JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Clemens Kriechbaumer

Submitted at
Institute for Machine Learning

Supervisor
Univ.-Prof. Dr. Sepp Hochreiter

Co-Supervisor
Philipp Seidl, MSc
Johannes Schimunek, LAss

January 2022

Attentive Interpretable Tabular Learning in Context of Drug Discovery

Master Thesis
to obtain the academic degree of
Master of Science
in the Master's Program
Artificial Intelligence

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Contents

Acknowledgements

Having time, curiosity and support which are the requirements to be able to learn something shouldn't be taken for granted. I had the privilege and circumstances that these factors came together and hope I'm able to appreciate this enough. In the following I want to thank all who made my studies possible. In case I forget to mention someone I want to apologize in advance.

First of all, I want to thank my family, parents Manuela and Bernhard and my sister Cornelia, without their unconditional support and background in these turbulent times, studying would not have been possible!

Beside the mentioned factors, if you want to learn something you need proper teachers, materials and guidance. And I must say that I really enjoyed the whole program, the classes and exercises and always had the feeling to learn something interesting and useful. Therefore I want to thank all my professors, teachers, tutors and lecturer, especially those from the Institute of Machine Learning. I also want to thank my master thesis supervisors Philipp Seidl and Johannes Schimunek for their guidance and valuable feedback!

Furthermore, I want to thank MIC, the company where I have worked for over 8 years, and related colleagues and supervisors who initially enabled me to study by providing a flexible and supporting working environment.

Personally I have the wish that I will be able to use the acquired knowledge appropriately, hopefully benefiting beings, but atleast not harming anyone. Furthermore I hope that I will not forget that behind all data, statistics, methods and algorithms is always a potential affected living being, which has a unique subjective perception different from mine, but similar to myself also strives for happiness and peacefulness!

Abstract

Despite the success of deep learning (DL) architectures in recent years, decision tree (DT) based methods like random forests (RF) or gradient boosting decision trees (GBDT) still perform well in domains like tabular data. Similarly in the context of drug discovery, DT based methods repeatedly have shown to achieve competitive results.

Recently new DL based architectures have been introduced in the attempt to resemble the success of DT based methods on tabular data. One of them is "TabNet: Attentive Interpretable Tabular Learning" introduced in 2019 by Arik S. and Pfister T. [7]. TabNet by design does feature selection and offers direct interpretability utilizing sparsemax, a sparse alternative to softmax. Reported results surpassed or were on par on various datasets when compared to DT based methods like GBDT. Based on reported results and the relevance of DT based methods in the field of drug discovery, TabNet has been extensively studied in context of drug discovery within this work.

Different drug discovery datasets, BBBP, BACE, HIV and SIDER have been used to compare TabNet with a baseline multilayer perceptron (MLP) and various reference results from literature including RF and GBDT. In none of the used datasets TabNet could outperform a baseline MLP or achieve results similar to RF or GBDT. Relative to the baseline MLP, TabNet performed worst on the SIDER dataset and best on the BBBP dataset with mean area under-ROC-curve (AUC) of $\sim 0.89 \pm 0.03$ for TabNet and $\sim 0.91 \pm 0.02$ for the MLP.

Furthermore in this work an attempt was made to empirically measure TabNet's build in interpretability capabilities in context of drug discovery. Following the experiment setup of recent work by Schimunek J. et al. in 2021 [90], the drug discovery hERG dataset together with known relevant baseline molecules have been used. The task was to rank most relevant atoms of the baseline molecules first. Results have been compared to a MLP, RF and GBDT together with various interpretability methods including integrated gradients, saliency and Shapley values sampling. In this particular experiment setup TabNet was not able to rank most relevant atoms first achieving a mean AUC of $\sim 0.49 \pm 0.06$. The best compared method was a MLP together with Shapley values sampling resulting in a mean AUC of $\sim 0.70 \pm 0.01$.

All experiments results including code, hyperparameters, metrics as well as the TabNet reimplementation is made publicly available to evaluate this work or to try out TabNet in a different domain.

1 Introduction

Non-deep learning machine learning methods like decision tree-based architectures such as random forest (RF) or gradient boosting decision tree (GBDT) are still very much relevant or even state of the art depending on domain and data in the field of machine learning. When looking at recent trends for applied machine learning architectures and methods ranked among data scientists, RF and GBDT are respectively top 2 and 3 of chosen methods ranking above deep learning methods like neural networks [98].

It might not be surprising as decision tree-based methods are performing well on structured or tabular data which are relevant in many business-related applications. On the other hand, for unstructured data, like text, audio, or image, which account for over 80% of the worldwide data [99], deep-learning has shown great success and established a de facto standard [38], [25]. These canonical architectures can learn or encode raw unstructured data in a meaningful and efficient representation, which is critical for solving subsequent problems.

Even though there is no common definition for structured or tabular data, data that is being organized in tabular form of samples as rows and features as columns is referred to as tabular data. Each sample consists of the same set of features. Unlike usual unstructured data, for tabular data, the available features can be a mixture of different data types like numerical, ordinal, categorical, and so on. Additionally, the lack of prior knowledge and sparsity, locally missing data for features, adds an additional challenge [24], [95].

In structured or tabular data raw features (e.g. income, age, nationality) often already represent a meaningful concept. The power of canonical deep learning architectures learning efficient representation based on large amount of data might not be as important as in unstructured data like image or audio, at least for the concrete task to solve. The lack of appropriate inductive bias (e.g. being able to select relevant features) for deep learning architectures in the context of tabular data often leads to over-parameterized and complex models [34], [48].

Even though tabular data and deep learning-based architectures might not be the perfect match it is worthwhile to apply and explore its potential. Besides the obvious task and motivation to improve the performance especially on large amount of data, deep learning offers gradient-based end-to-end learning which more easily enables architectures to integrate tabular data with any other unstructured data type (e.g combined with image data). Additionally, the ability to learn efficient and meaningful representations more easily enables domain adaption between data and tasks. This enables deep learning architectures to be applied on tabular data for transfer learning, generative modeling, or semi-supervised learning [7].

Similar to tabular data in the data domain of drug discovery for many tasks especially activity prediction, decision-tree based architectures like RF or GBDT are still very relevant. A recent extensive study comparing a wide area of machine learning methods including non deep learning and deep learning architectures, came to the conclusion that RF and GBDT perform reasonably well on most tasks and even achieve state of the art results on drug discovery classification tasks [45]. This seems to be an indication that

RF and GBDT are a more than reasonable candidate for both tabular data and drug discovery tasks alike.

Considering the mentioned additional potential of deep learning the motivation to explore newly emerging deep learning architectures and work specialized on tabular data in the context of drug discovery is an obvious motivation. One of recent proposed canonical deep learning architectures claiming to perform better or on par with decision-tree based architectures is "TabNet: Attentive Interpretable Tabular Learning" [7] proposed by Google Cloud AI end of 2019. Beside the promising results when compared to decision-tree based architectures the novel architecture uses a learnable feature selection mask, which can be used to explain which features were relevant for a given task. TabNet is described in detail within section 2. This newly promising architecture and the relevant potential of deep learning for the drug discovery domain leads to this works objectives, described in detail in section 1.2 and is the subject of this master thesis.

1.1 Related Work

Beside TabNet other recent works have explored and proposed specialized canonical deep learning architectures for tabular data. All of them have in common that they are inspired by decision tree-based architectures or use them as a baseline or reference for comparison. The following works are only briefly mentioned here, but are not focus of this work or explained in more detail.

NODE - Neural Oblivious Decision Ensembles [82], uses oblivious decision trees, which are differentiable. The complete architecture is based on an ensemble of differentiable trees.

DNF-NET - Disjunctive Normal Form Networks [2], a novel architecture using disjunctive normal neural form blocks, feature-selection masks and spatial localization weighting emulating decision-trees.

RLN - Regularization Learning Networks [93], which applies regularization coefficient as hyperparameters to each weight in a neural network. Those hyperparameters are tuned minimizing a new Counterfactual Loss.

Regularization is all you Need [46], proposing the usage of a mixture of 13 known regularization methods for plain multilayer perceptron (MLP) to be able to achieve state of the art performance on a large variate of datasets.

To the best of the authors knowledge, TabNet has not been extensively explored or studied in the context of drug discovery. The closest related works were two recent reported Kaggle challenge participants. Those used TabNet in drug discovery related datasets like Mechanism of Action (MoA) [5], [9]. Both only used TabNet as one of many architectures within an ensemble of models. The second mentioned work even claiming to have achieved the 2nd place out of over 4000 competitors. These promising results and the overall novel canonical deep learning architecture led to the author to explore TabNet in detail within this master-thesis.

1.2 Objectives

This master-thesis tries to answer the question if the TabNet is a viable option within the drug discovery domain. Beside the predictive performance another objective of this work is also to look into its capability to be used as a self-interpretable architecture. This leads to the following concrete objectives.

- **Compare the predictive performance of TabNet on various drug discovery datasets.**
  - Datasets chosen are BBBP, BASE, HIV, SIDER and hERG which are drug activity prediction datasets. Datasets are further described in section 3.
- **Empirically analyze and compare TabNet's build in interpretability capabilities.**
  - The hERG datasets (refer to section 3.6) and known relevant molecular substructures are used as a baseline
  - As metric, the ability to rank relevant substructures and their atoms first, is used
  - TabNet should further be compared to other interpretability methods like Integrated Gradients, Shapley Values Sampling or Saliency. The detailed setup is described in section 5.2 - Referred methods are explained in section 2.3.

1.3 Structure and Outline

This master-thesis is divided into the following sections. The introduction section 1 describes the background and motivation why this topic and architecture was chosen and should encourage the reader to continue exploring this work. The theoretical background section 2 covers relevant topics and theory and for this master thesis.

- **Machine Learning (ML)** (section 2.1)
  - **Decision-tree based methods**, with a focus on random forest (RF) and gradient-boosting decision tree (GBDT).
  - **Deep Learning (DL)**, describes the background for artificial neural networks (ANN), multilayer perceptron (MLP) and training using gradient descent and backpropagation.
- **Attentive Interpretable Tabular Learning (TabNet)** (section 2.2), gives a detailed insight into the TabNet architecture with its built-in interpretability capabilities.
- **Interpretable and Explainable Machine Learning** (section 2.3), tries to give an overview of various methods to explaining machine learning models.
  - Gradient based methods like integrated gradients (IG) or saliency which are especially relevant explaining deep learning related methods.
  - Perturbation base methods like Shapley value sampling and feature permutation which can be applied to arbitrary machine learning methods.

- **Drug Discovery** (section 2.4), gives a brief overview of the drug discovery process. Relevant applications like quantitative structure-activity relationship (QSAR) modelling are explained as well as used molecule descriptors.

In section 3, for this work chosen drug discovery related datasets and their backgrounds are explained. Datasets included are BBBP, BASE, HIV, SIDER and hERG. The later one is also used in the context of interpretability for which the relevant baseline and its systematic is described.

The concrete implementations and detailed information are presented in section 4 "Approach". This section describes the general setup as well as the verification of the reimplementation of TabNet prior to furter experiments.

In "Experiments and Results" (section 5), the drug discovery related experiments and their results are presented in detail. It includes the detailed process including hyperparameter selection as well as how achieved results were measured and compared.

In section 6 and 7 the discussion and conclusion regarding the objectives of this work is laid out.

1.4 Notation

If not explicitly defined otherwise in a given context the following mathematical notation is used within this master thesis.

Lower case letters like x or y denote scalar values like 7 or 0.98. Bold letters like $\mathbf{x}$ or $\mathbf{w}$ relate to vectors. Column vector notation is used in the form $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ using transpose T for better readability. Bold upper case letters like $\mathbf{W}$ or $\mathbf{X}$ denote matrices.

Dimensions are described using upper case letters like N or D. Using dimensions a size of vector is described as $\mathbf{x} \in \mathbb{R}^D$ meaning a vector having D real number elements. Similarly dimensions of matrices are given in the form $\mathbf{W} \in \mathbb{R}^{N \times D}$ describing a matrix having N rows or samples each represented by D features.

2 Theoretical Background

2.1 Machine Learning

The terms machine mearning (ML) and artificial intelligence (AI) are often used interchangeably, and not well distinguished. Therefore, before going into detail about machine learning both terms and their relation are pointed out.

Depending on the field of research there exists multiple definition for the term AI. In the following paragraphs some definitions are presented giving an insight into the wide range of possibilities and angles one can look at the topic. Based on the simple question "What is AI?" one possible answer is "It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable." [73]

A different angle is to categorize AI in terms of weak AI and strong AI [60], or narrow AI and artificial general intelligence (AGI). Weak refers to an AI which pretends to think, strong AI refers to a mind with mental states. Narrow AI exceeds humans in a specific limited area or task while AGI mimics and acts on the same level as a human mind but without consciousness [62].

In context of computer science, AI research and its potential applications can be categorized by two streams. The objective, thinking vs acting, and the type of decision making, humanly or rationally. This leads to a categorization of four research types [88, p. 5], [60].

- Cognitive modeling or thinking humanly, the ultimate goal would be to develop a machine imitating a mind.
- Turing test, implies acting humanly or show human like intelligence when interacting with humans.
- Laws of thought refers to thinking rationally, not necessarily give the same results as a human would do.
- Rational agent stream can be explained as acting rationally and refers to autonomous and intelligence agent following an objective achieving the rationally best outcome.

Based on different definitions and categorization it becomes clear that AI covers a wide range of topics and is still inaccurately defined. When looking at machine learning (ML) the research stream of "Rational Agent" within AI comes closest to ML which leads to the conclusion that ML is a branch of AI. Tom M. Mitchell states, that "Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data" [75].

Generally speaking machine learning involves "learning" by using experience, past events or collected facts in form of data. Usually in machine learning inductive learning

in which rules or pattern are identified, is followed by a deductive inference process during which those learned rules are applied to solve a concrete task. For traditional algorithms especially in context of computer science those rules are not inductively learned, but predefined following laws of nature or business objectives. Within Machine Learning there are multiple types of learning problems which can be categorized as follows [14]:

- **Supervised machine learning (SML)** includes a mapping in form of some input data $x \in X$ and corresponding target data $y \in Y$ for which a mapping function $g : X \to Y$ must be learned. This mapping function can be parameterized $y = g(x; \phi)$ for which a fixed set of parameters ϕ must be learned, or non-parameterized $y = g(x)$ which primarily depends on the size of data of x [11]. Depending on the type target data Y there are two main types supervised learning problems:
  - *Classification* tasks which involve the prediction of concrete class labels or
  - *Regression* tasks that involve the prediction of a numerical value
- **Unsupervised machine learning (UML)** describes methods which learn relationship within data $x \in X$ without any explicit given target data $y \in Y$. There is no "supervision" for the model during learning. Typical tasks are to find the "best" representation which usually is a simpler or more accessible version of x while still preserving relevant information [34]. Problems solved are for example:
  - *Clustering* problems, which try to find a predefined number of groups within data belonging together.
  - *Density estimation* which learns and summarizes the distribution of data.
- **Semi-supervised learning** is an approach which blurs supervised with unsupervised learning. For each input data $x_1, ...x_l \in X$ there exists target data $y_1, ..., y_l \in Y$. Additionally $x_{l+1}, ...x_n \in X$ exists without any corresponding target data and is referred to as unlabeled data. One simple approach is self-learning. In self-learning one case is to learn from target data similar to supervised learning and afterwards predict $\hat{y_{l+1}}, ...\hat{y_n} \in Y$. Afterwards use the combined set of $x_1, ...x_l + x_{l+1}, ...x_n \in X$ with target data $y_1, ..., y_l + \hat{y_{l+1}}, ...\hat{y_n} \in Y$ including predicted target data, usually thresholded to only include quality prediction, to relearn a model[63]. Other prominent approaches include generative models, graph-based learning or co-training [18].
- **Self-supervised learning** refers to reformulating an unsupervised learning problem such that a supervised learning method can be applied. In such a way it is possible to learn representations without direct target data $y \in Y$. Those learned representations can be used in a downstream task, one example is to fine tune those on actual target data, possible in a different data domain [54]. One simple but prominent example is an Autoencoder. Given input data $x \in X$ there exists a parameterized encoder function u which is used to learn a latent or hidden representation $h = v(x; \phi_{enc})$. Using a parameterized decoder function v such that one is able to reconstruct the original input data x as close as possible using the hidden latent representation, here referred to as $x' = v(h; \phi_{dec})$.

  The concept of encoder and decoder is an important concept within machine learning in general and for self-supervised learning in particular. Other prominent examples

are generative adversarial networks (GAN) [35] or variational autoencoder (VAE) [51].

- **Reinforcement learning (RL)** describes algorithms which can interact with an environment by learning from provided feedback seeking to achieve a defined goal or maximize a reward. As the learning involves a feedback loop, potentially noisy and delayed, there is no fixed data which is learned on [102].

Within machine learning beside the categorization of the learning problem also the learning technique is important and can be distinguished. Some important ones are briefly presented as follows:

- **Multitask learning** refers to the fact that a method can learn multiple tasks at once. Using the same input data $x \in X$ we have multiple target data $y_1 \in Y_1$, and $y_2 \in Y_2$ such that a function f (parameterized or non-parameterized) can learn to infer both set of target data $y_1, y_2 = f(x)$. This is especially useful in scenarios where the number of labeled data for multiple targets is not the same [34, p. 244].

- **Active learning** refers to strategies how to effectively use human or other expert systems in a machine learning data preparation loop. Especially in SML at some point in time those data must be annotated or labeled. In practice this process is very resource intensive and often labeling all data is not possible. Therefore, the machine learning method as learner must be able ask queries if stuck. There exists multiple query strategy frameworks. One simple approach is to start learning on few labeled data and relabel those data or new input data for which the learner or machine learning model cannot give a confident target prediction. This is refereed to as uncertainty sampling. Other strategies include variance reduction, expected error reduction or the query-by-committee approach [91].

- **Transfer learning** uses an already trained model which learned to solve one particular task within one data domain as a starting point for a subsequent task. Such a model is also referred to as pretrained model. Starting from a pretrained model a potential different task in a different but usually related data domain can be learned. This process is also referred to as fine tuning. Prominent examples for transfer learning can be found within the computer vision (CV) field or within the natural language processing (NLP) research stream [106].

- **Ensemble learning** describes the usage of multiple methods or learner $f_1, ...f_n$ which all are trained on the same input data $x \in X$ having target data $y \in Y$. An aggregation or voting function g is using individual learners output to return the overall decision $\hat{y} = g(f_1(x), ...f_n(x))$. The idea is that an ensemble of learner can overcome individual errors or problems. In many machine learning challenges and real world applications an ensemble of methods often achieves state of the art performance. Prominent examples of methods which are build around ensemble learning are random forests refer to section 2.1.1.1 or gradient boosting decision trees refer to 2.1.1.2. Both are using an ensemble of decision trees but ensemble learning is also able to combine multiple different types of methods or architectures [89].

Within the ML field there exists many different algorithms and methods to achieve the goal of learning. In the following some examples are listed and in the following subchapters for this work most relevant ones are described in detail.

- *Linear Regression* (section 2.1.2.1)
- *Neural network* (section 2.1.2.3)
- *k-nearest neighbors (k-NN)*
- *Support vector machine (SVM)*
- *Decision tree* (section 2.1.1)
- *Naive Bayes*
- *Perceptron and multilayer perceptron (MLP)* (section 2.1.2.3)
- *Recurrent neural network (RNN)*
- *...*

2.1.1 Decision-Tree based Methods

Decision trees (DT) are simple models which work by partitioning the input data into regions. A DT involves the selection of the best feature such that the data or distribution can be divided into most homogeneous parts. This is an iterative process which builds up the complete tree starting from a root node, selecting intermediate nodes and ends with homogeneous leaf nodes [55].

Consider a classification problem with input feature vector $\mathbf{x} = (x_1, ...x_m)$ and binary target $y \in 0, 1$. The actual training data consists of multiple input vectors $S = \mathbf{x_1}, ...\mathbf{x_n}$ with corresponding target data $T = y_1, ...y_n$. First we need to select an element $x_1, ...x_m$ of the feature space by which the overall training data can be split such the split is most homogeneous. In a binary case this could be in $T_{split} = y \in T | y = 1$.

In a complex feature space finding the optimal element by which to partition data is critical. Therefore, a concrete criteria or method is necessary to determine the optimal split. Two prominent splitting criteria are information gain and Gini impurity:

- **Information gain (IG)** is based on the concept of reduction of entropy. The idea is that larger information gain is based on a lower entropy and therefore a favored split. Consider a discrete random Variable $X = x_1, ...x_n$ with probability $P(x_1), ...P(x_n)$ the entropy (Shannon) can be written down as follows:

$$H(X) = -\sum_{i=1}^{n} P(x_i)\ log(P(x_i)) = \sum_{i=1}^{n} P(x_i)\ I(x_i) = \mathbb{E}[-log(P(X)) = \mathbb{E}[I(X)] \tag{2.1.1}$$

  The term $I(X)$ or $-log(P(X)$ refers to the self-information of information context. For example a dice having only one value on all sides would have no entropy (0). This is also referred to as purity or no surprise within information theory framework.

Information gain for a single subset or split can be described as the difference between, a prior entropy of the data and the entropy of the conditional data given a certain feature x_a:

$$IG(S, x_a) = H(S) - H(S|x_a) \tag{2.1.2}$$

More generally defined for a given node S_i the information gain is defined over its k splits or child nodes $S_1, \dots S_k$. $\frac{|S_j|}{|S|}$ denotes the number of elements in that split and is basically weighting the entropy:

$$IG = H(S, i) - \sum_{j=1}^{k} \frac{|S_j|}{|S|} H(S_j), \tag{2.1.3}$$

Lets consider a balanced binary classification task $T = 0, 1, 0, 1$ for which we split the data perfectly into two homogeneous groups $T_1 = 0, 0$ and $T_2 = 1, 1$. The IG would be calculated by using a prior entropy minus the two groups entropy $IG = H(T) - 1/2 * H(T_1) - 1/2 * H(T_2) = 1$. For the other extreme ($T_1 = 0, 1$, $T_2 = 0, 1$) the information gain would be $IG = H(T) - 1/2 * H(T_1) - 1/2 * H(T_2) = 0$. The later one would be considered an inferior split compared to the first split.

- **Gini impurity gain** g_G, is a concept using the Gini impurity (I_G) which is refereed to if an element from a set is randomly chosen and incorrectly labeled by randomly selecting a label from the distribution of labels. This can be defined as follows:

$$I_G(S) = 1 - \sum_{j=1}^{J} p_j(S)^2 \tag{2.1.4}$$

J denotes the number of target classes while $p_j(S)$ describes the frequency of occurrence of one particular class within one set. Naturally it follows if all labels in a set fall within one class the I_G would be 0. The g_G is defined as given:

$$g_G(S) = I_G(S) - \sum_{j=1}^{K} \frac{|S_k|}{|S|} \cdot I_G(S_k) \tag{2.1.5}$$

As an example consider the balanced binary classification tasks $T = 0, 1, 0, 1$ with again two perfectly homogeneous groups $T_1 = 0, 0$ and $T_2 = 1, 1$. The Gini impurity gain would be calculated as $g_G(T) = 1 - 1/2 * I_G(T_1) - 1/2 * I_G(T_2) = 1 - 1/2 * 0 - 1/2 - 0$ and considered perfect or pure.

To actually build up a decision tree multiple algorithms exist. One of the earliest one is ID3 (Iterative Dichotomiser 3) introduced in 1986 by Ross Quinlan [85]. ID3 supports categorical features and iterates over the available features in a greedy way using IG to identify the ideal splitting feature at each node. The successor to ID3 is C4.5 which supports also numerical features by dynamically partition a numerical feature into discrete set of intervals. An alternative algorithm is CART (Classification and Regression Trees) similar to C4.5 but directly supporting numerical features using a threshold to define splits [1].

General advantages of decision trees are that they are relatively simple and easy to interpret. Disadvantages are that they tend to overfit the data and are sensibly to outliers and small changes in the data. Therefore usually decision trees are combined as an ensemble of trees and trained using methods like random forest (RF) and gradient boosting decision tree (GBDT).

