Neural Decision Trees

Randall Balestriero*

* Electrical and Computer Engineering Department, Rice University, Houston TX, USA

* randallbalestriero@gmail.com

Abstract—In this paper we propose a synergistic melting of neural networks and decision trees (DT) we call neural decision trees (NDT). NDT is an architecture a la decision tree where each splitting node is an independent multilayer perceptron allowing oblique decision functions or arbitrary nonlinear decision function if more than one layer is used. This way, each MLP can be seen as a node of the tree. We then show that with the weight sharing asumption among those units, we end up with a Hashing Neural Network (HNN) which is a multilayer perceptron with sigmoid activation function for the last layer as opposed to the standard softmax. The output units then jointly represent the probability to be in a particular region. The proposed framework allows for global optimization as opposed to greedy in DT and differentiability w.r.t. all parameters and the input, allowing easy integration in any learnable pipeline, for example after CNNs for computer vision tasks. We also demonstrate the modeling power of HNN allowing to learn union of disjoint regions for final clustering or classification making it more general and powerful than standard softmax MLP requiring linear separability thus reducing the need on the inner layer to perform complex data transformations. We finally show experiments for supervised, semi-supervised and unsupervised tasks and compare results with standard DTs and MLPs.

Index Terms—Artifical Neural Networks, Multilayer PErceptrons, Decision Trees, Locality Sensitive Hashing, Oblique Decision Trees

I. INTRODUCTION AND DECISION TREE REVIEW

A. Motivations

HASHING. A hash function $H : X \to Y$ is a many-to-one mapping from a possibly infinite set to a finite set Harrison (1971) where we usually have $X \subset \mathbb{R}^d$ and $Y \subset \{1, ..., C\}$. As a result, the following function

$$H(x) = \arg\max_y p(y|x), \tag{1}$$

typical to discriminative models Jordan (2002), can be seen as a hashing function that has been optimized or learned given some dataset of N independent observations $\{(x_i, y_i)_{i=1}^N\}$ of input data x_i and corresponding label y_i. Decision trees (DT) are a special kind of discriminative model aiming at breaking up a complex decision into a union of simple binary decision a.k.a splitting nodes Safavian and Landgrebe (1990). In order to do so, DT learning involves a sequential top-down partitioning of the input space X into sub-regions Ω_k satisfying

$$\cup_k \Omega_k = X, \tag{2}$$
$$\Omega_k \cap \Omega_l = \emptyset, \forall k \neq l. \tag{3}$$

This partition is done so that the label distribution of the points w.r.t each region has minimal entropy. In particular, the optimum is obtained when all the points lying in a sub-region belong to the same class and this for all the sub-regions. Once trained, a new observation x is classified by first finding in which region $\Omega(x)$ it belongs to and by predicting the label specific to this region according to

$$H(x) = mode(\{y_i | x_i \in \Omega(x)\}), \tag{4}$$

for classification problems and

$$H(x) = mean(\{y_i | x_i \in \Omega(x)\}), \tag{5}$$

for regression problems. What brings DT among the most powerful discriminative technique for non cognitive tasks lies in the fact that the number of sub-regions Ω_k of X grows exponentially w.r.t their depth. This milestone is the core of the developed Hashing Neural Network (HNN) coupled with the acute modeling capacity of deep neural networks. We now describe briefly the standard univariate and multivariate decision trees, their advantages and drawbacks as well as motivations to extend them in a more unified and differentiable framework. In fact, as Quinlan said Quinlan (1994), decision trees are better when discrimination has to be done through a sequential testing of different attributes whereas ANN are good when knowledge of a simultaneous combination of many attributes is required, trying to get the best of both worlds seems natural.

B. Univariate and Oblique Decision Trees

Decision trees lead to a recursive space partitioning of the input space X through a cascade of simple tests Kuncheva (2004). In the case of univariate or monothetic DT, the local test for each splitting node is done by looking at one attribute att of $x \in X$ and comparing the taken values w.r.t. a threshold value b. If we have $x(att) < b$ then the observation is passed to the left child where another test is performed. This process is repeated until reaching a leaf in which case a prediction can be done for x. There is an intuition behind the sequence of simple tests performed by the tree to classify an object which is particularly useful in botany, zoology and medical diagnosis. By looking at all the leaves, one can see that they partition the input space into a set of axis-aligned regions. Since it has been proven that growing the optimal tree is NP-complete Hyafil and Rivest (1976) standard DT induction performs a greedy optimization by learning the best attribute and threshold for each split sequentially Quinlan (1986) unless everything is picked at random s.a. in Extremely Randomized DT Geurts et al. (2006). It is usually built in a top-down fashion but bottom-up and hybrid algorithms also exist. If a stopping criteria has been used during the growing phase it is called prepruning. Typical stopping criteria or prepruning can
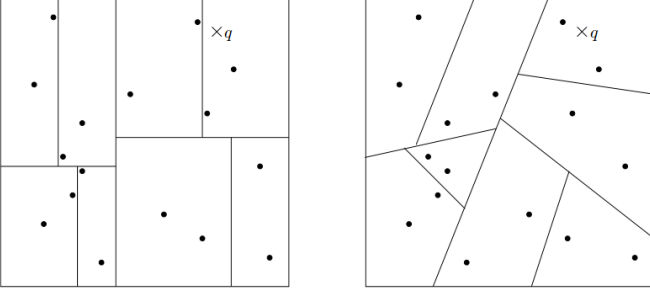
Fig. 2. Simple Tree

II. NEURAL DECISION TREES

We first introduce the neural decision tree (NDT) which is a soften version of decision trees allowing finely learned arbitrary decision surfaces for each node split and a global optimization framework. We first review briefly supervised LSH as the neural decision tree is a particular instance of a LSH framework.

A. Locality Sensitive Hashing

Locality Sensitive Hashing Gionis et al. (1999); Charikar (2002) aims at mapping similar inputs to the same hash value. In the case of trees, the hash value corresponds to the reached leaf. Learning this kind of function in a supervised manner has been studied Liu et al. (2012). For example in Xia et al. (2014) the similarity matrix induced by the labels is factorized into $H^T H$ and the features H are learned through a CNN. In Salakhutdinov and Hinton (2009) a deep autoencoder is learned in an unsupervised manner and the latent representation is then used for LSH. This last framework will be a special case of our unsupervised HNN with the main difference that the autoencoder will not just be trained to reconstruct the input but also provide a meaningful clustered latent space representation.

