MASTER'S THESIS

Bc. Martin Oharek

December 2019

Contents

Chapter 1

Introduction

Chapter 2

Classification

2.1 State-of-the-art classification models

2.1.1 Logistic regression

2.1.2 Support vector machines (SVM)

2.1.3 Random forest

Decision tree

Ensemble of decision trees

2.1.4 Neural network (NN)

Perceptron and neuron

Activation functions

Backpropagation algorithm

2.2 Evaluation metrics

Chapter 3

Decision-Tree-Inspired NN architecture

In this section is provided detailed derivation of the basic neural network architecture, which is further employed in all subsequently proposed models. The main idea is built on the article [1] describing 1 to 1 transformation of arbitrary regression tree (random forest regressor) to the specifically designed neural network (ensemble of neural networks).

Unfortunately, an approach presented in [1] is not uniformly convertible to the classification problem. Based on this issue, the alternative architecture and initial settings are proposed in order to simulate the exact behaviour of corresponding decision tree classifier.

This reformulation provides a sensible opportunity to enhance the decision tree performance, because parameters of newly constructed neural network could be better adapted with usage of the backpropagation algorithm of neural networks and therefore achieve superior classification.

3.1 Architecture and initial settings

The initial weights and biases of neurons and architecture of input layer and first two hidden layers will remain same among all proposed models. The settings was motivated by [1]. A sample of such architecture (only input layer and first two hidden layers) is illustrated in Figure 3.2 copying decisions of the decision tree depicted in 3.1, which splits the space by two hyperplanes as illustrated also in 3.1. Let's first consider neurons in the first and second hidden layer of network as perceptrons. This means, that activation function is

$$\tau(\boldsymbol{x}) = 2\mathbf{1}_{\boldsymbol{x}\geq 0} - 1 \quad , \tag{3.1}$$

where all math operations on vector $\boldsymbol{x}$ are element-wise. Also

$$(\mathbf{1}_{\boldsymbol{x}\geq 0})_i = \begin{cases} 1, & \text{if } \ x_i \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad , \tag{3.2}$$

where $(\mathbf{1}_{\boldsymbol{x}\geq 0})_i$ is i-th element of $\mathbf{1}_{\boldsymbol{x}\geq 0}$.

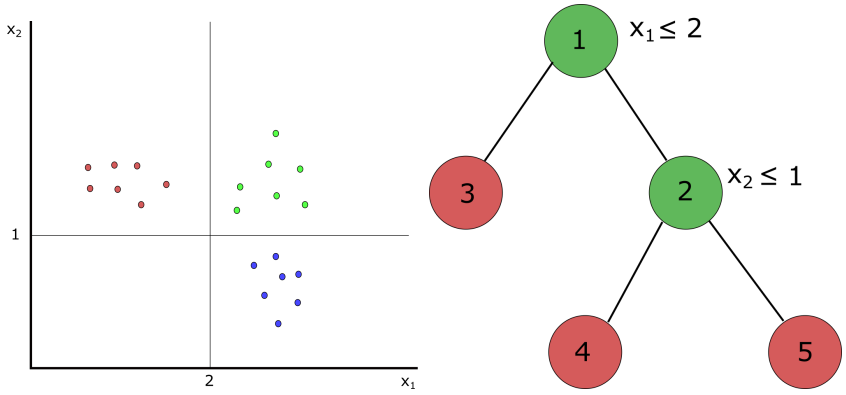

Figure 3.1: In the left Figure are depicted 3 color-coded classes divided by the decision tree in the right Figure, that are separated by two hyperplanes: $x_1 - 2 = 0$ and $x_2 - 1 = 0$. Inner nodes of the decision tree are green colored with corresponding split function next to them, whereas the leaves are red colored. All nodes are numbered.

So perceptrons from the first and second hidden layer outputs $+1$ or -1. Why is it chosen in such manner will be clear from further explanations.