2.1.1.1 Random Forest

Random forest (RF) describe the usage of multiple decision trees as an ensemble. The bases for RF was first introduced in 1995 by Tin Kam Ho [40] and refined in 2001 by Leo Breiman [13]. One of the main concept is called "bagging". For this concept two variants are used within RF.

- **Bagging** describes the basic technique of bootstrapping or bootstrap aggregation. Consider dataset $S = (x_1, ...x_n)$ with target data $T = (y_1, ...y_n)$ we sample with replacement K sub-datasets S_k, T_k consisting of m entries ($m < n$). For each of the sub-dataset a decision tree f_k is created or trained in a SML typical setting $T_k = f_k(S_k)$. During inference unseen data S' uses the overall result or prediction of the majority vote of the individual decision tree. This can be defined as follows:

$$\hat{T} = \frac{1}{K} \sum_{k=1}^{K} f_k(S') \tag{2.1.6}$$

- **Feature bagging**, similar to bootstrap aggregation using a random sample of individual entries within the data feature bagging refers to using a random sub-sample of features a during the build up of the individual trees. For example in the binary classification case typically only $\sqrt{a}$ are considered to determine the best split.

In contrast to decision trees a RF is less likely to overfit due to bootstrap aggregation. Due to the complexity of many decision trees combined in an ensemble to interpret the results or prediction is more difficult.

2.1.1.2 Gradient Boosting Decision Tree

Beside bootstrap aggregation like in RF another possibility to build up trees is using a technique called gradient boosting (GB). Before defining GB formally consider the overall model F which predicts values as $\hat{y} = F(x)$. To build up F GB uses M estimators which sequentially build up an ensemble which is used for the final prediction. Starting from an imperfect function F_m which could be a constant function or the average function $\hat{y}_i = \bar{y}$ another more refined function or estimator h_m is added.

$$F_{m+1}(x) = F_m(x) + h_m(x) = y \tag{2.1.7}$$

This could be reformulated as

$$h_m(x) = y - F_m(x) \tag{2.1.8}$$

and describes that in GB h_m is using the error or residual $y - F_m(x)$ as training data. In general boosting refers to the technique iteratively correct the error of the predecessor model. If h_m is fitted on the gradient g of a loss function L such as

$$g_m = \frac{\partial L(y, F_m(x))}{\partial F_m(x)} \tag{2.1.9}$$

the algorithm is referred to as gradient boosting or gradient boosting machine [32].

The general GB algorithm can be described as follows. Starting from a constant value function $F_0(x)$ using a differentiable loss function L with data $x_i, y_i{}_{i=1}^{N}$ GB can be described as follows [32]:

$$F_0(x) = \underset{p}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, p) \tag{2.1.10}$$

For $1 <= m <= M$ boost or improve the overall function model F by first calculate the pseudo-residual of the $m-1$ improved model F:

$$r_{im} = -\frac{\partial L(y_i, F_{m-1(x_i)})}{\partial F_{m-1}(x_i)} \tag{2.1.11}$$

Fit a base or weak learner (e.g. a decision tree) h_m using $x_i, r_i m_{i=1}^{N}$ as training data. Afterwards approximate the optimal coefficient p_m using line search:

$$p_m = \underset{p}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + p h_m(x_i)) \tag{2.1.12}$$

Within each step boost the overall model

$$F_m(x) = F_{m-1}(x) + p_m h_m(x) \tag{2.1.13}$$

which results in the final output model F_M.

When using a decision tree as a base or weak learner there exists multiple different implementations. Those differ in the way the decision tree is fitted and the residual or gradients are used to build up the individual trees. For details refer to the most prominent implementations which are XGBoost [19], LightGBM [47] and CatBoost [84].

2.1.2 Deep Learning

The bases for deep learning (DL), using neural networks (NN) and many of its fundamental concepts are around since decades. Due to the ever-increasing computational power, especially processing units performing exceptional well doing linear algebra (e.g., graphical processing units - GPU), the research field of neural networks reemerged as deep learning. The recently available computational power and break-through in research enabled neural networks with complex architectures with many billions of parameters and having multiple layers. Mainly the fact of having many layers within a NN lead to the term deep neural networks (DNN) and deep learning. In the following the core concepts of NN and DL are presented roughly following terminology and structure of Klambauer et al. [52].

2.1.2.1 Linear Model

A single output linear model describes a single output or target y which can be described by a linear combination of up to D features defined by input vector $\mathbf{x}$. This can be defined as:

$$y = \mathbf{w}^T\mathbf{x} + b \tag{2.1.14}$$

$y \in \mathbb{R}$ refers to the output, $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{w} \in \mathbb{R}^D$ are the coefficients or parameters. b refers to the bias or error term, within linear models referred to as noise. Within a multi task linear model the output becomes $\mathbf{x} \in \mathbb{R}^K$, the parameters become the matrix $\mathbf{W} \in \mathbb{R}^{K\times D}$ and the bias is defined as $\mathbf{b} \in \mathbb{R}^K$. Combined this can be written as:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.1.15}$$

For the single output linear model giving N samples such that $\mathbf{y} \in \mathbb{R}^N$ and $\mathbf{X} \in \mathbb{R}^{N\times D}$ the task is to minimize the empirical error $R_{emp}(\mathbf{y}, \mathbf{X}, \mathbf{w}, b)$. This describes the usage of an objective or loss function L on all available samples N to find the optimal parameters $\mathbf{w}, b$ such that R_{emp} becomes minimal.

For linear models one objective could be the squared residuals $(\mathbf{y} - (\mathbf{w}^T\mathbf{X} + b))^2$. If summed up over all N samples this is referred to as residual sum of squares (RSS). It follows that R_{emp} for linear model using RSS can be defined as:

$$R_{emp}(\mathbf{y}, \mathbf{X}, \mathbf{w}, b) = L(\mathbf{y}, \mathbf{X}|\mathbf{w}, b) = \frac{1}{2}(\mathbf{y} - (\mathbf{X}\mathbf{w} + b))^T(\mathbf{y} - (\mathbf{X}\mathbf{w} + b)) \tag{2.1.16}$$

To find the optional estimator $\hat{\mathbf{w}}, \hat{b}$ one needs to solve the following equation:

$$\hat{\mathbf{w}}, \hat{b} = \arg\min_{\mathbf{w},b}(L(\mathbf{y}, \mathbf{X}|\mathbf{w}, b)) \tag{2.1.17}$$

For a linear model there exists a closed solution using RSS as L which results in the best unbiased estimator for $\hat{\mathbf{w}}, \hat{b}$ [11, p. 146].

When the output is a binary classification the output becomes $y \in (-1, 1)$. Such regression model is possible if a "sign" function or step function is applied to the output of a liner model and can be described as:

$$y = sign(\mathbf{w}^T\mathbf{x}) \tag{2.1.18}$$

The sign function is thresholded around 0, meaning if the intermediate output a within the linear transformation $a = \mathbf{w}^T\mathbf{x} \geq 0$ $y = 1$ otherwise $y = -1$. This setup is also referred to as perceptron. For a perceptron there exists no closed form solution for the best estimator of parameters $\hat{\mathbf{w}}$. If data and classes are linear separable, a perceptron can be used and trained using a simple update rule based on the gradient w.r.t. $\mathbf{w}$. As the *sign* function is non continuous the overall result is either true, separable or not separable. Updating using the gradient therefore in sequential manor will not converge in non-separable target data scenario.

2.1.2.2 Logistic Regression

For non-linear scenarios the sigmoid function σ can be used to approximate the *sign* function in an probabilistic way. The output y is within the interval $[0, 1]$ and describes the probability that $y = 1$. σ is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1.19}$$

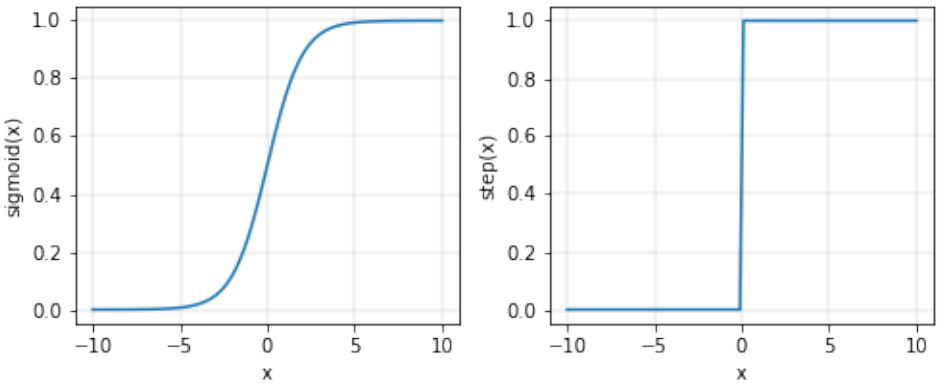


Figure 1: σ function on the right side as the approximation of an step function

When using σ on the output of a single target linear model this can be described as a single linear neuron with σ activation. In the context of neural networks and deep learning this is also referred to as an activation function. The linear neuron with σ activation is defined as:

$$y = \sigma(\mathbf{w}^T\mathbf{x}) \tag{2.1.20}$$

Analog to linear regression the optimal parameters of the model $\hat{\mathbf{w}}$ (for simplicity the bias term b is not included here) needs to be determined such that the R_{emp} becomes minimal. As loss function L for the R_{emp} in the logistic regression setting the binary cross entropy is used and defined as follows:

$$R_{emp}(y, \mathbf{x}, \mathbf{w}) = L(y, \mathbf{x}|\mathbf{w})) = -\Big(y \log\Big(\sigma(\mathbf{w}^T\mathbf{x})\Big) + (1 - y)log\Big(1 - \sigma(\mathbf{w}^T\mathbf{x})\Big)\Big) \tag{2.1.21}$$

There exists no closed form solution for the σ function therefore an iterative approach using the first order gradient of the L function w.r.t. $\mathbf{w}$ is used. First the gradient of σ w.r.t. $\mathbf{w}$ is defined as:

$$\frac{\partial\sigma(\mathbf{w}^T\mathbf{x})}{\partial\mathbf{w}} = \Big(\sigma(\mathbf{w}^T\mathbf{x})(1 - \sigma(\mathbf{w}^T\mathbf{x}))\Big)\frac{\partial\mathbf{w}^T\mathbf{x}}{\partial\mathbf{w}} = \Big(\sigma(\mathbf{w}^T\mathbf{x})(1 - \sigma(\mathbf{w}^T\mathbf{x}))\Big)\mathbf{x} \tag{2.1.22}$$

Using $\frac{\partial\sigma(\mathbf{w}^T\mathbf{x})}{\partial\mathbf{w}}$ the gradient of the binary cross entropy loss function as defined in equation 2.1.21 w.r.t. to $\mathbf{w}$ is given as:

$$\nabla_{\mathbf{w}} L(y, \mathbf{x}|\mathbf{w})) = \frac{\partial L(y, \mathbf{x}|\mathbf{w}))}{\partial \mathbf{w}} = (\sigma(\mathbf{w}^T\mathbf{x}) - y)\mathbf{x} \tag{2.1.23}$$

Using the gradient $\nabla_{\mathbf{w}}$ the parameters are updated until convergence of the loss function L or up to a defined maximum number of iterations T. This process is called gradient descent and given as:

$$\mathbf{w^t} = \mathbf{w^{t-1}} - \eta \nabla_{\mathbf{w^{t-1}}} \tag{2.1.24}$$

The term η determines the size of the step which is taken into the direction of a local minima of the loss function L. This is refereed to as learning rate and a key concept within gradient descent and DL.

2.1.2.3 Multilayer Perceptron

Even though within logistic regression a non-linear function σ is used, some non-linear problems like the XOR problem can not be solved with just a single layer or transformation in the form of $y = f(\mathbf{w}^T\mathbf{x})$ with f as an non linear activation function (e.g., σ) [74].

When using multiple layers or transformations in which the output of one layer is the input of a subsequent one the XOR problem can be solved. The basic form of a multiplayer perceptron (MLP) using a σ activation function is given as:

$$y = \sigma(\mathbf{w}^T\sigma(\mathbf{W}\mathbf{x})) \tag{2.1.25}$$

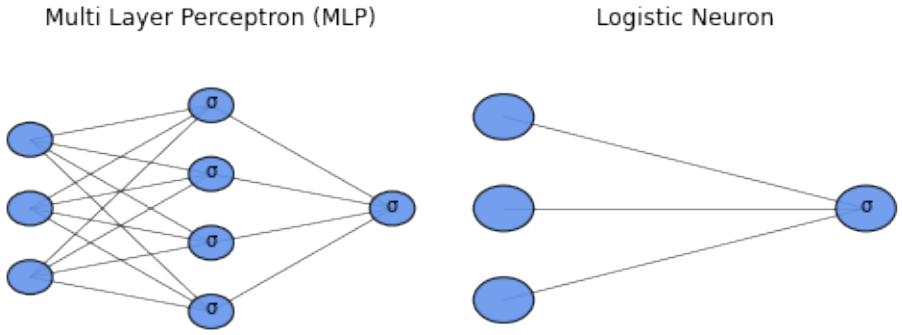


Figure 2: Multilayer perceptron (MLP) sample with 2 layers on the left, single layer linear neuron sample with σ activation (logistic neuron) on the right.

Within figure 2 an example MLP is shown with 2 layers. The middle layer is referred to as intermediate or hidden layer, in this example having 4 neurons. The principal architecture of any neural network is similar and only differs in the number of hidden layers, neurons, activation functions used and output activation. The term deep has no defined common meaning and only indicates a NN having multiple layers. This is also referred to as deep neural network (DNN).

The components of any MLP or NN(DNN) can be summarized as follows:

- Having multiple layers ($1 <= l < L$).
- Having multiple parameter sets in the form of $\mathbf{W}^l$ connecting layer $l-1$ with layer l.
- The size of each individual hidden layers H, referred to as nodes or neurons. This is also reflected in the dimension of the parameter matrix $\mathbf{W}^l \in \mathbb{R}^{H^l \times H^{l-1}}$.
- A parameter set $\mathbf{W}^0 \in \mathbb{R}^{H^l \times D}$ connecting the input data $x \in \mathbb{R}^D$ with the first hidden layer.
- A parameter set $\mathbf{W}^L \in \mathbb{R}^{K \times H^{l-1}}$ connecting the last hidden layer with the output data $y \in \mathbb{R}^K$.
- A non-linear activation function f. The output is referred to as the activation of a layer a^l. The function f is applied using the output of the linear transformation between the parameter set of the corresponding layer and the activation of the previous layers in form of $a^l = f(\mathbf{W}^l a+^{l-1})$.
- A bias term $\mathbf{b} \in \mathbb{R}^H$ which is added to the pre-activation $\mathbf{W}^l a^{l-1} + \mathbf{b}$. The term is optional and not further considered in the following equations.
- An output activation function τ depending on the task.

Defining the pre-activation $\mathbf{s}^l = \mathbf{W}^l \mathbf{a}^{l-1}$ the formal description of a MLP is given as follows, this is also referred to as the forward path:

$$\hat{\mathbf{y}} = \tau(\mathbf{W^L}(\mathbf{f}(\mathbf{W^{L-1}}(\mathbf{f}(\mathbf{W^{L-2}}(\dots \mathbf{f}(\mathbf{W^0 x}))))))) \tag{2.1.26}$$

In figure 3 an schematic DNN is shown. The edges represent individual parameters in the parameter matrix $\mathbf{W}^l$. This concrete example having $L = 4$, input dimension $D = 3$, with overall 605 parameters.

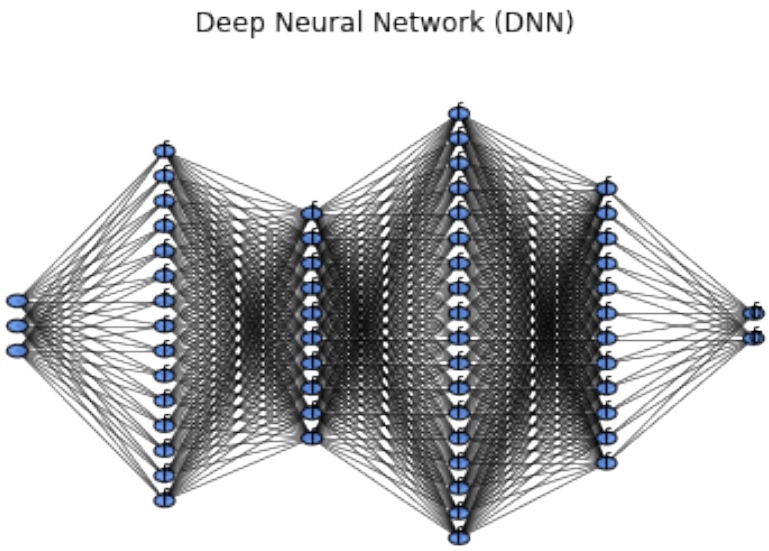


Figure 3: Multilayer perceptron (MLP) having many layers, referred to as neural network (NN) or deep neural network (DNN)

Analog to logistic regression, refer to section 2.1.2.2, a DNN is trained and optimized using gradient descent. Therefore using a differentiable loss function L the gradient w.r.t. each individual parameter set $\mathbf{W}^l$ is calculated.

For the parameters set $\mathbf{W}^L$ connecting the last hidden layer with the target data y this is defined as:

$$\frac{L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}^L} = \frac{L(\mathbf{y}, \tau(\mathbf{s}^L))}{\partial \mathbf{W}^L} = \frac{L(\mathbf{y}, \tau(\mathbf{s}^L)}{\partial \tau(\mathbf{s}^L)} \frac{\partial \tau(\mathbf{s}^L)}{\partial \mathbf{s}^L} \frac{\partial \mathbf{s}^L}{\partial \mathbf{W}^L} = \frac{L(\mathbf{y}, \tau(\mathbf{s}^L)}{\partial \tau(\mathbf{s}^L)} \frac{\partial \tau(\mathbf{s}^L)}{\partial \mathbf{s}^L} \mathbf{a}^{[l-1]T} \tag{2.1.27}$$

For any intermediate or hidden layer l and the corresponding parameter set $\mathbf{W}^l$ the gradient is calculated using the gradient of the loss function w.r.t. the pre-activation s^l. This is also referred to as error or delta δ and given as:

$$\delta^{[l]T} = \frac{L(\mathbf{y}, \tau(\mathbf{s}^L))}{\partial \mathbf{s}^l} \tag{2.1.28}$$

This error or delta is propagated back from the loss function l to the first parameter set $\mathbf{W}^0$ as shown in the following equation. This is referred to as **backpropagation** and one of the core principal for training DNN and for DL in general.

$$\delta^{[l-1]T} = \frac{L(\mathbf{y}, \tau(\mathbf{s}^L))}{\partial \mathbf{s}^l} \frac{\partial \mathbf{s}^l}{\partial \mathbf{s}^{l-1}} \tag{2.1.29}$$

Given the delta error for defined layer l the gradient of the overall forward path including the loss function w.r.t. any arbitrary parameter set $\mathbf{W}^l$ becomes:

$$\frac{L(\mathbf{y}, \tau(\mathbf{s}^L))}{\partial \mathbf{W}^l} = \mathbf{a}^{l-1} \delta^{[l]T} \tag{2.1.30}$$

2.2 Attentive Interpretable Tabular Learning

Within the paper "TabNet: Attentive Interpretable Tabular Learning" in 2019 by Arik S. and Pfizer T. a novel deep learning architecture TabNet was introduced [7]. Inspired by ensembles of DT like RF or GBDT, refer to section 2.1.1, its architecture uses a ensemble of DNN each referred to as decision step. Similar to a RF only a subset of features which are selected in the forward, are used within each step. This allows that the capacity within each step is used for the most salient features. In contrast to a RF the selected sub-features are not randomly selected but learned. The TabNet architecture by design explains which features are used and gives insight into each features contribution to the overall result or prediction.

Additionally, TabNet is differentiable and trained using gradient descent which enables end-to-end training and potentially combining it with additional data or objectives. This gives the advantage over DT based architectures for representation learning using self-supervised pre-training which potentially enables transfer learning and domain shifts.

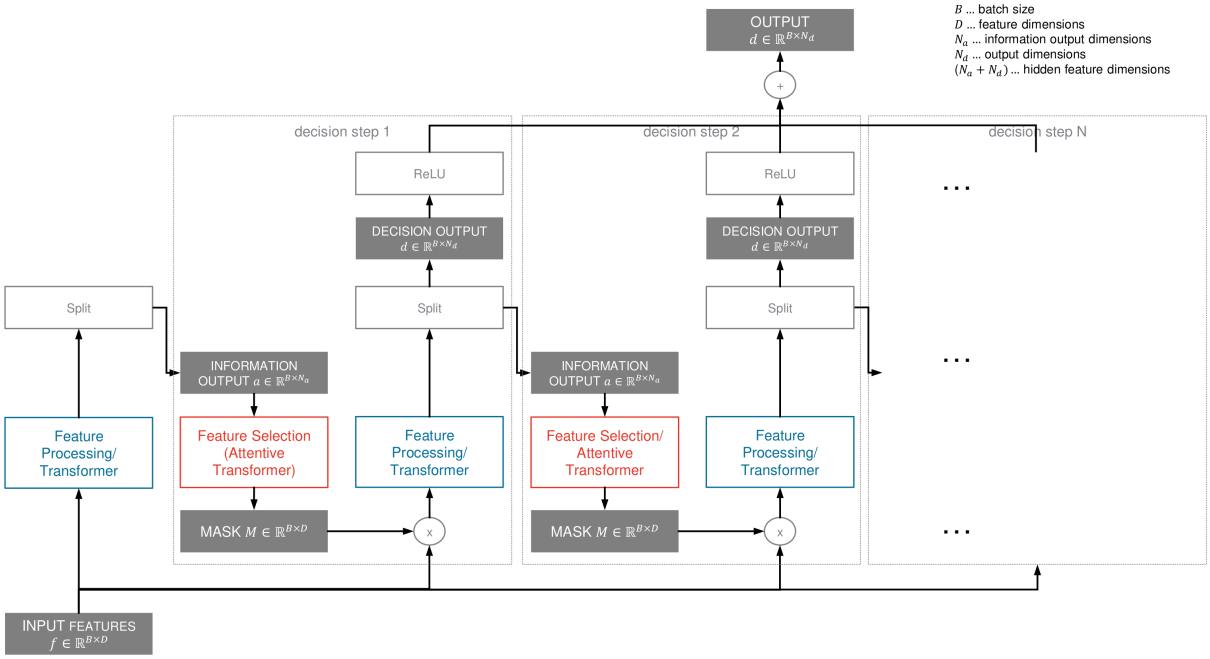


Figure 4: TabNet architecture overview

In figure 4 the overall TabNet architecture is shown. Inputs to the architecture are in the form $\mathbf{x} \in \mathbb{R}^{B \times D}$, where B refers to the sample size and D determines the number of features or feature dimensions. Categorical feature dimension are represented as learnable embeddings with dimension size 1. Raw inputs are not globally normalized but batch norm (BN) [44] is applied. Using the batch normalized inputs TabNet uses up to N_{steps} individual blocks called decision steps. Each step i consists of the following components:

- **Attentive Transformer** for feature selection which uses some information or output from the previous decision step $\mathbf{a}^{[i-1]} \in \mathbf{R}^{B \times N_a}$ and a prior scale $\mathbf{P}^{[i-1]} \in \mathbf{R}^{B \times D}$ to determine the features which are processed within the step i. The output is a learnable mask $\mathbf{M}^{[i]} \in \mathbf{R}^{B \times D}$ which is used for feature selection.

- **Feature Transformer** for feature processing, using the feature selection mask $\mathbf{M}^{[i]}$ to determine the most salient features for step i. Those are selected using element-wise multiplication with the batch normalized inputs $\mathbf{x}$ in the form $\mathbf{x}_i = \mathbf{M}^{[i]} \odot \mathbf{x}$.

