Differentiably Discovering Sets of Rules

Luis N. J. Paulus
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
luis.paulus@cispa.de

Jonas Fischer
Max Planck Institute for Informatics
Saarbrücken, Germany
jonas.fischer@mpi-inf.mpg.de

Jilles Vreeken
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
vreeken@cispa.de

Abstract

Association rule mining is a great way to gain insight into complex data, such as single-cell gene expression data, as it allows discovering conditional dependencies of the kind '*In samples where high expression of gene A and B are observed, C and D tend to be expressed too*'. Or at least, in theory. In practice, traditional methods for mining association rules tend to overwhelm the user with overly many, highly redundant, and mostly spurious results, while modern approaches that fix those problems do not scale to high-dimensional data. In this paper, we propose an end-to-end differentiable approach for mining high quality sets of rules that does scale to hundreds of thousands of features.

In particular, we suggest an inherently interpretable autoencoder that we call RuleNaps. Its hidden layer consists of two modules that together encode the found association rules. The first module encodes the most important *associations* between features (e.g., *ABCD*) while the second splits these into *conditional dependencies* (e.g., $AB \rightarrow CD$). By binarizing the weights in the forward pass, each neuron has a symbolic interpretation that allows us to trivially read out the rules. By considering the weights again as continuous-valued in the backward pass, we can use back-propagation to efficiently learn good rules even from very high-dimensional data. Experiments on synthetic and real-world data show that RuleNaps performs very well, outperforming the state of the art by a wide margin. In a case study, we confirm it also delivers in practice, revealing valuable insight into high-dimensional breast cancer data.

CCS Concepts

• **Information systems** → **Data mining**; • **Computing methodologies** → **Machine learning**.

Keywords

Association Rule Mining, Neural Networks, Interpretability

ACM Reference Format:

Luis N. J. Paulus, Jonas Fischer, and Jilles Vreeken. 2026. Differentiably Discovering Sets of Rules. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD 2026), August 9–13, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3770855.3817887

Resource Availability:

The source code of this paper has been made publicly available at https://doi.org/10.5281/zenodo.20508891.

KDD 2026, Jeju Island, Republic of Korea.

ACM ISBN 979-8-4007-2259-2/2026/08
https://doi.org/10.1145/3770855.3817887

1 Introduction

Suppose we are given a high-dimensional binary dataset on the presence or absence of genetic variants (columns) for different individuals (rows). How can we gain insight into the most salient conditional dependencies? At first glance, association rule mining [1], the classical data mining task of discovering rules $A \rightarrow B$ expressing items B are likely to appear in the context of items A, seems the right answer. In practice, however, existing methods fail to answer this question. Traditional approaches, which aim to discover *all* association rules $A \rightarrow B$ that pass a minimum support and a minimum confidence threshold [2, 25, 39, 40], tend to produce overwhelmingly many, mostly redundant and spurious rules [19]. Modern approaches, solve this pattern explosion by asking for a small *set of patterns* that generalize the data [7, 13]. These methods work well, but by their combinatorial nature, do not scale beyond a few thousand features, let alone hundreds of thousands.

In this paper, we take advantage of recent breakthroughs in differentiable pattern discovery [9, 14, 32] and propose the first continuously optimizable yet inherently interpretable approach to mining high-quality sets of association rules from high-dimensional data. In particular, we propose RuleNaps, a binarized autoencoder that we can efficiently optimize using back-propagation. The hidden layer of RuleNaps consists of two different modules, which together encode association rules $A \rightarrow B$. The first module encodes the most important *associations* in the data, e.g. $A \cup B$, and the second splits these into *conditional dependencies*, e.g. $A \rightarrow B$, such that the reconstruction loss of the input data is minimized.

To ensure the neurons in the hidden layer can be unambiguously interpreted, we use the same weights in the encoder and decoder. During the forward pass, we binarize these weights by interpreting them as Bernoulli random variables. This gives a crisp, symbolic interpretation of the hidden neurons, but also allows computing a meaningful loss for binary data. During the backward phase, we consider the weights as continuous-valued, which allows us to back-propagate errors. After training, we can straightforwardly read the discovered set of rules from the hidden layer. Altogether, RuleNaps is an elegant and efficient approach to a difficult, triply-exponentially-sized combinatorial problem.

We show that RuleNaps performs well in practice through an extensive set of experiments. On synthetic data it closely recovers the ground truth, outperforming state-of-the-art combinatorial and neural methods by a wide margin. It is robust to different sizes and frequencies of heads and tails, to noise, as well as varying dimensionality. On real world data, we show that unlike its competitors, RuleNaps returns reasonable numbers of rules, and does scale to

data with over 225 000 features. Finally, and most importantly, in a case study on breast cancer data, we show that RuleNaps reveals valuable insights, where it recovers rules that provide valuable new rules that align with domain knowledge and are promising directions for future study.

The main contributions of this paper are as follows.

(a) We propose RuleNaps, a differentiable binarized autoencoder for learning symbolic conditional dependencies.
(b) We show how to efficiently learn RuleNaps, how to tune hyperparameters, and how to read out rules after learning.
(c) We give an extensive set of experiments on synthetic and real-world data, comparing to five state of the art methods.

We postpone the pseudo-code and details on the experimental setup to the appendix. We make all code available online.[1]

2 Notation

We consider binary data $X \in \{0,1\}^{n \times m}$ with n rows (transactions) and m features (items). We refer to a single data entry in row i and column j as X_{ij}. A sample, or transaction $x_i \in X$ describes which features are present in row i, e.g., which genes are active given a sample from gene-expression data.

We want to find a set of rules R that together describe the most important conditional dependencies in the data. Formally, we define a rule as $A \to B$, where $A \subset \{1, 2, \ldots, m\}$, $B \subseteq \{1, 2, \ldots, m\}$, and $A \cap B = \emptyset$. We call A the antecedent or head and B the consequent or tail. $A \to B$ means the presence of features with indices in A increases the likelihood of the presence of features with indices in B.

Rules with an empty head, $\emptyset \to B$, capture unconditional dependencies. We call these patterns. The *support* of a pattern B is

$$supp(B) = \left|\left\{i \in \{1, \ldots, n\} \;\middle|\; \forall j \in B : X_{ij} = 1\right\}\right| .$$

and we define the *frequency* of a pattern B as

$$freq(B) = \frac{supp(B)}{n} .$$

The support of a rule, $A \to B$, corresponds to the support of the pattern $A \cup B$. For a given rule $A \to B$, we define its *confidence* by

$$conf(A \to B) = \frac{supp(A \cup B)}{supp(A)} ,$$

revealing how likely it is to see the tail in a given transaction when knowing that the head is present.

We use W to denote a matrix and write $(W)_{ij}$ for the element at row i and column j. For multivariate functions, we write f and q. We denote the only univariate function we use with λ and ease notation by defining its multivariate extension as $\lambda\big((y_1, \ldots, y_h)^\top\big) = \big(\lambda(y_1), \ldots, \lambda(y_h)\big)^\top$ for multi-dimensional inputs $y \in \mathbb{R}^h$. By $\frac{\partial f}{\partial W}$ we denote the partial derivative of a function f with respect to its weight parameters W.

3 Related Work

Pattern Mining. Association rule mining traditionally focuses on extracting *all* association rules from data that pass a minimum support and confidence threshold [2, 20, 39]. For non-trivial thresholds, this tends to result in overwhelmingly many, highly redundant, and mostly spurious rules [19]. Significant pattern mining [18, 34] employs statistical tests to filter out spurious patterns, but, as it considers rules individually, it cannot determine redundancy.

Pattern Set Mining. Pattern set mining [31] solves the problems of redundancy and spuriousness of results by instead asking for a *set of patterns* that together optimize a global criterion [8, 15, 29]. Grab [13] instantiates this idea for association rules, using combinatorial search to optimize a Minimum Description Length [17] criterion. As we will see in the experiments, Grab performs well, but does not scale to high-dimensional data.

