Casting Random Forests as Artificial Neural Networks (and Profiting from It)

Johannes Welbl

Heidelberg Collaboratory for Image Processing

While Artificial Neural Networks (ANNs) are highly expressive models, they are hard to train from limited data. Formalizing a connection between Random Forests (RFs) and ANNs allows exploiting the former to initialize the latter. Further parameter optimization within the ANN framework yields models that are intermediate between RF and ANN, and achieve performance better than RF and ANN on the majority of the UCI datasets used for benchmarking.*

1 Introduction

In supervised machine learning, both RF [1] classifiers and feedforward ANNs [2] are in widespread use for tackling classification problems. In this work, it will be demonstrated that the predictive behaviour of RFs can equally be transferred into the framework of two-layer ANNs, where it can serve as an initialization for training. This transferrability allows for a new functional interpretation of the RF model as special limit case of the general sigmoid ANN model.

ANNs are universal approximators [3], and with their many free parameters they are expressively very rich. However, their expressive power comes with the downside of increased overfitting risk, especially on small datasets. Conversely, RFs generalize well, but with their greedy tree construction process they yield fine, but often suboptimal classification performance.

Harnessing the novel RF-reformulation, a major practical intention is to exploit the benefits of RF and ANN models to overcome the shortcomings of the other. The novel network initialization allows for an optimization of RF predictions within the ANN framework and can possibly help ANNs to reduce their overfitting risk. The approach is developed in theoretical detail, later different training schemes are experimentally evaluated on various classification datasets.

Random forests In this section some notation will briefly be introduced. RFs are ensembles of decision trees [4] T_i, $i = 1, \dots, N_{Trees}$, (with N_{Trees} being the number of trees) which consist of both *inner nodes* $\mathcal{N}_i^{Inner}$ and *leaf nodes* $\mathcal{N}_i^{Leaf}$. For RFs using paraxial node splits, the split rule for $n \in \mathcal{N}_i^{Inner}$ and samples $x \in n$ is

$$x \in cl(n) \iff x_{f_n} < \theta_n \quad \text{and} \quad x \in cr(n) \iff x_{f_n} \geq \theta_n \tag{1}$$

* recommended for submission to the YRF2014 by Prof. Fred Hamprecht and Dr. Ullrich Köthe. The final publication is available at: link.springer.com/chapter/10.1007%2F978-3-319-11752-2_66

with f_n being the split feature of n, θ_n the threshold value, and $cl(n)$ and $cr(n)$ being the left and right child node of n.
For a tree leaf $L \in \mathcal{N}_i^{Leaf}$ the *ratio vote* is $y^L = (y_1^L, \dots, y_{N_{Labels}}^L)$ with $y_l^L \in [0,1]$ $\forall l = 1, \dots, N_{Labels}$, and $\sum_{l=1}^{N_{Labels}} y_l^L = 1$, where N_{Labels} is the number of labels.

Finally, the *path* $\mathcal{P}(L)$ to a leaf L will be the sequence of inner split nodes that has to be passed on the way from the tree root node n_0 to L:

$$\mathcal{P}(L) = (n_0, \dots, n_d) \quad \text{with} \quad n_0, \dots, n_d \in \mathcal{N}_i^{Inner} \text{ and } \quad L \subset n_d \subset \dots \subset n_0 \quad (2)$$

Thus, leaf membership is expressed in terms of conditions satisfied along $\mathcal{P}(L)$:

$$x \in L \iff \forall n_j \in \mathcal{P}(L) : \begin{cases} x_{f_{n_j}} < \theta_{n_j} & \text{if } L \subseteq cl(n_j) \\ x_{f_{n_j}} \geq \theta_{n_j} & \text{if } L \subseteq cr(n_j) \end{cases} \quad (3)$$

Artificial neural networks The ANNs chosen here are feedforward networks with two hidden layers (HLs). They use a tanh activation function in the HLs, and softmax activation for the output neurons, which allows for a probabilistic interpretation of the network output. During training, wrong predictions are backpropagated [5], and stochastic gradient descent (SGD) [6] is applied to maximize the likelihood of predicting true labels on randomized minibatches of training data.

2 Reformulating the RF as ANN

In this chapter, it will be described how a given pretrained RF can be reformulated as a two-layer ANN with the same predictive behaviour.