- **Split** block which separates the feature processing output into two parts the decision output $\mathbf{d}^{[i]} \in \mathbf{R}^{B \times N_d}$ and the information output $\mathbf{a}^{[i]} \in \mathbf{R}^{B \times N_a}$. The dimensions N_d and N_a add up to the overall features dimension $D = N_a + N_d$. The information output $\mathbf{a}^{[i]}$ is used as the input the subsequent step $i + 1$. The non-linear activation function ReLU $ReLU = max(0, x)$ is applied to each individual decision output $\mathbf{d}^{[i]}$ and furthermore aggregated with all other decision steps output:

$$\mathbf{d_{out}} = \sum_{i=1}^{N_{steps}} ReLU(\mathbf{d^{[i]}}) \tag{2.2.1}$$

2.2.1 Feature Transformer

Each feature transformer (FT) consists of multiple layers. Each layer l consists of a fully connected layer (FC) or linear transformation in the form:

$$FC_{FT}(\mathbf{x}) = \mathbf{W}^{[l]}\mathbf{x} \tag{2.2.2}$$

The parameters $\mathbf{W}$ are of dimension $\mathbf{W}^{[1]} \in \mathbb{R}^{H*2 \times D}$ for the first layer connecting input with subsequent layers and $\mathbf{W}^{[l]} \in \mathbb{R}^{H*2 \times H}$ for any following layer $1 < l <= N_{layers}$. The dimension H refers to the sum of the decision output dimension N_d and information output dimension N_a as $H = N_d + N_a$. The output of the FC transformation is followed by batch norm (BN).

To better support large batch size during training a variant called ghost BN (GBN) is used within the feature transformer. Within BN the statistics for mean μ and variance σ are dependent on the sample or batch size B. GBN split the complete batch $B_{complete}$ into J chunks in the form $B_{complete} = [B_1, B_2, \cdots, B_J]$. On each of the chunks the batch statistics are calculated and applied separately such as $[\mu_1, \mu_2, \cdots, \mu_J]$ and $[\sigma_1, \sigma_2, \cdots, \sigma_J]$. Experiments showed that this adaption of BN reduces the generalization error when large batch size are used [42].

As non-linearity a gated linear unit (GLU) [22] is used. Given the output of the FC transformation $\mathbf{h} \in \mathbb{R}^{H*2}$ which is spitted into two parts $\mathbf{h}_a \in \mathbb{R}^H$ and $\mathbf{h}_b \in \mathbb{R}^H$, GLU is defined using a sigmoid activation function σ, refer to section 2.1.2.2, as follows:

$$GLU(\mathbf{h}) = \mathbf{h}_a \odot \sigma(\mathbf{h}_b) \tag{2.2.3}$$

The output of the non-linearity is again of size $\mathbb{R}^H$, this also explains the dimensions of the FC transformation above being $H * 2$. GLU activation allows unimpeded information flow through the network similar to recurrent neural networks (RNN) with gating mechanism like long short-term memory (LSTM) [41].

Combining those blocks one layer can be summarized as:

$$L(\mathbf{x}) = GLU(BN(FC_{FT}(\mathbf{x}))) \tag{2.2.4}$$

The output of each layer l is connected the output of the previous layer using skip connection in the form:

$$L^{[l]}(\mathbf{h}) = \left(L^{[l]}(\mathbf{h}) + L^{[l-1]}(\mathbf{h})\right)\sqrt{0.5} \tag{2.2.5}$$

The factor $\sqrt{0.5}$ refers to a normalizing strategy for residual blocks which halves the variance of the sum of both terms [33].

To improve parameter efficiency of TabNet with multiple decision steps layers or more precise the parameters $\mathbf{W}$ of the FC block can be shared over all steps. The following figure 5 shows a feature transformer with two shared and two decision step individual layers.

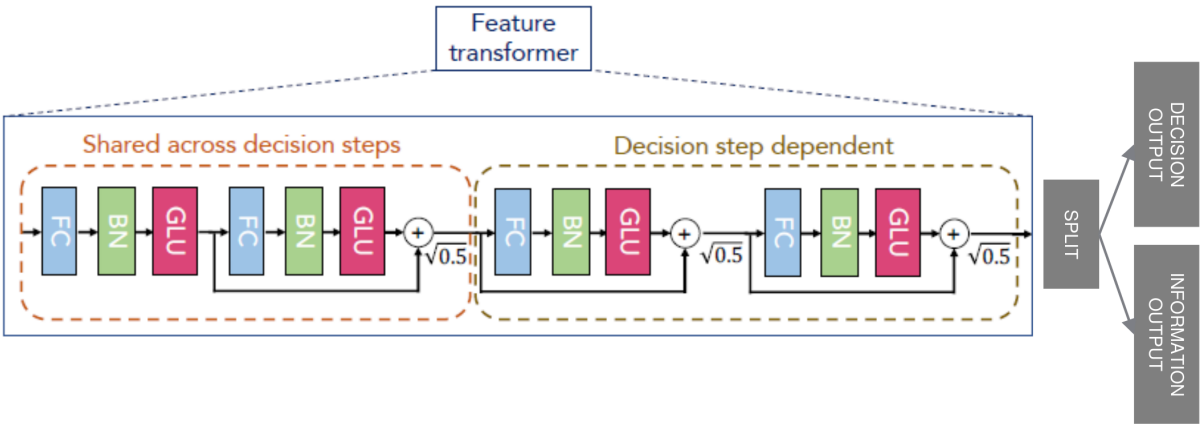


Figure 5: Feature Transformer [7]

The final output of feature transformer $\mathbf{h} \in \mathbb{R}^H$ is split into two parts, the decision step output $\mathbf{d} \in \mathbb{R}^{N_d}$ and the information which is used in subsequent steps $\mathbf{a} \in \mathbb{R}^{N_a}$. The dimensions add up to the hidden dimension in the form of $H = N_d + N_a$.

2.2.2 Attentive Transformer

The output of the previous decision step $\mathbf{a}^{[l-1]}$ is linear transformed followed by ghost BN, refer to section 2.2.1, in the form:

$$FC(\mathbf{a}^{[l-1]}) = BN(\mathbf{V}^{[l]}\mathbf{a}^{[l-1]}) \tag{2.2.6}$$

$\mathbf{V}^{[l]}$ describes the learnable parameters within the attentive transformer which subsequent determine which features are selected. The output of $FC(\mathbf{a}^{[l-1]})$ is multiplied with a prior scale term $\mathbf{P}^{[l-1]}$ which indicates what features have been used in previous decision steps. Finally the non-linear transformation $sparsemax$ is applied as follows which results in the feature selection mask $M^{[l]}$:

$$\mathbf{M}^{[l]} = sparsemax(\mathbf{P}^{[l-1]} \odot FC(\mathbf{a}^{[l-1]})) \tag{2.2.7}$$

The output of $sparsemax$ is a probability distribution with the property $\sum_{j=1}^{D} M_{b,j}^{[l]} = 1$, refer to section 2.2.3. Additionally, $sparsemax$ can output sparse probability meaning some entries $j \in D | M_{b,j}^{[l]} = 0$.

Given the relaxation parameter γ $\mathbf{P}^{[l]}$ is given as:

$$\mathbf{P}^{[l]} = \prod_{j=1}^{l} (\gamma - \mathbf{M}^{[j]}) \tag{2.2.8}$$

The relaxation parameter γ indicates if a feature is more likely used in more than one decision step. Starting from $\gamma = 1$ more flexibility in reusing features over multiple decision steps is provided as its value increases.

To further control sparsity TabNet propose to add a sparsity regularization term in form of entropy to the overall loss, with ϵ for numerical stability:

$$L_{sparse} = \sum_{l=1}^{N_{steps}} \sum_{b=1}^{B} \sum_{j=1}^{D} \frac{-M_{b,j}^{[l]} log(M_{b,j}^{[l]} + \epsilon)}{N_{steps} \cdot B} \tag{2.2.9}$$

The effect of sparsity regularization can be controlled with λ_{sparse} and is used in the overall loss term:

$$L = L + \lambda_{sparse} L_{sparse} \tag{2.2.10}$$

High values for $\lambda_{sparse} \uparrow$ encourage more sparsity of $\mathbf{M} \uparrow \downarrow$ while values approaching 0 for $lambda_{sparse} \downarrow$ result in less sparsity.

2.2.3 Sparsemax

Sparsemax activation function was introduced by Martins A. and Astudillo R. in 2016 [70]. Similar to $softmax$ the output follows a probability distribution. Given the input vector $\mathbf{z} \in \mathbb{R}^K$ first recap $softmax$ which is defined for $i = 1, \ldots, K$ as:

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.2.11}$$

As already mentioned the output $\mathbf{o} \in \mathbb{R}^K$ of $softmax(\mathbf{z})$ follows a probability distribution having the property $\sum_{i=1}^{K} o_i = 1$. As $softmax$ is fully differentiable the gradient of $softmax(\mathbf{z})_i$ w.r.t. one particular entry z_j is given as:

$$\frac{\partial softmax(\mathbf{z})_i}{\partial z_j} = softmax(\mathbf{z})_i (\delta_{ij} - softmax(\mathbf{z})_j) \tag{2.2.12}$$

δ_{ij} refers to the Kronecker delta which is defined as 1 if two variables i and j are equal and 0 otherwise:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.2.13}$$

The full Jacobian $J_{softmax(\mathbf{z})} \in \mathbb{R}^{K \times K}$ using the defined output $\mathbf{o} = softmax(\mathbf{z})$ is given as follows:

$$J_{softmax(\mathbf{z})} = diag(\mathbf{o}) - \mathbf{o}\mathbf{o}^T \tag{2.2.14}$$

In contrast to $softmax$ which has full support meaning $softmax(\mathbf{z})_i \neq 0$, for $sparsemax$ can output a sparse probability distribution and is defined as follows:

$$sparsemax(\mathbf{z}) = \underset{\mathbf{p}\in\Delta^{K-1}}{argmin}\|\mathbf{p}-\mathbf{z}\|^2 \tag{2.2.15}$$

This describes the Euclidean projection of the input $\mathbf{z}$ onto the probability simplex. A probability simplex refers to the geometric representation of a probability distribution in the form of a polytope, meaning an object having flat sides. The simplex Δ^{K-1} refers to $K-1$ dimensional object having K corners or vertices, e.g. Δ^2 refers to a line, Δ^3 refers to triangle and so on. For a probability simplex each corner represents the probability of a specific event or case $k \in K$.

For equation 2.2.15 closed form solution exists. One simple solution is described in algorithm 1.

Algorithm 1 Sparsemax closed form solution [70]

Require: Given input vector $\mathbf{z} \in \mathbb{R}^K$
Require: Define vector $[K] = [1, 2, \cdots, K]$
1: Sort $\mathbf{z}$ in in descending order: $\mathbf{z}_s = [z_{(1)} \geq \cdots \geq z_{(K)}]$
2: Find max index $k \in [K]$ for which $1 + k \cdot z_{s(k)}$ is still greater than the cumulative sum of $\mathbf{z}_s$ until index k, this can be expressed as function $k(\mathbf{z}_s)$:
$$k(\mathbf{z}_s) = max\Big\{k \in [K] | 1 + k \cdot z_{s(k)} > \textstyle\sum_{j\leq k} z_{s(j)}\Big\}$$
3: Compute the threshold $\tau(\mathbf{z})$ as the cumulative sum of $\mathbf{z}_s$ until the max support index $k(\mathbf{z}_s)$ minus 1 divided by the max index $k(\mathbf{z}_s)$:
$$\tau(\mathbf{z}) = \frac{(\sum_{j\leq k(\mathbf{z}_s)} z_j) - 1}{k(\mathbf{z}_s)}$$
4: **return** $max(0, z_i - \tau(\mathbf{z}))$

Furthermore the support S of $sparsemax$ is defined as all indices, for which the return value is above 0:

$$S(\mathbf{z}) = [i \in [K] | sparsemax(\mathbf{z})_i > 0] \tag{2.2.16}$$

Sparsemax is differentiable everywhere except at splitting points, similar to the activation functions ReLU which is not differentiable when evaluated at 0. Therefore the gradient of $sparsemax(\mathbf{z})_i$ w.r.t. z_j following [70] is defined as:

$$\frac{\partial sparsemax(\mathbf{z})_i}{\partial z_j} = \begin{cases} \delta_{ij} - \frac{\partial \tau(\mathbf{z})}{\partial z_j}, & \text{if } z_i > \tau(\mathbf{z}) \\ 0, & \text{otherwise} \end{cases} \tag{2.2.17}$$

$$\frac{\partial \tau(\mathbf{z})}{\partial z_j} = \begin{cases} \frac{1}{|S(\mathbf{z})|}, & \text{if } j \in S(\mathbf{z}) \\ 0, & \text{otherwise} \end{cases} \tag{2.2.18}$$

Given $z_j > \tau(\mathbf{z})$ if $j \in S(\mathbf{z})$:

$$\frac{\partial sparsemax(\mathbf{z})_i}{\partial z_j} = \begin{cases} \delta_{ij} - \frac{1}{|S(\mathbf{z})|}, & \text{if } i, j \in S(\mathbf{z}) \\ 0, & \text{otherwise} \end{cases} \tag{2.2.19}$$

The Jacobian $J_{sparsemax(\mathbf{z})} \in \mathbb{R}^{K\times K}$ using the the support function $S(\mathbf{z})$ in form of $\mathbf{s} = [1$ if $i \in S(\mathbf{z})$ otherwise 0 for all $i \in K]$ can be calculated as:

$$J_{sparsemax(\mathbf{z})} = diag(\mathbf{s}) - \frac{\mathbf{s}\mathbf{s}^T}{|S(\mathbf{z})|} \tag{2.2.20}$$

In the two-dimensional case softmax and sparsemax collapse to sigmoid and hard-sigmoid respectively. This can be seen in figure 6 presenting $softmax([t,0])_1$ and $sparsemax([t,0])_1$ in blue, plus their derivations $\frac{\partial softmax([t,0])_1}{\partial t}$ and $\frac{\partial sparsemax([t,0])_1}{\partial t}$ in orange.

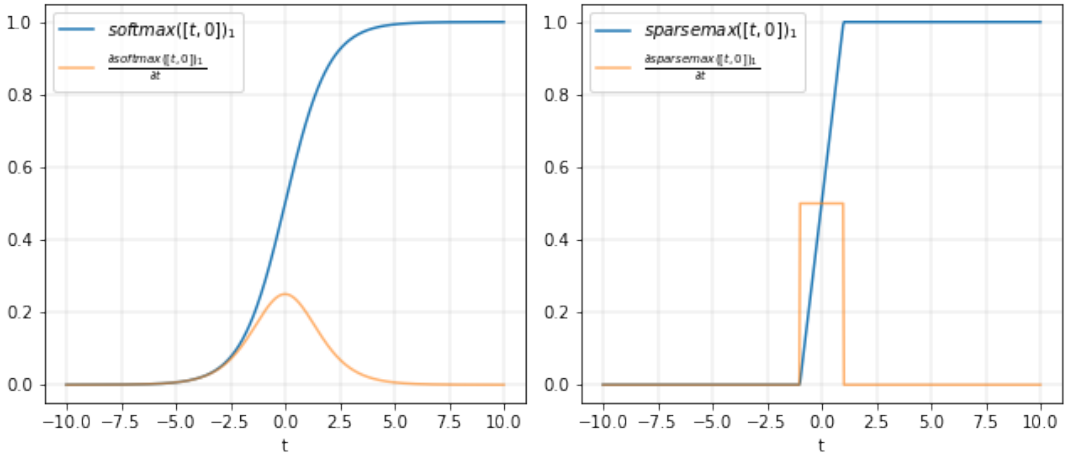


Figure 6: Softmax and sparsemax

Further work by Peters B. et al. [81] pointed out that softmax and sparsemax both can be expressed as a projection of an input vector $\mathbf{z} \in \mathbb{R}^K$ on a probability simplex which only differ in the choice of entropy regularization.

$$\begin{aligned} softmax(\mathbf{z}) &= \underset{\mathbf{p}\in\Delta^d}{argmax}(\mathbf{p}^T\mathbf{z} + H^S(\mathbf{p})) \\ sparsemax(\mathbf{z}) &= \underset{\mathbf{p}\in\Delta^d}{argmax}(\mathbf{p}^T\mathbf{z} + H^G(\mathbf{p})) \end{aligned} \tag{2.2.21}$$

Softmax is using Shannon entropy $H^S(\mathbf{p}) = -\sum_j p_j log(p_j)$ while sparsemax uses Gini entropy $H^G(\mathbf{p}) = \frac{1}{2}\sum_j p_j(1-p_j)$. Using a generalized form of entropy called Tsallis α entropy which is defined as:

$$H^T_\alpha(\mathbf{p}) = \begin{cases} \frac{1}{\alpha(\alpha-1)}\sum_j(p_j - p_j^\alpha), & \alpha \neq 1 \\ H^S(\mathbf{p}), & \alpha = 1 \end{cases} \tag{2.2.22}$$

Using Tsallis entropy a generalized or interpolated version of softmax and sparsemax called $\alpha - entmax$ can be defined:

$$\alpha - entmax(\mathbf{z}) = \underset{\mathbf{p}\in\Delta^d}{argmax}(\mathbf{p}^T\mathbf{z} + H^T_\alpha(\mathbf{p})) \tag{2.2.23}$$

Depending on α outputs are more $\alpha \uparrow$ or less sparse $\alpha \downarrow$. $\alpha - entmax$ is differentiable also w.r.t. α itself. Even though $\alpha - entmax$ was not applied within TabNet it is mentioned here as a generalization of sparsemax. Furthermore in figure 7 the two dimensional case $[t, 0]$ with different values for α with the corresponding gradients w.r.t. t is shown.

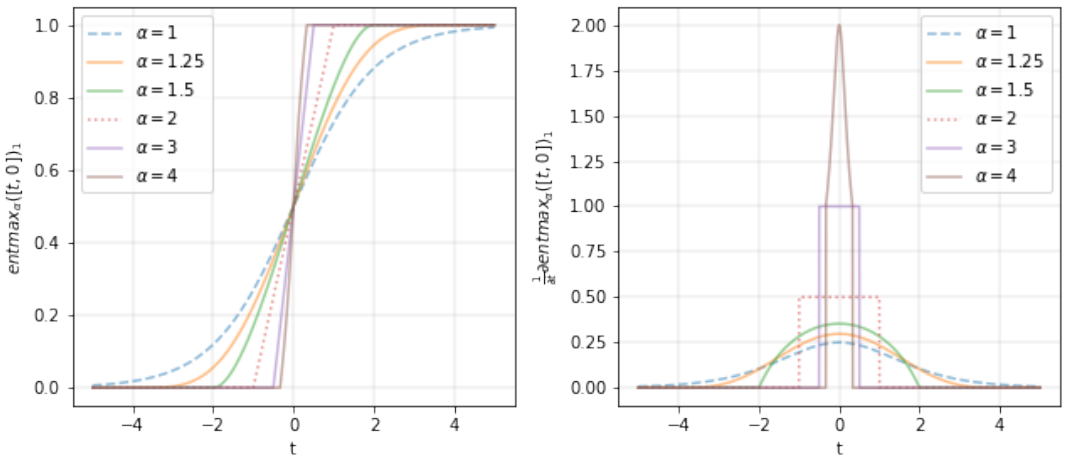


Figure 7: $\alpha - entmax$ evaluation and gradients

2.2.4 Interpretability

Before any input is processed within a decision step l, the input is multiplied with the sparse probability feature selection mask $\mathbf{M}^{[l]}$ as described in section 2.2.2. Therefore TabNet by design explains which features are used to determine the output or prediction and which are not. As the overall TabNet output is a linear combination of the individual decision steps outputs as shown in 2.2.1 the overall aggregated feature mask must also take into account the individual decision step contribution. Therefore for one concrete sample b a coefficient term $\eta_b^{[l]}$ is calculated as follows:

$$\eta_b^{[l]} = \sum_{i=1}^{N_d} ReLU(d_{b,i}^{[l]}) \tag{2.2.24}$$

This coefficient or scale term $\eta_b^{[l]}$ determines the importance or contribution of one particular decision step l and is used to scale the impact of the corresponding feature selection mask $\mathbf{M}^{[l]}$ accordingly. Using this scale term the overall feature importance or contribution mask $M_{agg,b}$ is given as follows:

$$\mathbf{M}_{agg,b} = \sum_{i=1}^{N_d} \eta_b^{[l]} \mathbf{M}^{[l]} \tag{2.2.25}$$

Beside the overall feature selection and attribution using $\mathbf{M_{agg}}$, TabNet allows insight which features together have been used in an individual step by just looking at individual feature selection masks $\mathbf{M}^{[l]}$. This design allows for a potential better interpretability of interactions of features within the TabNet architecture. This is additionally presented in the following figure 8.

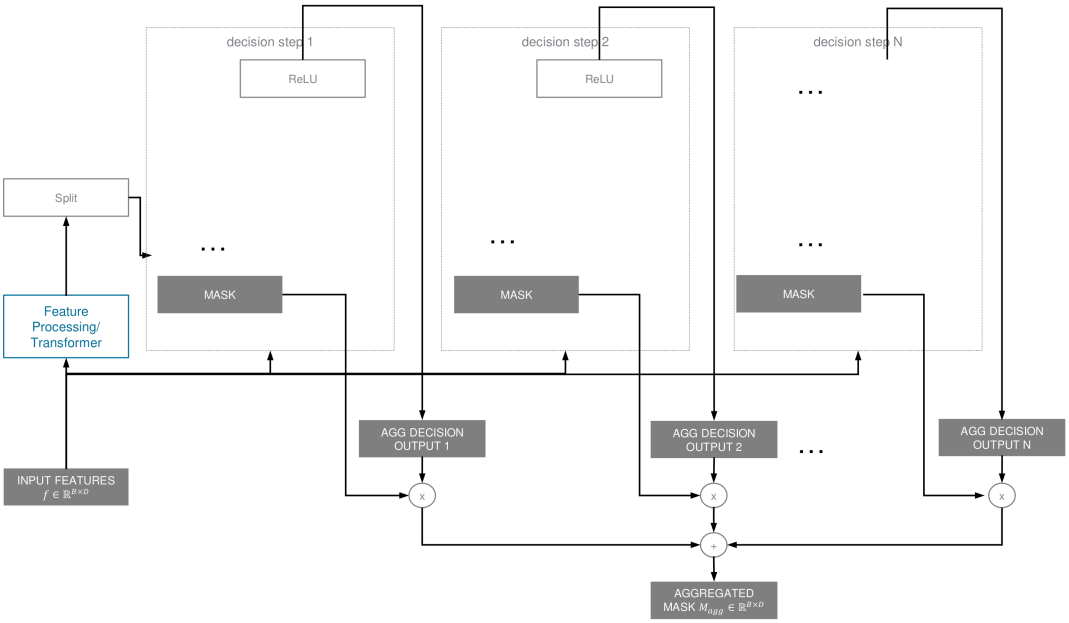


Figure 8: TabNet interpretability overview

Even though TabNet by design returns the selected features and their contribution to achieve a defined objective, it does not give an insight into the feature selection process itself. The question why a particular feature was not selected, e.g. because of the presence or absence of another feature can not be answered by TabNet directly.

2.3 Machine Learning Interpretability Methods

Beside the concrete result of a machine learning method, for humans to interpret the outcome, or get a more in depth explanation why a particular result was achieved is crucial. Especially for tasks in high stake domains like healthcare, medical research, security applications and many others. Recent developments in machine learning and in deep learning lead to complex models with billions of parameters which in many cases achieved state-of-the-art results but are also difficult or not at all to interpret [64]. To tackle this and other problems the research stream of explainable artificial intelligence (XAI) become more important in recent years. Even though there exists attempts [78], [10] to define XAI and terms such as "explainability" and "interpretability" a common agreed upon definition does not exists and those terms often are used interchangeably [68]. Nonetheless agreed upon goals and concepts of XAI can be found. Following the DARPA (US Defense Advanced Research Projects Agency) XAI should "produce more explainable models while maintaining high level of learning performance and enable human users to understand, appropriately trust, and effectively manage the emerging generation of artificially intelligent partners" [29]. When looking at concepts and problems which are subject of research within XAI the following can be found [65].

