NET-DNF: EFFECTIVE DEEP MODELING OF TABULAR DATA – APPENDICES

Anonymous authors
Paper under double-blind review

1 OR AND AND GATES

The (soft) neural OR and AND gates were defined as

$$\mathrm{OR}(\mathbf{x}) \triangleq \tanh\left(\sum_{i=1}^{d} \mathbf{x}_i + d - 1.5\right), \qquad \mathrm{AND}(\mathbf{x}) \triangleq \tanh\left(\sum_{i=1}^{d} \mathbf{x}_i - d + 1.5\right).$$

By replacing the $\tanh$ activation with a sign activation, and setting the bias term to 1 (instead of 1.5), we obtain exact binary gates,

$$\mathrm{OR}(\mathbf{x}) \triangleq \mathrm{sign}\left(\sum_{i=1}^{d} \mathbf{x}_i + d - 1\right), \qquad \mathrm{AND}(\mathbf{x}) \triangleq \mathrm{sign}\left(\sum_{i=1}^{d} \mathbf{x}_i - d + 1\right).$$

Consider a binary vector $\mathbf{x} \in \{\pm 1\}^d$. We prove that

$$\mathrm{AND}(\mathbf{x}) \equiv \bigwedge_{i=1}^{d} \mathbf{x}_i,$$

where, in the definition of the logical "and", -1 is equivalent to 0. If for any $1 \leq i \leq d$, $\mathbf{x}_i = 1$, then $\wedge_{i=1}^{d} \mathbf{x}_i = 1$. Conversely, we have,

$$\mathrm{AND}(\mathbf{x}) = \sum_{i=1}^{d} \mathbf{x}_i - d + 1 = d - d + 1 = 1,$$

and the application of the sign activation yields 1. In the case of the soft neural AND gate, we get $tanh(1) \approx 0.76$; therefore, we set the bias term to 1.5 to get an output closer to 1 ($tanh(1.5) \approx 0.9$).

Otherwise, there exists at least one index $1 \leq j \leq d$, such that $\mathbf{x}_j = -1$, and $\wedge_{i=1}^{d} \mathbf{x}_i = -1$. In this case,

$$\mathrm{AND}(\mathbf{x}) = \sum_{i=1}^{d} \mathbf{x}_i - d + 1 = \mathbf{x}_j + \sum_{i \neq j} \mathbf{x}_i - d + 1 \leq -1 + (d-1) - d + 1 = -1,$$

and by applying the sign activation we obtain -1. This proves that the $\mathrm{AND}(\mathbf{x})$ neuron is equivalent to a logical "AND" gate in the binary case. A very similar proof shows that

$$\mathrm{OR}(\mathbf{x}) \equiv \bigvee_{i=1}^{d} \mathbf{x}_i.$$

2 PROOF OF THEOREM 2

We bound the VC-dimension of a DNF formula in two steps. First, we derive an upper bound on the VC-dimension of a single conjunction, and then extend it to a disjunction of k conjunctions. We use the following simple lemma.

Lemma 1. *For every two hypothesis classes, $H' \subseteq H$, it holds that $VCDim(H') \leq VCDim(H)$.*

Proof. Let $d = VCDim(H')$. By definition, there exist d points that can be shattered by H'. Therefore, there exist 2^d hypotheses $\{h'_i\}_{i=1}^{2^d}$ in H', which shatter these points. By assumption, $\{h'_i\}_{i=1}^{2^d} \subseteq H$, so $VCDim(H) \geq d$. $\square$

For any conjunction on n Boolean variables (regardless of the number of literals), it is possible to construct an equivalent decision tree of rank 1. The construction is straightforward. If $\bigwedge_{i=1}^{\ell} x_i$ is the conjunction, the decision tree consists of a single main branch of ℓ internal decision nodes connected sequentially. Each left child in this tree corresponds to decision "1", and each right child corresponds to decision "0". The root is indexed 1 and contains the literal x_1. For $1 \leq i < \ell$, internal node i contains the decision literal x_i and its left child is node $i+1$ (whose decision literal is x_{i+1}). See the example in Figure 1.