Both the RF and the ANN model rely on linear separation. In the RF any $n \in \mathcal{N}_i^{Inner}$ is *linearly* divided into $cl(n)$ and $cr(n)$. In ANNs with sigmoid activity the neurons *linearly* split their input space into two halfspaces of either positive or negative activity. This common characteristic of the two classifiers enables the transfer from RF to ANN, which will first be described for a single decision tree T_i and later be extended to a whole RF. There will be three functionally different parts in the resulting ANN architecture (also visualized in Figure 1):

1. In the first hidden layer (HL_1) the neurons compute all tree split decisions $\forall n \in \mathcal{N}_i^{Inner}$ and indicate the split directions for a given input sample.
2. In the second hidden layer (HL_2) the information from HL_1 is combined to indicate if $x \in L$, $\forall L \in \mathcal{N}_i^{Leaf}$.
3. The weights from HL_2 to the output layer are set proportional to the ratio votes, effectively mimicing the RF voting system.

2.1 Designing the first hidden layer

The first HL is designed to indicate tree split directions of given network inputs x, i.e. $\forall n \in \mathcal{N}_i^{Inner}$ one dedicated HL_1-neuron will encode if $x_{f_n} \geq \theta_n$ or not.

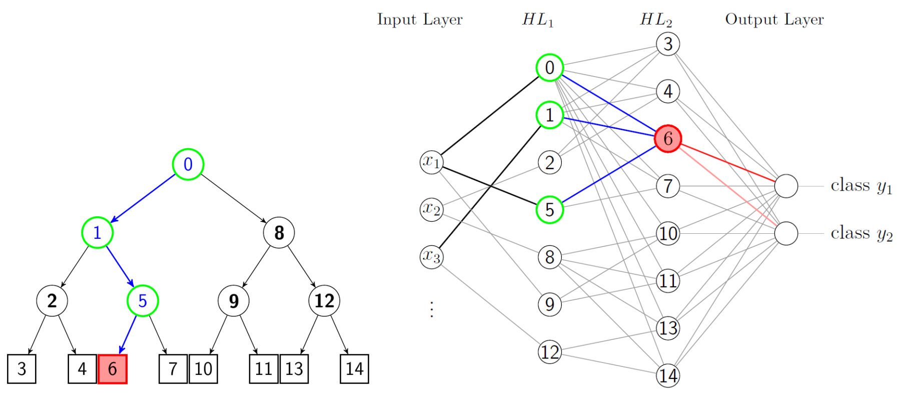

Fig. 1. A single decision tree (left) and its corresponding network structure (right). The circle nodes in the tree belong to $\mathcal{N}_i^{Inner}$ and the square nodes to $\mathcal{N}_i^{Leaf}$. The path to the red shaded leaf (6) consists of all light green nodes (0, 1, 5). Numbers in neurons correspond to numbers in tree nodes. The highlighted connections in the network are those relevant for the activity of the red neuron and its output vote.

Therefore, $\forall n \in \mathcal{N}_i^{Inner}$ one HL_1-neuron is created, named $HL_1(n)$, and then connected to input f_n. The corresponding weight and bias of $HL_1(n)$ are set to

$$w_{f_n,HL_1(n)} = str_{01} \quad \text{and} \quad b_{HL_1(n)} = -str_{01} \cdot \theta_n \tag{4}$$

with $str_{01} \gg 0$ a global linear scaling hyperparameter. No other connections between input layer and HL_1 are established, so the connectivity is sparse. With this choice of weights and biases, the following statements are equivalent

$$\begin{aligned} x \in cr(n) &\overset{(1)}{\iff} x_{f_n} \geq \theta_n \overset{(4)}{\iff} w_{f_n,HL_1(n)} \cdot x_{f_n} \geq -b_{HL_1(n)} \\ &\iff tanh(w_{f_n,HL_1(n)} \cdot x_{f_n} + b_{HL_1(n)}) \geq 0 \\ &\iff activity(HL_1(n)) \geq 0 \end{aligned}$$

and likewise for $cl(n)$. Hence, for a given input x positive activities of HL_1-neurons encode tree splits to the right, and negative activities tree splits to the left. This allows to rewrite leaf membership from (3) in terms of HL_1-activations:

$$x \in L \iff \forall n_j \in \mathcal{P}(L) : \begin{cases} activity(HL_1(n_j)) < 0 & \text{if } L \subseteq cl(n_j) \\ activity(HL_1(n_j)) \geq 0 & \text{if } L \subseteq cr(n_j) \end{cases} \tag{5}$$

The *strength*-hyperparameter str_{01} determines the contrast of the tanh activations: the larger str_{01} the sharper the transition from -1 to $+1$, and as str_{01} approaches infinity, the continuous tanh activation function converges to a binary step function. So when choosing str_{01} large, all HL_1-activations are close to either -1 or $+1$. This allows to operate with a differentiable approximation of a discontinous step activation function with nonzero gradient (important for network training), and later permits a relaxation of crisp tree node membership. This setup is equally applicable for RFs with oblique splits [7], but then $HL_1(n)$ must be connected to several features with weights according to the split tilt.