- **Trust** refers to an idea that results of a model must be trustworthy. This is usually measured using the raw performance or accuracy numbers of a model. But trust is also important in context of bias or imbalance within training data and how this problem is being addressed. Furthermore, beside the raw performance, it is also important to know for which examples or in which situation a model is more prone to errors.
- **Causality** describes the concept to infer causal relationships between input data from a model which function as the basis for further research hypothesis.
- **Informativeness** of a model is given if a human is able to extract additional information beside the raw output or prediction from a model.
- **Justification** is based on the idea of fair and ethical decision making. In the context of regulation for example the European Union stresses that individual effected by algorithm decisions have the right for explanation. The question why a prediction or output was generated by a model must be explained, based on a falsifiable proposition which provides a way to contest those proposition.
- **Transparency** of how a model outcome or prediction was determined. It refers to the problem that many machine learning models don't offer interpretability or explainability by design but function as a black-box which uses input data to determine an output or prediction. Without further tools or method transparency of the internal functionality is not given.

Enabling XAI and especially supporting transparency requires concrete interpretability methods and concepts. Therefore to further categorize interpretability methods the taxonomy shown in figure 9 is presented.

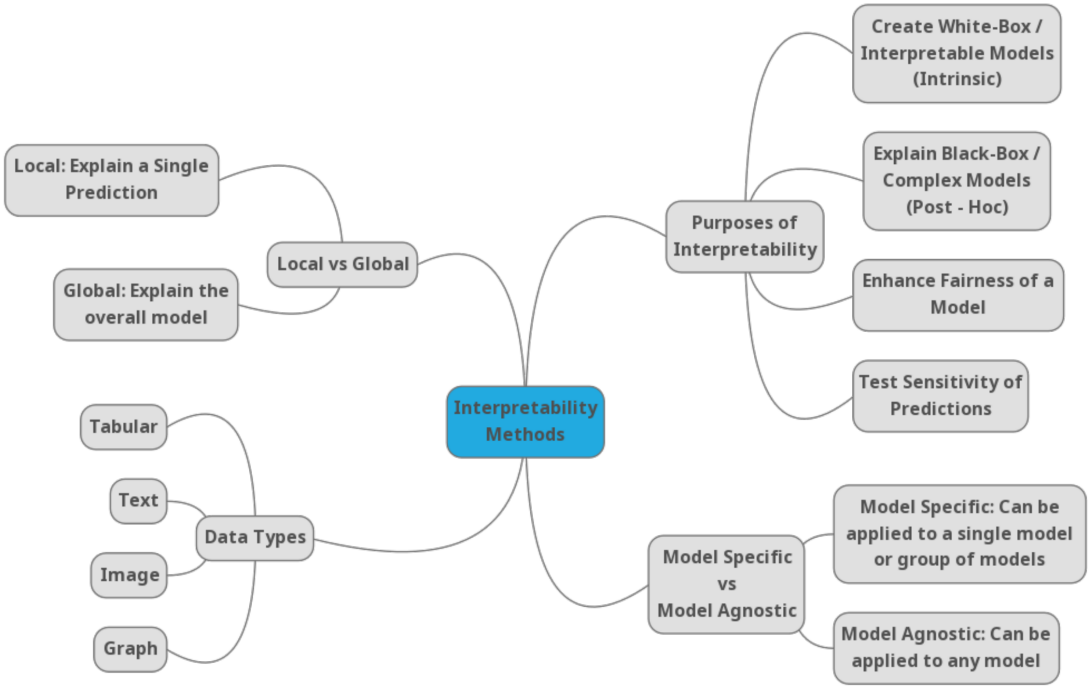


Figure 9: Taxonomy of interpretability methods [64]

Focusing on the categories "Local vs Global", "Model Specific vs Model Agnostic" and "Post-hoc Methods" the following sub chapters outline some recent implementations of interpretability methods focusing on deep learning architectures. A common task for interpretability methods is given a model $f \mid \mathbf{x} \rightarrow \mathbf{y}$ mapping from $\mathbf{x} \in \mathbb{R}^D$ to $\mathbf{y} \in \mathbb{R}^K$ find the feature attribution $\mathbf{a} \in \mathbb{R}^K$ using interpretability method $A \mid \mathbf{x} \rightarrow \mathbf{a}$ such that for each feature $[x_1, \ldots, x_D]$ there exists a attribution value $[a_1, \ldots, a_D]$ describing the contribution of each feature to the model output or prediction $\mathbf{y}$. This task is also referred to as feature attribution.

2.3.1 Integrated Gradients

Integrated gradients (IG) offers local feature attribution and is specific to models which are trained using gradient descent. Original introduced in 2017 [101] it became one of the most prominent feature attribution method. IG requires the definition of a baseline x' which is used to generate interpolations between $\mathbf{x}'$ and the actual input data $\mathbf{x}$. Using the interpolations the gradients of a model F w.r.t. $\mathbf{x}$ are calculated and aggregated. The following equation describes the calculation of IG for one particular feature dimension d within $\mathbf{x} \in \mathbb{R}^D$:

$$IG_d(\mathbf{x}) = (x_d - x'_d) \int_{\alpha=0}^{1} \frac{\partial F(\mathbf{x}' + \alpha(\mathbf{x} - \mathbf{x}'))}{\partial x_d} d\alpha \tag{2.3.1}$$

Due to difficulties calculation the integral for numerical reasons often an approximation using Riemann sum with m steps is used:

$$IG_d^{approx}(\mathbf{x}) = (x_d - x'_d) \sum_{k=1}^{m} \frac{\partial F(\mathbf{x}' + \frac{k}{m}(\mathbf{x} - \mathbf{x}'))}{\partial x_d} \frac{1}{m} \tag{2.3.2}$$

IG results in local or instance wise feature attribution values. Global feature attribution for a model can not be determined. Additionally relations or interactions between features can not be explained.

2.3.2 Saliency

The concept of saliency for feature attribution was first introduces in 2014 [97] in the context of computer vision to provide saliency maps of pixels contributing the most to a given prediction. Saliency provides local feature attribution values for gradient based methods. The idea is to use the gradient $\mathbf{w}$ of the model f_c for a concrete output or class c w.r.t. its input $\mathbf{x} \in \mathbb{R}^D$ to approximate the function output f_c by using a first-order Taylor expansion:

$$f_c(\mathbf{x}) \approx \mathbf{w}^T\mathbf{x} + b \tag{2.3.3}$$

The gradient $\mathbf{w} \in \mathbb{R}^D$ is given as:

$$\mathbf{w} = \frac{\partial f_c}{\partial \mathbf{x}} \tag{2.3.4}$$

The feature attribution values are given by either using the absolute gradient values $|\mathbf{w}|$, or the gradients $\mathbf{w}$ directly (relative). Saliency is an easy to compute feature attribution method but similar to IG does not describe relationships or interactions between features. Additionally, it suffers from the problem of saturated gradients which is relevant for models using ReLU activation [94].

2.3.3 Shapley Values

The original concept was introduced in 1951 by Lloyd Shapley [92] and tried to solve the problem of a fair and equal payment schema between participants who together produces a certain value or outcome. Similarly within a machine learning model the input data consists of multiple participants or feature dimensions which together produce an outcome or prediction. Therefore Shapley values can be used to identify the individual feature attribution or contribution to the overall model outcome. It produces local or instance wise feature attribution but in contrast to gradient based method is model agnostic. Given input data $\mathbf{x} \in \mathbb{R}^D$ having D dimensions which contribute to the outcome of a model f the Shapley value for the i feature or dimension can be written down as:

$$\psi_i(f) = \frac{1}{D} \sum_{\mathbf{s} \subseteq \mathbf{x} \setminus \{x_i\}} \binom{D-1}{|\mathbf{s}|}^{-1} \Big(f(\mathbf{s} \cup x_i) - f(\mathbf{s}) \Big) \tag{2.3.5}$$

The term $\mathbf{s} \subseteq \mathbf{x} \setminus \{x_i\}$ refers to a subset of features without the feature x_i for which the Shapley value is calculated. As the input to the model f is of dimension D the vector representing the subset $\mathbf{s} \in \mathbb{R}^D$ is also of the same size. In practice a baseline value (e.g. zero) is used replacing entries not present in the corresponding subset $\mathbf{s}$. For example given $\mathbf{x} = [x_1, x_2, x_3]$ there are 4 possible subsets excluding x_1 using zero as a baseline value, $\mathbf{s}_0 = [0, 0, 0]$, $\mathbf{s}_1 = [0, x_2, 0]$, $\mathbf{s}_2 = [0, 0, x_3]$ and $\mathbf{s}_3 = [0, x_2, x_3]$.

The marginal value or contribution when adding x_i is given by $f(\mathbf{s} \cup x_i) - f(\mathbf{s})$. It calculates the output for a given subset $\mathbf{s}$ including feature x_i minus the output using

just the subset $\mathbf{s}$. The term $\binom{D-1}{|\mathbf{s}|}^{-1}$ uses the number of combinations for the amount of features in $|\mathbf{s}|$ to scale the marginal contribution of x_i. For the mentioned example this results in the individual marginal values of $1 \cdot f([x_1, 0, 0]) - f([0, 0, 0])$, $\frac{1}{2} \cdot f([x_1, x_2, 0]) - f([0, x_2, 0])$, $\frac{1}{2} \cdot f([x_1, 0, x_3]) - f([0, 0, x_3])$, $1 \cdot f([x_1, x_2, x_3]) - f([0, x_2, x_3])$. Finally the individual marginal values are aggregated and scaled by $\frac{1}{D}$.

Given D feature dimension calculating the Shapley values requires 2^D subsets which needs to be considered. For many real world applications this is computational not feasible. In practice Shapley values are estimated using various techniques. One simple one is to only consider a sample of available subsets for calculations, this is also referred to as Shapley value sampling [17].

2.4 Drug Discovery

Development of new drugs is a complex and costly process having R&D costs of 2.6 billion USD per newly developed drug. The whole process until approval takes about 13.5 years, with 5.5 years before any clinical trails, also referred to as drug discovery process and 8 years for the remaining preclinical, clinical and approval phases [49].

Development of a new drug for a a selected disease starts with identification and validation of a potential target candidates. Based on the exploratory research results target candidates are tested in large screening tests to identify possible molecules which have desired affinity with the target. Those molecules referred to as HIT molecules are further analyzed to identify a lead compound which modify bind to targets in specific and selective way and modify its normal mechanism of action. The lead compound is further modified to improve its general biological activity as well as absorption, distribution, metabolism and excretion (ADME) behavior. If a promising lead compound could be successfully identified and modified pre-clinical and clinical trials phases starts [26]. In the following figure 10 an overview of the process is presented.

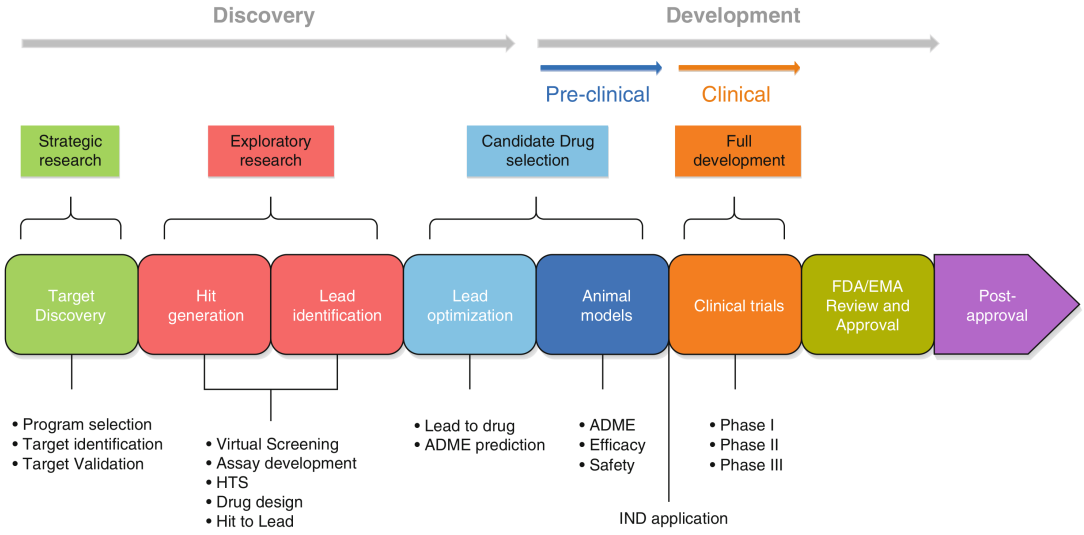


Figure 10: Drug development and discovery process overview [26]

During the drug discovery process various assays are applied which can be categorized by the type or environment the are carried out:

1. **In-vitro** ("in glass") - refers to an assay performed in wet lab in which in the context of drug discovery a molecule or ligand is tested for a desired biological effect onto a target.

2. **In-vivo** ("in the living") - describes an assay performed on living organisms (e.g. plants, animals).

3. **In-silico** - describes assays performed by computational simulations.

Especially in-silico assay helps to reduce the quantity of molecules which are candidates for further test in-vitro assays and therefore reduce the costs of drug discovery and development. This leads to an every increasing number of companies developing and applying ML and AI methods within the drug discovery process[26]. As an example in the following figure 11 only for HIT identification process within drug discovery an overview is presented. Especially on quantitative structure-activity relationship (QSAR) models, based on the assumption that similar structure have an similar effect on a target, ML and AI methods are often applied upon.

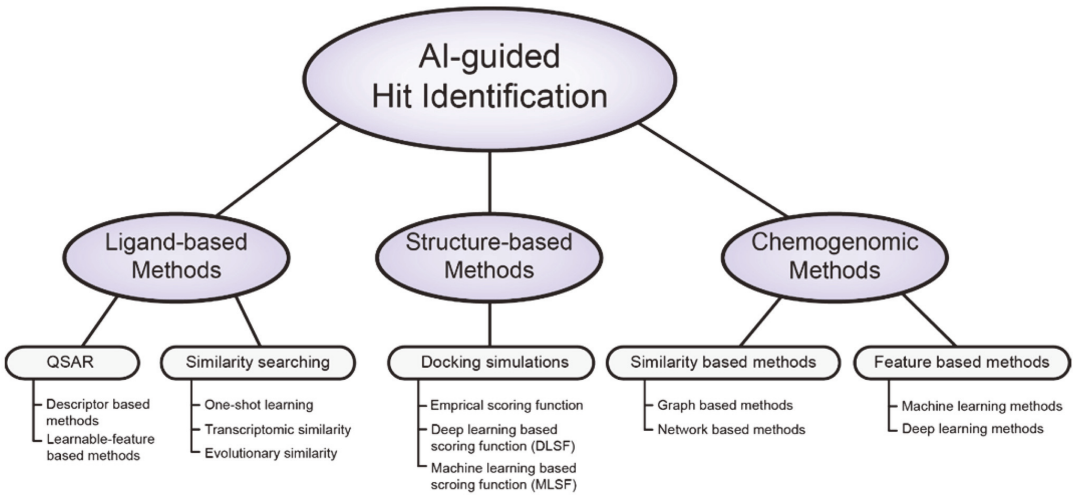


Figure 11: Drug discovery ML powered HIT identification methods overview [49]

Datasets and task related to QSAR applications are the main bases for experiments of this work and further described in section 3.

2.4.1 Quantitative Structure-Activity Relationship

One foundation of quantitative structure-activity relationship (QSAR) models is a study by Hansch et al.[37] in 1962 in which over 15 years the structure-activity relationship of growth regulators in plants have been analyzed finding out a suitable relationship which in the end turned out to be a dependency between the Hammett constant and hydrophobicity. Within QSAR applications the typical setup refers to a set of molecules for which a measurable biological property is known. Such a measurement usually refer to some potency value of a molecule having a corresponding effect and are typically originally measured in-vitro [20]. One widely used measurement is the half-maximum inhibitory concentration (IC_{50}). IC_{50} describes the potential of an substance to inhibit a biological activity by 50%. IC_{50} is often given as $-log_{10}(IC_{50})$ in mol/L and referred to

as pIC_{50}. Higher pIC_{50} values therefore indicate a more potent inhibitor [43]. Formally defined a QSAR model describes the activity or target a of a given compound or molecule M in the form of a function f:

$$a = f(M) \tag{2.4.1}$$

Depending on the task or target a such a QSAR model solves a regression (e.g. relating to the quantitative potency values) or as a classification problem (e.g. qualitative binary problem being active/inactive, toxic/non-toxic etc.) [20]. Given the nature of the problem, modeling QSAR using ML has shown great success and become a obvious choice in many cases [67].

2.4.2 Molecule Descriptors and Fingerprints

One key component for in silico drug discovery and QSAR modeling is how molecules and components are represented for computational processing. For this common descriptors and molecule fingerprints are introduced:

- **Simplified Molecular Input Line Entry Specification (SMILE)** - from theory point of view not a molecular descriptor or fingerprint but a common representation for molecules in a sequence of characters. Originally introduced in the 1980s [104] further development lead to the OpenSMILES standard in 2007. Their are many rules how to represent a molecule and described in the official specification [77]. An example for a SMILE representation of Aspirin is "O=C(C)Oc1ccccc1C(=O)O", this is also referred to as the SMILE string representation.
- **0D descriptors** - do not account for structural information within a molecule. Examples are number of atoms, number of bonds or other atomic properties [16]
- **1D descriptors** - usually describe information from factions or part of a molecule. Based on chemical functional groups properties examples are number of primary carbons, number salts or cyanante containing etc. [16].
- **2D descriptors** - are based on a 2D representation, usually in graph form, of a molecule. The structural topology is used like the adjacency and distance between atoms. Additionally topochemical information like properties of atoms (e.g. hybridization) can be incorporated [16].
- **3D descriptor** - are using the 3D representation of a molecule, like distance of bonds or angels but also conformer of a molecule are considered. It is a more complex process and requires more computational resources [16].
- **4D descriptor** - beside using the 3D representation also the interactions with other molecules in the 4th dimension can be considered [16].
- **Extended Connectivity Fingerprints (ECFP)** - is a widely used molecule representation which results in a fixed size fingerprint of a particular molecule. Creating a ECFP representation is an iterative process:

1. Each atom within a molecule a 32 bit integer is assigned. The integer is determined based on the Daylight atomic invariant rule. This rule is invariant regarding the number an atom is assigned within a molecule and consists of six properties: Number of nearest-neighbors (non-hydrogen), number of bonds (non-hydrogen), atomic number, atomic mass, atomic charge, bond order ignoring (ignoring hydrogen bonds) and number of connected hydrogen. Optional the property of an atom being part(1) of a ring or not(0) is added as a seventh property [96]. Determining those 6 properties results in an array is formed which is hashed resulting in a single 32 bit integer for given atom. In the following figure 12 for a sample molecule butyramide the individual atomic identifiers are presented.

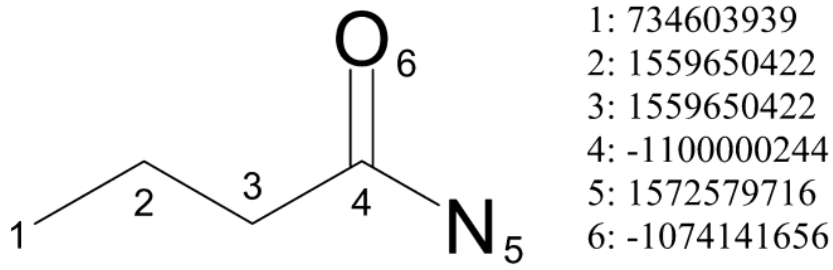


Figure 12: Atom identifier for butyramide using Daylight atomic invariants rule [87]

2. In the next step neighbor atom information are incorporated. For this an array is created starting with the center atom. For example considering atom 4 with identifier "-1100000244" in figure 12. All neighboring atoms with their integer identifiers are added including information about the bond type (1 single, 2 double, 3 triple and 4 for aromatic bonds) resulting in [1, -1100000244, 1, 1559650422, 1, 1572579716, 2, -1074141656]. This array again is hashed and getting a new integer identifier "-1708545601" which is added to the overall list of identifiers. After the first iteration this results for the mentioned example in the list [734603939, 1559650422, 1559650422, -1100000244, 1572579716, -1074141656, 863188371, -1793471910, -1789102870, 1708545601, -932108170, 2099970318] [87].

   This process can be repeated multiple times resulting in a larger radius around an atom considered, and is a hyperparameter for the ECFP molecule representation method.

3. To get a fixed size numerical representation, often desired and used as input to a corresponding QSAR model, a modulo operator (remainder of a division) is applied onto the individual identifiers. Using 1024 for the value of the modulo operator this would result in $[675, 118, 118, 12, 388, 552, 403, 602, 234, 447, 118, 270]$ for mentioned identifiers of butyramide. The fixed size representation is achieved by creating a binary vector of length 1024 and assigned 1 for all indices found in the modulo operator. Depending on the value of the modulo operator this lead to a high sparsity in the representation (if the value is large) or more bit collisions. Bit collisions appear if multiple identifiers end up assigning 1 to the same index. In the above mentioned this happened for the index 118 which

was assigned thrice. The process of creating a fixed size representation is also referred to as folding [87].

- **Molecular ACCess System (MACCS keys)** - refer to set of pattern in a molecule which must be present to define a fixed length binary vector. The set consists of 166 public available patterns resulting in a fixed size binary vector of length 166 representing a molecule. MACCS keys were original developed by MDL Information Systems and include 960 patterns but only a set of 166 patterns were made publicly available [27]

  Those patterns are described in the form of SMILE arbitrary target specification (SMARTS) pattern. As the name suggests it is related to SMILE representation and allows flexibility and precise substructure specification including wildcard matching. For the detailed specification refer to official specification [23].

3 Datasets

In this chapter for this master thesis relevant datasets are introduced. All datasets, except the first one which is used as reference to verify implementation details, belong to the domain of drug discovery.

3.1 Covertype

The dataset consist of 581012 samples describing forest cover types within a national park in Colorado US. The dataset consists of 54 features with 10 of them being numerical and 44 binary. The setting is a multi-class classification task for 7 different forest cover types. The dataset was collected and created in 1998 [12] and is freely available within the University of California Irvine (UCI) dataset collection [103].

The dataset is used within this master thesis to verify the implementation of TabNet based on the achieved performance before any subsequent experiments. The forest cover type dataset was used within the TabNet paper [7] as one dataset for which the architecture achieved better performance than other techniques such as GBDT or DNN.

Additionally Google published a repository exactly and only for this dataset testing TabNet [36] including the splitting process into training, validation, and testing part as well as random seeds used. The same splits (54% training, 26% validation, 20% testing) were used within this master thesis.

3.2 Blood-Brain-Barrier Penetration Dataset

The dataset consists of about 2000 molecules for each of which the characteristic whether the individual molecule can penetrate the blood-brain-barrier (BBB) or not is known. The BBB describes a membrane that separates circulating blood from extracellular fluid within the brain to protect the brain from intruding substances. Large molecules within drugs can not pass it and only about 2% of small molecules can. Therefore in case the passing of the barrier is desired within drug discovery finding such molecules which are able to penetrate those membrane is difficult. On the other hand for toxicology of newly developed drugs it is critical to test for penetration of the BBB [71].

3.3 Beta-Secretase 1 Dataset

The Beta-secretase 1 (BACE) dataset consists of molecules with binding results, in quantitative form for regression and in binary form for classification, for the human Beta-secretase 1 (BACE-1). The Beta-secretase 1 is an enzyme in humans, encoded in the BACE1 gene, which is mainly expressed within neuron cells. This enzyme plays a major role in the generation of amyloid-β peptides in neurons and are the main components in amyloid plaques which are found in human brains with Alzheimer disease (AD). Following the amyloid hypothesis, finding drugs which are BACE inhibitors in theory may prevent or at-least slow down AD [83].

The BACE dataset consists of about 1500 molecules represented as SMILE strings with binary classification whether being a BACE inhibitor (1) or not (0). Additionally the quantitative inhibitor property represented as IC_{50} value and the 2D structures of

molecules are available [100]. In this master thesis only the qualitative inhibit property is used in a binary classification setting.

The dataset is publicly available within the MoleculeNet dataset collection [105].