It follows that the hypothesis class of conjunctions is contained in the class of rank-1 decision trees. Therefore, by Lemma 1 and Theorem 1, the VC-dimension of conjunctions is bounded above by $n+1$.

We now derive the upper bound on the VC-dimension of a disjunction of k conjunctions. Let C be the class of conjunctions, and let $D_k(C)$ be the class of a disjunction of k conjunctions. Clearly, $D_k(C)$ is a k-fold union of the class C, namely,

$$D_k(C) = \left\{ \bigcup_{i=0}^{k} c_i \; | c_i \in C \right\}.$$

By Lemma 3.2.3 in (Blummer et al. 1989), if $d = VCDim(C)$, then for all $k \geq 1$, $VCDim(D_k(C)) \leq 2dk\log(3k)$. Therefore, for the class DNF_n^k, of DNF formulas with k conjunctions on n Boolean variables, we have

$$VCDim(DNF_n^k) \leq 2(n+1)k\log(3k).$$

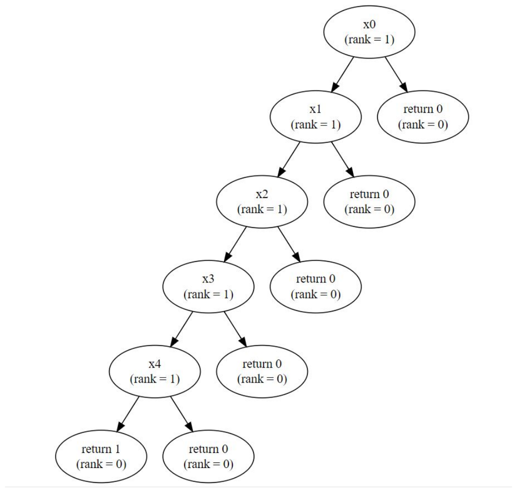

Figure 1: An example of a decision tree with rank 1, which is equivalent to the conjunction $x_0 \wedge x_1 \wedge x_2 \wedge x_3 \wedge x_4$.

3 Tabular Dataset Description

We use datasets (See Table 4) that differ in several aspects such as in the number of features (from 16 up to 200), the number of classes (from 2 up to 9), and the number of samples (from 10k up to 200k). To keep things simple, we selected datasets with no missing values, and that do not require preprocessing. All models were trained on the raw data without any feature or data engineering and without any kind of data balancing or weighting. Only feature wise standardization was applied.

Dataset	features	classes	samples	source	link
Otto Group	93	9	61.9k	Kaggle	kaggle.com/c/otto-group-product-classification-challenge/overview
Gesture Phase	32	5	9.8k	OpenML	openml.org/d/4538
Gas Concentrations	129	6	13.9k	OpenML	openml.org/d/1477
Eye Movements	26	3	10.9k	OpenML	openml.org/d/1044
Santander Transaction	200	2	200k	Kaggle	kaggle.com/c/santander-customer-transaction-prediction/overview
House	16	2	22.7k	OpenML	openml.org/d/821

Table 1: A description of the tabular datasets

4 EXPERIMENTAL PROTOCOL

4.1 DATA PARTITION AND GRID SEARCH PROCEDURE

All experiments in our work, using both synthetic and real datasets, were done through a grid search process. Each dataset was first randomly divided into five folds in a way that preserved the original distribution. Then, based on these five folds, we created five partitions of the dataset as follows. Each fold is used as the test set in one of the partitions, while the other folds are used as the training and validation sets. This way, each partition was 20% test, 10% validation, and 70% training. This division was done once [1], and the same partitions were used for all models. Based on these partitions, the following grid search process was repeated three times with three different seeds[2] (with the exact same five partitions as described before).