3.1.1 First hidden layer

The first hidden layer should copy decisions of all inner nodes present in the corresponding decision tree. The first hidden layer has same number of neurons as number of inner nodes. As was explained in the chapter about decision trees and random forests, each inner node $k \in \{1, ..., L-1\}$ of the decision tree, where $L-1$ is total number of inner nodes (in fact, if $L-1$ is total number of inner nodes, then L is total number of leaves present in the decision tree), possesses split function with parameters $j_k \in \{1, ..., n\}$, which is one dimension in a n-dimensional space that is used for split and also α_{j_k}, which is a threshold. Let's also define function s_k as

$$s_k(\boldsymbol{x}) = x_{j_k} - \alpha_{j_k} \quad . \tag{3.3}$$

It is apparent that equation $s_k(\boldsymbol{x}) = 0$ defines a hyperplane in $\mathbb{R}^n$, which splits the space in inner node k.

In order to obtain all decisions of inner nodes in the corresponding neurons of our neural network, we initialize weights in neuron k as $(0, .., 0, 1, 0, .., 0)^T$ with

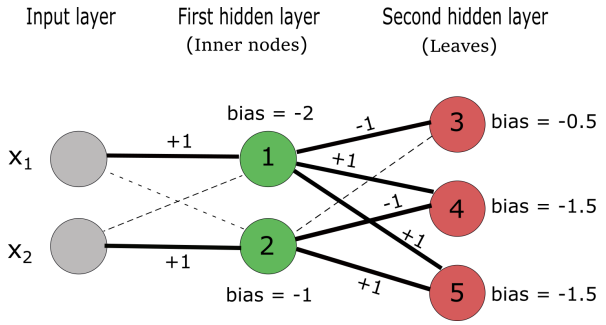

Figure 3.2: Transformation of the decision tree in Figure 3.1 to the neural network with two hidden layers. The first hidden layer detects the decisions of inner nodes same numbered as in 3.1. The second hidden layer retrieves leaf membership of input vector same as in the decision tree. Not null connections between neurons are bold higlighted with corresponding weights. Biases are written next to the neurons. All dashed connections indicates null connection (weight equals 0).

single 1 in j_k-th position and 0 otherwise. Bias is set to $-\alpha_{j_k}$. Hence, output of the first hidden layer is $(\tau(s_1(\boldsymbol{x})), \tau(s_2(\boldsymbol{x})), ..., \tau(s_{L-1}(\boldsymbol{x}))$ and it precisely copies decisions of inner nodes, with +1 indicating that input vector belongs to the right side of the hyperplane and -1 otherwise (and +1 if it belongs to the hyperplane). This is also done for the inner nodes outside the path of the input vector. The illustration can be seen in Figure 3.3.

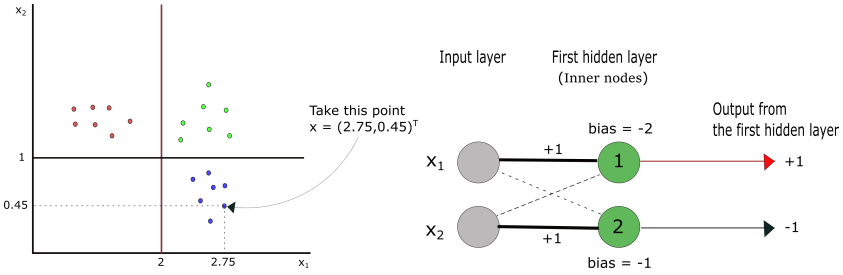

Figure 3.3: Space in the left figure is divided by red (in inner node number 1) and black (in inner node number 2) hyperplanes. If we consider point $\boldsymbol{x} = (x_1, x_2)^T = (2.75, 0.45)^T$ highlighted in left Figure, the output from the first hidden layer is +1 from neuron 1 (corresponding to the red hyperplane) and -1 from neuron 2 (corresponding to the black hyperplane).

3.1.2 Second hidden layer

From all inner node decisions obtained by the first hidden layer it should be possible to reconstruct the exact leaf membership of the input vector $\boldsymbol{x}$. This is the main task to accomplish by the second hidden layer. If there are $L-1$ neurons in the first hidden layer, then the second hidden layer consists of L neurons, each one corresponding to the one individual leaf in the decision tree.

We connect neuron m in the first hidden layer to neuron m' in the second hidden layer with not null weight if and only if inner node corresponding to the neuron m belongs to the path from root node to the leaf corresponding to the neuron m'. The weight is initialized to $+1$ if the split by inner node m is to the right child and -1 otherwise. If neuron m is not part of the path from root to the leaf, the weight is initialized always to 0.