B. Model

The main change we perform on a decision tree to make it differentiable is to replace the splitting function which can be seen as an indicator function into a sigmoid function

$$\Phi(x) = \frac{1}{1 + e^{-x}}. \tag{6}$$

We now interpret for each node i, j the output of $\Phi(x)_{i,j}$ as the probability that the instance x goes to the left child of the node, note that this is a generalized version of the node change suggested in Laptev and Buhmann (2014). For example, looking at the tree representation 2, we have

$$P(x \in L_1) = P(X_1 = \text{left}|x)P(X_{22} = \text{left}|X_1 = \text{left}, x). \tag{7}$$

In fact, if we denote the attributes and threshold value for each node $X_{i,j}$ in the tree as $att_{X_{i,j}}$ and $b_{X_{i,j}}$, we can denote the probability of a point to be passed to the left child through the use of a sigmoid function as

$$P(X_{i,j} = \text{left}|x, X_{\pi_{i,j}}) = 1_{x(att_{X_1}) < b_{X_1}} \quad \text{for DT,} \tag{8}$$

$$P(X_{i,j} = \text{left}|x, X_{\pi_{i,j}}) = \Phi_{i,j}(x) \quad \text{for NDT,} \tag{9}$$

Fig. 1. Random Projection Tree example on $2D$.

involve a validation set, criteria on the impurity reduction or on the number of examples reaching a node. Finally, hypothesis testing using a Chi-Square test Goulden (1939) to see if the class distribution of the children is different than the parent can also be used. Early stopping can be detrimental by stopping the exploration, known as the horizon effect Duda et al. (2012), a DT can instead be fully developed and then postpruned using some heuristics which is a way to regularize the splits. In fact, in the case of a fully developed DT and if there is not two identical objects x_i, x_j in the dataset with different class label, the decision tree can learn the dataset, leading to zero re-substitution error but making them instable. In fact, it can memorize the training set and thus a small change in the input would lead to a completely different fitted tree. The main limitation of univariate DT resides in the axis-aligned splits. This inherently implies that the performance of DT is not invariant to the rotation of the input space and for cognitive tasks, tests that are done only on one attribute at a time lead extremely poor results if some hand-crafted features are not provided. As a result, oblique decision trees have been developed for which a test is now done on a linear combination of the attributes of x. Since the splitting is still not differentiable, optimizing the cutting value and the weight vector w is usually done with Genetic Algorithms (GA)Cantu-Paz and Kamath (2003). Finally, unsupervised pre-processing has also been developed with Random Projection Trees Dasgupta and Freund (2008); Blaser and Fryzlewicz (2015) and PCA Trees Verma et al. (2009); Sproull (1991) to avoid the rotation problems. A partitioning example is presented in Fig. I-B for a univariate and oblique tree. The contributions of the paper are: learning arbitrary decision boundaries as opposed to axis-aligned or linear through a global optimization framework as opposed to greedy optimization via differentiable splitting nodes. The derivation of the HNN allowing the use of deep neural networks with a new deep hashing layer which is related to Locality Sensitive Hashing (LSH). Finally, the use of the developed HNN for supervised and unsupervised problems as well as classification and regression tasks.
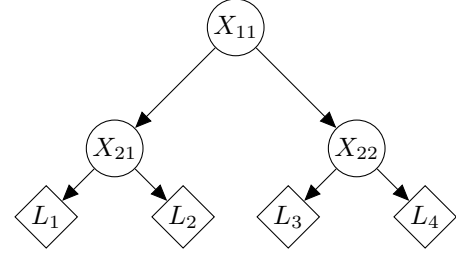
where $\Phi_{i,j}(x)$ denotes the sigmoid at node i,j containing the hyper-parameters and $\pi_{i,j}$ the decisions made by the parent of node i,j. More generally, we can apply the sigmoid after application of a general transformation of the data $G : \mathbb{R}^{\mathbb{D}} \Rightarrow \mathbb{R}$ such as a ANN as

$$P(X_{i,j} = \text{left}|x, X_{\pi_{i,j}}, G_{i,j}) = \Phi_{i,j}(G_{i,j}(x)). \qquad (10)$$

We now denote this probability directly as $\Phi_{i,j}(x)$ including the function $G_{i,j}$ and the parent decisions.

C. Local and Global Loss

We now introduce some notations in order to derive the overall loss function of the tree. We will derive the binary classification case only since the general case will be developed in the HNN section. We thus restrain $y \in \{0, 1\}$. In the case where each node is optimized in a top-down fashion we will first introduce some notation and derive the soft version of the usual DT splitting criteria. First, we have for each node i,j which is not a leaf

$$P = \sum_{n=1}^{n} y_n, \qquad (11)$$

$$N = \sum_{n=1}^{n} (1 - y_n), \qquad (12)$$

$$n_{left} = \sum_{n=1}^{n} \Phi(x_n), \qquad (13)$$

$$n_{right} = P + N - n_{left}, \qquad (14)$$

$$P_{left} = \sum_{n=1}^{n} \Phi(x_n) y_n, \qquad (15)$$

$$N_{left} = n_{left} - P_{left}, \qquad (16)$$

$$P_{right} = P - P_{left}, \qquad (17)$$

$$N_{right} = N - N_{left}, \qquad (18)$$

where these quantities represent respectively the number of positive and null observations, the observations going to the left and right node, the number of observation s.t. $y = 1$ going to the left child, the number of observation s.t. $y = 0$ going to the left child and similarly for the right child. We now present the main loss functions that can be used which are adapted from standard DT losses.

a) Information Gain (ID3De Mántaras (1991),C4.5Quinlan (2014)C5.0Im and Jensen (2005)): The information gain represents the amount by which the entropy of the class labels changes w.r.t the splitting of the dataset. It has to be maximized which happens when one minimizes the weighted sum of the local entropies, which in turn requires the class distribution of each region to converge to a Dirac. It is defined as

$$IG(x, y; \Phi) = E(P, N) - \frac{n_{left}}{P + N} * E(P_{left}, H_{left})$$
$$- \frac{n_{right}}{P + N} * E(P_{right}, N_{right}). \qquad (19)$$

b) Gini Impurity (CARTLewis (2000)): The Gini impurity is more statistically rooted. It has to be minimized and attains the global minimum at 0 happening when each region encodes only one class. It is defined for each region as

$$G(x, y) = 1 - \sum_{k} f_k^2, \qquad (20)$$

where f_k is the proportion of observation of class k. It symbolizes the expected classification error incurred if a class label was drawn following the class label distribution of the leaf. In fact, we have that

$$P(\hat{y}_i \neq y_i) = \sum_{k} P(\hat{y}_i = k) P(y_i \neq k)$$
$$= \sum_{k} f_k (1 - f_k)$$
$$= 1 - \sum_{k} f_k^2.$$

