Distilling a Neural Network Into a Soft Decision Tree

Marc-Olivier Bélanger , Rosanne Larocque , William Valiquette
Polytechnique Montréal
{marc-olivier.belanger, rosanne.larocque, william.valiquette}@polymtl.ca

Abstract

Although the performance of deep neural networks brought machine learning to the forefront of research in computer science, their inner working is still obscure and considered a black box as to how and why they give us the right output given a particular test case. One way to explain such decisions is to express the acquired knowledge of a neural network in a model that relies on hierarchical decisions, e.g. a soft decision tree, instead [1]. The experiments we showcase here are a reproduction of those described in the Google Brain Team's article *Distilling a Neural Network Into a Soft Decision Tree* [1].

1 Introduction

One of the greatest difficulty in understanding how and why a deep neural network outputs a class label given a specific test case is the distributed representation of the data across the hidden layers [1]. It is easy for the first and last hidden layers since we can understand what causes the activation of a unit in the first layer as well as the effects of the activation of a unit in the last layer [1]. However, it is much harder to establish a link between the other hidden layers' units activation and variables such as the input and the output [1]. On the other hand, it is easy to understand how and why a decision tree makes a classification simply by following the sequence of decisions from an input to its output. These decisions being directly based on the data, this gives us an interpretable way to explain the reasoning leading to an output of the tree.

Hinton and his collaborators [2] described a technique called distillation that offers a good trade-off between a deep neural network's generalization capabilities and a decision tree's interpretability. The idea of distillation is to transfer the knowledge acquired from a trained deep neural network to a distilled model that uses a soft target distribution [2]. In our case, distillation allows us to create a decision tree that makes soft decisions and mimics the black box input to output function discovered by the trained deep neural network [1]. Consequentially, we are then able to represent the link between inputs and outputs found by the deep neural network as a sequence of soft decisions in the decision tree. Finally, this distilled model performs slightly worse than the underlying deep neural network [1] but the benefits of its interpretability largely outweighs this loss.

In their 2006 article, Bucilua et al. were the first to introduce model distillation, then named *model compression*, to produce fast, compact and highly accurate models instead of using better performing models that are slower and more cumbersome [3]. This implies that, besides its added interpretability and comparable performance as stated previously, a distilled model will often be much faster than a deep neural network [1].

We start by introducing our methodology which lays the basis for our experiments.

2 Methodology

2.1 Convolutional Neural Network

The less interpretable, but highly performant, deep learning model we used is a regular small convolutional neural network. Our best model has two convolutional layers, one max-pooling layer, two dense layers and a few dropout layers in between. Our architecture is further detailed in Table 1. Otherwise, our model is trained with mini-batch gradient descent using the ADAM optimizer.

Layer Type	Units	Kernel Size	Stride
`Conv.2D`	32	3	1
`Conv.2D`	64	3	1
`MaxPool.2D`	-	2	1
`Dropout(0.25)`	-	-	-
`Dense`	128	-	-
`Dropout(0.25)`	-	-	-
`Dense`	10	-	-
`Softmax`	10	-	-

Table 1: Architecture of the Convolutional Network.

2.2 Hierarchical Mixture of Bigots

Similar to the original Google Brain Team's article, the model we implemented for the decision tree is a soft binary decision

tree where each inner node i has a learned filter W_i and a bias b_i. We can see that each inner node is thus a single-layer perceptron. Each leaf node l has a learned probability distribution Q_l. At each inner node, the probability of taking the leftmost branch is:

$$p_i(\mathbf{X}) = \sigma(\mathbf{X}W_i + b_i) \tag{1}$$

where $\mathbf{X}$ is the input image pixel matrix and σ is the sigmoid logistic function.

This model is actually a hierarchical mixture of experts [4] where the leaf nodes are the experts. However, the authors of the original article branded the experts as bigots, hence the name change, since they always produce the same distributions whatever the input data being fed after training [1]. As stated previously, this soft binary decision tree learns the filter for every inner node and the probability distribution for every leaf node. The hierarchy of filters is used to assign each example to a particular leaf according to the leaf's path probability. Each leaf's probability distribution is static and over the possible *k* output classes.

$$Q_k^l = \frac{exp(\phi_k^l)}{\sum_{k'} exp(\phi_{k'}^l)} \tag{2}$$

where Q_l is the probability distribution of the l^{th} leaf and each ϕ^l is a learned parameter of that leaf. We can see this is the softmax function.

