Neural networks for classification problem on tabular data

A S Nazdryukhin, A M Fedrak and N A Radeev
Novosibirsk National Research University, 2 Pirogova str., Novosibirsk, 630090, Russia
E-mail: a.nazdryukhin@g.nsu.ru, a.fedrak@g.nsu.ru

Abstract. This work presents the results of using self-normalizing neural networks with automatic selection of hyperparameters, TabNet and NODE to solve the problem of tabular data classification. The method of automatic selection of hyperparameters was realised. Testing was carried out with the open source framework OpenML AutoML Benchmark. As part of the work, a comparative analysis was carried out with seven classification methods, experiments were carried out for 39 datasets with 5 methods. NODE shows the best results among the following methods and overperformed standard methods for four datasets.

1. Introduction
The main direction of theoretical research and applied developments in machine learning is almost entirely represented by neural networks. Using convolutional neural networks, impressive results have been achieved in the field of image analysis and processing. For many natural language processing tasks – such as vector representation of text, machine translation, and speech recognition – recurrent neural networks and networks with an attention mechanism have also produced impressive results. Such achievements are not surprising: neural networks are more flexible and expressive than classical models (for example, than models based on decision trees). Deep learning allows you to express a representation of an object in terms of other, simpler representations. Thanks to the mechanism of learning representations, deep neural network architectures are able to independently extract and generate features.

One of the reasons why neural networks do not work well with tabular data is the discrete nature of data from tables. Neural networks are good for working with data that has context. For example, convolutional neural networks recognize an elephant in a picture because in neighboring pixels it is possible to select such objects as 4 large legs, a head, a trunk, two large ears, and a tail. Moreover, the objects are not clearly delineated; they smoothly merge into a single one-an elephant. It is the same with sounds where there is a beginning, peak, and fade and with texts where adjacent words form a single coherent context. All these data features make the feature spaces fairly smooth and continuous. With tables, the situation is different. In general, the values in different columns are not related to each other in any way. They have different types: text, categorical, numeric, date, time, and so on. Similarly, rows in tables have no relationships with each other. Both columns and rows can be rearranged within the table and this will not change the meaning of the data for a person, but any neural network will be disoriented. Due to these reasons, well-known neural networks are not able to efficiently extract features from tables.
On the one hand, this is intuitive—after all in tabular data feature generation has already been performed manually, which greatly restricts the mechanism for learning representations in neural networks. On the other hand, neural networks can still identify the most important features in training and, thanks to complex and differentiable activation functions, build a more complex feature description and produce a more flexible approximation of data than classical algorithms can allow.

Thus, the use of neural networks for working with tabular data is still justified. In this paper, it is proposed to use the NODE (Neural Obvious Decision Ensembles) architecture for classifying tabular data [1], based on an ensemble of trees optimized using gradient descent and built on the attention mechanism of the TabNet network [2]. It is important to note that the results of the original TabNet article were not reproducible, but we still tried to check the stability of the algorithm. We also used a fully connected neural network built on the principle of self-normalization for testing [3]. The advantage of Self-normalizing neural networks (SNN) is that they are resistant to noise in the data and can be extremely deep and, thus, can provide a high-level representation of features.

There are other approaches of classification tabular data using neural networks: in [4], the TAC (Tabular Convolution) mechanism is used, the essence of which is that the table rows are used as convolution kernels for a certain fixed image.

2. Materials and methods

2.1. Self-normalizing neural networks

Most modern neural networks use normalization techniques such as batch normalization, layer normalization, and weight normalization. However, the use of strict regularization techniques and stochastic gradient descent can introduce perturbations in the learning process. Convolutional and recurrent neural networks can compensate for such disturbances by redistributing weights. However, fully connected neural networks are not capable of this and therefore suffer from perturbations in the learning process, which leads to a high variance in the values of the cost function. The self-normalization technique allows you to build architectures that are resistant to perturbations that occur during the learning process.

The main advantage of self-normalizing neural networks is the lack of an explicit normalization mechanism. They "push" the mean and variance to the set values.

To describe self-normalization, we introduce a map $g$ such that:

$$g\left(\frac{\mu}{\nu}\right) = \left(\frac{\bar{\mu}}{\bar{\nu}}\right),$$

where $\mu, \nu, \bar{\mu}, \bar{\nu}$ are the mean and variance of the current and next layers of the neural network. That is, $g$ maps the mean and variance values from one layer to another.