The loss function per node is thus the weighted Gini impurity for each of the children defined as

$$i_{left} = 1 - (\frac{P_{left}}{n_{left}})^2 - (\frac{N_{left}}{n_{left}})^2, \qquad (21)$$

$$i_{right} = 1 - (\frac{P_{right}}{n_{right}})^2 - (\frac{N_{right}}{n_{right}})^2, \qquad (22)$$

$$G(x, y, \Phi) = \frac{n_{left}}{N + P} i_{left} + \frac{n_{right}}{N + P} i_{right}, \qquad (23)$$

with N_k the number of observation of class k.

c) Variance Reduction (CART Breiman et al. (1984)): Finally, to tackle regression problems, another measure has been derived, the variance reduction. In this case, one aims to find the best split so that the intra-region variance is minimal. It is defined as

$$\frac{\sum_{i=1}^{N} \Phi(x_i)(y_i - \tilde{y}_1)^2}{\sum_{i=1}^{N} \Phi(x_i)} + \frac{\sum_{i=1}^{N}(1 - \Phi(x_i))(y_i - \tilde{y}_2)^2}{\sum_{i=1}^{N}(1 - \Phi(x_i))}, \qquad (24)$$

with

$$\tilde{y}_1 = \frac{\sum_{i=1}^{N} \Phi(x_i) y_i}{\sum_{i=1}^{N} \Phi(x_i)}, \qquad (25)$$

$$\tilde{y}_2 = \frac{\sum_{i=1}^{N}(1 - \Phi(x_i)) y_i}{\sum_{i=1}^{N}(1 - \Phi(x_i))}. \qquad (26)$$

Note that a weighted version of this variance reduction can be used w.r.t the probability of each region. Learning a NDT is now straightforward and similar to learning a DT except that now for each node instead of searching heuristically or exhaustively for the splitting criteria, it is optimized through an iterative optimization procedure such as gradient descend. This already alleviates the drawbacks of non differentiability encountered in oblique trees where GA had to be used to find the optimal hyperplane. However, it is also possible to go further by not just optimizing each splitting node in a greedy manner but optimizing all the splitting nodes simultaneously. In fact, with the Neural Decision Tree framework, we are now able to optimize the overall cost function simultaneously on all the nodes. This will loose the sequential aspect of the tests. However, this means that even though there exists no analytical

solution for the global loss as it was the case in the greedy framework, the likelihood of being stuck in a local optimum is smaller. In fact, a non optimal split at a given node does not degrade all the children performances. Thus the global loss function corresponds to the loss function of each last split node weighted by the probability to reach it. This is defined explicitly for the Gini impurity as

$$Gini(Tree, \mathbf{x}) = \sum_{leaf} \left[\frac{\sum_i P(x_i \in leaf)}{P + N} \right] \left(\frac{N_{leaf}}{P_{leaf} + N_{leaf}} \right.$$
$$\left. - \left(\frac{N_{leaf}}{P_{leaf} + N_{leaf}} \right)^2 \right), \quad (27)$$

and for the entropy as

$$IG(Tree, \mathbf{x}) = E(P, N)$$
$$- \sum_{leaf} \left[\frac{\sum_i P(x_i \in leaf)}{P + N} \right] E\left(P_{leaf}, N_{leaf} \right), \quad (28)$$

where the quantities $\left[\frac{\sum_i P(x_i \in leaf)}{P + N} \right]$ denote the probability that a given point belongs to this leaf which is estimated on the training set. As a result these loss functions correspond to the generalization of the node loss function for all the leaves weighted by the probability to go into each of the leaves.

III. HASHING NEURAL NETWORK

A. Motivation

One can see that for the special case where $G(x) = x^T w + b$ we can rewrite the NDT as a perceptron where the output neurons all have a sigmoid function. The result of the output which will be an ordered chain 010011... is simply equivalent to the path of the corresponding tree that would put x to the corresponding leaf as shown in Fig. 10. This is the main motivation for rewriting the NDT as the HNN to then be able to add multiple layers and leverage deep learning framework combined with the NDT. This idea of combining the topology of DT with the learning capacities of ANN is not new. For example, combining ANN for latent space representation followed by DT is done in Chandra et al. (2007). Generating rules based on a trained ANN with DT is studied in Fu (1994); Towell and Shavlik (1993); Kamruzzaman and Hasan (2010); Craven and Shavlik (1994); Craven (1996). Using an ANN to filter a dataset prior to learning a DT is explored in Krishnan et al. (1999). Finally and more recently, a reformulation of regression tree as a sparse ANN as been done in Biau et al. (2016) in order to fine-tune the learned DT. In our case, the HNN can be summarized simply as a generalized NDT in the sense that it will learn regions so that the class distribution within each of the region has the minimal uncertainty, ultimately with only one class per region but not necessarily one region per class as opposed to current architectures. In addition, we will see that thanks to the LSH framework, we are able to also perform HNN learning in a semi-supervised way or even unsupervised.
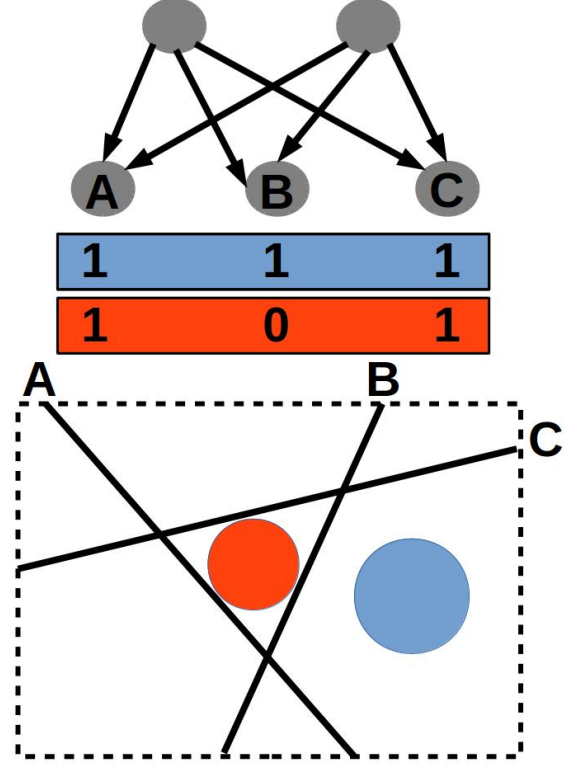