Based on this setting, it could be simply deduced that number of not null connections from the first hidden layer (weights) to the (arbitrary) neuron m' in the second hidden layer is same as length of the path from root to the leaf m'. This is illustrated in Figure 3.4.

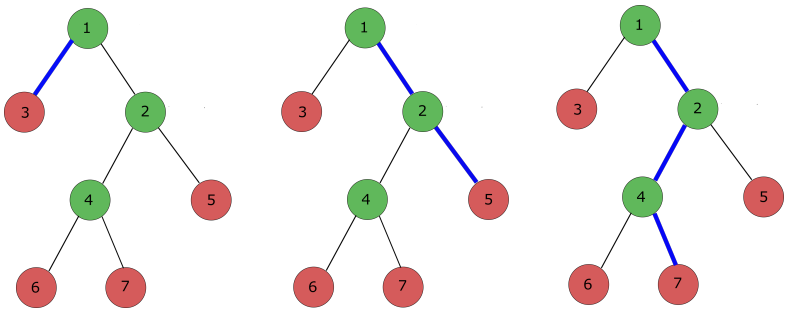

Figure 3.4: In the left figure is highlighted blue path from the root node 1 to the leaf 3. This has length 1 and also only one initialized not null connection from the first hidden layer exists, because only root node is part of the path. In the remaining figures are depicted paths for other leaves (blue colored) from the root node (except of leaf 6, which is in same depth as leaf 7). It is easy to see, that equality between length of the path from the root node to the particular leaf and number of not null connections from the first hidden layer to the corresponding neuron holds always.

If output from the first hidden layer is $\boldsymbol{v} = (\pm 1, \pm 1, ..., \pm 1)^T$, which encodes all decisions of inner nodes of the decision tree, then output from the neuron m' in the second hidden layer is $\tau(\sum_{i=1}^{L-1}(w_i^{m'} v_i) + bias(m'))$, where $\boldsymbol{w}^{m'} = (w_1^{m'}, w_2^{m'}, ..., w_{L-1}^{m'})^T$ is a vector of weights for connections to neuron m'. These weights are not null if the corresponding inner node is involved in the path from root to the leaf m' and are $+1$ if it is sent to the right child and -1 otherwise. For all inner nodes that are not involved in the root-leaf path are weights initialized to 0.

Desired behaviour of the neuron m' in the second hidden layer is to output

+1 if the input vector ends in leaf m' and -1 otherwise. For this purpose, $bias(m')$ must be correctly set. To do so, it is sufficient to notice that the sum $\sum_{i=1}^{L-1}(w_i^{m'}v_i)$ as the first term in the argument of $\tau(\cdot)$ function equals to the length of the path from root node to leaf m', if and only if the input ends in leaf m'. For convenience, let us denote this length as $l(m')$. It is a simple consequence of the fact, that the number of not null weights $w_{i_k}^{m'}$ is same as $l(m')$ and also they have same magnitude ($|w_{i_k}^{m'}| = 1$) and same sign as v_{i_k}, where $i_k \in \{1, ..., L-1\}, w_{i_k}^{m'} \neq 0$. Therefore, each member of the sum $\sum_{i=1}^{L-1}(w_i^{m'}v_i) = \sum_{i_k=1, w_{i_k}^{m'}\neq 0}^{L-1}(w_{i_k}^{m'}v_{i_k})$ equals to +1 and all members sum up to $l(m')$.

Moreover, if the input does not end in leaf m', then in the sum $\sum_{i=1}^{L-1}(w_i^{m'}v_i)$ exists a not null member in which interfere 2 integers (ones) with different signs, resulting in -1. Hence is clear, that if input does not end in leaf m', the sum holds inequality $\sum_{i=1}^{L-1}(w_i^{m'}v_i) \leq l(m') - 1 < l(m')$. For more precise intuition, the illustration is provided in Figure 3.5.