3.4 Human Immunodeficiency Viruses Dataset

The human immunodeficiency viruses (HIV) datasets consists of more than 40000 molecules describing their ability to inhibit HIV replication. HIV causes acquired immunodeficiency syndrome (AIDS) which causes failure of the immune system and per average leads to death within 9 to 11 years[39]. The dataset was introduced by the drug therapeutics program for AIDS antiviral screening [3].

The original HIV dataset consists of 3 categories being inactivity (not inhibiting), activity and moderate activity (inhibiting). The later two categories are combined leading to a binary classification task. Molecules are represented as smile strings. The dataset is publicly available within the MoleculeNet dataset collection [105].

3.5 Side Effect Resource Dataset

The side effect resource dataset (SIDER) consists of developed drugs available on the market with reported adverse drug reactions (ADR) during clinic trials. The datasets consists of over 1400 marketed drugs represented as smile strings with their reported side effect[61]. Beside the prepared dataset the side effect resource database exists[1].

The dataset publicly available within MoleculeNet [105] reports 27 categories for side effect represented as binary targets which lead to a multi target binary classification task. Per category 1 indicates that the side effect has been reported for corresponding drug and 0 if not has been reported.

3.6 Human Ether-à-go-go-Related Gene Dataset

The human ether-a-go-go related gene (hERG) dataset encodes a protein $K_v11.1$ relevant for the potassium ion channel within the heart. This channel mediates the electrical current flow of ions and helps regulating heart beats. When the ability of this channel to conduct electrical current is inhibited or effected this can lead to fatal disorder of the heart and subsequent to death.

Therefore within drug discovery molecules are regularly tested for their hERG characteristic. The dataset used within this master thesis consists of about 8000 molecules represented as SMILES string. For each molecule the hERG activity, 1 for blocking, 0 for non-blocking/decoy molecule and not further defined (n/d) is given. hERG active molecules are thresholded having $IC_{50} \leq 10\mu M$. The non-blocking or decoy molecules, molecules which are randomly extracted from chemical databases but resemble similarities with known hERG active molecules, are split using different IC_{50} thresholds. This results into 6 targets for non-blocking/decoy (0) if IC_{50} above $10, 20, 40, 60, 80, 100\ \mu M$ respectively and not defined (n/d) if below but above the active threshold $(IC_{50} > 10)\mu M$. This results into a multi binary target classification setting for which not every sample is fully defined for all targets (0, 1, n/d). In this work only the first target is used which in fact is fully defined (0 or 1) for all samples. The dataset is publicly available within the support

[1] http://sideeffects.embl.de/

information of the introducing paper [15].

Furthermore recent work [21] identified common motifs within hERG active and inactive molecules using maximum common substructures analysis on publicly available hERG data. Using various source data on hERG activity, which were collected using functional and binding assays and with two different thresholds ($1\mu M$, $10\mu M$) for IC_{50} binding affinity, 15 unique relevant substructures could be identified. Only complete ring structures and motifs with at-least 5 heavy non hydrogen atoms were considered. Some identified substructures were found within hERG active and inactive molecules. In table 1 all 15 substructures including their smile strings, IUPAC names, drawings and hERG active and inactive are shown. 5 substructures are exclusively found in hERG active molecules, 6 exclusively in hERG inactive molecules and 4 were found in both. The original differentiation between assay type and thresholds is not represented anymore, for those details including the number of matches for each substructure within the molecules refer to original authors work [21].

Following similar approach as done in the work by Schimunek J. et al. [90] this master thesis uses the described relevant substructures, especially ones which are exclusively related to hERG active, as a baseline. Refer to section 5.2.3 for details.

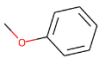
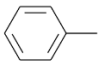
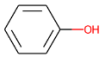
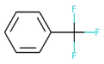
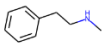
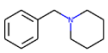
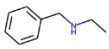
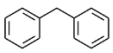
	smiles	iupac names	molecule	active	inactive
1	CCNCc1ccccc1	N-(phenylmethyl)ethanamine			**1**
2	CCOc1ccccc1	Ethoxybenzene		**1**	
3	CCc1ccccc1	Ethylbenzene		**1**	**1**
4	COc1ccccc1	Methoxybenzene			**1**
5	Cc1ccccc1	methylbenzene			**1**
6	NCCc1ccccc1	2-phenylethanamine		**1**	**1**
7	Oc1ccccc1	Phenol		**1**	**1**
8	c1cc(C)ccc1C	1,4-Dimethylbenzene			**1**
9	c1ccccc1C(F)(F)F	Trifluoromethylbenzene		**1**	**1**
10	c1ccccc1CCNC	N-methyl-2-phenylethanamine		**1**	
11	c1ccccc1CN2CCCCC2	1-(phenylmethyl)piperidine		**1**	
12	c1ccccc1CNCC	N-(phenylmethyl)ethanamine		**1**	
13	c1ccccc1Cc1ccccc1	phenylmethylbenzene		**1**	
14	n1ccccc1	Pyridine			**1**
15	n1ccccc1C	2-Methylpyridine			**1**

Table 1: hERG identified relevant substructures including the SMILE string, IUPAC chemical name, hERG active and inactive status [21]

4 Approach

4.1 Implementation and Software

4.1.1 Software

The initial part of this master thesis was first to reimplement the TabNet architecture based on the original paper as described in section 2.2. The original authors released a public available version, but not parameterize-able and using an outdated autograd backend (Tensorflow v1 [69]). Additionally the implementation was mainly focused on the Covertype dataset [36]. Even though other reimplementations of TabNet can be found it was decided to reimplement the original TabNet architecture to get better insight and being more flexible adding additional features and debugging options.

A recent version of Pytorch [79] is used as a autograd backend for the core TabNet components. Additionally Pytorch-Lighting (PL) [31] was used as a wrapper for handling training loops, better logging capabilities and callbacks such as early stopping and checkpointing. PL also allows for more easy GPU utilization including multi GPU training and distributed GPU training over many machines. DL baselines for experiments were also implemented using Pytorch and PL. For random forests the scikit-learn [80] framework was used and as gradient boosting decision tree implementation XGBoost [19].

The open source ML lifecycle stack MLflow [76] was used for experiment tracking including metric and parameter tracking. It was self hosted using a S3[1] compatible backend storage for artifact storing like model checkpoints and plots. Details of the self hosted experiment setup can be found in [57].

4.1.2 Hyperparameters Search

For hyperparameters search Optuna [4] and the Ax [8] framework have been used. The goal of the hyperparameters search is to find the optimal set of hyperparameters to achieve the best result of an underlying unknown objective function, in this case a ML model. Iteratively a set of hyperparameters are chosen and used to train the underlying model resulting in a measurable result or metric. To try and compare all possible combinations of hyperparameters doing grid search is usually not feasible. Therefore usually random hyperparameters search or other methods like Bayesian optimization is utilized.

Both mentioned frameworks are open source Bayesian hyperparameters optimization tools which iteratively update a prior belief about the impact of hyperparameters to a functions output or metric, in this case a ML model. Goal is to get a better posterior understanding about their relationship. First a number of initial runs, using randomly sampled hyperparameter sets, are used to build up a surrogate function describing the prior knowledge about the relationship between hyperparameters and function (ML model) outputs. This surrogate function or Gaussian process is a joint distribution of random variables, in this case hyperparameters which follows a multivariate normal distribution. Using the surrogate function an acquisition function is used to determine which hyperpa-

[1] Simple Storage Service - object storage over a web service - first introduced by Amazon in 2006 [6]

rameter set to use next. This is done by sampling from the surrogate function either by choosing a point of highest probability of improving (PoI) or by choosing the point with highest expected improvement (EI) [30].

This iteratively process by choosing a new set of hyperparameters using the acquisition function and updating the surrogate function based on the results is repeated a predefined number of times or runs. Within each experiment in this work the hyperparameter search space as well as the number of total runs to find the optimal hyperparameters is described.

4.1.3 Optimizer and Schedulers

For DL parameter optimization algorithm for gradient descent as described in section 2.1.2 the variants Adam [50] and AdamW [66] were used. For learning rate schedulers a linear scheduler with warmup or an exponential decaying one were applied. After each parameter update or training step the learning rate is updated. For a linear scheduler with warmup this can be described using learning rate η at given training step s training for N_{steps} with N_{warmup} warm up steps as follows:

$$\eta_{new} = \begin{cases} \eta \cdot \frac{s}{N_{warmup}}, & \text{if } s < N_{warmup} \\ \eta \cdot \frac{N_{steps}-s}{N_{steps}-N_{warmup}}, & \text{otherwise} \end{cases} \tag{4.1.1}$$

An simple exponential decaying scheduler with its settings decaying rate β and decaying step λ is given by:

$$\eta_{new} = \eta \cdot \beta^{\frac{s}{\lambda})} \tag{4.1.2}$$

Additionally the following figure 13 plots the behavior of the learning rate using different settings.

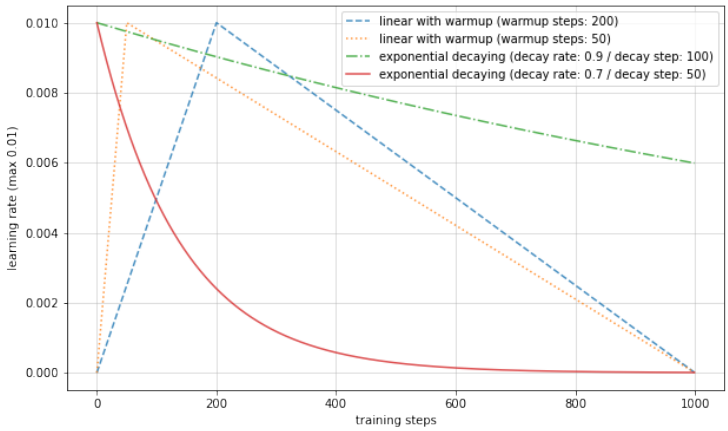


Figure 13: Linear and exponential decaying learning rate scheduler

The authors of TabNet used Adam optimizer with an exponential decaying learning rate scheduler in all of their experiments. Further within their release code repository [36] gradient clipping was applied. Gradient clipping is a technique to manually clip gradients if the norm or values of the gradient are above a defined threshold to mitigate

the exploding gradient issue and numerical problems. The reimplementation of TabNet within this master thesis has the option to apply gradient clipping. The exact setup (optimizer, scheduler and gradient clipping) is described within each of the experiments.

4.1.4 Infrastructure and Hardware

All experience were performed on a regular desktop computer with Ubuntu 20.04 as operating system powered by an AMD Ryzen 3700X with 64GB memory. Two GPUs, NVIDIA GTX 1070 with 8GB memory and NVIDIA GTX 1080 Ti with 11GB were used in a single node multi GPU setup for training the various DL models. Depending on the hyperparameters TabNet can be considered a compact DL architecture with usually between a few 100K parameters to maximum of single digit million parameters. Therefore it is feasible to train a TabNet based model from scratch without any hardware barrier and can not be compared to architectures like transformers with tens of millions of parameters.

For source control Git with GitHub[1] as backend was used. The TabNet reimplementation including all source codes including the experiments is available within a public repository [58]. The experiment metrics including parameters used are available under a public available domain[2] for at least 1 year after release of this work. The self hosted environment is based on MLFlow and described in [59]. For testing the framework pytest [56] was used. All core components of the TabNet reimplementation, this includes the individual components like attentive transformer, feature, decision step or sparsemax implementation were tested including gradient checking which especially important for the sparsemax part.

4.1.5 TabNet Implementation Details

Based on described architecture in section 2.2 the TabNet architecture was implemented. Due to the novel character in following the most relevant and untypical hyperparameters are explained. Some of them are added or adapted compared to the authors TabNet implementation [36] and offer an additional flexibility and options.

- **Input size** - refers to input feature dimensions
- **Feature size** - refers to internal hidden size for feature representation. The decision size N_d and the information output size N_a together are the feature size.
- **Decision size** - this is TabNet output size.
- **Number layers** - how many layers (FC, BN and GLU) each decision step within the feature transformer is using.
- **Number shared layers** - how many of those layers are shared between the decision steps. This significantly reduces the models trainable parameter count. If there are shared layers they are first run through within the feature transformer.
- **Number steps** - how many decision steps are used in the model.

[1] `https://github.com/`

[2] All experiments are available at `mlflow.kriechbaumer.at` for atleast 1 year after release of this work - later on experiments can be made available after contacting the author

- **Gamma,** γ - this is the relaxation parameter which controls if an individual feature is more likely used in multiple steps or not. Ifγ is close to 1 reusing of features is more strict if γ gets larger reusing is more relaxed. Details can be found in equation 2.2.8.
- **Relaxation type** - beside the value for γ this additional hyperparameter allows additional control of the relaxation hyperparameter.
  - γ *fixed* - this is the default value, handling gamma as a hyperparameter in the same way as the authors TabNet implementation.
  - γ *not applied* - Previous feature selection masks are not considered and the equation 2.2.8 is not applied.
  - γ *shared trainable* - converts γ from a hyperparameter to one trainable parameter shared across all decision steps.
  - γ *trainable* - converts γ from a hyperparameter to a trainable parameter per decision step.
- **Attentive type** - The authors TabNet implementation used Sparsemax exclusively for feature selection. This reimplementation allows to use alternative attentive methods. Those alternative methods include the α-entmax function with arbitrary alphas as described in section 2.2.3 using the α-entmax authors implementation as backend [28].
  - *Sparsemax* - default value and using sparsemax for feature selection.
  - *Entmax1.5* - uses the numerically optimized version of α-entmax when $\alpha = 1.5$.
  - α-*entmax* - uses α-entmax with an arbitrary value for $\alpha > 1$
  - α-*entmax* - α *shared trainable* - uses α-entmax but α itself becomes a trainable parameter shared across all decision steps.
  - α-*entmax* - α *trainable* - uses α-entmax but α itself becomes a trainable parameter per decision step.
- **Epsilon** ϵ - a small value for numerical stability when calculating the sparsity regularization term in equation 2.2.9.
- λ **sparse** - Controls the amount of sparsity regularization as described in equation 2.2.10.
- **Momentum** - the momentum term for calculating the running average for the mean μ and variance σ statistics in all batch norm used in the model.
- **Virtual batch size** - This describes the chunk size for ghost batch norm as described in section 2.2.1. Additionally if the value is not given no batch norm is applied within the feature and attentive transformer of the TabNet model.
- **Normalize input** - Controls if batch norm is applied onto the raw input features or not. The original paper describes to use raw (not normalized)features for the TabNet model, but first apply batch norm. Note for the input batch norm default batch norm (not ghost batch norm) is used.

The amount of hyperparameters also emphasize the novel character of the TabNet architecture but is especially relevant and potential problematic for hyperparameter search.

4.1.6 TabNet Metrics and Logging

The TabNet reimplementation allows for logging of the average sparsity, or one average how many individual feature dimension over a given dataset have not been considered. Given batch size B and feature dimension N using the Kronecker delta δ and the aggregated feature selection mask $\mathbf{M} \in \mathbb{R}^{B \times N}$ this is calculated as follows:

$$m_{sparsity} = \frac{1}{B \cdot N} \sum_{b=1}^{B} \sum_{i=1}^{N} \delta_{\mathbf{M}_{bi},0} \tag{4.1.3}$$

The sparsity metric $m_{sparsity}$ can be calculated on the overall level using the aggregated feature selection masks, as described in equation 2.2.25, or using individual feature selection mask per decision step.

Other metrics used were accuracy and the area under the receiver operator characteristic (AUROC) curve.

4.1.7 Verification

Before any experiments within the drug discovery domain were performed the reimplementation was verified based on the achieved results using the Covertype dataset, refer to section 3.1. As all relevant hyperparameter were known including random seeds and dataset splits this resulted in the following setup.

- *Hyperparameters*: Number of decision steps $N_{steps} = 5$, hidden feature size $N = 128$, TabNet decision output size $N_d = 64$, relaxation parameter gamma $\gamma = 1.5$, regularization parameter $\lambda_{sparse} = 0.0001$, batch norm momentum $m_B = 0.3$ with virtual batch size $B_V = 512$ for ghost batch norm, training batch size $B = 16384$, number shared layers 2 and number of individual layers 2
- *Optimizer and Scheduler*: Adam with learning rate $\eta = 0.02$ with exponential decaying learning rate scheduler using rate $\beta = 0.95$ decaying every 500 steps was used. Gradient clipping based on absolute gradient value above 2000 was performed.
- *Data preprocessing*: The same splits as described in section 3.1 were used. Per categorical dimension (44 binary feature dimensions) 1 dimensional embeddings are trained.

Using this setup the model was trained about 3 hours with described infrastructure, refer to section 4.1.4. The results of the reimplementation "my TabNet" compared with the TabNet authors results can be seen in following table 2. The TabNet reimplementation achieves similar results than the TabNet authors reported results. The minor differences were not further analyzed but could be explained in the more than 4 times higher maximum training steps, potential numerical differences resulting from varying initialization strategies used, as well as different and variations in software packages.

Model	Test accuracy (%)
XGBoost	89.34
LightGBM	89.28
CatBoost	85.14
AutoML Tables	94.95
TabNet [1]	96.99
my TabNet [2]	96.38

Table 2: TabNet reimplementation verification results

Furthermore the effect of using a different attentive strategies using the Covertype dataset was briefly explored. Within the following table 3 the achieved accuracy as well as the average sparsity as explained in equation 4.1.3 is visible. Achieved values are not representative, due to no hyperparameter tuning or repeated experiments with different random seeds and splits. Those runs are a teaser and give a better intuition especially about the sparsity behavior of different attentive types.

Model	Test accuracy (%)	Sparsity (%)
TabNet	96.38	75.80
TabNet (γ shared trainable)	96.59	32.03
TabNet ($\alpha = 3.0$-entmax)	94.28	86.34
TabNet ($\alpha = 1.5$-entmax)	96.34	27.10
TabNet (Softmax)	96.07	0

Table 3: TabNet reimplementation using different attentive strategies

In addition in figure 14 the training behavior of TabNet on the Covertype dataset is shown for different attentive strategies is illustrated. On the y axis the corresponding values for the TabNet loss term (cross entropy loss plus sparsity regularization), the achieved accuracy and the sparsity as given in equation 4.1.3 are shown. Data was collected on the validation Covertype dataset split during training. Additionally experiments or different effects of attentive strategies have not been further explored within this master thesis.

[1] about 130K training steps

[2] about 30K training steps - for all details refer to `https://mlflow.kriechbaumer.at/#/experiments/39/runs/fb30421c0cb34cdda54a7d961a09c7ad`

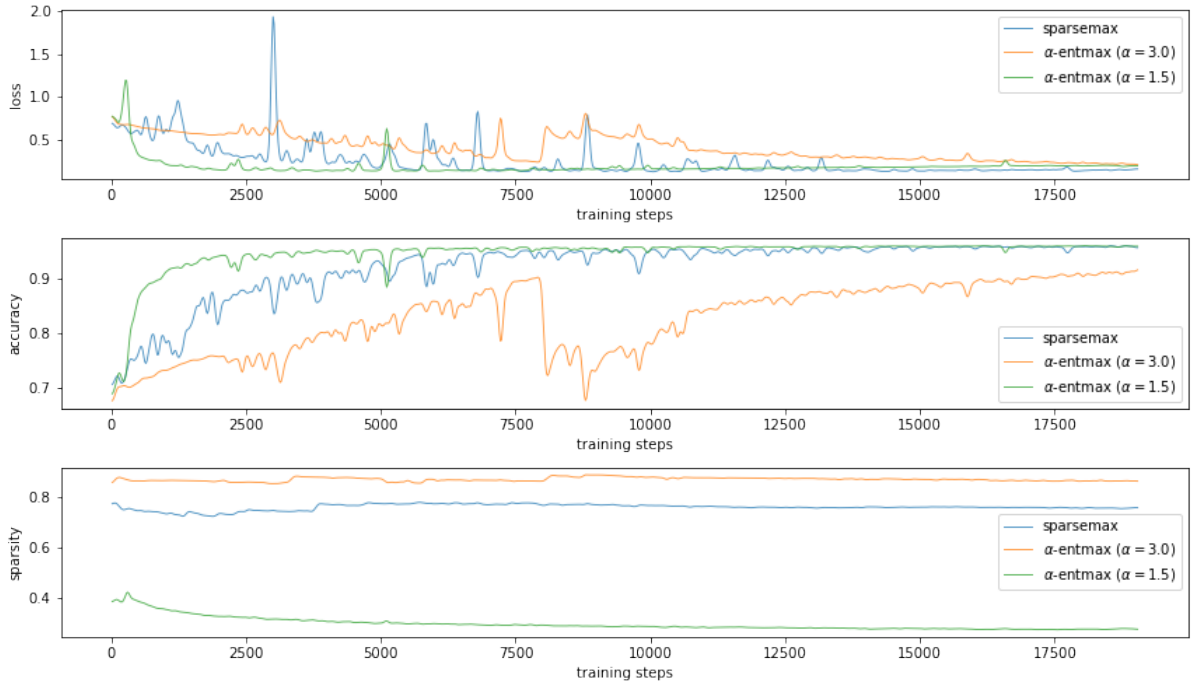


Figure 14: Training behavior of TabNet on the Covtype dataset using different attentive strategies

4.1.8 Interpretability and Plotting

The TabNet reimplementation allows for plotting of the feature selection masks as a heatmap, either using aggregated M_{agg} or individual $M^{[l]}$ feature selection mask. In figure 15 an example of the plotted M_{agg} for 25 samples of the Covertype test dataset using the trained TabNet model with attentive type sparsemax is shown. The corresponding labels per sample are displayed on the y axis and on the x axis the feature dimensions and names is given. In figure 16 all individual $M^{[l]}$ for all the 5 decision steps on the same 25 samples are visualized.