Fig. 3. Example of sub-region query

B. The Hashing Layer

In the HNN framework, the last layer now plays the role of an hashing function hence its hashing layer name. As a result, its output does not represent anymore $p(y|x)$ but $p(x \in L)$ where L is any of the sub-region encoded by the network, and the output of the neurons correspond to a prediction of the path taken in a decision tree. From the sub-region membership, a prediction policy can be used based on the most present training data class present for example. This fact already highlights the ability to also make confidence prediction. In fact, if a region is ambiguous, different kind of predictions can be done based on the problem at hand. We no

1) Supervised Case: The number of output neurons $\#out$ must be of at least $\log_2(C)$ where C is the number of classes in the task at hand. In fact, one needs a least C different regions and it is clear that the number of different region that can be modeled by an hashing layer is $2^{\#out}$. For the case where the number of output neurons is greater, then the ANN has the flexibility to learn different sub-regions for each class. This is particularly interesting for example if the latent representation is somehow clustered yet the number of cluster is still greater than the number of classes and the cluster of same class are not necessarily neighbors. In order to derive our loss function for multiclass problems we first need to impose the formulation of y as a one-hot vector with a 1 at the index of the class.

From that we have the following quantities

$$out_n(x) = \Phi(x^T w_n + b_n), \forall n \in \{1, ..., \#out\} \quad (29)$$
$$P = \{0, 1\}^{\#out} \quad (30)$$

and finally we call *chain* each element of P which fully and uniquely identify each of the sub-regions encoded in the network. We thus have

$$p(x_i \in chain) = \prod_n out_n(x_i)^{1_{chain(n)=1}} (1-out_n(x_i))^{1_{chain(n)=0}} \quad (31)$$

the average y per sub-region is thus the weighted mean of all the y w.r.t their membership probability, this will give the class distribution per *chain* as

$$y_{chain} = \sum_i y_i p(x_i \in chain), \quad (32)$$

where we recall that y_i is a one-hot vector and thus y_{chain} is a vector of size C which sums to 1 and is nonnegative. Thus it is $p(y|chain)$. Finally, we can now compute the measure of uncertainty E for each *chain* whether it is the entropy, the Gini impurity or any other differentiable function and write the final loss function as the weighted sum of these uncertainties weighted by the probability to reach each sub-regions as

$$loss = \sum_{x \in X} \sum_{chain \in P} p(chain)E(y_{chain}), \quad (33)$$

where as for the NDT the estimated probability to reach a sub-region is simply

$$p(chain) = \frac{\sum_{i=1}^N p(x_i \in chain)}{N}. \quad (34)$$

Note that all the classification loss can be used with class weights in order to deal with unbalanced dataset which occur often in real world problems. For regression problems we first define the local variance of the outputs as

$$V(leaf) = \frac{\sum_i (y_i - \tilde{y}_{chain})^2 p(x_i \in chain)}{\sum_i p(x_i \in chain)}, \quad (35)$$

where

$$\tilde{y}_{chain} = \frac{\sum_i y_i p(x_i \in chain)}{\sum_i p(x_i \in chain)}. \quad (36)$$

As a result we can rewrite the complete regression loss as

$$loss = \sum_{x \in X} \sum_{chain \in P} p(chain)V(leaf). \quad (37)$$

Note that it is possible to not weight the local loss functions by the probability to reach the region.

Finally, note that this hashing layer can obviously be used after any already known neural network layer such as densely connected Gardner and Dorling (1998) or convolutional Krizhevsky et al. (2012) layers or even recurrent based layers Gers et al. (2000) making HNN acting on deep and latent representation rather than the input space. The training is thus done through back-propagation with the presented loss.

2) Semi-Supervised Learning: One extension not developed in the paper concerns the semi-supervised case for which the standard discriminative loss is used for labeled examples s.a. the Information Gain of the Entropy and the unsupervised loss is used for the unlabbeled examples as well as the labeled examples. The unlabeled loss would tipycally be the intra region variance which is similar to the loss of k-NN algorithm or GMM with identity covariance matrix. This result in aggregating into same region parts of the space with high density while constraining that two different labels never occur inside one region. Extensions of this will be presented as well as validation results in order to validate this hybrid loss between Decision Trees and k-NN.

3) Unsupervised Case: In the last section a loss function was derived for the case where we have access to all the labels y_i of the training inputs x_i. The semi-supervised framework is basically a deep-autoencoder on which a hashing layer is connected to the latent space, namely the middle layer. As a result, the reconstruction loss is applied to all inputs and the standard hashing layer loss is used when labels are available. For unsupervised however, the deep autoencoder is trained again coupled with an hashing layer but this time the hashing layer is unsupervised. We now derive analytically the loss function of the hashing layer in the unsupervised case. There are two ways to do it, it can either be random to become a known LSH function such as MinHash or one can try to find clusters so that the intra-cluster variance is minimal similarly to a k-NN Fukunaga and Narendra (1975) approach. The variance per region is defined as

$$V_{chain}(x) = \frac{\sum_i (x_i - \tilde{x}_{chain})^2 p(x_i \in chain)}{\sum_i p(x_i \in chain)}, \quad (38)$$

where the local sub-region means is defined as

$$\tilde{x}_{chain} = \frac{\sum_i p(x_i \in chain)x_i}{\sum_i p(x_i \in chain)}. \quad (39)$$

As a result, the overall loss is simply

$$loss = \sum_{x \in X} \sum_{chain \in P} p(chain)V(x). \quad (40)$$

If we denote by G the last layer of the neural network before the hashing layer, the final unsupervised loss thus becomes

$$loss = \sum_{x \in X} ||G^{-1}(G(x)) - x||^2 + \sum_{chain \in P} p(chain)V(G(x)), \quad (41)$$

and for the semi-supervised case

$$loss = \sum_{x \in X} ||G^{-1}(G(x)) - x||^2$$
$$+ \sum_{chain \in P} p(chain)E(G(x))1_{x \text{ has a label}}. \quad (42)$$

Concerning the random strategy, similarly to Extremely Randomized Trees, it is possible to adopt an unlearn approach for which the hyperplanes w_b are drawn according to a Normal distribution. This way, we follow the LSH framework for which we have

$$p(out_n(x_i) \neq out_n(x_j)) \propto d(x_i, x_j), \quad (43)$$

where d is a distance measure.