Algorithm 1: Grid Search Procedure

```
Input: model, configurations_list
results_list = [ ]
for i=1 to n_partitions do
    val_scores_list = [ ]
    test_scores_list = [ ]
    train, val, test = read_data(partition_index=i)
    for c in configurations_list do
        trained_model = model.train(train_data=train, val_data=val, configuration=c)
        trained_model.load_weights_from_best_epoch()
        val_score = trained_model.predict(data=val)
        test_score = trained_model.predict(data=test)
        val_scores_list.append(val_score)
        test_scores_list.append(test_score)
    end
    best_val_index = get_index_of_best_val_score(val_scores_list)
    test_res = test_scores_list[best_val_index]
    results_list.append(test_res)
end
mean = mean(results_list)
sem = standard_error_of_the_mean(results_list)
Return: mean, sem
```

The final mean and sem[3] that we presents in all experiments are the average across the three seeds. Additionally, as can be seen from Algorithm 1, the model that was trained on the training set (70%) is the one that is used to evaluate performance on the test set (20%). This was done to keep things simple. The loading wights command is relevant for the neural network models. While for the XGBoost, the framework handles the optimal number of estimators on prediction time (accordingly to early stopping on training time).

[1]We used seed number 1.
[2]We used seed numbers 1, 2, 3.
[3]For details, see: docs.scipy.org/doc/scipy/reference/generated/scipy.stats.sem.html

4.2 Training Protocol

The Net-DNF and the FCN were implemented using Tesnorflow. To make a fair comparison, for both models, we used the same batch size[4] of 2048, and the same learning rate scheduler (reduce on plateau) that monitors the training loss. We set a maximum of 1000 epochs and used the same early stopping protocol (30 epochs) that monitors the validation score. Moreover, for both of them, we used the same loss function (softmax-cross-entropy for multi-class datasets and sigmoid-cross-entropy for binary datasets) and the same optimizer (Adam with default parameters).

For Net-DNF we used an initial learning rate of 0.05. For FCN, we added the initial learning rate to the grid search with values of $\{0.05, 0.005, 0.0005\}$.

For XGBoost we set the maximal number of estimators to be 2500, and used an early stopping of 50 estimators that monitors the validation score.

All models were trained on GPUs - Titan Xp 12GB RAM.

Additionally, in the case of Net-DNF, we took a symmetry-breaking approach between the different DNNFs. This is reflected by the DNNF group being divided equally into four subgroups where, for each subgroup, the number of conjunctions is equal to one of the following values $[6, 9, 12, 15]$, and the group of conjunctions of each DNNF was divided equally into three subgroups where, for each subgroup, the conjunction length is equal to one of the following values $[2, 4, 6]$. The same approach was used for the parameter p of the random mask. The DNNF group was divided equally into five subgroups where, for each subgroup, p is equal to one of the following values $[0.1, 0.3, 0.5, 0.7, 0.9]$. In all experiments we used the same values.

4.3 Grid Parameters – Tabular Datasets

4.3.1 Net-DNF

Net-DNF (42 configs)	
hyperparameter	**values**
number of formulas	$\{64, 128, 256, 512, 1024, 2048, 3072\}$
feature selection beta	$\{1.6, 1.3, 1., 0.7, 0.4, 0.1\}$

4.3.2 XGBoost

XGBoost (864 configs)	
hyperparameter	**values**
number of estimators	$\{2500\}$
learning rate	$\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$
max depth	$\{2, 3, 4, 5, 7, 9, 11, 13, 15\}$
colsample by tree	$\{0.25, 0.5, 0.75, 1.\}$
sub sample	$\{0.25, 0.5, 0.75, 1.\}$

To summarize, we performed a crude but broad selection (among 42 hyper-parameter configurations) for our Net-DNF. Results were quite strong, so we avoided further fine tuning. To ensure extra fairness w.r.t. the baselines, we provided them with significantly more hyper-parameter tuning resources (864 configurations for XGBoost, and 3300 configurations for FCNs).

4.3.3 Fully Connected Networks

The FCN networks are constructed using Dense-RELU-Dropout blocks with L_2 regularization. The network's blocks are defined in the following way. Given depth and width parameters, we examine two different configurations: (1) the same width is used for the entire network (e.g., if the width is 512 and the depth is four, then the network blocks are [512, 512, 512, 512]), and (2) the width

[4]For Net-DNF , when using 3072 formulas, we set the batch size to 1024 on the Santander Transaction and Gas datasets and when using 2048 formulas, we set the batch size to 1024 on the Santander Transaction dataset. This was done due to memory issues.

parameter defines the width of the first block, and the subsequent blocks are reduced by a factor of 2 (e.g., if the width is 512 and the depth is four, then the network blocks are [512, 256, 128, 64]). On top of the last block we add a simple linear layer that reduce the dimension into the output dimension. The dropout and L_2 values are the same for all blocks.