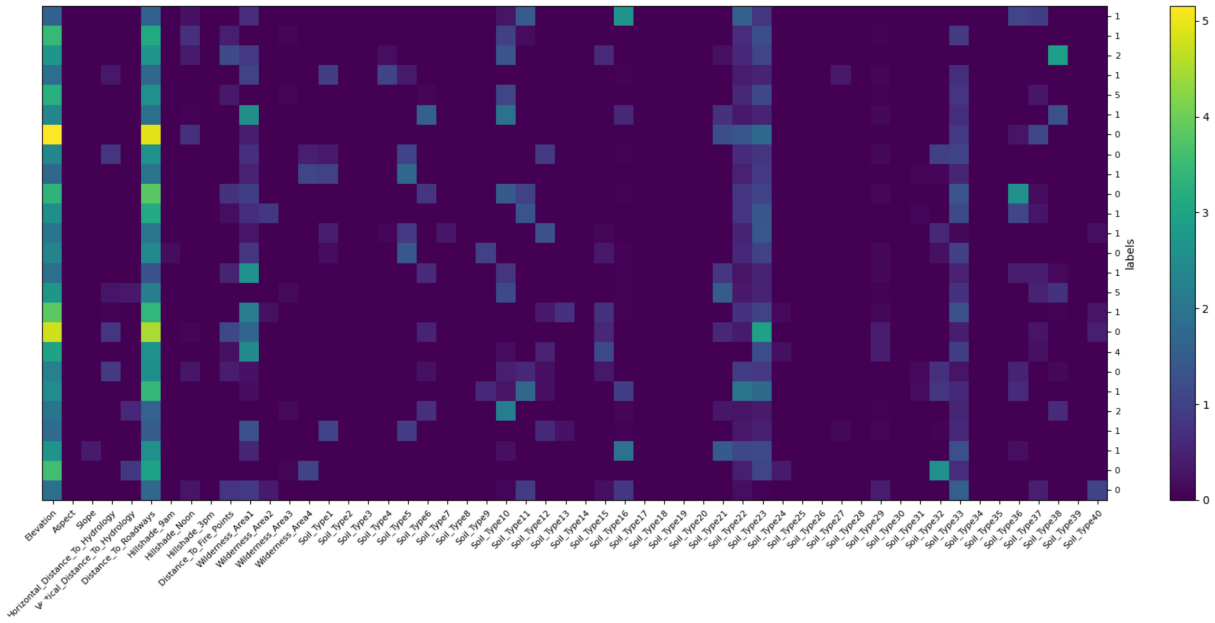


Figure 15: Aggregated mask M_{agg} visualized on sample entries of the Covertype test dataset

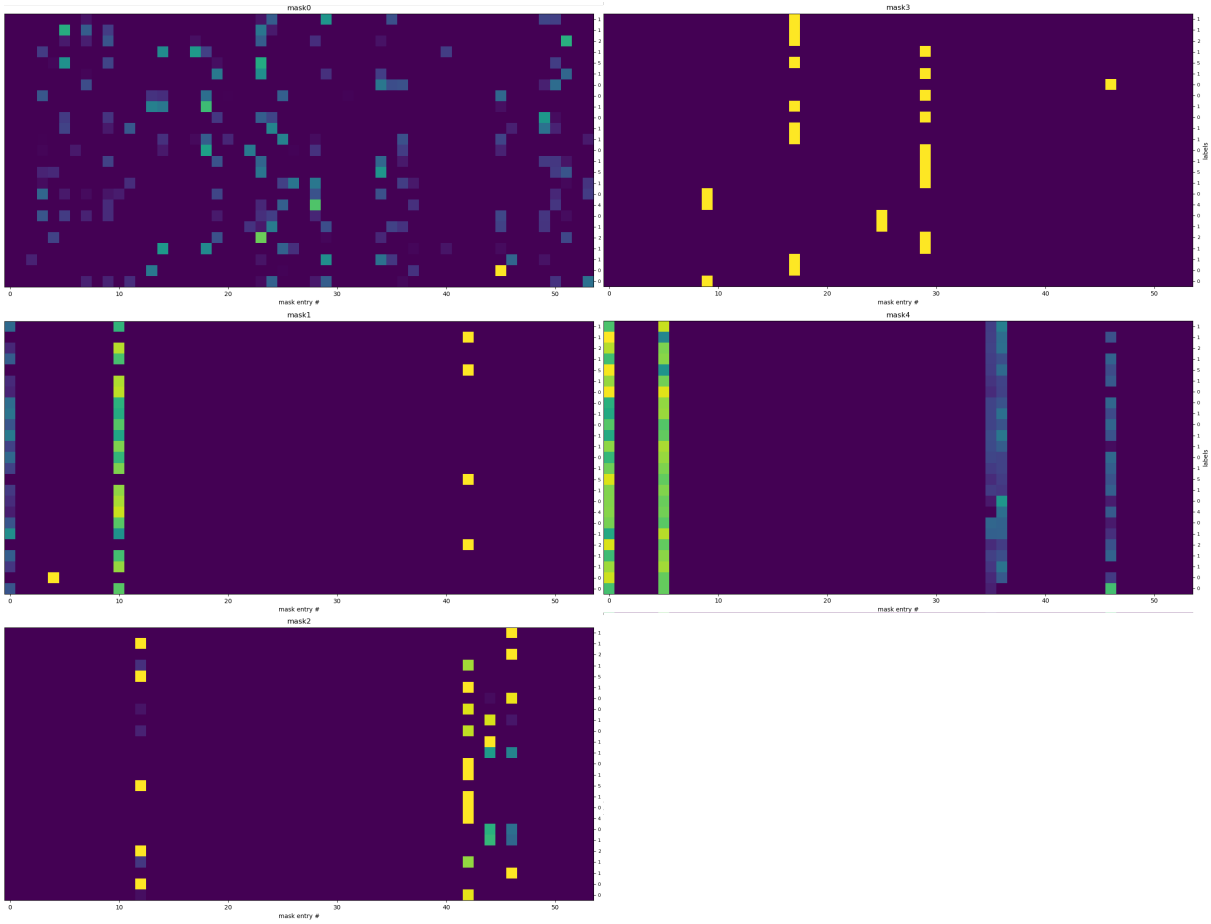


Figure 16: Individual masks on samples of the Covertype test dataset

By using the visualized feature selection mask it can be seen that some features, like for Covertype "Elevation" or "Distance to Roadways" seams to be important for all individual samples. When looking at the overview of all masks (figure 16) it can be seen that within the TabNet architectures individual decision steps specialize on different features. Using all entries in a dataset and calculate the mean and standard deviation for each feature dimension within the $\mathbf{M}_{\mathbf{agg}}$ the average importance and ranking of each feature can be determined. An example is shown in figure 17 for which on the left the average feature ranking using sparsemax and on the right using α-entmax with $\alpha = 3.0$ is presented. The concrete ranking with "Elevation" being the most important feature on average confirms the first impression while only looking at 25 samples as given in figure 15. Furthermore it shall be noted that the α-entmax with $\alpha = 3.0$ variant is much more sparse. It only uses 20 feature dimensions, meaning considering all test samples only 20 features were considered to be relevant.

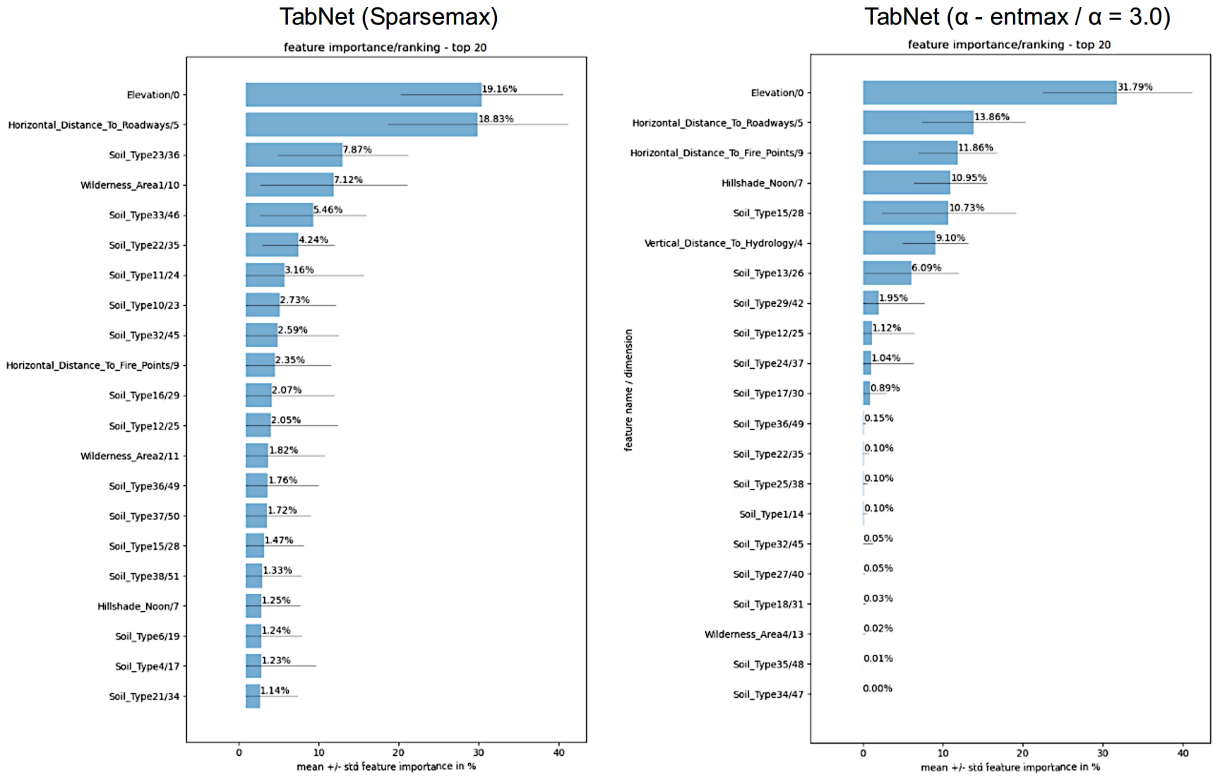


Figure 17: Average feature importance ranking of the Covertype test dataset using two different attentive types

5 Experiments and Results

Using the described datasets in section 3 the first objective was to determine the general predictive performance of TabNet within the drug discovery domain.

5.1 Predictive Performance Experiments

Datasets used were BBBP, BACE, HIV and SIDER, for details refer to section 3. In the following the experiment setup as well as the achieved results are presented.

5.1.1 Data Preparation

All datasets were randomly split into a training, validation and testing part with a percentage ratio of 80, 10 and 10 within each experiment run. Using the extended-connectivity fingerprint (ECFP) descriptor, molecules in the dataset were represented as a binary matrix in the form of $\mathbf{X} \in \mathbb{B}^{N \times D}$. When using the ECFP molecule descriptor (refer to section 2.4.2) the folding size defines the number of feature dimension D. For D various values were tested. For BBBP D was tested with (512, 4096, 12288), BACE, HIV and SIDER with (512, 4096).

5.1.2 Baseline

For direct comparison, a baseline model using a simple MLP utilizing a ReLU activation was trained. Using a Bayesian hyperparameter tuning framework, refer to section 4.1.2, the optimal hyperparameter for the baseline MLP were determined first. As a search space dropout probability p was chosen from $[0.0, 0.05, 0.1, 0.3]$, hidden size H chosen between 16 and 256 for 1 up to 8 layers L, and the learning rate η was defined to be between 0.00001 and 0.001. Other fixed hyperparameters were, maximum training steps defined as 1000, batch size set to 256 and a linear learning rate scheduler with warm up using Adam optimizer was applied. Best hyperparameter were selected out of 25 hyperparameter search runs. All hyperparameters search runs used the same seed for dataset splitting. Early stopping was applied using only the validation dataset part as a guideline. For each of the various feature dimension $D = [512, 4096, 12288]$ hyperparameter search was applied which lead to more than 300 training runs just to deterine the optimal hyperparameter for the corresponding dataset and folding size used. The found hyperparameters for the baseline MLP are presented in the following table 4:

Dataset	Number Layers	Hidden Size	η	Dropout
BBBP (D=512)[1]	4	167	$1.818 \cdot 10^{-5}$	0.0
BBBP (D=4096)[2]	8	44	$1.724 \cdot 10^{-5}$	0.05
BBBP (D=12288)[3]	5	211	$1.724 \cdot 10^{-5}$	0.1
BACE (D=512)[4]	7	150	$4.722 \cdot 10^{-5}$	0.3
BACE (D=4096)[5]	1	179	$6.907 \cdot 10^{-5}$	0.1
HIV (D=512)[6]	1	232	$1.97 \cdot 10^{-3}$	0.0
HIV (D=4096)[7]	1	198	$5.09 \cdot 10^{-5}$	0.3
SIDER (D=512)[8]	2	178	$8.24 \cdot 10^{-3}$	0.3
SIDER (D=4096)[9]	4	189	$5.16 \cdot 10^{-4}$	0.3

Table 4: Hyperparameter found for MLP baseline models

5.1.3 TabNet

Analog to the baseline model Bayesian hyperparameter search was applied first. Hyperparameter selected were decision size $N_d = [8, 16, 24, 32, 64, 128]$, relaxation parameter $\gamma = [1.0, 1.2, 1.5, 2.0]$, sparsity regularization $\lambda_{sparse} = [0.0, 10^{-6}, 10^{-4}, 0.001, 0.1]$ and the number of decisions steps N_{steps} (between 3 and 10). For the feature transformer the number of shared layers was set to 2 and the number of individual steps to 2. The maximum training steps, batch size, scheduler and optimizer were the same as used within the baseline model. Learning rate was selected within the same range as the baseline model. Batch normalization was not used. Again best hyperparameter were selected out of 25 hyperparameter search runs with the same seed for dataset splitting resulting in over 300 training runs. Early stopping was applied within the hyperparameter search runs.

Dataset	N_{steps}	N_d	γ	$lambda_{sparse}$	η
BBBP (D=512)[10]	5	128	1.0	$1.0 \cdot 10^{-6}$	$3.433 \cdot 10^{-5}$
BBBP (D=4096)[11]	8	16	1.5	0.0	$6.449 \cdot 10^{-4}$
BBBP (D=12288)[12]	6	32	1.2	$1.0 \cdot 10^{-3}$	$8.699 \cdot 10^{-4}$
BACE (D=512)[13]	5	24	1.5	0.0	$1.0 \cdot 10^{-3}$
BACE (D=4096)[14]	4	24	2.0	$1.0 \cdot 10^{-6}$	$3.118 \cdot 10^{-4}$
HIV (D=512)[15]	6	128	1.5	0.0	$2.48 \cdot 10^{-4}$
HIV (D=4096)[16]	3	128	1.2	0.0	$4.48 \cdot 10^{-4}$
SIDER (D=512)[17]	4	64	1.5	$1.0 \cdot 10^{-4}$	$2.64 \cdot 10^{-3}$
SIDER (D=4096)[18]	3	24	1.0	$1.0 \cdot 10^{-3}$	$7.98 \cdot 10^{-4}$

Table 5: Hyperparameter found for TabNet models

[1]Refer to `https://mlflow.kriechbaumer.at/#/experiments/73` for all 25 run details
[2]Refer to `https://mlflow.kriechbaumer.at/#/experiments/69` for all 25 run details
[3]Refer to `https://mlflow.kriechbaumer.at/#/experiments/70` for all 25 run details
[4]Refer to `https://mlflow.kriechbaumer.at/#/experiments/82` for all 25 run details
[5]Refer to `https://mlflow.kriechbaumer.at/#/experiments/87` for all 25 run details
[6]Refer to `https://mlflow.kriechbaumer.at/#/experiments/232` for all 25 run details
[7]Refer to `https://mlflow.kriechbaumer.at/#/experiments/235` for all 25 run details
[8]Refer to `https://mlflow.kriechbaumer.at/#/experiments/253` for all 25 run details
[9]Refer to `https://mlflow.kriechbaumer.at/#/experiments/253` for all 25 run details

5.1.4 Results

The final results have been determined by repeating or retraining each of the experiment setups using selected hyperparameters from scratch 20 times. For each of the experiment runs different random seeds for dataset splitting have been used. The random seed related to the model, e.g. for initialization, stayed the same. The results achieved on the test set can be seen in the following table 6 for BBBP, for BACE within table 7 for HIV within table 8 and for SIDER in table 9. As metric the mean AUROC and standard deviation of the 20 runs are presented. For additional comparison, results from work by Jiang et al. [45] as well as a recent result from Ramsauer et al. [86] were used. Both used a similar setup with multiple random split to determine their results.

Model	**BBBP mean AUROC ± std**
SVM[45]	0.919 ± 0.028
XGBoost[45]	0.926 ± 0.026
RF[45]	**0.927 ± 0.025**
GCN[45]	0.903 ± 0.027
GAT[45]	0.898 ± 0.033
DNN[45]	0.898 ± 0.033
MPNN[45]	0.879 ± 0.030
Attentive FP[45]	0.887 ± 0.032
Hopfield[86]	0.910 ± 0.026
TabNet $(D = 512)$[1]	0.874 ± 0.032
TabNet $(D = 4096)$[2]	0.875 ± 0.022
TabNet $(D = 12288)$[3]	0.891 ± 0.030
Baseline MLP $(D = 512)$[4]	0.877 ± 0.032
Baseline MLP $(D = 4096)$[5]	0.902 ± 0.022
Baseline MLP $(D = 12288)$[6]	0.913 ± 0.022

Table 6: TabNet results on BBBP test set showing mean AUROC ± standard deviation for 20 random splits

[10]Refer to `https://mlflow.kriechbaumer.at/#/experiments/54` for all 25 run details
[11]Refer to `https://mlflow.kriechbaumer.at/#/experiments/61` for all 25 run details
[12]Refer to `https://mlflow.kriechbaumer.at/#/experiments/59` for all 25 run details
[13]Refer to `https://mlflow.kriechbaumer.at/#/experiments/83` for all 25 run details
[14]Refer to `https://mlflow.kriechbaumer.at/#/experiments/86` for all 25 run details
[15]Refer to `https://mlflow.kriechbaumer.at/#/experiments/246` for all 25 run details
[16]Refer to `https://mlflow.kriechbaumer.at/#/experiments/261` for all 25 run details
[17]Refer to `https://mlflow.kriechbaumer.at/#/experiments/251` for all 25 run details
[18]Refer to `https://mlflow.kriechbaumer.at/#/experiments/258` for all 25 run details
[1]Refer to `https://mlflow.kriechbaumer.at/#/experiments/56` for all 20 run details
[2]Refer to `https://mlflow.kriechbaumer.at/#/experiments/63` for all 20 run details
[3]Refer to `https://mlflow.kriechbaumer.at/#/experiments/62` for all 20 run details
[4]Refer to `https://mlflow.kriechbaumer.at/#/experiments/74` for all 20 run details
[5]Refer to `https://mlflow.kriechbaumer.at/#/experiments/71` for all 20 run details
[6]Refer to `https://mlflow.kriechbaumer.at/#/experiments/72` for all 20 run details

Model	BACE mean AUROC ± std
SVM[45]	0.893 ± 0.020
XGBoost[45]	0.889 ± 0.021
RF[45]	0.890 ± 0.022
GCN[45]	0.898 ± 0.019
GAT[45]	0.886 ± 0.023
DNN[45]	0.890 ± 0.024
MPNN[45]	0.838 ± 0.027
Attentive FP[45]	0.876 ± 0.023
Hopfield[86]	**0.902 ± 0.023**
TabNet $(D = 512)$[1]	0.858 ± 0.029
TabNet $(D = 4096)$[2]	0.854 ± 0.032
Baseline MLP $(D = 512)$[3]	0.876 ± 0.033
Baseline MLP $(D = 4096)$[4]	0.893 ± 0.032

Table 7: TabNet results on BACE test set showing mean AUROC ± standard deviation for 20 random splits

Model	HIV mean AUROC ± std
SVM[45]	0.822 ± 0.020
XGBoost[45]	0.816 ± 0.020
RF[45]	0.820 ± 0.016
GCN[45]	**0.834 ± 0.025**
GAT[45]	0.826 ± 0.030
DNN[45]	0.797 ± 0.018
MPNN[45]	0.811 ± 0.031
Attentive FP[45]	0.822 ± 0.026
Hopfield[86]	0.815 ± 0.023
TabNet $(D = 512)$[5]	0.752 ± 0.028
TabNet $(D = 4096)$[6]	0.755 ± 0.019
Baseline MLP $(D = 512)$[7]	0.775 ± 0.027
Baseline MLP $(D = 4096)$[8]	0.785 ± 0.024

Table 8: TabNet results on HIV test set showing mean AUROC ± standard deviation for 20 random splits

[1]Refer to `https://mlflow.kriechbaumer.at/#/experiments/85` for all 20 run details
[2]Refer to `https://mlflow.kriechbaumer.at/#/experiments/89` for all 20 run details
[3]Refer to `https://mlflow.kriechbaumer.at/#/experiments/84` for all 20 run details
[4]Refer to `https://mlflow.kriechbaumer.at/#/experiments/88` for all 20 run details
[5]Refer to `https://mlflow.kriechbaumer.at/#/experiments/248` for all 20 run details
[6]Refer to `https://mlflow.kriechbaumer.at/#/experiments/262` for all 20 run details
[7]Refer to `https://mlflow.kriechbaumer.at/#/experiments/238` for all 20 run details
[8]Refer to `https://mlflow.kriechbaumer.at/#/experiments/237` for all 20 run details

Model	SIDER mean AUROC ± std
SVM[45]	0.630 ± 0.021
XGBoost[45]	0.642 ± 0.020
RF[45]	0.646 ± 0.022
GCN[45]	0.634 ± 0.026
GAT[45]	0.627 ± 0.024
DNN[45]	0.631 ± 0.028
MPNN[45]	0.598 ± 0.031
Attentive FP[45]	0.623 ± 0.026
Hopfield[86]	**0.672 ± 0.019**
TabNet $(D = 512)$[1]	0.588 ± 0.032
TabNet $(D = 4096)$[2]	0.582 ± 0.027
Baseline MLP $(D = 512)$[3]	0.629 ± 0.028
Baseline MLP $(D = 4096)$[4]	0.644 ± 0.018

Table 9: TabNet results on SIDER test set showing mean AUROC ± standard deviation for 20 random splits

5.2 Interpretability Experiment

The second part of the experiments uses the hERG dataset and identified relevant components, refer to section 3.6, in an attempt to empirically analyze TabNets interpretability capabilities, using the feature selection mask, to identify and rank most relevant features. The experiment setup follows the approach from Schimunek J. et al. [90] and uses hERG identified relevant components as a ground truth for most relevant features. The experiment setup is explained in the following sections.

5.2.1 Data Preparation

hERG dataset was split in a ratio of 60:20:20 for training, validation and testing parts. Molecules were represented in a combination of extended-connectivity fingerprints (ECFP) with a folding size of $D_{ecfp} = 1024$, MACCS fingerprints $D_m = 167$ and 826 SMARTS (refer to section 2.4.2) pattern from Mayr A. et al. [72] with $D_t = 826$. This resulted in a combined feature dimension of $D = 2017$ and input data $\mathbf{X} \in \mathbb{R}^{B \times 2017}$ with B as the batch size.

5.2.2 Atomic Attribution and Mapping

Each molecule mol_i consists of J_i individual atoms which vary from molecule to molecule. Using a descriptor one molecule i can be represented as a vector $\mathbf{x_i} \in \mathbb{R}^D$. D depends on the descriptor and corresponding settings used. The descriptor is a function mapping from molecule and its atomic structure to its numerical representation in the form of $d \mid mol_i \rightarrow \mathbb{R}^D$. In similar way a atomic contribution mapping can be identified using

[1]Refer to `https://mlflow.kriechbaumer.at/#/experiments/252` for all 20 run details
[2]Refer to `https://mlflow.kriechbaumer.at/#/experiments/259` for all 20 run details
[3]Refer to `https://mlflow.kriechbaumer.at/#/experiments/254` for all 20 run details
[4]Refer to `https://mlflow.kriechbaumer.at/#/experiments/257` for all 20 run details

a mapping function $m \mid \mathbb{R}^D \rightarrow \mathbb{R}^{J_i}$ for each descriptor. This mapping describes each feature dimension d and its contribution factor to each atom j. Using a feature attribution methods for a trained model, for example as given by TabNets feature masks $\mathbf{M_{agg}}$ (section 2.2.4) or other feature attributions methods (section 2.3) the feature attribution $\mathbf{v_i} \in \mathbb{R}^D$ for one particular molecule i can be identified. Using the atomic mapping function $m(\mathbf{v_i})$ one can calculate the atomic attribution $\mathbf{a_i} \in \mathbb{R}^{J_i}$ for a particular molecule i for each descriptor. When using multiple molecule descriptors (as in this experiment setup) the overall atomic attribution for one particular molecule i is given by the sum of all atomic attribution over all descriptors.

In the following figure 18 the atomic mapping for one concrete sample of the hERG dataset with 20 heavy atoms (D2392 with smile "O1c2c(C(=O)C[C@H]1c1ccc(O)cc1)c(O)cc(O)c2" and IUPAC name "(2S)-5,7-dihydroxy-2-(4-hydroxyphenyl)chroman-4-one") which is hERG inactive is shown. In the example an ECFP fingerprint is used with an unrealistic folding size of 32 and radius of 1 for visualization purpose. The strength of each individual edge indicates the different factors for individual contribution of one feature dimension to the actual atom. Furthermore in figure 19 the same atom with a given atomic attribution is presented using feature attributions (e.g. from TabNet or other feature attribution methods). Using the mapping function m presented in the form of edges the concrete atomic attribution for each atom within the hERG inactive sample is determined.