C. Training

1) Iterative Optimization Schemes: Since there is no analytical form to find the optimal weights and bias inside a deep neural network, one has to use iterative optimization methods. Two of the main possibilities are Genetic Algorithms Davis (1991) and Gradient based methods. We focus here on the advances for gradient based methods as it is the most popular optimization technique nowadays. Put simply, the update for each free parameter W, the update rule is

$$W = W - \alpha \nabla W + \beta f(W), \tag{44}$$

where α is the learning rate and β a regularization parameter applied on some extra function f. A common technique to find the best learning rate and regularizer is cross-validation but new techniques have been developed allowing an adaptive learning rate and momentum which are changed during training Yu et al. (2006); Yu and Liu (2002); Hamid et al. (2011); Nawi et al. (2011). Whatever activation function is used, one can also add new parameters to the input in order to scale the input as presented in He et al. (2015). Finally, many tricks are studies for better back-propagation in LeCun et al. (2012) and a deep study of the behavior of the weights during learning is carried out in LeCun et al. (1991). We now derive the explicit gradient for the case where the loss E is the Gini impurity. It is clear that

$$\frac{d}{dW} loss = \sum_n \frac{dloss}{dout_n} \frac{dout_n}{dW}$$
$$= \sum_{chain \in P} \sum_n \frac{dp(chain)E(y_{chain})}{dout_n} \frac{dout_n}{dW} \tag{45}$$

with $\frac{dp(chain)E(y_c hain)}{dout_n}$ a scalar and $\frac{dout_n}{dW}$ a matrix. We now derive explicitly the derivative:

$$\frac{dout(x_i)}{dW} = \begin{pmatrix} (-1)^{1_{chain(1)=0}} out_1(x_i)(1 - out_1(x_i))x_i^T \\ (-1)^{1_{chain(2)=0}} out_2(x_i)(1 - out_2(x_i))x_i^T \\ \vdots \end{pmatrix}, \tag{46}$$

which is of size $(\#out, \#in)$ and is basically on each row the input x_i which might be the output of another upper layer weighted by the activation, it is similar to a sigmoid based layer except for the indicator function which helps to determine the sign. Now if we now denote the true output by

$$\sigma_n^{chain(n)}(x_i) = out_n(x_i)^{1_{chain(n)=1}}(1 - out_n(x_i))^{1_{chain(n)=0}}, \tag{47}$$

which includes the indicator function for clarity, we have

$$\frac{\partial}{\partial out_n} p(chain) = \frac{\partial}{\partial out_n} \frac{\sum_i \prod_n \sigma_n(x_i)}{N}$$
$$= \frac{\sum_i \sum_k (-1)^{1_{chain(k)=0}} \prod_{n \neq k} \sigma_n^{chain(n)}(x_i)}{N}. \tag{48}$$

Note that the derivative of the probability w.r.t the output is thus simply a sum of the products of the other output neurons where the neuron considered in the derivative determines the sign applied. In short, we see that this changes linearly as

one fixes all the neurons but the one considered for variations, which is natural. We now derive the final needed derivate:

$$\frac{\partial}{\partial out_n} E(y_{chain}) = \frac{\partial}{\partial out_n}(1 - \sum_k f_k^2)$$

$$= -\sum_k \frac{\partial}{\partial out_n} \left(\frac{\sum_i y_{i,k} p(x_i \in chain)}{\sum_j \sum_i y_{i,j} p(x_i \in chain)} \right)^2$$

$$= -\sum_k \frac{\partial}{\partial out_n} \left(\frac{\sum_i y_{i,k} \prod_n \sigma_n^{chain(n)}(x_i)}{\sum_j \sum_i y_{i,j} \prod_n \sigma_n^{chain(n)}(x_i)} \right)^2$$

$$= -\sum_k 2f_k \left(\frac{(\sum_i y_{i,k} \sum_l (-1)^{1_{chain(l)=0}} \prod_{n \neq l} \sigma_n^{chain(n)}(x_i))}{(\sum_j \sum_i y_{i,j} \prod_n \sigma_n^{chain(n)}(x_i))^2} \right.$$

$$\times (\sum_j \sum_i y_{i,j} \prod_n \sigma_n^{chain(n)}(x_i)) \Bigg)$$

$$+ \sum_k 2f_k \left(\frac{(\sum_i y_{i,k} \prod_n \sigma_n^{chain(n)}(x_i))}{(\sum_j \sum_i y_{i,j} \prod_n \sigma_n^{chain(n)}(x_i))^2} \right.$$

$$\times (\sum_j \sum_i y_{i,j} \sum_l (-1)^1_{chain(l)=0} \prod_{n \neq l} \sigma_n^{chain(n)}(x_i)) \Bigg)$$

Finally, note that performing stochastic gradient descent in an option when dealing with large training set. It has also been shown to improve the convergence rate.

2) Regularization Techniques: Initially, regularization was done by adding to the standard cost function a regularization term, typically the L1 or L2 norm of the weights. This imposed the learned parameters to be sparse. From that new kind of regularization techniques have been developed such as dropout Hinton et al. (2012), Srivastava et al. (2014). In dropout, each neuron has a nonzero probability to be deactivated (simply output 0) forcing the weights to avoid co-adaptation. This probabilistic deactivation can be transformed into adding some Gaussian noise to each of the neuron outputs which in this case force the weights to be robust to noise. The motivation here is not necessarily to avoid overfitting, in fact, if using ensemble methods, over fitting is actually better to then leverage variance reduction from averaging Krogh (1996). However another type of regularization can be used on the distribution of the data across the regions. One such type might be

$$\sum_{chain \in P} ||p(chain) - \frac{1}{|P|}||^2 \tag{49}$$

so that regions become more equally likely.