FCN (3300 configs)	
hyperparameter	**values**
depth	$\{1, 2, 3, 4, 5, 6\}$
width	$\{128, 256, 512, 1024, 2048\}$
L_2 lambda	$\{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 0.\}$
dropout	$\{0., 0.25, 0.5, 0.75\}$
initial learning rate	$\{0.05, 0.005, 0.0005\}$

4.4 Ablation Study

All ablation studies experiments were conducted using the grid search process as described in 4.1. In all experiments, we used the same training details as described on 4.2 for Net-DNF. Where the only difference between the different experiments is the addition or removal of the components.

The single hyperparameter that was fine-tuned using the grid search is the 'feature selection beta' on the range $\{1.6, 1.3, 1., 0.7, 0.4, 0.1\}$, in experiments in which the feature selection component is involved. In the other cases, only one configuration was tested in the grid search process for a specific number of formulas.

4.5 Feature Selection Analysis

The input features $\mathbf{x} \in \mathbb{R}^d$ of all six datasets were generated from a d-dimensional Gaussian distribution with no correlation across the features, $\mathbf{x} \sim \mathbb{N}(0, I)$. The label $\mathbf{y}$ is sampled as a Bernoulli random variable with $\mathbb{P}(\mathbf{y} = 1|\mathbf{x}) = \frac{1}{1+logit(\mathbf{x})}$, where $logit(\mathbf{x})$ is varied to create the different synthetic datasets ($\mathbf{x}_i$ refers to the ith entry):

1. **Syn1**: $logit(\mathbf{x}) = exp(\mathbf{x}_1\mathbf{x}_2)$
2. **Syn2**: $logit(\mathbf{x}) = exp(\sum_{i=3}^{6} \mathbf{x}_i^2 - 4)$
3. **Syn3**: $logit(\mathbf{x}) = -10\sin(2\mathbf{x}_7) + 2|\mathbf{x}_8| + \mathbf{x}_9 + exp(-\mathbf{x}_{10}) - 2.4$
4. **Syn4**: if $\mathbf{x}_{11} < 0$, logit follows **Syn1**, else, logit follows **Syn2**
5. **Syn5**: if $\mathbf{x}_{11} < 0$, logit follows **Syn1**, else, logit follows **Syn3**
6. **Syn6**: if $\mathbf{x}_{11} < 0$, logit follows **Syn2**, else, logit follows **Syn3**

We compare the performance of a basic FCN on three different cases: (1) **oracle (ideal) feature selection** – where the input feature vector is multiplied element-wise with an input oracle mask, whose ith entry equals 1 iff the ith feature is relevant (e.g., on Syn1, features 1 and 2 are relevant, and on Syn4, features 1-6, and 11 are relevant), (2) **our (learned) feature selection mask** – where the input feature vector is multiplied element-wise with the mask $\mathbf{m}_t$, i.e., the entries of the mask $\mathbf{m}_s$ (see Section 2.3) are all fixed to 1, and (3) **no feature selection**.

From each dataset, we generated seven different instances that differ in their input size, $d \in [11, 50, 100, 150, 200, 250, 300]$. Where when the input dimension d increases, the same logit function is used. Each instance contains 10k samples that were partitioned as described in Section 4.1. We treated each instance as an independent dataset, and the grid search process that is described in Section 4.1 was done for each one.

The FCN that we used has two dense hidden layers [64, 32] with a RELU activation. To keep things simple, we have not used drouput or any kind of regularization. The same training protocol was used for all three models. We used the same learning rate scheduler, early stopping protocol, loss function and optimizer as appear in Section 4.2[5]. We use a batch size of 256, and an initial learning

[5]We noticed that in this scenario, a large learning rate or large batch size leads to a decline in the performance of the 'FCN with the feature selection'. While the simple FCN and the 'FCN with oracle mask' remains approximately the same.

rate of 0.001. The only hyperparameter that was fine-tuned is the ‘feature selection beta’ in the case of ‘FCN with feature selection’ on the range $\{1.3, 1., 0.7, 0.4\}$. For the two other models, only a single configuration was tested in the grid search process.