2.2 Designing the second hidden layer

HL_2 is designed to the end that $\forall L \in \mathcal{N}_i^{Leaf}$ there is one neuron that indicates whether $x \in L$. Therefore, $\forall L \in \mathcal{N}_i^{Leaf}$ one HL_2-neuron is created and named $HL_2(L)$. It is connected to all HL_1-neurons $HL_1(n_j)$ with $n_j \in \mathcal{P}(L)$ and not connected to the other neurons in HL_1. $HL_2(L)$ combines the split information from the activations of all $HL_1(n_j)$, and for determining whether all conditions from (5) are satisfied, $HL_2(L)$ computes a softAND function of the $HL_1(n_j)$ activities. For this the incoming weights of $HL_2(L)$ are set according to the direction that $\mathcal{P}(L)$ takes at each n_j:

$$w_{HL_1(n_j),HL_2(L)} = \begin{cases} -str_{12} & \text{if } L \subseteq cl(n_j) \\ +str_{12} & \text{if } L \subseteq cr(n_j) \end{cases}, \qquad str_{12} > 0 \tag{6}$$

Assuming str_{01} large $\Rightarrow$ $HL_1(n_j)$-activations approximate either -1 or $+1$, so the incoming weights (6) act as a matched filter for the specific $HL_1(n_j)$-activation pattern (5) which appears iff $x \in L$. For fixed str_{12} this maximizes the activation response of $HL_2(L)$ if the HL_1-pattern (5) is present.

Note that for large str_{01} all datapoints x are mapped into the very corners of the hypercube of all HL_1-activations. Also the datapoints in the orthant defined by (5) (case $x \in L$) are located in one hypercube corner and are linearly separable from the other datapoints ($x \notin L$) in other hypercube corners. The bias determines the threshold of the softAND function to separate the cases:

$$b_{HL_2(L)} = -str_{12} \cdot [len(\mathcal{P}(L)) - 1] \tag{7}$$

with $len(\mathcal{P}(L))$ the number of tree nodes n_j on $\mathcal{P}(L)$. In case of infinite str_{01} where $\forall n_j : activity(HL_1(n_j)) \in \{-1, 1\}$ this bias choice maximizes the margin between the different HL_1-activation patterns appearing in the two cases $x \in L$ and $x \notin L$. Thus for large str_{01} the activity of $HL_2(L)$ effectively distinguishes the two cases and indicates if $x \in L$. Again, choosing a large global scaling hyperparameter ($str_{12} \gg 0$) makes the activities in HL_2 more contrastive.

2.3 Transferring the RF voting system

So now, $\forall L \in \mathcal{N}_i^{Leaf}$ there is one HL_2-neuron which indicates whether $x \in L$. Next, all activities of HL_2 are linearly rescaled from $[-1, 1]$ to $[0, 1]$ and $HL_2(L)$ is connected to all outputs l, setting the weights proportional to the ratio vote:

$$w_{HL_2(L),l} = y_l^L \cdot str_{23} \tag{8}$$

with y_l^L the vote of L for label l, and $str_{23} > 0$ another scaling hyperparameter.

This concludes the ANN-reformulation of a single decision tree. For a complete RF, the HLs of several network structures are concatenated, so that the votes of all trees are expressed in the connections to the output neurons, effectively averaging the votes of the whole tree ensemble.

3 Training schemes and relaxations

The resulting network is now ready for training for which three approaches will be investigated. The *sparse* training scheme consists of the application of a standard network optimization algorithm (here: SGD) on the sparsely connected network. The *relaxed* training scheme has two differences to the *sparse*: (i) Allow for full connectivity between subsequent layers. While still initializing the previously non-existent connections close to 0, they will be free to change during network optimization, and hence establish cross-connections between functionally unrelated RF modules. Possibly more complex interactions between different RF elements can be captured this way. (ii) Decrease the global *strength* hyperparameters str_{01} and str_{12} to make the sigmoid activations less contrastive. This can be interpreted as a relaxation of crisp to fuzzy tree node membership. Samples not merely fall into one direction per split and one final leaf but simultaneously into several tree branches and leaves. Finally, the *vote* training scheme restricts network optimization to only weights of the last layer. While maintaining pure tree structure in the layers before, merely the RF votes will collectively be optimized. After training, this scheme enables a retransfer of the network into a classical RF with an altered (and possibly enhanced) voting system.