By the Banach theorem, $g$ exists and has a fixed point to which the mean and variance converge in the learning process.

The $g$ mapping is set using the selu activation function, which must be set for all neurons:

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$$

where $\lambda, \alpha$ are parameters that depend on the choice of a fixed point. The calculation of these parameters is shown in [3].

It is convenient to choose a fixed point for many architectures with the values $\mu = 0, \nu = 1$. Then $\lambda = 1.0507009873554804934193349852946, \alpha = 1.6732632423543772848170429916717$ [3] and the graph $\text{selu}(x)$ takes the following form (figure 1).
You can also use an initialization scheme for faster convergence of data to the zero mean and unit variance LeCun initialization scheme. The weights of the neural network are taken from a normal distribution with parameters:

\[ E(w_i) = 0, \]
\[ Var(w_i) = \frac{1}{n}. \]

Also the alpha dropout technique is also introduced in [4]. The standard technique of strict regularization of neural networks-dropout (exception) does not allow using the principle of self-normalization due to an increase in variance and further divergence of the learning process. The basic idea is that the alpha dropout should keep the mean and variance stationary.

When a dropout occurs, the neuron is not activated, i.e.:

\[ \lim_{x \to -\infty} selu(x) = -\lambda\alpha := \alpha'. \]

Then the alpha dropout should set the neuron outputs to \( \alpha' \) in accordance with the dropout probability \( 1 - q \) and the dropout value \( d \sim B(1, q) \). When performing the affine transformation \( x \mapsto a(xd + \alpha'(1 - d)) + b \), we obtain that:

\[ E(a(xd + \alpha'(1 - d)) + b) = \mu, \]
\[ Var(a(xd + \alpha'(1 - d)) + b) = \nu. \]

Thus, when performing an alpha dropout, the mean and variance values are preserved. The above parameters \( a, b \) are calculated for a specific fixed point. With zero mean and unit variance, they are equal to:

\[ a = \frac{1}{\sqrt{q + \alpha'^2q(1 - q)}}, \]
\[ b = \frac{\alpha'^2(1 - q)}{\sqrt{q + \alpha'^2q(1 - q)}}. \]
So, using the self-normalization mechanism allows you to train deep architectures with a fairly small spread of the cost function during the training process.

Self-normalization of neural networks does not allow you to determine the appropriate architecture for a specific task, because it is just an approach.

To search for the optimal hyperparameters such as deep neural networks, the number of neurons in a particular layer, etc., was used hyperopt [6] - library with open source code that implements the following selection hyperparameters algorithms: random search [7], Tree of Parzen Estimators (TPE) [8], Adaptive TPE [9].

2.2. TabNet
TabNet developers turned to the world of classic machine learning algorithms, to the most famous and one of the simplest of them – the decision tree. This algorithm was recreated in the language of neural networks. A special neural network module was developed, which in the special case simulates the operation of a decision tree, and in the general case, this module is able to create more complex combinations of features. Hence, there are also more complex feature spaces than the decision tree. However, the decision tree logic underlying the module allowed the architecture to work well with tabular data.

To model a decision tree using a neural network, a series of "steps" is created. The step consists of an attention mechanism, a mask, a feature transformer, and an activation function. The attention mechanism uses the Sparsemax function [11] - modification of the classic known Softmax function.

\[
\text{sparsemax}(z) := \underset{p \in \Delta^{k-1}}{\text{argmin}} \| p - z \|^2.
\]

This function has a simpler form on the graph. If Softmax is a smooth continuous monotonically increasing curve, the Sparsemax is a piecewise continuous function, which is on the interval \((-\infty; -1]\) is a constant 0, and \([1; +\infty)\) is 1. On the interval \([-1; 1]\), the function is a straight line connecting the point \((-1; 0)\) and \((1; 1)\). In other words, the curve is defined as follows:

\[
\text{sparsemax}(x) = \begin{cases} 
1 & \text{if } x > 1 > \\
\frac{(x + 1)}{2} & \text{if } -1 \leq x \leq 1 \\
0 & \text{if } x 
\end{cases}
\]

and has a graph:

![Sparsemax Curve](image-url)
The point of using this function is that very small values are simply zeroed. At the same time, the function has an important differentiability property for training a neural network. The derivative function is also given piecewise. You need to use Sparsemax with understanding, because otherwise there is a high probability of encountering the problem of fading gradients. The authors of the Sparsemax article even give an example of a special cost function that solves the problem of damped gradients (if you use it instead of cross-entropy loss).