D. Toy Dataset

We now present the application of the HNN on two simple toy datasets for binary classification, the two moons and circle dataset. Each one presents nonlinearly separable data points yet the boundary decision can very effectively be represented by a small union of linear plans. The main result is that the number of parameters needed with the HNN is smaller than when using an ANN. For all the below examples, the ANN is made of 3 layers with topology $2 : 3 : 1$ which is the smallest network able to tackle these two problems. As we will see,
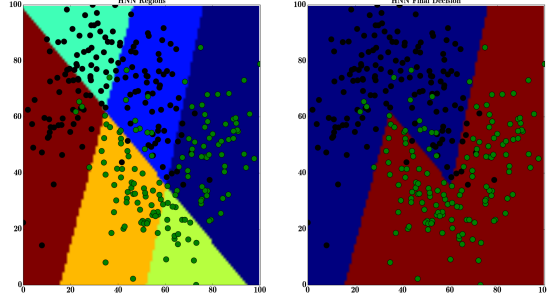
Fig. 4. Noisy Two-Moons Dataset, learned regions and final binary classification regions for HNN.
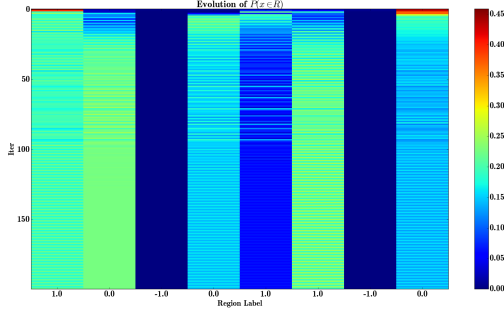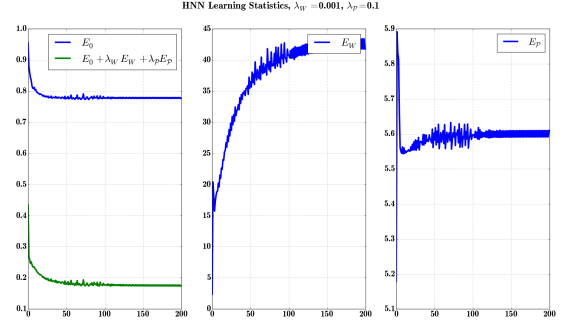

Fig. 6. Evolution of the errors during training. On the left is the pure error as defined in 33 and with addition of the regularization terms. On the middle is the norm of the weight and on the right the distance with respect to a uniform distribution of the points in each region.

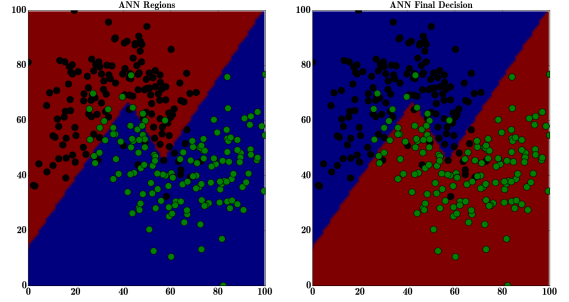Fig. 5. Evolution of $p(chain)$ over the 8 regions learned during training.

Fig. 7. Noisy Two-Moons Dataset, learned regions and final binary classification regions for HNN.

Fig. 8. Evolution of $p(y = 0)$ and $p(y = 1)$ during training of the MLP.

the HNN only requires one layer and the number of needed parameters for similar decision boundary goes for from 13 for ANN to 9 for HNN. This simple example shows that the reformulation of the hashing layer helps to avoid overfitting in general. In fact, overfitting is not necessarily learning the training set but it is also using a more flexible model that needs be Hawkins (2004). Since with the HNN we are able to obtain the same decision boundaries yet with less parameters, it means that the ANN architecture was somehow sub-optimal.

1) Two-moon Dataset: The two-moon dataset is a typical example of nonlinear binary classification. It can be solved easily with kernel based methods or nonlinear classifiers in general. As we will show, even though the used HNN has only one layer, by the way it combines the learned hyper-plans it is possible to learn nonlinear decision boundaries. In Fig. 4 one can see the HNN with $\#out = 3$ after training. The boundary decision is similar in shape with the one learned form a $2 : 3 : 1$ MLP shown in Fig. 7. In fact, it is made up of 3 combined hyper-plans. One can see in Fig. 5 the evolution of the probability to reach each of the 8 regions during training as opposed to the 2 regions for the case of the MLP presented in Fig. 8. We also present in Fig. 6 the evolution of the error and regularizations during training for the HNN. As one can see, regions starting with almost no points can recover and become preponderant whereas useful regions can be disregarded at any point in the training. Similarly we have for Fig. 9 the case with the ANN. It is interesting to see that the convergence rate is also faster with the HNN. In fact it converges in 50 iterations

whereas ANN converges in around 150 iterations.

2) Two-circle Dataset: The two-circle dataset is another simple yet meaningful dataset. It presents two circles with same center with different radius, as a consequence one is inside the other and the binary classification task is to discriminate between the two. It is quite straightforward to see that a hand-craft change of variable $(x, y) \rightarrow (r, \theta)$ can make this problem linearly separable yet we will see how HNN and ANN solve this discrimination problem. We present in Fig. 10 the result of the HNN with 3 neurons. Again, the decision boundary is also presented for the case of an ANN
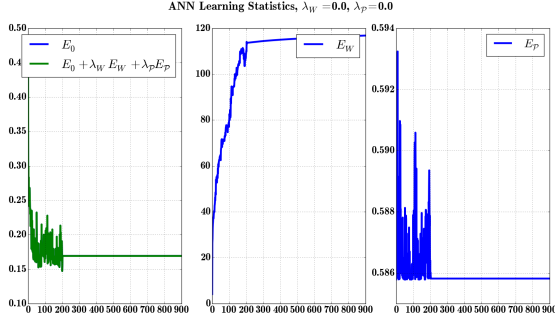
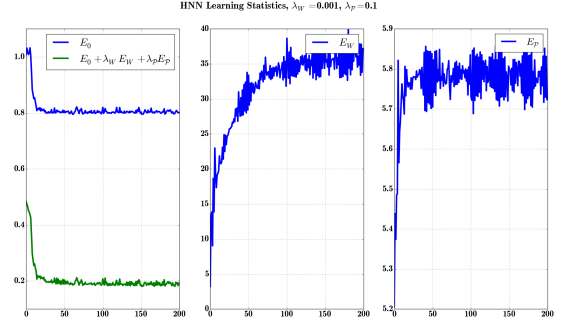Fig. 9. Evolution of the errors during training for the ANN.

