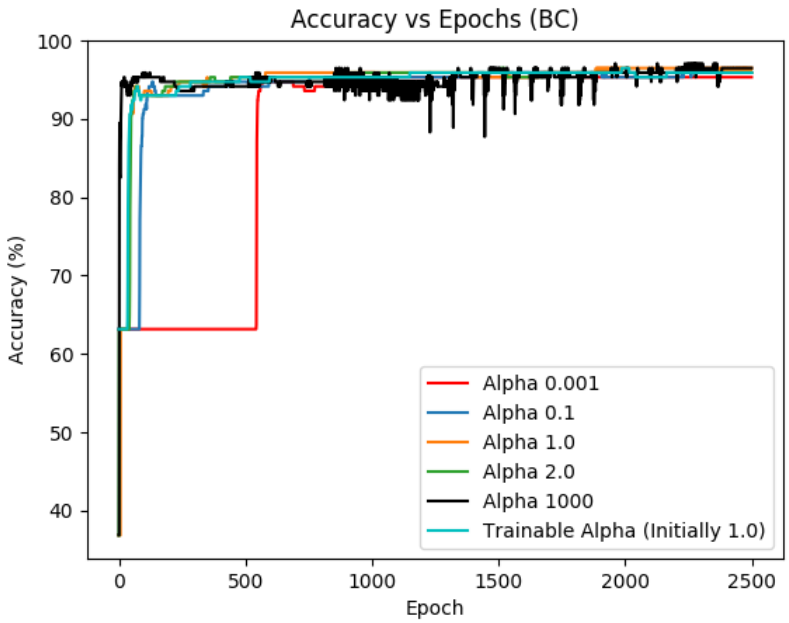




Alpha

Alpha, the temperature value, is multiplied by the linear calculation before the sigmoid activation function at each probability calculation of a DDT. It is the same for every node and determines the ‘confidence’ of the model. Again, the sigmoid activation function places all

number values between 0 and 1, with very high positive inputs near 1, very negative inputs near 0, and inputs close to 0 near 0.5. Therefore, as alpha approaches infinity, positive results of the linear calculation become more positive while negative results become more negative, and the split probability for an input approaches 0 or 1. This means that for every input, a single path and leaf will be used to generate a label, mimicking the hard split decision of a decision tree. As alpha approaches 0, the decision probability becomes closer to 0.5 after sigmoid, and there is no confidence to the tree.



What I found when testing various alpha values is that a frozen value between 0.1 and 2.0 generally produces an optimal result. An arbitrarily high alpha values can still potentially produce high maximum accuracy, as high certainty down a single path for each input could still perform well after some training (similar to a single linear neural network for each input); however, if a small change made to a node's weights ends up changing the path of an input, the certainty of the switch will be nearly 100%, and the input will go down a path it presumably has never been trained on, causing the sharp drops in accuracy shown by the black plot above. If alpha is very low, the model will simply not benefit in any way. Because of the low certainty of the model, the DDT cannot create specific probability distributions for certain leaves, as each input would reach every leaf with the same probability. As such, the weights of all of the nodes simply must become large enough in magnitude to cancel out the very low alpha; the accuracy of the model eventually becomes comparable to that of with a more standard alpha, but it takes much longer for the weights to reach a sufficiently high value to make this happen.

Alternatively, alpha can be parameterized. In other words, as the model determined how the tree is to fit to inputs, it also decides its own confidence with its probability calculations. In practice, I found that a trainable alpha will yield a result that is comparable to that of a standard alpha between 0.1 and 2.0, but the alpha itself only steadily increases as this happens, regardless of the current accuracy. As there are some downsides to having a very large alpha, parameterizing the alpha could be harmful to the model's consistency if value gets high enough.

Future Work

Concerning the DDT, there are both problems to fix that could hinder the efficiency of the model and possible directions the model could take in the future. First, more work could be done to fix Deep DDT's failure to fit to the entire tree, perhaps by further encouraging 50/50 splits early on in training. Second, stabilizing a parameterized alpha and preventing it from constantly increasing as the model trains could make a DDT with a parameterized alpha useful in some situations (this could potentially help to fix the Deep DDT's probability issues, as well). Finally, the interpretability of the Deep DDT could be improved through easier insight into the input-space-partitioning that occurs through probability splits.

Although the current version of the Deep DDT might require more work to function as well as the model allows, there are still variations to the model that could greatly improve it. The linear calculation that creates each parent node's data representation could be swapped out for something like a convolutional layer, potentially making the data representation more useful for image data.

Finally, certain data representations or path probabilities could be useful for more than just each node's two child nodes. An interesting direction for future research is to allow for a breaking of the tree-like structure, allowing for previously disconnected nodes to connect and for the model to take the form of a Directed Acyclic Graph.