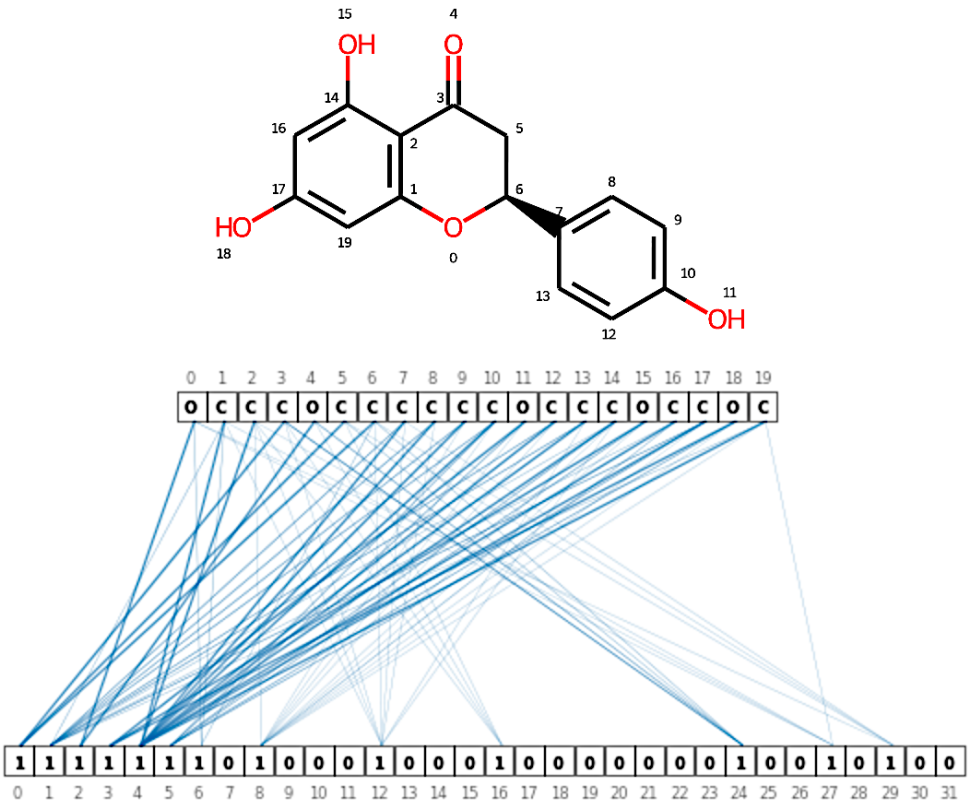


Figure 18: Atomic mapping example showing one hERG inactive molecule sample. On top a hERG inactive molecule including atom indices is presented. The bottom gives a sample for the corresponding ECFP fingerprint.

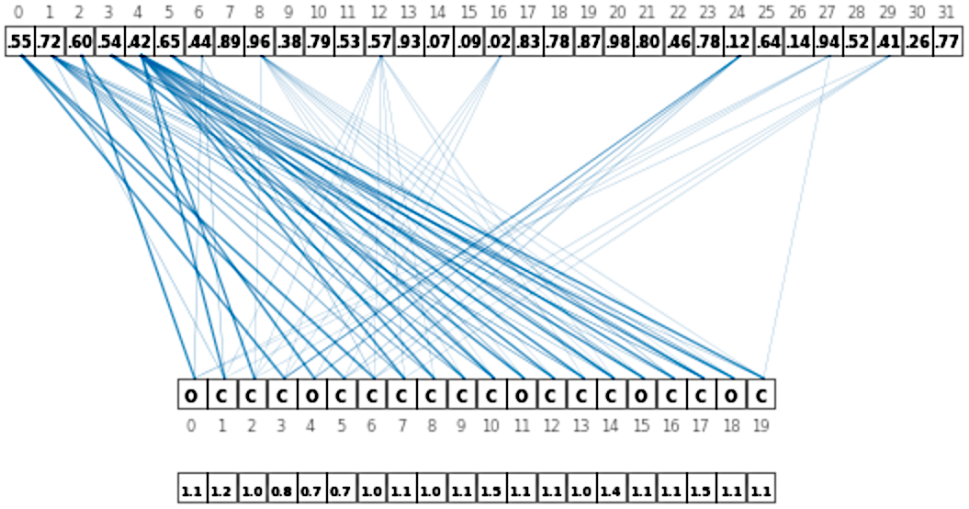


Figure 19: Atomic attribution example for one hERG inactive molecule sample. On top the feature attribution is given and on bottom the calculated atomic attribution

The implementation of the individual mapping functions m for each descriptor type (ECFP, MACCSkeys, and DeepTox [72]) was kindly provided by Schimunek J. [90]. As the determination of the atomic mapping is computational expensive, due to the fact that the atomic mapping is individual per molecule and descriptor, within this master thesis work the atomic mapping was precomputed and cached. The atomic mapping stores the information which feature dimension of a descriptor contributes to which atom index by what factor. The blue edges in figure 18 refer a sample atomic mapping which can be precomputed and cached. Given feature attributions using the precomputed atomic mapping the atomic attribution can be calculated without the usage of the actual mapping function m. This significantly improves experiment runtime.

5.2.3 Metric and Evaluation Methodology

To evaluate the identified atomic attribution the hERG identified relevant substructures, as described in section 3.6, are used as a ground truth. A perfect atomic attribution would attribute and rank those atom indices higher which do match a relevant ground truth substructure. To measure the ability to rank atomic attribution which match a relevant substructure the area under the receiver operator characteristic curve (AUROC) was used. To illustrate this, in figure 20 the hERG relevant substructure Ethoxybenzene (with smile "CCOc1ccccc1") is presented matching a hERG molecule (same as given in figure 18) at atom indices indicated in red.

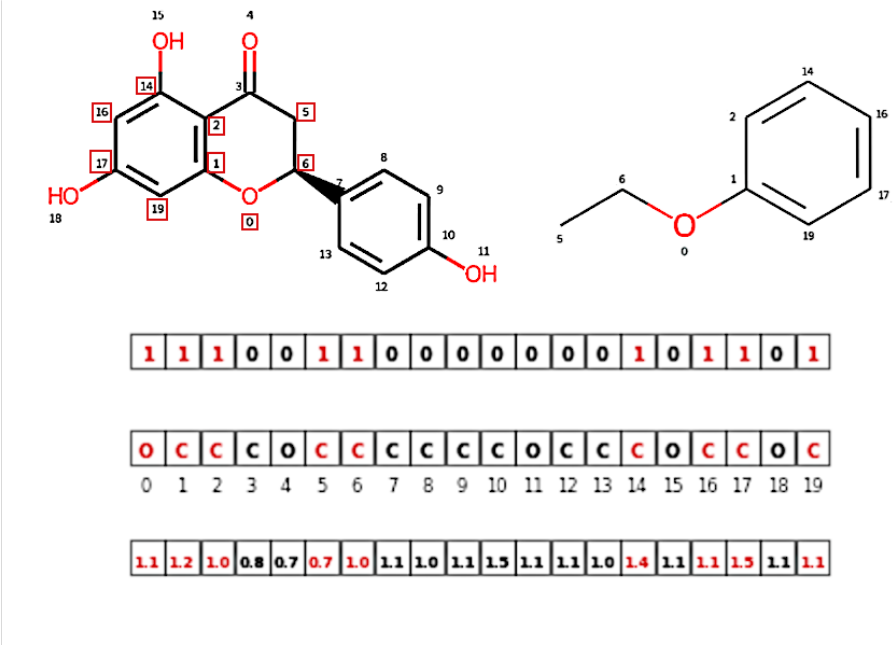


Figure 20: hERG relevant substructure matching. Top left shows a large hERG molecule with a matching relevant substructure to its right. Matching atomic indices are marked in red. On bottom the relevant numerical atomic attribution is given.

Using the determined atomic attribution the AUROC between the matching atom indices represented by the binary vector **s** and atomic attribution **a**

$$\mathbf{s} = [1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0]$$
$$\mathbf{a} = [1.1, 1.2, 1.0, 0.8, 0.7, 0.7, 1.0, 1.1, 1.0, 1.1, 1.5, 1.1, 1.1, 1.0, 1.4, 1.1, 1.1, 1.5, 1.1, 1.1]$$

is 0.59. As reference for intuition, a perfect atomic attribution ranking on a fictive sample having $\mathbf{s} = [0, 1, 0, 0, 1]$ could be $\mathbf{a}$ =[0.5, 0.7, 0.3, 0.1, 0.6] with AUROC of 1.0 while a worst case scenario could be given $\mathbf{a} = [0.5, 0.7, 0.3, 0.0, 0.6]$ with AUROC of 0.0.

Given the described methodology using the AUROC to determine the ability to rank relevant atoms based on their atomic attribution first, as ground truth only unique hERG active substructures were considered. Those hERG active substructures (also refer to table 1) are ethoxybenzene (smile "CCOc1ccccc1"), N-methyl-2-phenylethanamine (smile "c1ccccc1CCNC"), 1-(phenylmethyl)piperidine (smile "c1ccccc1CN2CCCCC2"), N-(phenylmethyl)ethanamine (smile "c1ccccc1CNCC") and phenylmethylbenzene (smile "c1ccccc1Cc1ccccc1"). Using those 5 ground truth molecules the final score was calculated as follows:

1. Train model using the hERG dataset with the binary classification objective of a molecule being either hERG active or inactive (only the first target of the hERG dataset as described in section 3.6 was considered).

2. Get all hERG activity predictions for the test part of the dataset using the trained model.

3. Calculate the feature attributions using a feature attribution method (e.g. TabNet M_{agg} or methods described in section 2.3)
4. Use the feature attributions to get all atomic attributions utilizing the precomputed and cached atomic mappings.
5. Consider samples for which the hERG activity was predicted as inactive.
6. For all those samples calculate the ranking score (AUROC) using the atomic attributions on each ground truth molecule (5 relevant substructures). Obviously the value is only calculated if the ground truth molecule is indeed a substructure of a particular hERG molecule.
7. The final score is the mean over all the individual ranking scores for each hERG sample and potential substructure.

This process, using the inactive hERG predictions together with substructures which are most relevant for hERG active molecules, might seam to be non-intuitive but follows the counterfactual thinking approach anaolog to the setup used by Schimunek et al. [90]. The idea is that despite of one molecule being predicted as hERG inactive the typically substructure found in hERG active molecules should still be considered important.

5.2.4 Training

The experiment setup and evaluation methodology was applied onto different ML model architectures and feature interpretability methods. This allows for a broader picture and gives reference results for TabNets interpretability capabilities. Additionally this follows the work of Schimunek et al. [90] in which interpretability methods were first empirically evaluated using the hERG dataset with a trained MLP as a bases. For model architectures **TabNet**, using the reimplementation described in section 4.1.5, a **Baseline MLP**, using the same baseline implementation as described in section 5.1.2, **Random Forest (RF)**[1] and **Gradient Boosting Decision Tree (GBDT)**[2] were evaluated.

Given the model architectures the following combinations of interpretability methods were evaluated.

- **TabNets** M_{agg} - only used and relevant for the TabNet architecture
- **Integrated Gradients (IG)**[3] - applied after training *TabNet* and a baseline *MLP*
- **Saliency (absolute and relative)**[3] - applied after training *TabNet* and a baseline *MLP*
- **Input** $\times$ **Gradient**[3] - applied after training *TabNet* and a baseline *MLP*
- **Shapley Value Sampling**[3] - applied for all model types

[1] Using the scikit-learn implementation [80]
[2] Using the XGBoost implementation [19]
[3] Using the captum implementation [53]

Before any interpretability method was applied to evaluate interpretability capabilities as described in section 5.2.3 for each model type hyperparameter search using Bayesian hyperparameter optimization as described in section 4.1.2 was applied. For each model architecture the following optimal hyperparameters were determined over 30 trials each.

- **TabNet** - The best found TabNet architecture[1] uses 3 decision steps with decision size $N_d = 16$, $\gamma = 1.5$ and $\lambda_{sparse} = 10^{-4}$ for sparsity regularization. As learning rate 0.01 was determined with a batch size of 256. AdamW optimizer with weight decay of 0.0001 together with an exponential decaying learning rate scheduler having decaying rate $\beta = 0.9$ over 800 steps was applied. Batch normalization was applied using momentum 0.95 and a virtual batch size for ghost batch norm of 64.
- **Baseline MLP** - The best found architecture[2] is a 3 layer MLP with hidden size of 128. Dropout was not applied. Batch normalization was determined with a momentum of 0.9. Model was trained with a batch size of 512 using Adam without weight decay utilizing a linear scheduler with a maximum learning rate of 0.001.
- **Random Forest (RF)** - The best RF found[3] consists of 200 decision trees. Bootstrap sampling was not applied. The splitting criteria used was entropy or information gain as described in section 2.1.1. The minimum amount of samples for one split was determined to be 5 with at least 2 samples per split leaf. For the maximum number of features considered, the square root of feature dimensions (in this case $\sqrt{2017}$) or around 44 for the hERG dataset was determined. The maximum depth of one decision tree was set to 40.
- **Gradient Boosting Decision Tree (GBDT)** - The best GBDT found[4] uses learning rate of 0.1 for boosting and has a maximum depth of 24 per decision tree trained.

5.2.5 Results

Given the optimal hyperparameters found each model architecture was repeatedly trained using 20 different random splits of the hERG dataset. In the following table 10 the result of the binary classification task whether a molecule is hERG active or inactive is presented using the mean AUROC as well as the standard deviation on the test part over 20 consecutive random splits.

Model	hERG mean AUROC ± std	min AUROC	max AUROC
TabNet[5]	0.826 ± 0.017	0.793	0.859
MLP[6]	0.878 ± 0.010	0.864	0.895
RF	**0.895 ± 0.008**	0.879	0.916
GBDT[7]	0.890 ± 0.009	0.877	0.903

Table 10: Comparative results on hERG test set showing mean AUROC ± standard deviation for 20 random splits

[1]Refer to `https://mlflow.kriechbaumer.at/#/experiments/212` for all 30 trials
[2]Refer to `https://mlflow.kriechbaumer.at/#/experiments/205` for all 30 trials
[3]Refer to `https://mlflow.kriechbaumer.at/#/experiments/217` for all 30 trials
[4]Refer to `https://mlflow.kriechbaumer.at/#/experiments/220` for all 30 trials

After each consecutive training runs the mentioned interpretability methods were applied using the trained models using the methodology described in section 5.2.3. In the following tables 11, 12 and 13 the achieved score (using AUROC) ranking relevant atoms first is presented. Methods are ordered descending from best to worst achieved performance. TabNets performance is shown in table 11.

Interpretability Method[1]	**mean AUROC ± std**	min AUROC	max AUROC
Shapley Value Sampling	**0.649 ± 0.069**	0.451	0.727
Integrated Gradients (IG)	0.637 ± 0.069	0.465	0.721
Input × Gradient	0.611 ± 0.041	0.523	0.683
Saliency (relative)	0.534 ± 0.023	0.493	0.588
TabNet	0.485 ± 0.06	0.429	0.685
Saliency (absolute)	0.468 ± 0.024	0.430	0.526

Table 11: Comparative ranking score of different interpretability methods applied on a trained TabNet model

Interpretability Method[2]	**mean AUROC ± std**	min AUROC	max AUROC
Shapley Value Sampling	**0.703 ± 0.014**	0.671	0.721
Integrated Gradients (IG)	0.685 ± 0.018	0.652	0.711
Input × Gradient	0.673 ± 0.015	0.643	0.694
Saliency (relative)	0.653 ± 0.022	0.594	0.692
Saliency (absolute)	0.411 ± 0.015	0.378	0.432

Table 12: Comparative ranking score of different interpretability methods applied on a trained MLP

Interpretability Method	**mean AUROC ± std**	min AUROC	max AUROC
GBDT+Shapley Value Sampling[3]	0.614 ± 0.012	0.599	0.632
RF+Shapley Value Sampling[4]	0.584 ± 0.013	0.564	0.595

Table 13: Comparative ranking score of different interpretability methods applied on RF and GBDT

[5]Refer to `https://mlflow.kriechbaumer.at/#/experiments/214` for all 20 run details
[6]Refer to `https://mlflow.kriechbaumer.at/#/experiments/211` for all 20 run details
[7]Refer to `https://mlflow.kriechbaumer.at/#/experiments/222` for all 20 run details
[1]Refer to `https://mlflow.kriechbaumer.at/#/experiments/214` for all 20 run details
[2]Refer to `https://mlflow.kriechbaumer.at/#/experiments/211` for all 20 run details
[3]Refer to `https://mlflow.kriechbaumer.at/#/experiments/222` for all 20 run details
[4]Refer to `https://mlflow.kriechbaumer.at/#/experiments/218` for all 20 run details

6 Discussion

6.1 Performance

Using 5 different datasets within the drug discovery domain performing close to 1000 experiment runs (including hyperparameter search and repeated random split runs) it could not be shown that TabNet performs better than other ML architectures within this data domain using the described approach and scenario.

The first experiment using BBBP dataset (refer to section 3) was repeated using 3 different ECFP fingerprint folding sizes (512, 4096 and 12288). Similar as described in the original TabNet paper [7], the idea was that due to feature selection, TabNet potentially has a higher learning capacity and can focus easier on more salient features. Due to the nature of ECFP fingerprints becoming more sparse with larger folding sizes TabNet potentially can focus better on salient (non sparse) entries or features. But comparing the results for different folding/feature dimension sizes, using more feature dimensions seems not to improve the results for TabNet substantially. In fact an in parallel trained MLP seems to benefit more from increased folding size than TabNet. Therefore subsequent datasets and experiments were only trained using two different folding or feature dimension sizes (512 and 4096).

For none of the evaluated datasets TabNet was within the top performing architectures. Relative to a trained baseline MLP, TabNet performed best for the BBBP dataset as seen in figure 6. It performed worst using the HIV dataset as shown in figure 8. As the performance was not only compared to reference work but also to a baseline MLP, which was trained using the same environment, data preparation and setup, it can be argued that potential inaccuracies got eliminated or at least minimized. In fact when looking at the performance of the trained baseline MLPs and comparing it to reported reference DNN results, some of them (e.g. for SIDER refer to figure 9) could be improved.

6.1.1 Limitations

Compared to a typical MLP, TabNet has more hyperparameters to optimize. The hyperparameter search space used was inspired by the original paper, but still the number of search runs (25) using Bayesian optimization might not be sufficient to find optimal TabNet hyperparameter. Investing the same time and effort in training TabNet and baseline MLP models, MLP based models and their found hyperparameters outperformed corresponding TabNet models. Due to limited computational resources, further optimization runs were not feasible. Also the original TabNet paper [7] despite describing the search space for hyperparameters did not mention the method or the number of training runs to determine the optimal hyperparameters for the corresponding datasets. Therefore it remains unclear whether TabNets performance would significantly increase when investing more into hyperparameter search.

6.2 Interpretability

Applying the described experiment setup (refer to section 5.2), results indicate that using TabNet's feature selection mask for identifying most relevant atoms is not well suited compared to other methods.

Shapley value sampling performed best ranking most relevant atoms, independent of the underlying ML architecture. Overall using a simple MLP as ML architecture together with Shapley value sampling achieved the best performance ranking relevant atoms. The second best method was integrated gradient together with an MLP. Results indicate that both methods are suited to rank relevant atoms, this is in line with results from Schimunek et al. [90]. Furthermore DT based methods (RF and GBDT) have been analyzed and surprisingly, despite achieving a slightly better prediction result, together with Shapley value sampling performed worse in ranking most relevant atoms.

6.2.1 Limitations

One limitation or argument why the used experiment setup might not be best suited for TabNet is the possibility, that the counterfactual thinking approach (described in section 5.2) might not match with TabNets feature selection mask. Within the counterfactual thinking approach baseline atoms are supposed to still be ranked high, despite other atoms being more important for a concrete prediction result. If TabNets feature selection mask (being sparse) does not select some relevant baseline atoms at all those can not be ranked high or considered at all. Results in table 11 partially confirm this, but in edge cases, when looking at the minimum and maximum achieved ranking score, TabNet could in some experiment runs rank relevant atoms appropriately. This is also reflected in the higher variance of the ranking score when using TabNet compared to the baseline MLP.

Furthermore, the output of TabNet consists of the actual prediction and the corresponding sparse feature selection mask with its feature importance values related to the prediction. Therefore TabNets feature selection mask only explains why a certain feature was selected for the actual prediction. Other interpretability methods like Shapley value sampling or integrated gradients (applied onto a trained TabNet model) give feature importance value for both outputs, the prediction and the feature selection mask. It follows that those methods can attribute importance to features which might be relevant for TabNets feature selection process itself. For this reason comparative results of TabNet's feature selection mask with interpretability methods applied onto TabNet (as shown in figure 11) should be considered with caution.

Another general limitation of the overall assessment of TabNet's interpretability capabilities, is the validity of the chosen baseline and experiment setup. The baseline consists of 5 relevant components or molecules. Additional and more baseline molecules could increase the experiments informative value.

7 Conclusion and Outlook

As shown in the original paper [7] TabNet outperformed and was compared especially to decision tree (DT) based methods like Gradient Boosting Decision Trees (GBDT) using various tabular datasets. As DT based methods in many cases still achieve state of the art results within various drug discovery tasks, TabNet seemed a potential fitting architecture.

In this master thesis the TabNet architecture has been studied extensively using various datasets in the context of drug discovery. Results indicate that TabNets predictive performance does not reach the level of a baseline MLP and other ML methods like Random Forests (RF) or Gradient Boosting Decision Trees (GBDT) over a set of 5 different drug discovery tasks. Therefore it could not be confirmed, that it is an exceptionally fitting architecture for this data domain.

As TabNet by design offers built-in feature selection and feature attribution, which is a direct additional output in the forward path, an attempt was made to empirically measure and compare it with other interpretability methods. Within the chosen scenario, to rank most relevant atoms in a molecule first, other methods like Shapley value sampling or integrated gradients together with a MLP outperformed TabNet. But as discussed in section 6.2.1 those results should be considered with caution as it is highly dependent on the experiment setup and the baseline, defining which atoms are actually important within a molecule. Therefore a final statement whether TabNet is suitable as an interpretability method within the drug discovery domain can not be made based on the chosen scenario.

Even though the overall results of TabNet did not match the expectation it is still a new architecture with many novel features and ideas quite different from other methods. Especially the attempt to select features and be interpretable by design is interesting for a deep learning architecture. Furthermore, the possibility to end-to-end train TabNet or add other data types to handle multi-modal data and still offer direct interpretability could be appropriate for various scenarios. Similarly as done in past ML challenges, TabNet due to its novel and different structure, could be a suitable candidate for an additional method within an ensemble of ML models.

Considering all the results and experience with TabNet the following aspects could be bases for further research.

As TabNet is utilizing sparsemax for feature selection and attribution, further developments and alternatives like $\alpha-$entmax (refer to section 2.2.3) would be an obvious choice for additional experiments. Replacing sparsemax in TabNet with $\alpha-$entmax variants and empirically measure the impact would be interesting. Additionally, directly learning α in $\alpha-$entmax as a parameter (not hyperparameter) could be beneficial. Also removing or adapting the relaxation hyperparameter γ, which affects to what extend features are reused in multiple decision steps within TabNet, would be an option. Effectively utilizing γ requires a prior knowledge about the training data and its features. If this is not available one possibility could be to instead train γ as a parameter. This would eliminate one hyperparameter and the effect could be subject to one research question.

Even though in one of the experiments (using hERG) multiple molecule fingerprints

have been used in a combined way, an extensive experiment setup comparing various molecule descriptors and fingerprints together with TabNet could provide additional insights. As in all other experiments extended connectivity fingerprints (ECFP) were used (although with different fingerprint sizes), further experiments could clarify the impact of this setup. Furthermore, as TabNet sparsely selects most relevant features, using multiple molecule descriptors or fingerprints in a combined way could be an option to find out which one is the most relevant (at least for TabNet). This setup could additional be explored and compared with a other ML methods together with interpretability methods like Shapley value sampling or integrated gradients.

A different research direction would be to explore and utilize optimized sparse linear transformation[1] within TabNet. As TabNet heaviliy utilizes sparsemax and depending on the hyperparameter chosen has many intermediate sparse matrices, using optimized sparse linear transformation could potentially lead to very compact, efficient and fast models.

Hopefully this work provides a useful insight into TabNet within the context of drug discovery and points out potential limitations, but also gives direction and inspiration for further research or applications involving this novel and interesting deep learning architecture.

[1] an example would be Pytorch sparse implementation `https://pytorch.org/docs/stable/sparse.html`

References

[1] *1.10. Decision Trees*. scikit-learn. URL: https://scikit-learn/stable/modules/tree.html (visited on 10/19/2021).

[2] Ami Abutbul et al. "DNF-Net: A Neural Architecture for Tabular Data". In: *arXiv:2006.06465 [cs, stat]* (June 11, 2020). arXiv: 2006.06465. URL: http://arxiv.org/abs/2006.06465 (visited on 10/12/2021).