4 Experimental evaluation

Next, several experiments are performed to investigate the novel initialization and different training schemes on classification datasets from UCI Machine Learning Repository [8] (*Breast Cancer Wisconsin (Diagnostic), Ionosphere, Sonar, Landsat, Pima Indians Diabetes, Heart Disease* and *German Credit*). The continuous feature dimensions of these datasets are in a generic preprocessing step linearly scaled to $[-1, 1]$. For each of the datasets the novel ANN initializations are examined with the three introduced training schemes, and for comparison also RFs and randomly initialized ANNs are tested. Each experimental setup is repeated 30 times on a randomly permuted separation into 3/4 training data and 1/4 test data (if training/test data not prespecified). Samples with missing values are discarded. The trees in the RF are grown applying the entropy criterium until a minimum number of 10 samples per node is reached. The best split feature is chosen among $\sqrt{M}$ random features, with M the input dimensionality. All RFs possess $N_{Trees} = 200$ trees (50 for Landsat). To account for the random number of nodes in RFs and transcoded RFs, randomly initialized 2-layer networks with different sizes of 40 to 3500 neurons per HL are trained, and the best result is chosen. All randomly initialized ANNs are initialized with biases 0 and weights following a narrow Gaussian $\mathcal{N}(0, \sigma^2)$, $\sigma = 0.01$.

Hyperparameters are set to $str_{01} = 1000$, $str_{12} = 1000$ and $str_{23} = 0.1$. For the relaxation str_{12} is decreased to 10, which is chosen as the lowest feasible value while evaluating the congruence of predictions of RFs and transcoded RFs on an exponentially scaled hyperparameter grid. SGD network training is applied using a batchsize of 100 and a momentum of 0.99. Learning rate is manually selected for each dataset so that training convergence is guaranteed. All networks are trained until convergence. To harness the power of parallelized GPU-computation the python package *theano* [9] is used. For all datasets RF construction and network training to convergence happens within seconds to minutes on a custom GPU. Table 1 shows the final test perfomances for all experimental setups.

5 Discussion of results and conclusion

A trivial result is that RF-initialized networks yield better test error at the start of training than randomly initialized networks, effectively giving the network a warmstart at the prediction level of the RF.

Comparing the performances of RFs with randomly initialized ANNs, it becomes apparent that the latter at times overfit (where RF performance is better than ANN), and that overfitting is mitigated by the novel network initialization as transcoded RF.

On the other hand, RF predictions are often improved by SGD-optimization within the ANN framework. The *sparse* scheme outperforms *both* ANN and RF in most cases. The *vote* optimization training (which allows for a retransfer into a classical RF) is not as competitive but still improves the RF predictions for several datasets. *Relaxed* training yields similar results to *sparse* training, hence the benefits of RF-initialization are not limited to sparsely connected networks but are equally available for the more widespreadly used fully connected ANNs.

In conclusion, the newfound theoretical link between RFs and ANNs can fruitfully be exploited to enhance both classifiers in practice. Optimizing RFs within the ANN framework often improves the predictive performance of the two original models. The benefits of both can be united in one novel hybrid classifier.

	RF	ANN	sparse	relaxed	vote
Wisconsin	4.8 (1.7)	3.4 (1.7)	**2.9 (1.3)**	3.5 (1.9)	4.1 (1.8)
Ionosphere	6.5 (2.2)	11.0 (3.3)	**6.2 (2.0)**	6.9 (2.7)	6.7 (1.9)
Sonar	21.5 (5.1)	18.0 (5.5)	**14.4 (4.8)**	16.0 (5.2)	14.8 (4.1)
Landsat	10.9 (0.3)	10.1 (0.4)	9.1 (0.3)	9.1 (0.4)	**9.0 (0.4)**
Pima	**24.5 (2.1)**	28.9 (3.0)	26.8 (3.1)	26.4 (2.3)	26.6 (2.6)
Heart	**16.3 (4.4)**	21.8 (4.3)	19.5 (4.1)	19.5 (4.3)	19.0 (4.1)
Credit	25.6 (2.2)	28.3 (2.7)	**24.6 (1.8)**	24.7 (2.9)	25.3 (2.7)

Table 1. Predictive performance on test data for different classifiers and training schemes across seven datasets. Table entries are the empirical mean test error with standard deviation in parentheses. Bold writing highlights the winning classifier.

References

1. Breiman, L.: Random forests. Mach. Learn. **45**(1) (October 2001) 5–32
2. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems **2** (1989) 303–314
3. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5) (July 1989) 359–366
4. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA (1984)
5. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. MIT Press, Cambridge, MA, USA (1986) 318–362
6. Bottou, L.: Online learning and stochastic approximations (1998)
7. Murthy, S.K., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. Journal of Artificial Intelligence Research **2** (1994) 1–32
8. Bache, K., Lichman, M.: UCI machine learning repository (2013)
9. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy). (June 2010) Oral Presentation.