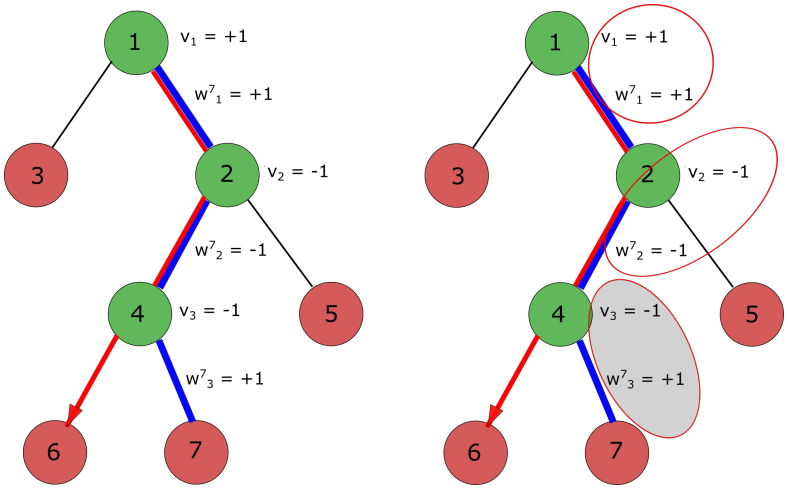

Figure 3.5: Demonstration of inequality $\sum_{i=1}^{L-1}(w_i^{m'}v_i) \leq l(m') - 1 < l(m')$ in case of leaf number 7 (corresponding to one neuron in the second hidden layer of our network), if input does not end in leaf m'. The red path in the picture indicates real path of input in the decision tree. Our sample input ends in leaf number 6, as could be seen from the illustration. The output from the first hidden layer would therefore be $\boldsymbol{v} = (v_1, v_2, v_3)^T = (+1, -1, -1)^T$. But weights corresponding to neuron 7 (leaf 7) are $\boldsymbol{w}^7 = (w_1^7, w_2^7, w_3^7)^T = (+1, -1, +1)^T$. After multiplying the values in red circles and summing the results up, we obtain $\sum_{i=1}^{L-1}(w_i^7 v_i) = 2 < 3$, where 3 means the length of the root-leaf path. A decrease is caused due to the interference of different signs in a grey circle.

After these considerations, the reasonable choice of $bias(m')$ is

$$bias(m') = -l(m') + 0.5 \tag{3.4}$$

and then $\sum_{i=1}^{L-1}(w_i^{m'} v_i) + bias(m')$ has following property:

$$\sum_{i=1}^{L-1}(w_i^{m'} v_i) + bias(m') \begin{cases} > 0, & \text{if input ends in leaf } m' \\ < 0, & \text{otherwise} \end{cases} \tag{3.5}$$

With respect to the property of $\tau(\cdot)$ function argument in (3.5), the second hidden layer outputs a vector of $(-1, ..., -1, +1, -1, ..., -1)^T$ with a single positive 1 indicating the correct leaf membership of an input.

To retain this $\tau(\cdot)$ argument property, it is sufficient to choose any other arbitrary constant in (3.4) in range $(0, 1)$ instead of 0.5. But to stay consistent with [1], we also used the proposed value of 0.5 in conducted experiments.

At this stage, we already defined architecture and initial weights and biases settings of first two hidden layers in order to transform decision tree into neural network with the same properties. All that remains is to gain classification predictions from the second hidden layer.

3.1.3 Output layer

In this section is proposed architecture and initial setting for output layer, that will gain same predictions as the decision tree. Unfortunately, same method proposed for regression trees in [1] is not directly applicable in the classification case. Therefore we propose an alternative for the classification case, that give same classification outcomes as the corresponding decision tree.

The output layer will be constructed as follows: The number of neurons in output layer is equal to number of classes we desire to classify. Each neuron corresponds to only one particular class (one label). Neuron with the highest activation represents the predicted class of neural network. In the experiments were used two types of neurons in output layer - with sigmoid and softmax activation functions, as were discussed in Chapter [Activation functions]. To get the same performance as the decision tree, we must retrieve probability distributions stored in leaves from the leaf membership encoded in the second hidden layer. If the output layer outcomes the same probability values for classes as the decision tree, hence the neural network performs alike.