Fig. 12. Evolution of the error and regularization for the HNN with 3 neurons.
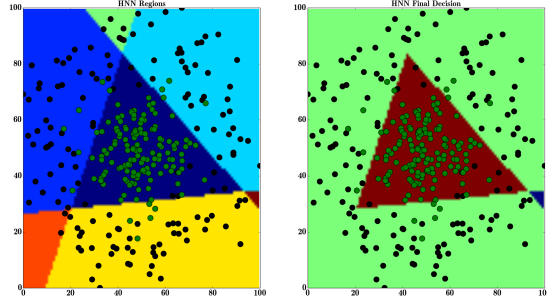

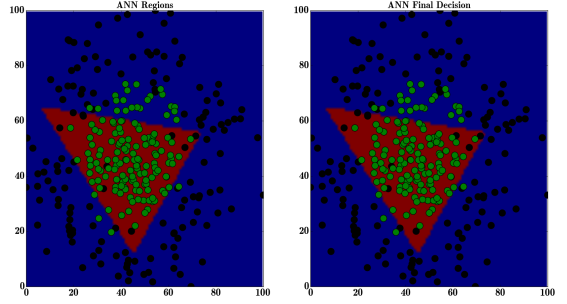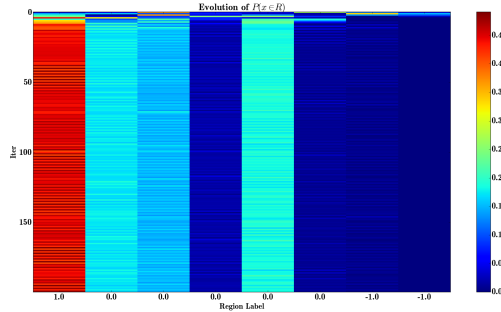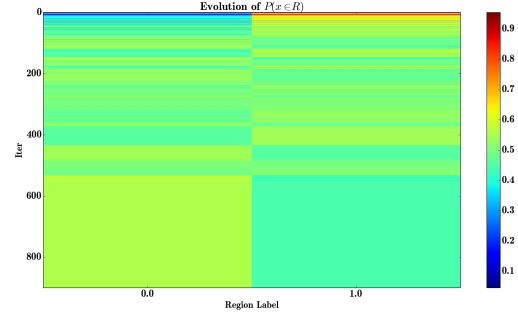Fig. 10. Regions and decision boundary for the HNN in the two-circle dataset.

Fig. 13. ANN Decision Boundary for the two-circle dataset

Fig. 11. Evolution of $p(chain)$ during training for the 8 possible regions of the HNN on the two-circle dataset.

Fig. 14. ANN $p(chain)$ for the two classes

with topology $2 : 3 : 1$ in Fig. 13. We also present in Fig. 11 for the HNN and Fig. 14 for the ANN the evolution of the probability to reach the sub-regions during training. As can be seen in Fig. 12 and 15 the convergence rate is significantly faster for the HNN. In fact the convergence is done in about 20 iteration whereas the neural network needs a bit more than 300 iterations. Finally, we present the case where we used 4 neurons for the HNN on the two-circle dataset. Note that this puts the number of parameters to 12 which is still less than the number of parameters used in the ANN 13. Yet if one wanted to have the same modeling ability with a MLP, the minimum topology would be $2 : 4 : 1$ which contains 17 parameters. In fact, it grows exponentially w.r.t the number of hidden neurons whereas for the HNN it grows linearly since
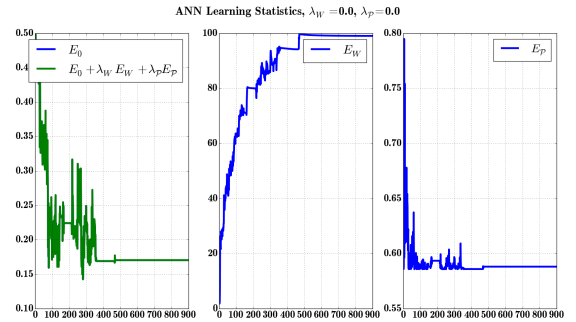

Fig. 15. ANN training statistics including the error and the regularizations
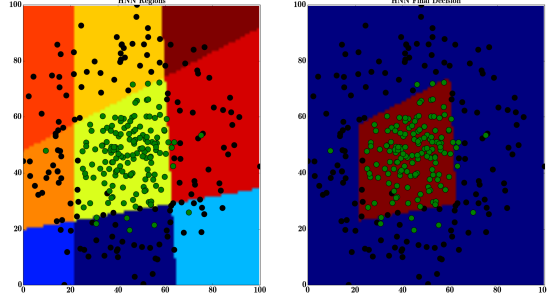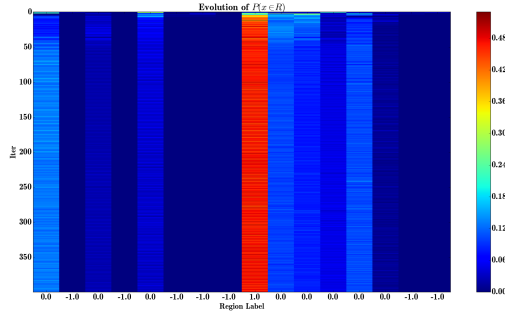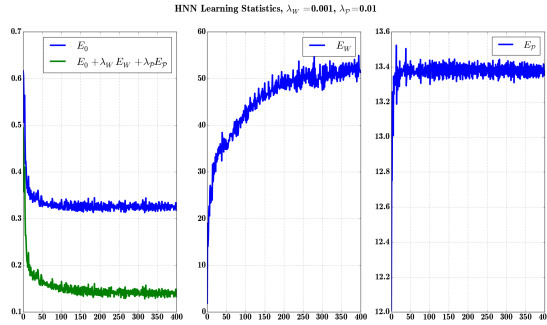
Fig. 16.   4 neurons HNN

Fig. 17.   4 neurons HNN, evolution of $p(chain)$.

Fig. 18.   4 neurons HNN, evolution of the error and regularization errors during training.

the modeling capacity grows exponentially. We present in Fig. 16, Fig. 17 and Fig. 18 the results for the 4 neurons HNN.

IV. FUTURE WORK

This paper introduces a new way to improve neural networks through the analysis of the output activations and the loss function. In fact, it has been demonstrated for example in Tang (2013) a SVM type loss is used instead of the standard cross-entropy. As a result the generalization loss has been diminished. Yet the important point was that the cross-entropy of the new trained network was far from optimal or even close to what one could consider as satisfactory. This suggests that different loss functions do not just affect the learning but

also the final network. As a result, one part of the future work is to study the impact of the loss function on different training set with fixed topologies. This includes the training with standard neural networks and the HNN for the case where $\#out = \log_2(n)$. This could lead to a new framework aiming at learning the loss function online.