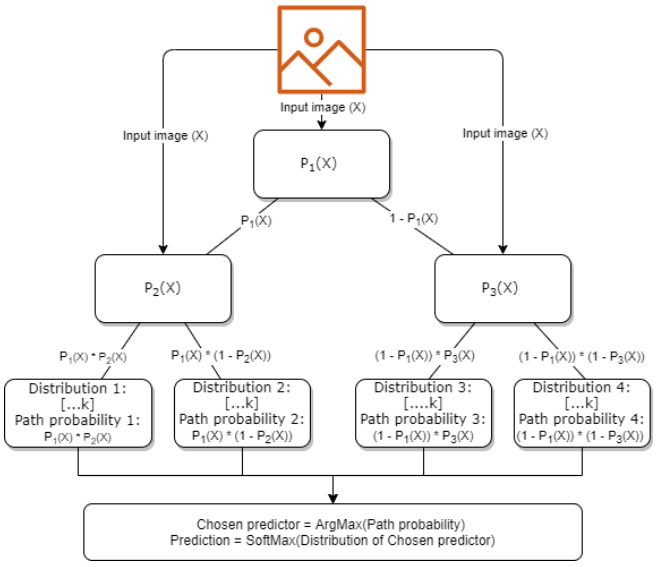

Figure 1: Example of a Soft Decision Tree of Depth 2

We can see on Figure 1 that the filter of each inner node is learned directly from the input data being fed to the model and not from the activation of the node's parent. The path probability of any given node is defined recursively as the cumulative multiplication of the node's parent probability if the node is the left child or of its parent's complement if the node is the right child. The leaf selected to make the prediction for a given example is the one with the maximum path probability. The output prediction itself is the selected leaf's probability distribution Q_l.

The model is trained using a loss function that seeks to minimize the cross entropy between each leaf, weighted by its path probability, and the target distribution [1].

$$L(x) = -\left(\sum_{l \epsilon Leaf Nodes} P^l(\mathbf{X}) \sum_k T_k log(Q_k^l)\right) \tag{3}$$

where $\mathbf{X}$ is the input image pixel matrix and T_k is the target distribution for a single training example. $P^l(\mathbf{X})$ is the path probability of leaf *l* given the input $\mathbf{X}$.

There was a mistake in the original Google Brain Team's article for the loss function. Indeed, they were calculating the logarithm of the outer sum whose value happens to be negative since the logarithm of the probability distribution Q_l in the inner sum is negative. This operation is of course undefined, so we simply removed the outer logarithm calculation and obtained a proper loss value. Formula 3 reflects that change.

We also implemented the penalty term α_i from the original article to regularize the model. This penalty is to encourage each inner node to make equal use of both left and right children [1]. Specifically, it is meant to avoid getting stuck at poor solutions during training or on plateaus in which one or more of the inner nodes always assigns almost all the probability to one of its child [1]. This latter situation implies the gradient of the logistic function for that decision would always be very close to zero [1]. The penalty value is the cross entropy between the desired average distribution 0.5, 0.5 for both the left and right children and the actual average distribution α_i, (1-α_i) where α_i is given by:

$$\alpha_i = \frac{\sum_X P^i(X) p_i(X)}{\sum_X P^i(X)} \tag{4}$$

where $P^i(X)$ is the path probability from the root node to *i*. The penalty summed over all the inner nodes *C* is thus given by:

$$C = -\lambda \sum_{i \epsilon Inner Nodes} 0.5log(\alpha_i) + 0.5log(1 - \alpha_i) \tag{5}$$

where λ is a hyper-parameter that determines the magnitude of the penalty.

2.3 Distilled Neural Network

As mentioned earlier, the main difference between training a regular model and training a distilled model resides in the labels used. Instead of using hard labels such as one hot vectors, training a distilled model uses what we call soft labels.

Those soft labels actually come from the distribution created by the more complex model we want to distill. Therefore, we will not have one hot vectors, but vectors of the classes distribution that were learned by the more complex model, the convolutional network in our case. The idea is that the loss would be more accurate doing so. To illustrate this idea, let us consider a binary classifier trying to classify a sample hardly labeled as [0 1]. Let us say the result of its softmax layer gives [0.55 0.45]. It will then classify the sample as [1 0] and will be severely penalized for its mistake. If we chose soft labels instead, our target could be something similar to [0.45 0.55]. Using this soft label, the loss of the simpler model would not be as bad as using the one hot label. Hence, it is assumed to make our simpler model more robust to outliers. Using soft labels of good enough quality would thus help improve the model's performance while keeping it interpretable. To summarize the steps of distilling a neural network, we first have to train and optimize a deep neural network with the regular hard labels, then we conceive a simpler model and train it on the predictions made by the network instead of directly on the data.

3 Experiment

3.1 Dataset, Metrics and Baseline

As in the original article, all models were trained on the MNIST dataset. It consists of 70k images of handwritten digits. We have decided to split it into three subsets. We took 54k images for the training subset, 6k images for the validation subset and 10k images for the test subset. Models' performances are compared based on their accuracies on the test set, a simple metric indicating the percentage of correctly classified samples.

For this experiment, the goal was to compare the performance of a hardly interpretable model, e.g. a convolutional neural network, with that of a more interpretable yet slightly less performant model i.e. a soft binary decision tree in our case. Hence, our baseline compares our convolutional neural network with a regularly trained soft decision tree. Then, we compare the convolutional network with a distilled soft decision tree, hoping to reduce the expected performance loss using a more interpretable model.