Neural Rule Mining. A step closer towards our goal, Berteloot et al. [6] recently proposed Arm-ae, a two-step neural approach for mining association rules. They first train an autoencoder, from which they then extract rules by inferring which single feature (rule tail) has the strongest dependency on what set of input features (rule head). Karabulut et al. [22] proposed a similar approach in which the model is also probed for dependencies to extract a set of rules. Unlike RuleNaps, both are inherently limited to rules with single-item tails.

Neural Pattern Set Mining. BinaPs [14] is a neural approach to pattern set mining. BinaPs is essentially an autoencoder, designed such that the hidden layer can be interpreted as a set of itemsets. Walter et al. [32] and Chataing et al. [9] extend this idea to the supervised setting for mining class-related patterns. Our architecture is also based on an autoencoder, but designed such that it can model *conditional* dependencies across all features, not just a class label, i.e. association rules. We compare to BinaPs in the experiments.

Rule-Based Classification. Complementary to our setting, there exists work on rule-based classification. Montañez et al. [26], for example, combine classical association rule mining with stacked autoencoders and multilayer perceptrons. Decision rules [12, 23, 28, 33, 37] and rule lists [10, 24, 35, 36] both use rules where the tails are the target classes to obtain interpretable and well-performing classifiers. These methods are by design restricted to the supervised setting, and therefore out of scope for this paper.

Positioning. RuleNaps is highly scalable, inherently interpretable, and allows mining association rules without imposing length constraints on the head or tail, does not require post-processing, and is applicable to unsupervised data.

4 RuleNaps

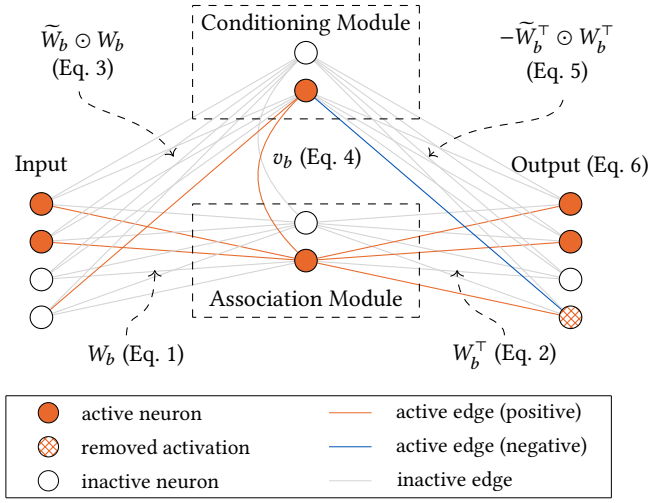
Next, we present RuleNaps, an interpretable neural approach for association rule mining. We first give the intuition, then the details.

4.1 In a Nutshell

Our goal is to mine a set of relevant rules R in a given binary dataset. The resulting rules should be non-redundant, so multiple rules should not encode the same information about the data. Additionally, we want to avoid finding rules that are spurious in the way that they contain co-occurrences of features that are random and not due to an underlying dependence. Most importantly, mining should be scalable to high-dimensional datasets.

To accomplish this, we propose RuleNaps, a neural architecture based on a differentiable autoencoder. It consists of two modules,

[1] https://eda.group/rulenaps/



$$W = \begin{bmatrix} 0.1 & 0.1 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.1 & 0.4 \end{bmatrix} \xrightarrow{\mathcal{B}(\cdot)} W_b = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

$$v^\top = \begin{bmatrix} 0.9 & 0.8 \end{bmatrix} \xrightarrow{\mathcal{B}(\cdot)} v_b^\top = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$\widetilde{W} = \begin{bmatrix} 0.9 & 0.9 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.9 & 0.6 \end{bmatrix} \xrightarrow{\mathcal{B}(\cdot)} \widetilde{W}_b = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\text{masked encoder weight: } \widetilde{W}_b \odot W_b = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{masked decoder weight: } -\widetilde{W}_b \odot W_b = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Rules: $R^1 = \{\emptyset \longrightarrow \{3,4\}\}$ resp. $R^2 = \{\{1,2\} \longrightarrow \{4\}\}$

Figure 1: *Left:* A small example for the architecture of RuleNaps with a hidden layer of size 2 for data with 4 features. The network weights associated with the second neuron encode the rule $\{1,2\} \longrightarrow \{4\}$. The second neuron in the Association Module reconstructs the pattern $\{1,2,4\}$ although feature 4 is not present. The corresponding neuron in the Conditioning Module detects the absence of 4 and removes the reconstructed item in the output. The network eventually only reconstructs the rule head $\{1,2\}$. *Right:* The continuous and binarized weights of the Conditioning Module, the Association Module, and the connection between them (note that $\widetilde{W} = 1 - W$), as well as the masked weights during the forward pass in the Conditioning Module and the rules extracted from the weight matrix W.

both of which have an encoding and a decoding layer (cf. Figure 1). RuleNaps is trained to reconstruct the input while being limited by a bottleneck (Association Module). At the same time, it is trained to encode conditional dependencies (Conditioning Module) through a bottleneck that takes input features and patterns learned in the Association Module to reconstruct features conditionally. The careful design of the modules allows us to directly extract interpretable rules from the trained weights.

A rule $A \longrightarrow B$ is then encoded in the following way. If an input sample contains a sufficiently large part of $A \cup B$, the Association Module will reconstruct $A \cup B$. The Conditioning Module will remove B from this reconstruction whenever too few elements of B are present in the input. For example, when only the head of the rule, A is present in the input, the Association Module generates $A \cup B$, the Conditioning Module removes B, such that in the end only A is reconstructed.

At its core, the Association Module follows the layout of BinaPs [14]. During the forward pass, the weights of each neuron should represent which features are part of the pattern encoded by the neuron. To achieve this symbolic interpretation, we model the weights as Bernoulli variables and sample binary values in the forward pass. The greater the weight, the higher the chance we sample a 1, meaning the connected feature is part of the pattern. We then use the transposed version of the weights W^T for decoding to ensure there is no disambiguity between how patterns are encoded and decoded. This design incentivizes the network to encode co-occurring features through the same neuron, leading to a more stable activation and reconstruction under the stochastic binarization. Distributing the encoding of such features across different neurons would require several stochastic events to align for the desired reconstruction.

The Association Module can only pick up associations and thus always reconstructs $A \cup B$. However, as the rule tail B is not always present, we need a second module to account for that. The *Conditioning Module* removes items in B from the reconstruction if the items in B are not present in the input (while items in A are). This module contains a mirrored version of each neuron in the Association Module, reusing the weights from the Association Module but now consider them the other way around: the lower the weight, the higher the chance we sample a 1. When computing the Conditioning Module, we also flip the binary input so that if an item is reconstructed by the Conditioning Module, this item is not present in the original input. So, if items in B are not present in the input, but they are part of a pattern $A \cup B$ (a neuron in the Association Module), the Conditioning Module's corresponding neuron will activate and mask out B in the final reconstruction. We provide an example computation in Figure 1.

4.2 Architecture

Association Module. For the Association Module, which consists of a single, modified linear layer, we use continuous weight parameters $W \in [0,1]^{h \times m}$, where h is the size of the hidden layer. For the output of this layer, we use the transposed version $W^\top$ of the weights. The weights associated with a neuron in the hidden layer represent a pattern where $(W_b)_{ij} = 1$ means that feature j is present in the pattern encoded by neuron i.

Conditioning Module. In the Conditioning Module, we follow a similar architecture, with a linear layer of same size as in the

Association Module, using weights $\widetilde{W} = 1 - W$. Here, W is the weight matrix from the Association Module. The input to this module are the flipped versions of the input mapping input 1s to 0s and vice versa. The weights of each neuron of this module are, by construction, the rule tail.

We denote the weight vector that connects the bottleneck neurons between the Association and Conditioning Module with $v \in [0,1]^h$ and its binarized version with $v_b \in \{0,1\}^h$.

We ensure that the minimal values of W and $\widetilde{W}$, respectively v, are set to $\frac{1}{m}$, respectively $\frac{1}{h}$, after each weight update. This way, at least one 1 will be sampled on expectation in v and per row in W to avoid the model getting stuck in a local minimum.