During the training, attention masks are trained. After receiving the attention masks, the results of the algorithm are interpreted by the activation values of the masks. To begin with, examples are passed through the network, changing the values in only one attribute. This way, you can find out which element of the mask is responsible for the observed feature. Once you have this information, you can interpret the mask activations in the test cases. For example, if the cells corresponding to salary, amount of expenses, and interest on the deposit were activated, then we can conclude that the characteristics associated with the financial condition of the person were important for the classification of the bank’s client.

In general, TabNet is not only a very original development – a neural network that works well with tabular data, but also has such an important property as interpretability.

2.3. NODE

As mentioned above, one of the main areas of the neural network approach to working with tabular data is modeling decision trees. The main idea of NODE (Neural Obvious Decision Ensembles) is an attempt to make tree functions differentiable.

NODE, like CatBoost, one of the most powerful gradient boosting implementations, uses Obvious Decision Trees (ODT). Their special feature is that predicates at the same level must be the same, which simplifies the construction of the tree. However, NODE also uses antialiasing for differentiability of trees.

If you enter the notation $d$ – depth of the tree, $f \in \mathbb{R}^d$ – sign of predicate partitioning, $b \in \mathbb{R}^d$ – threshold level of partitioning, $R \in \mathbb{R}^d$ – tensor of dimension $d$, then ODT can be written as follows:

$$h(x) = R[\mathcal{H}(f_1(x) - b_1), ..., \mathcal{H}(f_d(x) - b_d)],$$

where $\mathcal{H}$ is the Heaviside function, which is an indicator of getting into the list.

Further, the weight of the $i$-th feature is determined using the decimating function $\text{entmax}$ [10]:

$$\hat{f}_i(x) = \sum_{j=1}^{d} x_j \text{entmax}(F_{ij}),$$

where $F_{ij}$ – trainable weights.

$$\sigma(x) = \text{entmax}([x, 0]),$$

$$c_i(x) = \sigma(\frac{\hat{f}_i(x) - b_i}{\tau_i}).$$

where $b_{bi}$, $ti$ are hyperparameters.

Then Differentiable Oblivious Decision Trees – the weighted sum of leaf outputs - are formally written as:

$$h_t(x) = \sum_{i_{d-1},...,i_{d}} R_{i_{d-1},...,i_{d}} \ast C_{i_{d-1},...,i_{d}}(x),$$

where $C(x) = \left[ 1 - c_1 \right] \oplus \left[ 1 - c_2 \right] \oplus \ldots \oplus \left[ 1 - c_d \right].$

The NODE neural network architecture layer NODE consists of such decision trees. Layer output-concatenation of tree outputs.

In general, the NODE ensemble architecture resembles the DenseNet architecture, where the output of each level consists of a concatenation of the outputs of trees at this level and accepts as input a concatenation of the outputs of all previous levels. The final outputs are averaged.
As a cost function, you can either use the popular cross-entropy or mean squared error, or any other functions.

3. Results
The OpenML AutoML Benchmark was used to test the architectures described above [10]. This benchmark is interesting because it allows testing on thirty-nine datasets and comparing the results with popular AutoML models.

The table below shows the results of testing TabNet, NODE, and self-normalizing neural networks on several datasets. The results of the five models that performed best on the above datasets are also shown.

Measurement of the performance of the TabNet architecture on the benchmark was performed using the implementation [12]. For testing NODE, the implementation was used [13]. The SNN-based architecture SNN was implemented using PyTorch and hyperopt. The implementations used were not adapted for all datasets in the benchmark, so some measurements were not made.