[3] *AIDS Antiviral Screen Data - NCI DTP Data - NCI Wiki*. URL: https://wiki.nci.nih.gov/display/ncidtpdata/aids+antiviral+screen+data (visited on 12/07/2021).

[4] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[5] Lombardi Alessandro et al. *Mechanism of Action (MoA) Prediction - Kaggle Competition*. URL: https://www.epfl.ch/labs/mlo/wp-content/uploads/2021/05/crpmlcourse-paper839.pdf (visited on 10/13/2021).

[6] *Amazon S3*. In: *Wikipedia*. Page Version ID: 1057018762. Nov. 24, 2021. URL: https://en.wikipedia.org/w/index.php?title=Amazon_S3&oldid=1057018762 (visited on 12/29/2021).

[7] Sercan O. Arik and Tomas Pfister. "TabNet: Attentive Interpretable Tabular Learning". In: *arXiv:1908.07442 [cs, stat]* (Dec. 9, 2020). arXiv: 1908.07442. URL: http://arxiv.org/abs/1908.07442 (visited on 10/12/2021).

[8] *Ax · Adaptive Experimentation Platform*. URL: https://ax.dev//index.html (visited on 11/21/2021).

[9] baosenguo. *Kaggle-MoA 2nd Place Solution*. original-date: 2020-12-09T02:24:45Z. Oct. 12, 2021. URL: https://github.com/baosenguo/Kaggle-MoA-2nd-Place-Solution (visited on 10/13/2021).

[10] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58 (June 1, 2020), pp. 82–115. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2019.12.012. URL: https://www.sciencedirect.com/science/article/pii/S1566253519308103 (visited on 11/13/2021).

[11] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. New York: Springer, 2006. 738 pp. ISBN: 978-0-387-31073-2.

[12] Jock A. Blackard and Denis J. Dean. "Comparative accuracies of neural networks and discriminant analysis in predicting forest cover types from cartographic variables". In: *Second Southern Forestry GIS Conference*. 1998, pp. 189–199.

[13] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 1, 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324 (visited on 10/17/2021).

[14] Jason Brownlee. *14 Different Types of Learning in Machine Learning*. Machine Learning Mastery. Nov. 10, 2019. URL: https://machinelearningmastery.com/types-of-learning-in-machine-learning/ (visited on 10/15/2021).

[15] Chuipu Cai et al. “Deep Learning-Based Prediction of Drug-Induced Cardiotoxicity”. In: *Journal of Chemical Information and Modeling* 59.3 (Mar. 25, 2019). Publisher: American Chemical Society, pp. 1073–1084. ISSN: 1549-9596. DOI: 10.1021/acs.jcim.8b00769. URL: https://doi.org/10.1021/acs.jcim.8b00769 (visited on 06/19/2021).

[16] Paula Carracedo-Reboredo et al. “A review on machine learning approaches and trends in drug discovery”. In: *Computational and Structural Biotechnology Journal* 19 (Jan. 1, 2021), pp. 4538–4558. ISSN: 2001-0370. DOI: 10.1016/j.csbj.2021.08.011. URL: https://www.sciencedirect.com/science/article/pii/S2001037021003421 (visited on 12/07/2021).

[17] Javier Castro, Daniel Gómez, and Juan Tejada. “Polynomial calculation of the Shapley value based on sampling”. In: *Computers & Operations Research* 36.5 (2009). Publisher: Elsevier, pp. 1726–1730.

[18] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, eds. *Semi-supervised learning*. Adaptive computation and machine learning. OCLC: ocm64898359. Cambridge, Mass: MIT Press, 2006. 508 pp. ISBN: 978-0-262-03358-9.

[19] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[20] Artem Cherkasov et al. “QSAR Modeling: Where Have You Been? Where Are You Going To?” In: *Journal of Medicinal Chemistry* 57.12 (June 26, 2014). Publisher: American Chemical Society, pp. 4977–5010. ISSN: 0022-2623. DOI: 10.1021/jm4004285. URL: https://doi.org/10.1021/jm4004285 (visited on 12/07/2021).

[21] Paul Czodrowski. “hERG Me Out”. In: *Journal of Chemical Information and Modeling* 53.9 (Sept. 23, 2013), pp. 2240–2251. ISSN: 1549-9596, 1549-960X. DOI: 10.1021/ci400308z. URL: https://pubs.acs.org/doi/10.1021/ci400308z (visited on 06/19/2021).

[22] Yann N. Dauphin et al. “Language Modeling with Gated Convolutional Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 17, 2017, pp. 933–941. URL: https://proceedings.mlr.press/v70/dauphin17a.html (visited on 10/31/2021).

[23] *Daylight Theory: SMARTS - A Language for Describing Molecular Patterns*. URL: https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html (visited on 12/08/2021).

[24] *Deep Learning with Structured Data*. Manning Publications. URL: https://www.manning.com/books/deep-learning-with-structured-data (visited on 10/11/2021).

[25] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv:1810.04805 [cs]* (May 24, 2019). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 10/11/2021).

[26] Robin Duelen et al. “Medicinal Biotechnology for Disease Modeling, Clinical Therapy, and Drug Discovery and Development”. In: Aug. 17, 2019, pp. 89–128. ISBN: 978-3-030-22140-9. DOI: 10.1007/978-3-030-22141-6_5.

[27] Joseph L. Durant et al. “Reoptimization of MDL keys for use in drug discovery”. In: *Journal of Chemical Information and Computer Sciences* 42.6 (Dec. 2002), pp. 1273–1280. ISSN: 0095-2338. DOI: 10.1021/ci010132r.

[28] *entmax*. original-date: 2019-05-31T13:54:08Z. Nov. 16, 2021. URL: https://github.com/deep-spin/entmax (visited on 11/22/2021).

[29] *Explainable Artificial Intelligence*. URL: https://www.darpa.mil/program/explainable-artificial-intelligence (visited on 11/13/2021).

[30] *Exploring Multi-Objective Hyperparameter Optimization*. URL: https://blog.fastforwardlabs.com/2021/07/07/exploring-multi-objective%20-hyperparameter-optimization.html (visited on 11/21/2021).

[31] William Falcon et al. “PyTorch Lightning”. In: *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3 (2019).

[32] Jerome H. Friedman. “Greedy function approximation: A gradient boosting machine.” In: *The Annals of Statistics* 29.5 (Oct. 2001). Publisher: Institute of Mathematical Statistics, pp. 1189–1232. ISSN: 0090-5364, 2168-8966. DOI: 10.1214/aos/1013203451. URL: https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-5/Greedy-function-approximation-A-gradient-boosting-machine/10.1214/aos/1013203451.full (visited on 10/21/2021).

[33] Jonas Gehring et al. “Convolutional Sequence to Sequence Learning”. In: *Proceedings of the 34th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 17, 2017, pp. 1243–1252. URL: https://proceedings.mlr.press/v70/gehring17a.html (visited on 10/31/2021).

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[35] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: *arXiv:1406.2661 [cs, stat]* (June 10, 2014). arXiv: 1406.2661. URL: http://arxiv.org/abs/1406.2661 (visited on 12/09/2021).

[36] *google-research/tabnet at master · google-research/google-research*. GitHub. URL: https://github.com/google-research/google-research (visited on 11/15/2021).

[37] Corwin Hansch et al. “Correlation of Biological Activity of Phenoxyacetic Acids with Hammett Substituent Constants and Partition Coefficients”. In: *Nature* 194.4824 (Apr. 1962). Bandiera_abtest: a Cg_type: Nature Research Journals Number: 4824 Primary_atype: Research Publisher: Nature Publishing Group, pp. 178–180. ISSN: 1476-4687. DOI: 10.1038/194178b0. URL: https://www.nature.com/articles/194178b0 (visited on 12/08/2021).

[38] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: (Dec. 10, 2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (visited on 10/11/2021).

[39] *HIV*. In: *Wikipedia*. Page Version ID: 1058749602. Dec. 5, 2021. URL: https://en.wikipedia.org/w/index.php?title=HIV&oldid=1058749602 (visited on 12/07/2021).

[40] Tin Kam Ho. "The random subspace method for constructing decision forests". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.8 (Aug. 1998). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 832–844. ISSN: 1939-3539. DOI: 10.1109/34.709601.

[41] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997). Conference Name: Neural Computation, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[42] Elad Hoffer, Itay Hubara, and Daniel Soudry. "Train longer, generalize better: closing the generalization gap in large batch training of neural networks". In: (May 24, 2017). URL: https://arxiv.org/abs/1705.08741v2 (visited on 10/31/2021).

[43] *IC_{50}*. In: *Wikipedia*. Page Version ID: 1034669993. July 21, 2021. URL: https://en.wikipedia.org/w/index.php?title=IC50&oldid=1034669993 (visited on 12/08/2021).

[44] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: (Feb. 11, 2015). URL: https://arxiv.org/abs/1502.03167v3 (visited on 10/31/2021).

[45] Dejun Jiang et al. "Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models". In: *Journal of Cheminformatics* 13.1 (Feb. 17, 2021), p. 12. ISSN: 1758-2946. DOI: 10.1186/s13321-020-00479-8. URL: https://doi.org/10.1186/s13321-020-00479-8 (visited on 11/27/2021).

[46] Arlind Kadra et al. "Regularization is all you Need: Simple Neural Nets can Excel on Tabular Data". In: *arXiv:2106.11189 [cs]* (June 21, 2021). arXiv: 2106.11189. URL: http://arxiv.org/abs/2106.11189 (visited on 10/11/2021).

[47] Guolin Ke et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://papers.nips.cc/paper/2017/hash/6449f44%20a102fde848669bdd9eb6b76fa-Abstract.html (visited on 10/23/2021).

[48] Guolin Ke et al. "TabNN: A Universal Neural Network Solution for Tabular Data". In: (Sept. 27, 2018). URL: https://openreview.net/forum?id=r1eJssCqY7 (visited on 10/12/2021).

[49] Hyunho Kim et al. "Artificial Intelligence in Drug Discovery: A Comprehensive Review of Data-driven and Machine Learning Approaches". In: *Biotechnology and Bioprocess Engineering* 25.6 (Dec. 1, 2020), pp. 895–930. ISSN: 1976-3816. DOI: 10.1007/s12257-020-0049-y. URL: https://doi.org/10.1007/s12257-020-0049-y (visited on 12/07/2021).

[50] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Jan. 29, 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 11/21/2021).

[51] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv:1312.6114 [cs, stat]* (May 1, 2014). arXiv: 1312.6114. URL: http://arxiv.org/abs/1312.6114 (visited on 12/09/2021).

[52] Günter Klambauer et al. *Lecture Notes: Deep Learning and Neural Networks I - Winter Semester 2019*. Institute for Machine Learning, Johannes Kepler University Linz. Oct. 2019.

[53] Narine Kokhlikyan et al. *Captum: A unified and generic model interpretability library for PyTorch*. 2020. arXiv: 2009.07896 [cs.LG].

[54] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. "Revisiting Self-Supervised Visual Representation Learning". In: *arXiv:1901.09005 [cs]* (Jan. 25, 2019). arXiv: 1901.09005. URL: http://arxiv.org/abs/1901.09005 (visited on 10/16/2021).

[55] S. B. Kotsiantis. "Decision trees: a recent overview". In: *Artificial Intelligence Review* 39.4 (Apr. 1, 2013), pp. 261–283. ISSN: 1573-7462. DOI: 10.1007/s10462-011-9272-4. URL: https://doi.org/10.1007/s10462-011-9272-4 (visited on 12/09/2021).

[56] Holger Krekel et al. *pytest x.y*. 2004. URL: https://github.com/pytest-dev/pytest.

[57] Clemens Kriechbaumer. *clemens33/mlflow*. original-date: 2021-05-02T12:03:36Z. Aug. 12, 2021. URL: https://github.com/clemens33/mlflow (visited on 11/20/2021).

[58] Clemens Kriechbaumer. *clemens33/thesis-tex: Master Thesis in AI*. GitHub. URL: https://github.com/clemens33/thesis-tex (visited on 11/21/2021).

[59] Clemens Kriechbaumer. *Master Thesis (AI) - TabNet Experiments*. URL: https://mlflow.kriechbaumer.at/#/ (visited on 11/21/2021).

[60] Niklas Kühl et al. "Machine Learning in Artificial Intelligence: Towards a Common Understanding". In: *arXiv:2004.04686 [cs]* (Mar. 27, 2020). arXiv: 2004.04686. URL: http://arxiv.org/abs/2004.04686 (visited on 10/14/2021).

[61] Michael Kuhn et al. "The SIDER database of drugs and side effects". In: *Nucleic Acids Research* 44 (D1 Jan. 4, 2016), pp. D1075–1079. ISSN: 1362-4962. DOI: 10.1093/nar/gkv1075.

[62] Ray Kurzweil. *The Singularity Is Near: When Humans Transcend Biology*. New York: Penguin Books, Sept. 26, 2006. 672 pp. ISBN: 978-0-7394-6626-1.

[63] Alexander Ligthart, Cagatay Catal, and Bedir Tekinerdogan. "Analyzing the effectiveness of semi-supervised learning approaches for opinion spam classification". In: *Applied Soft Computing* 101 (Mar. 1, 2021), p. 107023. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2020.107023. URL: https://www.sciencedirect.com/science/article/pii/S1568494620309625 (visited on 10/16/2021).

[64] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable AI: A Review of Machine Learning Interpretability Methods". In: *Entropy* 23.1 (Jan. 2021). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 18. DOI: 10.3390/e23010018. URL: https://www.mdpi.com/1099-4300/23/1/18 (visited on 11/10/2021).

[65] Zachary C. Lipton. "The Mythos of Model Interpretability". In: *arXiv:1606.03490 [cs, stat]* (Mar. 6, 2017). arXiv: 1606.03490. URL: http://arxiv.org/abs/1606.03490 (visited on 11/13/2021).

[66] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *arXiv:1711.05101 [cs, math]* (Jan. 4, 2019). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101 (visited on 11/21/2021).

[67] Junshui Ma et al. "Deep neural nets as a method for quantitative structure-activity relationships". In: *Journal of Chemical Information and Modeling* 55.2 (Feb. 23, 2015), pp. 263–274. ISSN: 1549-960X. DOI: 10.1021/ci500747n.

[68] Ričards Marcinkevičs and Julia E. Vogt. "Interpretability and Explainability: A Machine Learning Zoo Mini-tour". In: *arXiv:2012.01805 [cs]* (Dec. 3, 2020). arXiv: 2012.01805. URL: http://arxiv.org/abs/2012.01805 (visited on 10/11/2021).

[69] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.%20tensorflow.org/.

[70] Andre Martins and Ramon Astudillo. "From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification". In: *Proceedings of The 33rd International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 1938-7228. PMLR, June 11, 2016, pp. 1614–1623. URL: https://proceedings.mlr.press/v48/martins16.html (visited on 11/01/2021).

[71] Ines Filipa Martins et al. "A Bayesian approach to in silico blood-brain barrier penetration modeling". In: *Journal of chemical information and modeling* 52.6 (2012). Publisher: ACS Publications, pp. 1686–1697.

[72] Andreas Mayr et al. "DeepTox: Toxicity Prediction using Deep Learning". In: *Frontiers in Environmental Science* 3.80 (2016).

[73] John McCarthy. "What Is Artificial Intelligence?" In: (Nov. 1, 2007). URL: http://jmc.stanford.edu/articles/whatisai/whatisai.pdf (visited on 10/14/2021).

[74] Marvin Minsky and Seymour Papert. *Perceptrons*. Perceptrons. Oxford, England: M.I.T. Press, 1969.

[75] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[76] *MLflow - A platform for the machine learning lifecycle*. MLflow. URL: https://mlflow.org/ (visited on 11/20/2021).

[77] *OpenSMILES specification*. URL: http://opensmiles.org/opensmiles.html (visited on 12/08/2021).

[78] Sebastian Palacio et al. "XAI Handbook: Towards a Unified Framework for Explainable AI". In: *arXiv:2105.06677 [cs]* (May 14, 2021). arXiv: 2105.06677. URL: http://arxiv.org/abs/2105.06677 (visited on 11/12/2021).

[79] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[80] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[81] Ben Peters, Vlad Niculae, and André F. T. Martins. “Sparse Sequence-to-Sequence Models”. In: *arXiv:1905.05702 [cs]* (June 12, 2019). arXiv: 1905.05702. URL: http://arxiv.org/abs/1905.05702 (visited on 11/01/2021).

[82] Sergei Popov, Stanislav Morozov, and Artem Babenko. “Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data”. In: *arXiv:1909.06312 [cs, stat]* (Sept. 19, 2019). arXiv: 1909.06312. URL: http://arxiv.org/abs/1909.06312 (visited on 10/11/2021).

[83] Jangampalli Adi Pradeepkiran et al. “Protective effects of BACE1 inhibitory ligand molecules against amyloid beta-induced synaptic and mitochondrial toxicities in Alzheimer’s disease”. In: *Human Molecular Genetics* 29.1 (Jan. 1, 2020), pp. 49–69. ISSN: 0964-6906. DOI: 10.1093/hmg/ddz227. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7001603/ (visited on 11/16/2021).

[84] Liudmila Prokhorenkova et al. “CatBoost: unbiased boosting with categorical features”. In: *arXiv:1706.09516 [cs]* (Jan. 20, 2019). arXiv: 1706.09516. URL: http://arxiv.org/abs/1706.09516 (visited on 10/23/2021).

[85] J. R. Quinlan. “Induction of decision trees”. In: *Machine Learning* 1.1 (Mar. 1, 1986), pp. 81–106. ISSN: 1573-0565. DOI: 10.1007/BF00116251. URL: https://doi.org/10.1007/BF00116251 (visited on 10/19/2021).

[86] Hubert Ramsauer et al. “Hopfield Networks is All You Need”. In: *arXiv:2008.02217 [cs, stat]* (Dec. 22, 2020). arXiv: 2008.02217. URL: http://arxiv.org/abs/2008.02217 (visited on 04/06/2021).

[87] David Rogers and Mathew Hahn. “Extended-Connectivity Fingerprints”. In: *Journal of Chemical Information and Modeling* 50.5 (May 24, 2010). Publisher: American Chemical Society, pp. 742–754. ISSN: 1549-9596. DOI: 10.1021/ci100050t. URL: https://doi.org/10.1021/ci100050t (visited on 12/08/2021).

[88] Stuart J. Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Englewood Cliffs, N.J: Prentice Hall, 1995. 932 pp. ISBN: 978-0-13-103805-9.

[89] Omer Sagi and Lior Rokach. “Ensemble learning: A survey”. In: *WIREs Data Mining and Knowledge Discovery* 8.4 (2018), e1249. ISSN: 1942-4795. DOI: 10.1002/widm.1249. URL: https://onlinelibrary.wiley.com%20/doi/abs/10.1002/widm.1249 (visited on 10/17/2021).

[90] Johannes Schimunek et al. *Poster: Comparative assessment of interpretability methods of deep activity models for hERG*. In 19th International Workshop on (Quantitative) Structure-Activity Relationships in Environmental and Health Sciences. June 2021.

[91] Burr Settles. *Active Learning Literature Survey*. Technical Report. Accepted: 2012-03-15T17:23:56Z. University of Wisconsin-Madison Department of Computer Sciences, 2009. URL: https://minds.wisconsin.edu/handle/1793/60660 (visited on 10/16/2021).

[92] Lloyd S Shapley. *Notes on the N-person Game–I: Characteristic-point Solutions of the Four-person Game*. Rand Corporation, 1951.

[93] Ira Shavitt and Eran Segal. "Regularization Learning Networks: Deep Learning for Tabular Datasets". In: *arXiv:1805.06440 [cs, stat]* (Oct. 23, 2018). arXiv: 1805.06440. URL: http://arxiv.org/abs/1805.06440 (visited on 10/12/2021).

[94] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning Important Features Through Propagating Activation Differences". In: *Proceedings of the 34th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 17, 2017, pp. 3145–3153. URL: https://proceedings.mlr.press/v70/shrikumar17a.html (visited on 11/14/2021).

[95] Ravid Shwartz-Ziv and Amitai Armon. "Tabular Data: Deep Learning is Not All You Need". In: *arXiv:2106.03253 [cs]* (June 6, 2021). arXiv: 2106.03253. URL: http://arxiv.org/abs/2106.03253 (visited on 10/11/2021).

[96] Manish Sihag. *A beginner's guide for understanding Extended-Connectivity Fingerprints(ECFPs)*. ChemicBook. Mar. 25, 2021. URL: https://chemicbook.com/2021/03/25/a-beginners-guide-for-understanding-extended-connectivity-fingerprints.html (visited on 12/08/2021).

[97] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *arXiv:1312.6034 [cs]* (Apr. 19, 2014). arXiv: 1312.6034. URL: http://arxiv.org/abs/1312.6034 (visited on 11/14/2021).

[98] *State of Data Science and Machine Learning 2020*. URL: https://www.kaggle.com/kaggle-survey-2020 (visited on 10/11/2021).

[99] *Structured vs Unstructured Data 101: Top Guide*. Datamation. May 21, 2021. URL: https://www.datamation.com/big-data/structured-vs-unstructured-data/ (visited on 10/11/2021).

[100] Govindan Subramanian et al. "Computational Modeling of Beta-Secretase 1 (BACE-1) Inhibitors Using Ligand Based Approaches". In: *Journal of Chemical Information and Modeling* 56.10 (Oct. 24, 2016). Publisher: American Chemical Society, pp. 1936–1949. ISSN: 1549-9596. DOI: 10.1021/acs.jcim.6b00290. URL: https://doi.org/10.1021/acs.jcim.6b00290 (visited on 11/15/2021).

[101] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic Attribution for Deep Networks". In: *arXiv:1703.01365 [cs]* (June 12, 2017). arXiv: 1703.01365. URL: http://arxiv.org/abs/1703.01365 (visited on 11/14/2021).

[102] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[103] *UCI Machine Learning Repository: Covertype Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/covertype (visited on 06/20/2021).

[104] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *Journal of Chemical Information and Computer Sciences* 28.1 (Feb. 1, 1988). Publisher: American Chemical Society, pp. 31–36. ISSN: 0095-2338. DOI: 10.1021/ci00057a005. URL: https://pubs.acs.org/doi/abs/10.1021/ci00057a005 (visited on 12/08/2021).

[105] Zhenqin Wu et al. "MoleculeNet: A Benchmark for Molecular Machine Learning". In: *arXiv:1703.00564 [physics, stat]* (Oct. 25, 2018). arXiv: 1703.00564. URL: http://arxiv.org/abs/1703.00564 (visited on 11/15/2021).

[106] Fuzhen Zhuang et al. “A Comprehensive Survey on Transfer Learning”. In: (June 23, 2020). arXiv: 1911.02685. URL: http://arxiv.org/abs/1911.02685 (visited on 10/16/2021).

List of Figures

List of Tables

List of Acronyms

AGI artificial general intelligence

AI artificial intelligence

AIDS acquired immunodeficiency syndrome

ANN artificial neural network

AUROC area under the receiver operating characteristics

BBBP blood-brain-barrier penetration

BN batch normalization

CV computer vision

DL deep learning

DNN deep neural network

DT decision tree

EI expected improvement

FC fully connected

GAN generative adversarial network

GBDT gradient boosting decision tree

GLU gated linear unit

HIV human immunodeficiency viruses

IG integrated gradients

k-NN k-nearest neighbors

LSTM long short-term memory

ML machine learning

MLP multilayer perceptron

NLP natural-language processing

NN neural network

PoI probability of improvement

QSAR quantitative structure-activity relationship

ReLU rectified linear unit

RF random forest

RL reinforcement learning

RNN recurrent neural network

SIDER side effect ressource

SML supervised machine learning

SVM support vector machine

UML unsupervised machine learning

VAE variational autoencoder

XAI explainable artificial intelligence

Statutory Declaration

I hereby declare under oath that the submitted Master Thesis has been written solely by me without any third-party assistance, information other than the provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited. The submitted document here present is identical to the electronically submitted text document.

Dörnbach, January 10th 2022

location, date

(signature)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Dörnbach, 10. Jänner 2022

.....

Ort, Datum

.....

(Unterschrift)