4.3 Optimization

Forward Pass. In the forward pass through the Association Module, we first create a binarized version W_b of W by independently sampling from Bernoulli distributions where the weights define the probability of sampling a one, i.e. $(W_b)_{ij} \sim \mathcal{B}\big((W)_{ij}\big)$ for all i, j. We define the non-linear activation function λ by

$$\lambda(x) = \text{round}\big(\text{clamp}(x, 0, 1)\big) \,,$$

where

$$\text{clamp}(x, a, b) = \begin{cases} a & \text{if } x < a \,, \\ x & \text{if } a \leq x \leq b \,, \\ b & \text{if } b < x \end{cases}$$

is the clamping function as defined by Fischer and Vreeken [14]. Using the binarized weights W_b, we compute the encoding $z \in \{0,1\}^h$ for input $x \in \{0,1\}^m$ as

$$z = \lambda\Big(f_{W_b}(x)\Big) \,, \tag{1}$$

where $f_W(x) = Wx$ is the standard linear layer operation. We decode z by computing

$$\hat{x} = \lambda\Big(f_{W_b^\top}(z)\Big) \,. \tag{2}$$

In the forward pass through the Conditioning Module, we first set the weights to be flipped versions of the weights of the Association Module, $\widetilde{W} = 1 - W$. The core idea is that rule tail items B will have lower weights (e.g., close to 0.5 in the toy example in Figure 1) in the Association Module, hence can be active in both the Association Module and the Conditioning Module when flipping the weights. The reason for these lower weights is that during learning, negative training signals will dampen the weights of B for a pattern of $A \cup B$ whenever A is there but B is not, separating head from tail items by weight magnitude.

We then perform binarization to obtain $\widetilde{W}_b$ similarly as above, using the weights to initialize Bernoulli variables to sample from. Finally, for the computation of the Conditioning Module we mask the weights of $\widetilde{W}_b$ by W, so only those items actually part of $A \cup B$ will be propagated.

More formally, we define a masked linear layer as

$$q_{W,M}(x) = (M \odot W)x \,,$$

where $x \in \{0,1\}^m$ denotes an input, $W \in \mathbb{R}^{h \times m}$ a weight, and $M \in \mathbb{R}^{h \times m}$ a mask. The output $y \in \{0,1\}^h$ of the encoder function is then defined as

$$y = \lambda\Big(q_{\widetilde{W}_b, W_b}(1 - x)\Big) \,. \tag{3}$$

An example of this process is visualized on the right-hand side in Figure 1.

In this example, we depict the weights of a toy network with two neurons and their binarized versions. We see that the masking eliminates the third feature in the second row of the weights of the Conditioning Module $\widetilde{W}_b$. That is because the corresponding weight of the Association Module $(W_b)_{23}$ is 0, meaning that the second neuron does not depict any association with this feature. Thus, it makes sense to mask this feature in the weights of the Conditioning Module.

We add an additional dependence of the neurons in the association layer to the neurons in the conditioning layer by adding connections from the ith neuron in the Association Module to the ith neuron in the Conditioning Module for all $i \in \{1, \ldots, h\}$. We compute the neuron activation in the bottleneck of the conditioning layer as

$$\tilde{y} = y \odot (v_b \odot z) \,, \tag{4}$$

where the conditioning encoder output y is multiplied with the Hadamard product of v_b and the association encoder output z. To decode $\tilde{y}$, we compute

$$\tilde{x} = \lambda\Big(q_{-\widetilde{W}_b^\top, W_b^\top}(\tilde{y})\Big) \,, \tag{5}$$

using the transposed versions of the weights. Note that here we have a negative matrix, i.e., all 1s are mapped to -1, such that this module is able to *remove* items from the rule tail that have been wrongly reconstructed by the Association Module.

For the final reconstruction we simply add up the outputs of the two modules as

$$x' = \hat{x} + \tilde{x} \,. \tag{6}$$

Concluding the forward pass, we can now compute a loss function.

Objective Function. We optimize both modules jointly by weighing False Negative respectively False Positive errors by α resp. $(1 - \alpha)$, namely

$$L_\alpha(x, x') = \sum_{j=1}^{m} \big(\alpha(1 - x_j) + (1 - \alpha)x_j\big)(x'_j - x_j)^2 \,,$$

where $x \in \{0,1\}^m$ denotes the original data sample, $x' \in \{0,1\}^m$ the autoencoder output, and $\alpha \in (0, \frac{1}{2}]$ a hyperparameter for weighing different types of errors. Additionally, we use two regularizers. To regularize against long patterns, we define a penalty r_h over the weights in the row-wise (*h*orizontal) direction. Conversely, to regularize against patterns that share features with other patterns, we define a penalty r_v over the weights in the column-wise (*v*ertical) direction. With these two regularizers, we can calibrate the rule length and the amount of feature overlap between the rules we want to have. Formally, we define them as

$$r_h(W) = \sum_i^h \left(\sum_j^m (W)_{ij} - \frac{1}{m} \right)^2 \,, \quad r_v(W) = \sum_j^m \left(\sum_i^h (W)_{ij} - \frac{1}{m} \right)^2$$

for weight matrix W [32].

Putting everything together, our final objective function is

$$\mathcal{L}_{\alpha,\beta,\gamma}(X, X', W) = \sum_{i=1}^{n} L_\alpha(X_i, X_i') + \beta r_h(W) + \gamma r_v(W) \,,$$

where X is the input, X' is the final output, and α, β, γ are hyperparameters that allow to control the impact of different error types and the regularizers.

Backward Pass. In practice, we use gradient descent on the previously defined objective to optimize our autoencoder parameters W. Since our forward pass uses binarized versions of the weight matrix, we resort to the straight-through estimator (STE) [4] that has proven valuable for such symbolic autoencoder architectures before [14, 32].

In particular, for a linear layer we obtain partial derivatives with respect to input x and weights W by

$$\frac{\partial f_{W_b}}{\partial x} = W^\top g \qquad \text{and} \qquad \frac{\partial f_{W_b}}{\partial W} = g x^\top \,,$$

where g denotes the upstream gradient and the derivative is taken w.r.t. to the *continuous* W, ignoring the binarization that was performed on top of it during the forward pass (STE).

The derivatives of the masked linear layer operation in the Conditioning Module are analogously defined by

$$\frac{\partial q_{W_b,M}}{\partial x} = (M \odot W)^\top g \qquad \text{and} \qquad \frac{\partial q_{W_b,M}}{\partial W} = g x^\top$$

given mask M. For propagating gradients backwards through activation functions, we again use the STE.

Hyperparameter Tuning. On top of the usual hyperparameters learning rate, number of epochs, and batch size, RuleNaps has four additional hyperparameters, namely α, β, γ, and the size of the hidden layer h. To find good values for them, we validate our model on a hold-out set consisting of 10% of the data, optimizing for lowest equally-weighted reconstruction loss. For these validation runs, we eliminate the stochasticity of the network weights and use deterministic versions. In particular, we binarize the weight matrices W and $\widetilde{W}$ as we do in the forward pass, but set the connection weights v between the two modules to 1. We do so because during validation, we want every non-present tail to be removed if the head is present in the input. For more specific information on hyperparameter tuning, we refer to Appendix A.2.

Complexity. Given a a dataset with n rows and m columns and training the model with a hidden layer size of h for t iterations, the resulting complexity is $O(tmnh)$. This holds when using a simple matrix multiplication algorithm. To relate this to the traditional search and enumeration-based algorithms, we calculate the number of possible sets of rules. Using the binomial formula, we calculate the number of possible rules as $\sum_{k=2}^{m} \left(\binom{m}{k} \cdot 2^k \right) = 3^m$ and thus the number of possible rule sets as 2^{3^m} [13]. Besides the stark difference of achieving a polynomial runtime through RuleNaps opposed to the exponential runtime of many of the traditional works, more importantly our algorithm natively works on GPUs, since it is primarily consisting of matrix multiplication. This allows us to scale to data with hundreds of thousands of features in this unsupervised setting, that even polynomial time heuristics, such as Grab, fail to work on.