Table 1. Results of the conducted testing.

| Datasets                  | metric | TabNet | NODE | SNN  | autosklearn | autoweka | h2automl | tpot | tunedforest |
|---------------------------|--------|--------|------|------|-------------|----------|----------|------|-------------|
| Adult                     | auc    | 0.917  | 0.847| 0.930| 0.908       | 0.926    | 0.927    | 0.909|             |
| Airlines                  | auc    | 0.686  | 0.694| 0.725| 0.690       | 0.732    | 0.723    | 0.661|             |
| Albert                    | auc    | 0.717  | 0.742| 0.742| 0.763       | 0.738    | 0.738    |      |             |
| Amazon_employment         | auc    | 0.647  | 0.590| 0.856| 0.809       | 0.879    | 0.870    | 0.863|             |
| APSFailure                | auc    | 0.977  | 0.972| 0.644| 0.991       | 0.965    | 0.992    | 0.990| 0.991       |
| Australian                | auc    | 0.699  | 0.891| 0.693| 0.935       | 0.929    | 0.926    | 0.927| 0.909       |
| bank-marketing            | auc    | 0.918  | 0.916| 0.905| 0.937       | 0.827    | 0.937    | 0.934| 0.931       |
| blood-transfusion         | auc    | 0.734  | 0.801| 0.746| 0.757       | 0.741    | 0.756    | 0.724| 0.689       |
| Car                       | logloss| 1.028  | 0.014| 0.701| 0.010       | 0.243    | 0.002    | 0.000| 0.047       |
| christine                 | auc    | 0.698  | 0.817| 0.720| 0.830       | 0.802    | 0.826    | 0.813| 0.810       |
| cnae-9                    | logloss| 2.197  | 0.307| 0.325| 0.171       | 0.873    | 0.155    | 0.196| 0.297       |
| connect-4                 | logloss| 0.517  | 0.492| 5.515| 0.426       | 0.741    | 0.345    | 0.400| 0.478       |
| Covertype                 | logloss| 0.347  | -    | -    | 0.104       | 3.895    | 0.163    | 0.107| 0.161       |
| credit-g                  | auc    | 0.647  | 0.801| -    | 0.783       | 0.753    | 0.789    | 0.786| 0.796       |
| dilbert                   | logloss| 0.087  | 0.129| 0.239| 0.097       | 1.787    | 0.052    | 0.217| 0.329       |
| Dionis                    | logloss| 1.049  | -    | -    | 4.120       | 2.883    | 1.662    | 1.361| 1.016       |
| fabert                    | logloss| 1.411  | 0.987| 1.051| 0.778       | 3.037    | 0.754    | 0.826| 0.806       |
| Fashion-MNIST             | logloss| 0.330  | 0.329| -    | 0.354       | 0.581    | 0.294    | 0.651| 0.362       |
| guillermo                 | auc    | 0.712  | 0.817| -    | 0.901       | 0.878    | 0.910    | 0.819| 0.903       |
| Helena                    | logloss| 2.843  | 2.752| -    | 3.447       | 13.043   | 3.186    | 3.245| 3.559       |
| Higgs                     | auc    | 0.802  | 0.794| -    | 0.793       | 0.677    | 0.814    | 0.802| 0.803       |
| Jannis                    | logloss| 0.701  | 0.741| -    | 0.705       | 6.271    | 0.681    | 0.732| 0.729       |
| jasmine                   | auc    | 0.827  | 0.854| 0.794| 0.884       | 0.861    | 0.888    | 0.885| 0.889       |
| jungle_chess_2pcs_raw_endgame_complete   | logloss| 0.299  | 0.065| -    | 0.234       | 1.559    | 0.218    | 0.198| 0.402       |
| kc1                       | auc    | 0.794  | 0.810| 0.668| 0.843       | 0.814    | 0.836    | 0.841| 0.842       |
| KDDCup09                  | auc    | -      | -    | -    | 0.834       | 0.791    | 0.830    | 0.818| 0.786       |
Table 1. Results of the conducted testing (table continuation).