Let's denote output from the second hidden layer as
$\boldsymbol{r} = (-1, ..., -1, +1, -1, ..., -1)^T$, where the position of $+1$ indicates the leaf where the input falls in. For each leaf $l \in \{1, ..., L\}$ we denote a probability vector $\boldsymbol{p^l} = (p_1^l, p_2^l, ..., p_C^l)^T$ with probabilities of individual classes, that are stored in leaf l, where C is total number of classes. If $\boldsymbol{r}$ has $+1$ as the first element (corresponding to the first leaf), i.e. $r_1 = +1$, the output layer should outcome $\boldsymbol{p}^1$. If $r_2 = +1$, the outcome should be $\boldsymbol{p}^2$ etc.

If we initialize biases in the output layer to 0, then appropriate initialization weights could be obtained by solving the system of linear equations with a

regular matrix $\mathbb{A}$

$$\mathbb{A} = \begin{pmatrix} 1 & -1 & -1 & \dots & -1 \\ -1 & 1 & -1 & \dots & -1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -1 & \ddots & \ddots & \ddots & -1 \\ -1 & -1 & \dots & -1 & 1 \end{pmatrix} \quad . \tag{3.6}$$

Matrix $\mathbb{A} \in \mathbb{R}^{L \times L}$, where L is a total number of leaves in the decision tree. The determinant of matrix $\mathbb{A}$ is

$$\det\mathbb{A} = \begin{vmatrix} 1 & -1 & -1 & \dots & -1 \\ -1 & 1 & -1 & \dots & -1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -1 & \ddots & \ddots & \ddots & -1 \\ -1 & -1 & \dots & -1 & 1 \end{vmatrix} = (-1)^{2L-3} \cdot 2^{L-1} \cdot (L-2) \tag{3.7}$$

For the proof of (3.7) see Chapter [Proves]. Matrix $\mathbb{A}$ is therefore always regular with except of $L = 2$. In this case, $\det\mathbb{A} = 0$ and matrix $\mathbb{A}$ is singular. But for $L = 2$, the decision tree has only root node and 2 leaves, which is rarely a well-functional model in practical use. It could have a good performance almost only in case of data with significantly unambigous geometric deployment, where occurs only 2 classes and exists one hyperplane that sufficiently separates them. As the conclusion, we will almost never encounter such an elementary model.

In case of invertible activation function $\sigma(\cdot)$ in the output layer (in our experiments were used sigmoid and softmax activation functions, which are both invertible), we obtain appropriate weights for neuron c (also represents class $c \in \{1, ..., C\}$ that this neuron corresponds to) in output layer after solving the following system of linear equations:

$$\mathbb{A} \begin{pmatrix} w_1^c \\ w_2^c \\ \vdots \\ w_L^c \end{pmatrix} = \sigma^{-1}(\begin{pmatrix} p_c^1 \\ p_c^2 \\ \vdots \\ p_c^L \end{pmatrix}) \quad , \tag{3.8}$$

where $\boldsymbol{w}^c = (w_1^c, ..., w_L^c)^T$ are weights of connections from the second hidden layer to neuron c in the output layer and $\sigma^{-1}(\cdot)$ is inverse function of $\sigma(\cdot)$. In other words, the appropriate weights for neuron c in the output layer are obtained as

$$\begin{pmatrix} w_1^c \\ w_2^c \\ \vdots \\ w_L^c \end{pmatrix} = \mathbb{A}^{-1}(\sigma^{-1}(\begin{pmatrix} p_c^1 \\ p_c^2 \\ \vdots \\ p_c^L \end{pmatrix})) \quad . \tag{3.9}$$

After initialization of biases to 0 and solving the system of linear equations for all neurons in the output layer (or computing the inverse of matrix $\mathbb{A}$) and for appropriate activation function chosen in advance, we gain also all initialization weights. This initial setting causes the neural network to output same predictions as from the corresponding decision tree and hence to get equally performing classification model.

3.2 NN models with decision-tree-initialization

Describe all five competitive models that we use in experiments with illustrations and algorithms. Discuss replacement of τ with tanh activation function and its influence on performance and backpropagation. Divide basic model to sparse setting and full connected setting. In sparse setting we allow to train with backpropagation only notnull connections, to preserve decision tree interpretation. That will help us also to demonstrate the effect of backpropagation. In full connected setting we train all parameters.

Chapter 4

Experiments

Bibliography

[1] Neural Random Forest. Gerard Biau, Erwan Scornet, Johannes Welbl