4.4 Rule Extraction

A key property of our architecture is that after training, we can extract a set of symbolic rules directly from the model weights. For that, we first create a matrix $P \in \{0,1\}^{h \times m}$ containing the discovered patterns by binarizing W with a *pattern extraction threshold* $\tau \in [0, 1)$. We formally define the values of P by

$$(P)_{ij} = \begin{cases} 1 & \text{if } (W)_{ij} > \tau \\ 0 & \text{otherwise} \end{cases} \,,$$

yielding a matrix that contains the binary encoded patterns as rows. Next, we compute

$$W_P = P \odot W \,,$$

where we use P to mask the continuous weights. In W_P, each row contains the continuous weights for features of the found patterns, and 0 for features that are not relevant.

To extract rules, we have to distinguish for each 1 in W_P if the corresponding feature belongs to the head or the tail. We achieve this using per-rule thresholds that we compute from the model to split up the weights in two groups. We define $\rho^+, \rho^- \in \mathbb{R}^h$ by

$$\rho_i^{\pm} = \max_j \left((W_P)_{ij} \right) \pm \widehat{\min_j} \left((W_P)_{ij} \right) \,,$$

where $\widehat{\min}(\cdot)$ gives the minimum without considering the masked 0-values. For neuron i, we use the exact middle between the maximal and minimal weights $\frac{\rho_i^+}{2}$ as the threshold that differentiates between head and tail feature. If $(W_P)_{ij}$ is below this threshold, feature j is part of the tail of rule i, and else, it is part of the head.

We then extract the head features from neuron i by

$$A_i = \begin{cases} \left\{ j \,\middle|\, (W_P)_{ij} > \frac{\rho_i^+}{2}, j \in \{1, \ldots, m\} \right\} & \text{if } \rho_i^- > \delta \\ \emptyset & \text{otherwise} \end{cases}$$

and analogously the corresponding tail features by

$$B_i = \begin{cases} \left\{ j \,\middle|\, (W_P)_{ij} < \frac{\rho_i^+}{2} \wedge (W_P)_{ij} \neq 0, j \in \{1, \ldots, m\} \right\} & \text{if } \rho_i^- > \delta \\ \left\{ j \,\middle|\, (W_P)_{ij} > 0 \in \{1, \ldots, m\} \right\} & \text{otherwise} \end{cases}$$

with the *negligibility threshold* $\delta \in [0, 1]$ that specifies the difference between the weights necessary to be considered as rules instead of patterns. If the gap is smaller than δ, we consider the result to be negligible and just extract a rule of the form $\emptyset \rightarrow p$ for pattern p in the respective row.

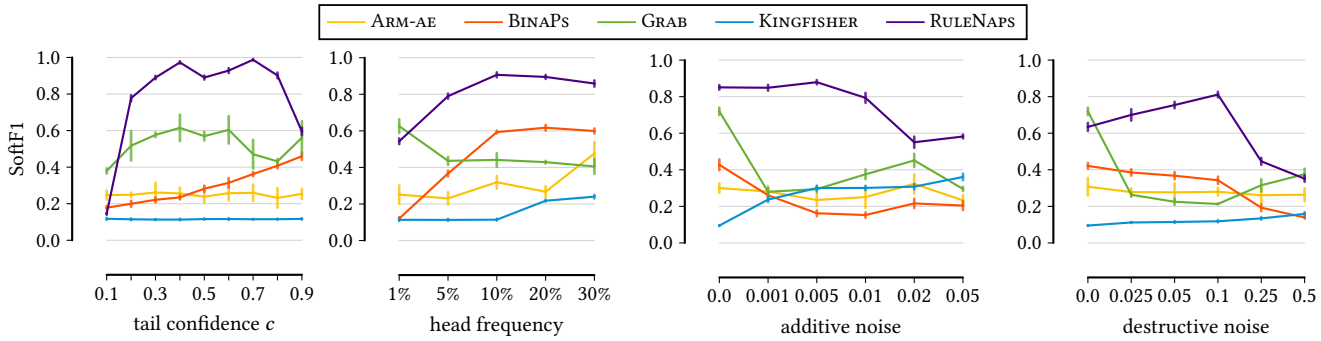
We finally define the set of discovered rules R by

$$R = \left\{ A_i \rightarrow B_i \,\middle|\, B_i \neq \emptyset, i \in \{1, \ldots, h\} \right\} \,,$$

omitting rules where the head and tail are both empty. In this way, we can extract rules of the form $A \rightarrow B$ for each row in the weight matrix W. We do not use v for the rule extraction. We show an example of the rule extraction process in Figure 4 in the appendix.

5 Experiments

In this section, we empirically evaluate RuleNaps. We compare it to five other state-of-the-art algorithms, namely Arm-ae [6], a neural association rule miner, BinaPs [14], a neural pattern miner, Grab [13], a combinatorial association rule miner, and Kingfisher [18], a significant association rule miner. In addition, we compare to Aerial+ [22], a recently proposed neural association



(a) F1 score (↑ better) **(b) F1 score (↑ better)** **(c) F1 score (↑ better)** **(d) F1 score (↑ better)**

Figure 2: *Robustness:* To evaluate robustness, we test four different settings in which we vary the tail confidence of the planted rules (a), the head frequency of the planted rules (b), the amount of additive noise (c), and the amount of destructive noise (d). We show the results in terms of F1 score, averaged over five datasets per configuration. The results of RULENAPS are better than those of the other algorithms in almost all settings.

rule miner, but found it tends to return overly many rules. To not clutter presentation, we postpone its results to the Appendix.

For AERIAL+, ARM-AE, BINAPS, and RULENAPS we use the same GPU setup as for ours, if permitted by the implementation. For details regarding the experimental setup and parameterization of the algorithms, we refer to Appendix A.2. We make our code available for research purposes.[2]

5.1 Synthetic Data

We first evaluate RULENAPS on data with known ground truth. To this end we generate data as follows.

Data Generation. We create a data matrix consisting of $n = 1000$ rows and $m = 1000$ columns, containing only zeros initially. We draw the lengths of the heads and tails for 200 rules from a Poisson distribution with $\lambda_h = \lambda_t = 1.5$, ensuring a minimal length of 1 for head and tail each. Per rule, we choose only unused features to avoid creating new rules from overlapping features. We sample the confidence c for each rule from a uniform distribution with the interval $[0.5, 1.0]$. For each rule, we randomly sample $freq = 5\%$ of the rows and set the head features to 1, and in $100c\%$ of those rows, we set the features of the corresponding tail to 1. We thus allow multiple rules per row. Afterward, we apply additive noise by flipping $\sigma_{add} = 0.01\%$ of the zeros in the data, and destructive noise by flipping $\sigma_{destr} = 0.1\%$ of the ones corresponding to a planted rule. To ensure that the results are more significant, we generate five datasets for each data configuration using different random seeds. We show error bars to indicate ±1 standard deviation.

Scoring. Evaluating the quality of the discovered rules against the ground truth is non-trivial; a standard F1 score, for example, only considers exact answers to be correct and therewith penalizes minor mistakes (e.g. one missing item in a long rule) very harshly. To avoid this, we extend the soft F1 score proposed by Walter et al. [32] to rules. We give the formal definition in Appendix A.1. The general idea is to take the harmonic mean of precision and recall scores defined using Jaccard distance. We postpone the plots depicting recall and precision to the Appendix.

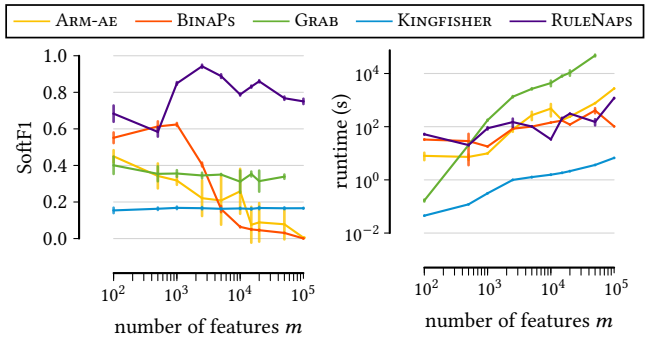
Varying Rule Confidence. First, we evaluate how each method fares for varying rule confidences. To this end, we generate data where all rules have $c \in \{0.1, 0.2, \ldots, 0.9\}$ and show results for RULENAPS and competitors in Figure 2a. We see that RULENAPS outperforms its competitors by a large margin. With the exception of the lowest and highest rule confidences,[3] RULENAPS stably obtains soft F1 scores between 0.8 to 1.0. Its closest competitor is GRAB, obtaining scores around 0.6. The other methods perform worse.