With the analysis of trees one natural extension of the HNN is boosting or its simpler form, bagging. Doing this ensemble methods with complete ANN might be difficult due to high complexity to already train one network. As a result, some techniques such as dropout have been used and analyzed as a weak way to perform model averaging or bagging. A solution in our case would be to perform bagging or boosting of only the hashing layer part, namely the last layer of the HNN. This way the workers live on the same latent space but act on different pieces of it and their combination is used for improving the latent space representation. Another approach would be to see model averaging from a bayesian point of view Penny et al. (2006). The model evidence is given by

$$p(y|m) = \int p(y|\theta, m)p(\theta|m)d\theta, \tag{50}$$

and the Bayesian Model Selection (BMS) is

$$m_{MP} = \arg\max_{m \in M} p(m|y). \tag{51}$$

When dealing with one model m, the inference $p(\theta|y, m)$ is dependent w.r.t to the chosen model. In order to take into account the uncertainty in the choice of the model, model averaging can be used. Model Averaging (BMA) formulates the distribution

$$p(\theta|y) = \sum_m p(\theta|y, m)p(m|y), \tag{52}$$

look for a review in Hoeting et al. (1999). This whole framework can be used with a nonparametric PGM with a new define probability for a neural network, and $p(x|ANN)$ as the reconstruction error and $p(ANN)$ based on the model complexity for example or the topology of the connections.

Finally, an important aspect resides in the correlation between different $out_n(x_i)$ and $out_m(x_i)$ for different inputs. In fact, they have to be not correlated otherwise it means that the hyperplanes are scaled versions of each others.

V. CONCLUSION

In this paper we presented an extension of DTs and ANNs to provide a unified framework allowing to take the best of both approaches. The ability to hash a dataset into an exponentially large number of regions which is the strenfgth of DT coupled with the ability to learn any boundary decision for those region coming from the ability of ANN to model arbitrary functions. We leverage the differentiable of our approach to derive a global loss function to train all the nodes simultaneously with respect to the resulting leaves entropy showing robustness to poor local optimum DTs can fall in. The differentiability of the model allows easy integration in many machine learning pipeline allowing the extension of CNNs to robust semi-supervised clustering for example. In addition, the ability to learn arbitrary union of regions of the space to perform a per

class clustering reduces the necessary condition of having fully linearized the dataset. This shall reduce the required depth of today's deep architectures. Finally, this network has now the capacity to be information theoretically optimal as the minimum required number of neurons in supervised problems is $\log_2(C)$ as opposed to C for usual soft-max layers where C is the number of classes. Finally, the possibility to apply this framework in supervised as well as unsupervised settings might lead to interesting behavior through the clustering property of the latent representations as the experiments showed promising results.

References

G. Biau, E. Scornet, and J. Welbl. Neural Random Forests. *ArXiv e-prints*, April 2016.

Rico Blaser and Piotr Fryzlewicz. Random rotation ensembles. *J Mach Learning Res*, 2:1–15, 2015.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

Erick Cantu-Paz and Chandrika Kamath. Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(1):54–68, 2003.

Rohitash Chandra, Kaylash Chaudhary, and Akshay Kumar. The combination and comparison of neural networks with decision trees for wine classification. *School of sciences and technology, University of Fiji, in*, 2007.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

Mark Craven and Jude W Shavlik. Using sampling and queries to extract rules from trained neural networks. In *ICML*, pages 37–45, 1994.

Mark W Craven. *Extracting comprehensible models from trained neural networks*. PhD thesis, University of Wisconsin–Madison, 1996.

Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 537–546. ACM, 2008.

Lawrence Davis. Handbook of genetic algorithms. 1991.

R López De Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine learning*, 6 (1):81–92, 1991.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

LiMin Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(8): 1114–1124, 1994.

Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE transactions on computers*, 100(7):750–753, 1975.

Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14): 2627–2636, 1998.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

Cyril Harold Goulden. Methods of statistical analysis. 1939.

Norhamreeza Abdul Hamid, Nazri Mohd Nawi, Rozaida Ghazali, and Mohd Najib Mohd Salleh. Accelerating learning performance of back propagation algorithm by using adap-

tive gain together with adaptive momentum and adaptive learning rate on classification problems. In *Ubiquitous Computing and Multimedia Applications*, pages 559–570. Springer, 2011.

Malcolm C Harrison. Implementation of the substring test by hashing. *Communications of the ACM*, 14(12):777–779, 1971.

Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Jennifer A Hoeting, David Madigan, Adrian E Raftery, and Chris T Volinsky. Bayesian model averaging: a tutorial. *Statistical science*, pages 382–401, 1999.

Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

Jungho Im and John R Jensen. A change detection model based on neighborhood correlation image analysis and decision tree classification. *Remote Sensing of Environment*, 99(3):326–340, 2005.

A Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841, 2002.

SM Kamruzzaman and Ahmed Ryadh Hasan. Rule extraction using artificial neural networks. *arXiv preprint arXiv:1009.4984*, 2010.

R Krishnan, G Sivakumar, and P Bhattacharya. Extracting decision trees from trained neural networks. *Pattern Recognition*, 32(12), 1999.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Peter Sollich Anders Krogh. Learning with ensembles: How over-fitting can be useful. In *Proceedings of the 1995 Conference*, volume 8, page 190, 1996.

Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

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

Yann LeCun, Ido Kanter, and Sara A Solla. Second order properties of error surfaces: Learning time and generalization. In *Advances in neural information processing systems*, pages 918–924, 1991.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Roger J Lewis. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, pages 1–14, 2000.

Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2074–2081. IEEE, 2012.

Nazri Mohd Nawi, Norhamreeza Abdul Hamid, RS Ransing, Rozaida Ghazali, and Mohd Najib Mohd Salleh. Enhancing back propagation neural network algorithm with adaptive gain on classification problems. *networks*, 4(2), 2011.

Will Penny, J Mattout, and N Trujillo-Barreto. Bayesian model selection and averaging. *Statistical Parametric Mapping: The analysis of functional brain images. London: Elsevier*, 2006.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

John Ross Quinlan. Comparing connectionist and symbolic learning methods. In *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects*. Citeseer, 1994.

S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. 1990.

Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7): 969–978, 2009.

Robert F Sproull. Refinements to nearest-neighbor searching ink-dimensional trees. *Algorithmica*, 6(1-6):579–589, 1991.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Yichuan Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

Geoffrey G Towell and Jude W Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine learning*, 13(1):71–101, 1993.

Nakul Verma, Samory Kpotufe, and Sanjoy Dasgupta. Which spatial partition trees are adaptive to intrinsic dimension? In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 565–574. AUAI Press, 2009.

Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, volume 1, page 2, 2014.

Chien-Cheng Yu and Bin-Da Liu. A backpropagation algorithm with adaptive learning rate and momentum coefficient. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1218–1223. IEEE, 2002.

Lean Yu, Shouyang Wang, and Kin Keung Lai. An adaptive bp algorithm with optimal learning rates and directional error correction for foreign exchange market trend prediction. In *Advances in Neural Networks-ISNN 2006*, pages 498–503. Springer, 2006.