| Datasets       | metric | TabNet | NODE | SNN   | autosklearn | autoweka | h2oautoml | tpot     | tunedforest |
|----------------|--------|--------|------|-------|-------------|----------|------------|----------|-------------|
| kr-vs-kp       | auc    | 0.994  | 0.999| 0.987 | 1.000       | 0.976    | 1.000      | 1.000    | 1.000       |
| mfeat-factors  | logloss| 18.238 | 0.117| 0.465 | 0.099       | 0.627    | 0.105      | 0.138    | 0.201       |
| MiniBooNE      | auc    | 0.943  | 0.981|       | 0.985       | 0.961    | 0.987      | 0.981    | 0.982       |
| nomao          | auc    | 0.992  | 0.992| 0.912 | 0.996       | 0.984    | 0.996      | 0.995    | 0.995       |
| numerai28.6    | auc    | 0.525  | 0.521|       | 0.529       | 0.520    | 0.532      | 0.525    | 0.521       |
| phoneme        | auc    | 0.927  | 0.951| 0.756 | 0.963       | 0.957    | 0.968      | 0.969    | 0.966       |
| riccardo       | auc    | 0.997  | 0.996|       | 1.000       |          | 1.000      | 0.992    | 1.000       |
| Robert         | logloss| 1.910  | 0.730| 0.230 | 2.303       |          | 1.553      | 4.041    | 1.687       |
| segment        | logloss| 0.371  | 0.194| 0.441 | 0.060       | 0.501    | 0.047      | 0.052    | 0.069       |
| Shuttle        | logloss| 0.016  | 0.013|       | 0.001       | 0.015    | 0.000      | 3.444    | 0.001       |
| sylvine        | auc    | 0.700  | 0.980| 0.875 | 0.990       | 0.975    | 0.990      | 0.992    | 0.984       |
| vehicle        | logloss| 13.183 | 0.377|       | 0.395       | 2.105    | 0.353      | 0.414    | 0.486       |
| Volkert        | logloss| 1.043  | 0.955|       | 0.945       | 1.110    | 0.821      | 1.011    | 0.979       |

4. Conclusions
Based on the measurements made, a few conclusions can be drawn:

TabNet is generally an excellent "out-of-box solution", since even the "default" configuration allows you to achieve results comparable to good solutions. Since TabNet uses the attention mechanism, it needs a large amount of data to realize its potential. These features make TabNet a good solution for big data.

SNN often shows results that are close to the best across the entire benchmark. Presumably, better results can be achieved if in some cases more advanced and complex methods of automatic selection of hyperparameters are used, other than those proposed in the article. In other words, in order to unlock the potential of self-normalizing neural networks in some cases it is necessary to fine-tune the model for specific datasets. This area is promising for further research.

NODE is the most powerful solution presented in this article. NODE shows the best results among all solutions on four datasets and comparable results on almost all others.

References
[1] Popov S, Morozov S and Babenko A 2019 Neural Oblivious Decision Ensembles for Deep Learning on Tabu-lar Data. ArXiv abs/1909.06312.
[2] Arik S Ö and Pfister T 2020 TabNet: Attentive Interpretable Tabular Learning. arXiv:1908.07442v5.
[3] Buturović L and Miljković D 2020 A novel method for classification of tabular data using convolutional neural networks. bioRxiv 2020.05.02.074203; doi: https://doi.org/10.1101/2020.05.02.074203. 2020
[4] Klambauer G, Unterthiner T, Mayr A and Hochreiter S 2017 Self-Normalizing Neural Networks. In Advances in Neural Information Processing Systems (NIPS).
[5] SELU: PyTorch 1.8.1 Documentation 2020. URL:https://pytorch.org/docs/stable/generated/torch.nn.SELU.html
[6] Bergstra J, Yamins D and Cox D D 2013 Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. Proc. of the 30th International Conference on Machine Learning (ICML 2013). June 2013. P. I-115 to I-23.
[7] Bergstra J and Bengio J 2012 Random Search for Hyperparameter Optimization. Journal of Machine Learning Research 1(10) 281-305.
[8] Bergstra J S, Bardenet R, Bengio Y and Kégl B 2011 Algorithms for hyper-parameter optimization. Proceedings of the 25th Annual Conference on Advances in Neural Information Processing Systems, Granada, Spain, 6–12 December 2011 p 2546–2554.

[9] Electric Brain: Learning to Optimize 2018. URL: http://web.archive.org/web/20191120001641/https://www.electricbrain.io/blog/learning-to-optimize

[10] Gijsbers P, LeDell E, Thomas J, Poirier S, Bischl B and Vanschoren J 2019 An Opensource AutoML Benchmark. AutoML Workshop at 6th ICML.

[11] Martins Andre F T and Astudillo Ramon F 2016 From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. URL: https://arxiv.org/pdf/1602.02068.pdf

[12] Arık Sercan O and Pfister Tomas 2020 TabNet: Attentive Interpretable Tabular Learning. URL: https://arxiv.org/pdf/1908.07442.pdf

[13] Popov S, Morozov S and Babenko A 2020 Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data. URL: https://arxiv.org/pdf/1909.06312.pdf