Varying Rule Frequency. Next, we evaluate sensitivity to rule frequency. To this end we generate data with rule frequencies $freq \in \{0.01, 0.05, 0.1, 0.2, 0.3\}$. We show the results in Figure 2b. We see that RULENAPS again outperforms its competitors, with a clear trend that for higher rule frequencies it obtains better F1 scores. GRAB only performs well at very low frequencies, but then quickly deteriorates. At a respectable distance, BINAPS is our closest competitor, showing a similar improvement trend for higher frequencies. Although KINGFISHER again suffers from low precision due to returning overly many rules, it is encouraging that it improves for higher rule frequencies.

Noise Robustness. Next, we evaluate robustness to additive resp. destructive noise. We generate two data setups, the first one containing data with $\sigma_{add} \in \{0.0, 0.001, 0.005, 0.01, 0.02, 0.05\}$, and the second one consisting of data with $\sigma_{destr} \in \{0.0, 0.025, 0.05, 0.1, 0.025, 0.5\}$. We run each method, and give the results in Figure 2c resp. 2d. We see that GRAB, BINAPS, ARM-AE, and KINGFISHER all perform similarly for both noise types, with F1 scores of around 0.3, without clear trends. RULENAPS is robust to both noise types. In the additive setting, its result quality degrades for $\sigma_{add} = 0.02$, which corresponds to an SNR ratio of 1.9dB. In the other setting, the result quality degrades for $\sigma_{destr} = 0.25$ which corresponds to

[2] https://eda.group/rulenaps/

[3] Note that for very high (e.g. ≥ 90%) resp. very low rule confidence (e.g. ≤ 10%), it becomes inherently hard, up to impossible, to differentiate a ground truth rule $A \to B$ from a single pattern $\emptyset \to AB$ (high confidence) resp. two patterns $\emptyset \to A$ and $\emptyset \to B$ (low confidence).



(a) F1 score (↑ better) **(b) Runtime (↓ better)**

Figure 3: *Scalability:* We show the F1 score of all algorithms (a) together with the runtimes (b) on datasets with an increasing number of columns, averaged over five datasets per configuration. For large number of features, RULENAPS clearly outperforms all other algorithms by a wide margin while maintaining a reasonable runtime. Runtime axis is provided on a *log-scale*.

an SNR ratio of 6dB. Overall, it outperforms the other algorithms in almost all settings.

Scalability in m. Next, we evaluate scalability to high dimensions. We generate synthetic data with $n = 4000$ and $m \in \{100, 500, 1000, 2500, 5000, 10000, 15000, 20000, 50000, 100000\}$, and plant $r = 500$ rules in each dataset, or until all items were covered by a rule. We show the results in Figure 3.

First, we consider the quality of the rules as shown in Figure 3a. Like before, KINGFISHER is unable to retrieve a good set of rules, the score stays below 0.2 for each of the settings. The rules found by GRAB yield a score of nearly 0.4 throughout all of the settings (except for the last one where it did not finish). ARM-AE yields a similar score in the beginning, but its result quality quickly degrades with an increased number of features. BINAPS performs a bit better on the low-dimensional datasets, but its result quality degrades as well for an increased number of features. We see that RULENAPS is competitive already for settings with fewer than 1000 features, but takes an increasingly big lead for higher dimensionalities.

Second, we consider the corresponding runtimes of the methods. We see that the combinatorial search employed by GRAB does not scale well, taking an order of magnitude longer than the neural approaches ARM-AE, BINAPS, and RULENAPS, which perform comparably.

Statistical Significance. We assess statistical significance of soft F1 score over all $n = 36$ synthetic datasets. We list the average ranks in Table 4 in the appendix. A Friedman (omnibus) test across all methods yields $\chi^2 = 86.4$ and $p = 7.6 \times 10^{-18}$. Furthermore, we compute one-sided paired Wilcoxon signed-rank tests (RULENAPS versus baselines) with Holm-Bonferroni multiple test correction. Missing values due to failed runs are imputed as 0.0. The results are shown in Table 5 in the appendix. This underlines that RULENAPS is significantly better on these datasets.

5.2 Real-world Data

Next, we quantitatively evaluate RULENAPS on real-world data. We show that our algorithm finds an interpretable set of rules on such datasets. To this end, we consider seven datasets from different domains and of varying dimensionalities. We use *DNA* amplification data [27], *Abstracts* of research papers at the ICDM conference [11], data on traffic *Accidents* [16] in Belgium, food purchase data at *Instacart* [14], data from the 1000 *Genomes* project about human genetic variation preprocessed by Fischer and Vreeken [14], and two datasets containing gene expressions for breast cancer, *BRCA-N* and *BRCA-S* preprocessed by Walter et al. [32]. We provide data statistics and results in Table 1.

We see that RULENAPS discovers rules with varying length heads and tails, adapting to the complexity of the data. This also holds for the adaptation of BINAPS, but not for ARM-AE because there the desired maximum length has to be determined beforehand. Furthermore, the set of rules found by RULENAPS is succinct and therefore interpretable, similarly to the one found by GRAB despite not explicitly optimizing model complexity. A feature of GRAB is that it optimizes model complexity through a theoretically justified measure that enables discovery of meaningful rule sets but is excessively costly prohibiting scaling to large-scale data such as Genomes, data that RULENAPS can process. For the last three listed datasets, ARM-AE, KINGFISHER, and the adaptation of BINAPS fail to find succinct sets of rules, and GRAB does not terminate within 24 hours. RULENAPS has an adequate runtime considering the scale of the datasets. AERIAL+ does not seem to be designed for such large datasets as it finds an extreme amount of rules. We postpone reporting the corresponding results in Table 3 in the Appendix.

In Table 2, we show for the rules RULENAPS finds how many are unique and how many are shared with other methods. Additionally, we provide examples for rules only RULENAPS finds as well as rules that at least one other algorithm finds. Overall, these results show that RULENAPS consistently recovers a small shared core where the methods agree, while uncovering many additional dataset-specific rules that are not returned by existing approaches.

5.3 Case Study: Breast Cancer Data

In cooperation with a trained Computational Molecular Biologist, we analysed rules RULENAPS discovered in the TCGA Breast Cancer data splitted by subtypes (BRCA-S). Each head and tail of a rule were subjected to a Gene Set Overrepresentation Analysis [38] in pathways annotated in the KEGG pathway database [21], which hosts manually curated accurate pathway maps representing our knowledge of molecular interactions. In a nutshell, a statistical significance test assesses whether a given set of genes, specified by our discovered rule, is overrepresented in a molecular pathway compared to a background set of genes, which are here genes covered in the BRCA-S dataset.