3.2 Results

The results obtained with the convolutional neural network are pretty similar to those presented by the original article. The Google Brain Team achieved 99.21% on the test set with two convolutional layers and one dense layer. We obtained 99.28%. Our convolutional network is slightly deeper as it has two dense layers and some dropout layers. However, their article did not mention the number of units in any of their layers, so we can't compare the global size of the network.

What was more tedious was to achieve their results with the decision trees. The decision tree trained with the hard labels got 90.75% in our experiment compared to 94.45% in the original experiment. This is due to the difference in the hyper-parameters used. They mentioned using a depth of 8 for their tree, but we found better results with a depth of 7. Moreover, they did not mention anything further on any other hyper-parameter. The gap between our performances indicates that we clearly chose a different set of parameters. When using the distilled model with the same hyper-parameters, we achieved 92.09% accuracy, which represents an improvement of 1.34%. The original article improved their model by 2.31%. Although our performances are lower than those from the original article, we still observe an improvement when using the soft labels.

Model	Labels	Acc. Val	Acc. Test
Conv. Net	Hard (one hot)	99.29	99.28
SDT	Hard (one hot)	90.09	90.75
SDT	Soft	90.85	92.09

Table 2: Performances of our models.

Besides having good performances, the interesting characteristic of those trees is their interpretability. On Figure 2, we can see a visualisation of the filters of each inner node in a tree of depth 4. The leaves on the figure show their learned distributions. As we can see, the filters of the upper levels tend to be more abstract, and the deeper we get in the tree, the more discriminant they become. It shows the hierarchy of decisions made by the model in an intuitive way. The interpretation process can be summarized as follows: we start by asking very wide questions and as we go deeper in the tree, our questions are more and more precise as we get more information. The last inner node on the right of Figure 2 is peculiarly eloquent. Most of its weights are focused on the line closing the digit 3 to form the digit 8, guiding its final decision.

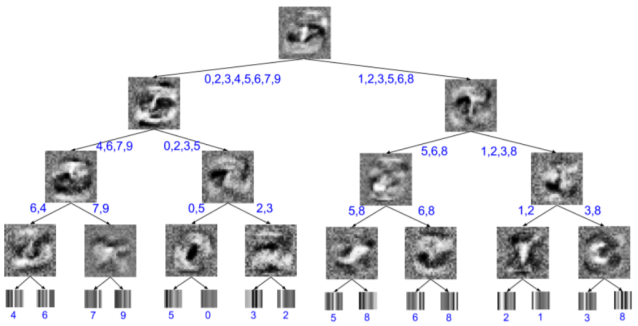

Figure 2: Visualisation of the weights and distributions learned by a decision tree of depth 4 [1]

The code to reproduce this experiment is available on GitHub[1].

4 Discussion

The approach we used to learn more about model distillation started by reading the Google Brain Team's original article but also the other articles cited in it. In its essence, model distillation was not a very difficult concept to grasp for us.

[1] https://github.com/pinkTheRock/Explainable-Deep-Learning

We started our implementation from scratch and actually built the convolutional network entirely by ourselves. We then scoured the Internet for implementations of soft decision trees and found numerous GitHub repositories doing just that. We still made our own implementation but inspired ourselves from them[2].

Once we had the code structure for the decision tree, it was pretty straightforward for us to implement the formulas detailed in the original article. We spent some time trying to mount our soft decision tree on a Google Collaboratory GPU in order to be able to compare its inference time with that of the convolutional network, but it turned out that training the tree was taking longer than if done on a CPU. It was thus impossible for us to confirm or infirm the hypothesis stated in the introduction that a distilled model is often much faster than a deep neural network.

In summary, our approach to learn about model distillation was to read the original article and its cited articles that seemed the most relevant to the subject and looking on the Internet for implementations of soft decision trees.

5 Conclusion

Through this article, we reproduced the model distillation experiment of the Google Brain Team as described in their article *Distilling a Neural Network Into a Soft Decision Tree* [1]. We described and implemented their method of using a trained neural network to create a more interpretable model in the form of a soft decision tree trained on the network's predictions instead of directly from the data. This way, the obtained distilled model retained the underlying neural network's generalization capabilities as well as the decision tree's interpretability. Although it performed slightly worse than the convolutional neural network, we managed to demonstrate through our experiments that the distilled model performed better than a soft decision tree trained directly on the data. This marks a great step towards better explaining the inner working of the black box that deep neural networks are.

References

[1] Nicholas Frosst and Geoffrey Hinton. *Distilling a neural network into a soft decision tree*. Nov. 2017. URL: https://arxiv.org/abs/1711.09784.

[2] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. Mar. 2015. URL: https://arxiv.org/abs/1503.02531?ref=hackernoon.com.

[3] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. "Model compression". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06* (2006). DOI: 10.1145/1150402.1150464.

[4] Michael Jordan and Robert Jacobs. "Hierarchical Mixtures of Expert and the EM Algorithm". In: *Neural Computation* 6 (Aug. 2001). DOI: 10.1162/neco.1994.6.2.181.

[2] https://github.com/kimhc6028/soft-decision-tree/tree/abe7a9b1690490cb450793d3417c7078ef1a6df1