Among the discovered rules, almost all showed statistically significant enrichment ($p < .05$ after FDR correction) for specific pathways. The discovered dependencies also often reflect highly specific knowledge about molecular dependencies in breast cancer. For example, one discovered rule is enriched for the "NOD-like receptor signaling pathway" in the head, where NOD-based signaling is linked to tumorigenesis [30]. As tail, this rule showed various

			RuleNaps (ours)				Arm-ae				Grab				BinaPs				Kingfisher			
Dataset	n	m	$\lvert R\rvert$	$\lvert\overline{A}\rvert$	$\lvert\overline{B}\rvert$	t	$\lvert R\rvert$	$\lvert\overline{A}\rvert$	$\lvert\overline{B}\rvert$	t	$\lvert R\rvert$	$\lvert\overline{A}\rvert$	$\lvert\overline{B}\rvert$	t	$\lvert R\rvert$	$\lvert\overline{A}\rvert$	$\lvert\overline{B}\rvert$	t	$\lvert R\rvert$	$\lvert\overline{A}\rvert$	$\lvert\overline{B}\rvert$	t
DNA	1.3k	392	370	5.9	2.3	23s	784	1.5	1.0	11s	147	1.1	1.7	5s	99	5.7	3.4	31s	942	1.5	1.0	1s
Abstracts	859	3.9k	99	1.3	1.5	25s	7.9k	1.5	1.0	40s	29	0.9	1.2	28s	1.6k	0.6	11.6	15s	7.5k	2.9	1.0	1s
Accidents	340k	468	55	1.1	1.2	43s	936	1.5	1.0	2m	138	1.0	1.1	33s	129	2.6	1.7	29m	-	-	-	-
Instacart	2.7M	1.2k	13	1.0	1.0	28m	2.5k	1.5	1.0	17m	884	1.0	1.26	1s	815	0.4	8.8	24m	8	1.9	1.0	10s
Genomes	2.5k	225k	547	124	166	17m	453k	1.5	1.0	4h	-	-	-	-	10k	6.4	7.0	13m	0	0.0	0.0	1s
BRCA-N	222	20k	21	6.8	21.1	57s	40k	1.5	1.0	2m	-	-	-	-	15k	9.6	10.4	4m	0	0.0	0.0	1s
BRCA-S	187	20k	38	3.2	6.8	30s	40k	1.5	1.0	3m	-	-	-	-	3.5k	8.6	9.5	15s	0	0.0	0.0	1s

Table 1: *Results on real-world data.* We provide dataset size as number of samples n and number of features m. For each algorithm, we report the number of discovered rules ($\lvert R\rvert$), average length of heads ($\lvert\overline{A}\rvert$) and tails ($\lvert\overline{B}\rvert$), and wall-clock runtime (t). Failed runs (e.g. due to exceeding the time limit of 24h) are indicated by '-'.

Dataset	#unique	#shared	Unique example	Shared example
DNA	159	21	[] → [179, 181, 182]	[165] → [166]
Abstracts	95	4	[classif] → [classifi, learn]	[machin, vector] → [support]
Accidents	43	6	[108, 125] → [72, 135]	[20] → [48]
Instacart	7	4	[Vegetarian Sausage] → [Deli Slices]	[Candy Bars] → [Chocolate Bar]
Genomes	547	0	[] → [PTPN22, NOXA1, EMILIN3]	none
BRCA-N	19	0	[G118193, G169607, G113368] → [G66279, G117650, G163808]	none
BRCA-S	38	0	[G154451] → [G73861, G4468]	none

Table 2: *Overlap analysis.* We report the number of rules unique to RuleNaps and shared with other methods, along with examples, for all real–world datasets.

inflammatory and immune-related molecular processes, including differentiation of different types of T-cells. Put simply, this rule thus provides us with an indication of which processes might be a consequence of the cancer-supporting NOD-like receptor signaling.

Also, we discover a rule that, among other interactions, is enriched for the "TNF signaling pathway" in the head of the rule, and for "Natural killer cell mediated cytotoxicity" in the tail of the rule. Intriguingly, treatment of breast cancer through TNF-α is known to enhance the susceptibility of breast cancer cells to Natural Killer cell-mediated killing [3], a dependency that RuleNaps discovered unsupervised and from the gene expression data alone. This shows that RuleNaps can serve as a discovery tool for new hypotheses in complex scientific domains and large-scale datasets.

6 Discussion

The empirical evaluation shows that RuleNaps discovers a concise set of relevant association rules in binary datasets.

RuleNaps successfully finds most of the planted ground truth rules in synthetic data with various different properties. It recovers low- and high-confidence rules and is quite robust to noise. It almost always outcompetes state-of-the-art algorithms, with the exception of low-frequency settings, where it performs on par. Most notably, RuleNaps scales very well to large-scale datasets with hundreds of thousands of features, both in terms of runtime as well as in terms of quality of the results. In comparison, we found that Arm-ae and Aerial+ suffer from the pattern explosion and return overly many rules, Kingfisher performs badly as its goal of significant rule mining is slightly different from ours, and that while Grab performs well on smaller and low-noise data it does not scale to high-dimensional datasets.

BinaPs [14] is conceptually a close cousin to RuleNaps, and also scales to large and high dimensional data. It is designed to discover associations, rather than rules. In our experiments we find that it cannot trivially be extended to discover rules, which in our simple attempt are at best mediocre. We elaborate on the algorithm we use to this end in Appendix A.2. Running the extraction algorithm of RuleNaps on the trained weights of BinaPs does not improve the result (see Appendix Figure **??**). In particular, BinaPs suffers from redundancy and poor separation of head and tail features, which shows that associations and dependencies in the data have to be treated differently.

On real-world data, RuleNaps is the only method that reliably finds a reasonably-sized, and thus interpretable, set of rules on real-world data, as opposed to either 0 or many thousands. These experiments also underline that RuleNaps is applicable to data with millions of rows and hundreds of thousands of columns. Our case study on a Breast Cancer dataset suggests that the discovered rules in complex scientific data contain meaningful information.

This shows that RuleNaps is an efficient approach for association rule mining that overcomes the scalability shortcomings of existing approaches. Although the interpretable network architecture allows for mining a succinct set of rules, it also has limitations. The results of RuleNaps are based on associations, without guarantees regarding causality, and hence need to be treated with care.

Furthermore, due to the stochasticity of the network weights, RuleNaps does not give a guarantee to obtain the same results unless run with the same seed and hyperparameters. We describe in Appendix A.2 how to tune hyperparameters. We found that a classical automated hyperparameter search using a hold-out data split for evaluation works well in practice.

It would be interesting to further analyze the continuous version of the network weights in the future to see if we can gain more information about the feature dependencies in the data. For example, they might entail information about multi-layer hierarchies between features. Also, it would be interesting to adapt the architecture such that it works for continuous data.

7 Conclusion

In this paper, we considered the problem of association rule mining at scale, discovering rules in binary data of millions of transactions (or samples) and hundreds of thousands of items (or features). This type of tabular data naturally matches complex datasets, such as in scientific domains, yet existing methods fail to scale to such data, and even on smaller data usually yield overly redundant, and too many to interpret rules.

To tackle this problem, we proposed RuleNaps, a neural autoencoder architecture that naturally encodes symbolic rules while allowing for differentiation, enabling gradient-based optimization on GPUs. Across considered benchmarks, it outperformed existing methods, including neural architectures for rule mining, by a wide marging. It further discovered valuable novel insights into complex and high-dimensional single-cell Genomics data that aligns with domain knowledge from the field. RuleNaps hence discovers meaningful, interpretable sets of rules at unprecedented scale.

Acknowledgments

Luis Paulus is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – GRK 2853/1 “Neuroexplicit Models of Language, Vision, and Action” - project number 471607914.

References

[1] R. Agrawal, T. Imielinksi, and A. Swami. 1993. Mining association rules between sets of items in large databases. In *SIGMOD*. ACM, 207–216.
[2] R. Agrawal and R. Srikant. 1994. Fast Algorithms for Mining Association Rules. In *VLDB*. 487–499.
[3] F. Barberini, R. Pietroni, S. Ielpo, V. Lucarini, D. Nardozi, O. Melaiu, M. Benvenuto, C. Focaccetti, C. Palumbo, F. Rossin, D. Fruci, D. Olive, L. Masuelli, R. Bei, and L. Cifaldi. 2025. Combined IFN-γ and TNF-α treatment enhances the susceptibility of breast cancer cells and spheroids to Natural Killer cell-mediated killing. *Cell Death & Disease* 16 (2025).
[4] Y. Bengio, N. Léonard, and A. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv* (2013), 12 pages. cs.LG:1308.3432
[5] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 24 (2011).
[6] T. Berteloot, R. Khoury, and A. Durand. 2024. Association rules mining with auto-encoders. In *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 51–62.
[7] B. Bringmann and A. Zimmermann. 2007. The Chosen Few: On identifying valuable patterns. In *ICDM*. 63–72.
[8] B. Bringmann and A. Zimmermann. 2009. One in a million: picking the right patterns. *Knowl. Inf. Sys.* 18, 1 (2009), 61–81.
[9] T. Chataing, J. Perez, M. Plantevit, and C. Robardet. 2024. DiffVersify: a Scalable Approach to Differentiable Pattern Mining with Coverage Regularization. In *ECML PKDD*. Springer, 407–422.
[10] C. Chen and C. Rudin. 2018. An optimization approach to learning falling rule lists. In *International conference on artificial intelligence and statistics*. PMLR, 604–612.
[11] T. De Bie, K.-N. Kontonasios, and E. Spyropoulou. 2010. A Framework for Mining Interesting Pattern Sets. *SIGKDD Explor.* 12, 2 (2010), 92–100.
[12] L. Dierckx, R. Veroneze, and S. Nijssen. 2023. Rl-net: Interpretable rule learning with neural networks. In *PAKDD*. Springer, 95–107.
[13] J. Fischer and J. Vreeken. 2019. Sets of robust rules, and how to find them. In *ECML PKDD*. Springer, 38–54.
[14] J. Fischer and J. Vreeken. 2021. Differentiable pattern set mining. In *KDD*. ACM, 383–392.
[15] E. Galbrun, P. Cellier, N. Tatti, A. Termier, and B. Crémilleux. 2018. Mining Periodic Patterns with a MDL Criterion. In *ECML PKDD*. Springer, 535–551.
[16] K. Geurts, G. Wets, T. Brijs, and K. Vanhoof. 2003. Profiling High Frequency Accident Locations Using Association Rules. In *ATRB*. National Academy, 1–18.
[17] P. Grünwald. 2007. *The Minimum Description Length Principle*. MIT Press.
[18] W. Hämäläinen. 2012. Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures. *Knowl. Inf. Sys.* 32, 2 (2012), 383–414.
[19] J. Han, H. Cheng, D. Xin, and X. Yan. 2007. Frequent Pattern Mining: Current Status and Future Directions. *Data Min. Knowl. Disc.* 15 (2007), 55–86. Issue 1.
[20] J. Han, J. Pei, and Y. Yin. 2000. Mining frequent patterns without candidate generation. In *SIGMOD*. ACM, 1–12.
[21] M. Kanehisa and S. Goto. 2000. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research* 28, 1 (2000), 27–30. https://doi.org/10.1093/nar/28.1.27
[22] E. Karabulut, P. Groth, and V. Degeler. 2025. PyAerial: Scalable association rule mining from tabular data. *SoftwareX* 31 (2025), 102341.
[23] R. Kusters, Y. Kim, M. Collery, C. de Sainte Marie, and S. Gupta. 2022. Differentiable Rule Induction with Learned Relational Features. In *NeSy*. CEUR, 15 pages.
[24] H. Lakkaraju, S. H. Bach, and J. Leskovec. 2016. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1675–1684.
[25] H. Mannila, H. Toivonen, and A. I. Verkamo. 1994. Efficient Algorithms for Discovering Association Rules. In *KDD*. 181–192.
[26] C. A. C. Montañez, P. Fergus, C. Chalmers, N. H. A. H. Malim, B. Abdulaimma, D. Reilly, and F. Falciani. 2020. SAERMA: Stacked Autoencoder Rule Mining Algorithm for the interpretation of epistatic interactions in GWAS for extreme obesity. *IEEE Access* 8 (2020), 112379–112392.
[27] S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. 2006. DNA copy number amplification profiling of human neoplasms. *Oncogene* 25, 55 (2006), 7324–7332.
[28] L. Qiao, W. Wang, and B. Lin. 2021. Learning accurate and interpretable decision rule sets from neural networks. In *AAAI*. AAAI, 4303–4311.
[29] A. Siebes, J. Vreeken, and M. van Leeuwen. 2006. Item Sets that Compress. In *SDM*. SIAM, 393–404.
[30] F. J. Velloso, A. R. Campos, M. C. Sogayar, and R. G. Correa. 2019. Proteome profiling of triple negative breast cancer cells overexpressing NOD1 and NOD2 receptors unveils molecular signatures of malignant cell proliferation. *BMC genomics* 20, 1 (2019), 152.
[31] J. Vreeken and N. Tatti. 2014. Frequent Pattern Mining. In *Frequent Pattern Mining*, Charu C. Aggarwal and Jiawei Han (Eds.). Springer, Chapter Interesting Patterns, 105–134.
[32] N. P. Walter, J. Fischer, and J. Vreeken. 2024. Finding interpretable class-specific patterns through efficient neural search. In *AAAI*. AAAI, 9062–9070.
[33] Z. Wang, W. Zhang, N. Liu, and J. Wang. 2021. Scalable rule-based representation learning for interpretable classification. In *NeurIPS*. Curran, 30479–30491.
[34] G. I. Webb. 2007. Discovering Significant Patterns. *Mach. Learn.* 68, 1 (2007), 1–33.
[35] S. Xu, N. P. Walter, and J. Vreeken. 2025. Neural Rule Lists: Learning Discretizations, Rules, and Order in One Go. In *NeurIPS*. Curran, 11 pages.
[36] L. Yang and M. van Leeuwen. 2022. Truly unordered probabilistic rule sets for multi-class classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 87–103.
[37] Y. Yang, W. Ren, and S. Li. 2024. HyperLogic: Enhancing Diversity and Accuracy in Rule Learning with HyperNets. In *NeurIPS*. Curran, 3564–3587.
[38] G. Yu, L.-G. Wang, Y. Han, and Q.-Y. He. 2012. clusterProfiler: an R package for comparing biological themes among gene clusters. *OMICS: A Journal of Integrative Biology* 16, 5 (2012), 284–287. https://doi.org/10.1089/omi.2011.0118
[39] M. J. Zaki. 2000. Scalable Algorithms for Association Mining. *IEEE TKDE* 12, 3 (2000), 372–390.
[40] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. 1997. New algorithms for fast discovery of association rules. In *KDD*. ACM, 283–286.

(1) Infer pattern matrix:
$$W = \begin{bmatrix} 0.1 & 0.1 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.1 & 0.4 \end{bmatrix} \xrightarrow{>\tau=0.2} P = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$
(2) Mask weight matrix using P:
$$W_P = P \odot W = \begin{bmatrix} 0.0 & 0.0 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.0 & 0.4 \end{bmatrix}$$
(3) Calculate $\rho^- = \begin{bmatrix} \rho_1^- \\ \rho_2^- \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.4 \end{bmatrix}$ and $\rho^+ = \begin{bmatrix} \rho_1^+ \\ \rho_2^+ \end{bmatrix} = \begin{bmatrix} 1.6 \\ 1.2 \end{bmatrix}$
(4) Apply threshold $\frac{\rho_i^+}{2}$ on W_P to differentiate between heads and tails if $\rho_i^- > \delta = 0.1$
(5) Extract set of discovered rules:
$R = \{\emptyset \rightarrow \{3,4\}, \{1,2\} \rightarrow \{4\}\}$

Figure 4: *Rule extraction example.* Example of how to extract the set of rules from W. Note that we use the same weights as in Figure 1.

			Aerial+			
Dataset	n	m	$\lvert R\rvert$	$\overline{\lvert A\rvert}$	$\overline{\lvert B\rvert}$	*time*
DNA	1.3k	392	152k	1.0	1.0	3m
Abstracts	859	3.9k	155M	1.0	1.0	23m
Accidents	340k	468	219k	1.0	1.0	4h
Instacart	2.7M	1.2k	-	-	-	-
Genomes	2.5k	225k	-	-	-	-
BRCA-N	222	20k	385M	1.0	1.0	7h
BRCA-S	187	20k	386M	1.0	1.0	6h

Table 3: *Additional results on real-world data.* Results of Aerial+ on the real-world datasets.

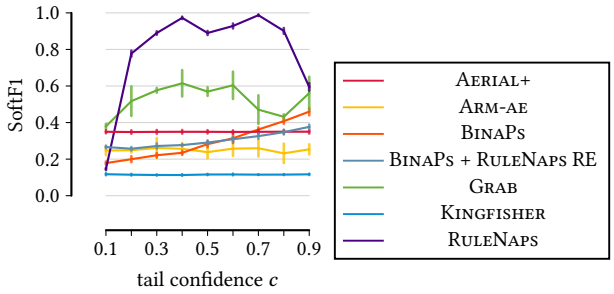


Figure 5: *Additional results on synthetic data.* Experimental results of the datasets in Figure 2a with the results of Aerial+. Furthermore, we show that running our rule extraction algorithm on the trained weights of BinaPs does not lead to good results.

A Appendix

A.1 Evaluation Metric

We define the soft precision on rules by

$$SoftPrec(R, R^*) = \frac{1}{|R|} \sum_{A \rightarrow B \in R} \max_{A^* \rightarrow B^* \in R^*} \frac{1}{2} \left(\frac{|A \cap A^*|}{|A \cup A^*|} + \frac{|B \cap B^*|}{|B \cup B^*|} \right)$$

and analogously the soft recall on rules by

$$SoftRec(R, R^*) = \frac{1}{|R^*|} \sum_{A^* \rightarrow B^* \in R^*} \max_{A \rightarrow B \in R} \frac{1}{2} \left(\frac{|A \cap A^*|}{|A \cup A^*|} + \frac{|B \cap B^*|}{|B \cup B^*|} \right)$$

where R denotes the set of discovered rules and R^* the set of ground truth rules. We then define

$$SoftF1(R, R^*) = \frac{2 \cdot SoftPrec(R, R^*) \cdot SoftRec(R, R^*)}{SoftPrec(R, R^*) + SoftRec(R, R^*)}$$

as the soft F1 score on rules.

A.2 Experimental Setup

Algorithm Parameters. We fix the pattern extraction threshold $\tau = 0.2$ as originally specified for BinaPs. We fix the negligibility threshold $\delta = 0.01$. We optimize α, β, γ, the learning rate, the batch size, and the number of epochs by minimizing the reconstruction loss on a hold-out set consisting of 10% of the samples. Often, setting the hidden layer size h to half the number of features, limited to maximally 1000, leads to good results. Optimizing h might improve the result, but we recommend beginning with this initial value. The same holds for setting α to 0.5. We start with Latin Hypercube Sampling for initial coverage in the search space and then proceed by applying the Tree-structured Parzen Estimator approach [5]. We use the Adam optimizer for weight optimization.

For Grab, we do not change any parameters as the authors strongly advise against it. For Kingfisher, we set the minimum confidence to 50%. We use Fisher's exact test with a significance threshold of $\alpha = 0.001$. We set the minimum frequency threshold to 1% of the number of rows. We mine the best one million rules. Apart from that, we use the default parameters. For BinaPs, we tuned the learning rate, the batch size, and the number of epochs using the reconstruction loss on a hold-out set.

Aerial+ is implemented in a way such that the algorithm cannot run on data with columns that do not contain at least one 1. As our synthetic datasets can contain such columns by design, we remove all of these columns from the dataset before running Aerial+. When running with default parameters on our synthetic data, we find over hundreds of millions of rules for some settings. Therefore, we follow the official documentation and set ant_similarity=0.8 and cons_similarity=0.9 and limit antecedents to length 1 to avoid rule explosion. For the BRCA-N and BRCA-S datasets, we reduced the layer dimensionalities to 4096 and 1024 to avoid running out of CUDA memory. On the Genomes dataset, we ran out of memory either way.

For Arm-ae, we optimize the batch size, the learning rate, and the number of epochs by optimizing the loss on a hold-out set. As the runtime of the algorithm increases significantly for a larger number of columns, we refrain from optimizing the hyperparameters on the last four settings in the experiment testing scalability to a high number of features (from 20,000 columns onwards). We

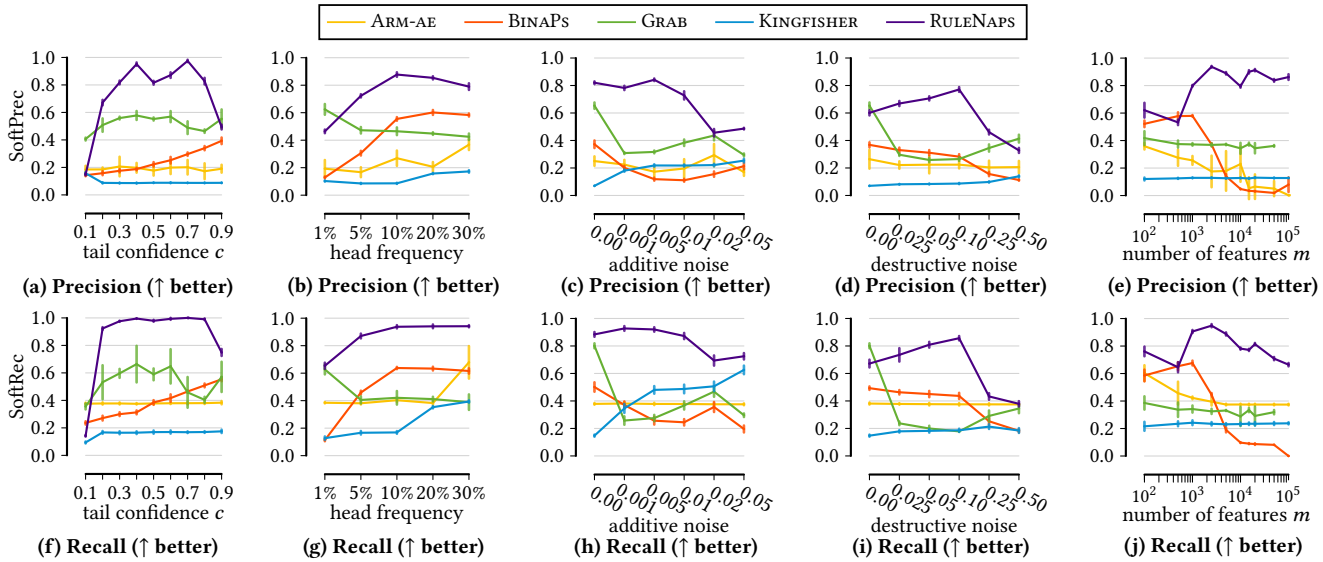


Figure 6: *Precision and Recall.* We show soft precision (a)-(d), resp. (e), and soft recall (f)-(i), resp. (j), of all experiments in Figure 2, resp. Figure 3.

used values that worked well in the other experiments. For those settings and the *BRCA-N*, *BRCA-S*, and *Genomes* datasets, we reduce all hidden layer sizes to 1024 to ensure the algorithm terminates. The authors stopped training after the difference of loss between two consecutive epochs was lower than 0.1 as this worked well for their datasets. However, this does not seem to translate to other datasets. In the synthetic data we generate, we create rules that do not share features, so each feature is present in at most one rule. We therefore only mine one rule per consequent. Due to this, we do not have to optimize the similarity threshold. We fix the maximal antecedent length to 2 as used by the authors.

Reproducibility. For all experiments, we draw the same configuration five times with random seeds 1, 2, 3, 4, and 5. We fix the random seed before every algorithm run to 1.

Computational Resources. We implemented RuleNaps using PyTorch and used machines with NVIDIA DGX A100 and AMD Rome 7742 CPUs, or with Dell R7525 and AMD EPYC 7f32 CPUs for the experiments. We used the same setup for Aerial+ and Arm-ae as they are also implemented in Python. We ran the other algorithms on Intel(R) Xeon(R) Gold 6244 CPUs with 256GB RAM.

Algorithms. We used an implementation of Kingfisher sent to us by the author upon request as we could not find an implementation. For BinaPs, Grab and for Arm-ae, we used the source code published by the authors. For Aerial+, we used the dedicated PyAerial Python package.

Rule Extraction for BinaPs. We extract rules from the patterns found by BinaPs using Algorithm 1. When running existing rule extraction algorithms [2] on the patterns found by BinaPs, we suffer a combinatorial explosion and find up to a million rules. We therefore decide to use a straight-forward extraction algorithm.

Algorithm 1 ExtractRulesFromPatterns

Require: Set of patterns P, data X

```
rules ← ∅
r ← MergeCooccurringPatterns(P, X)
rules ← rules ∪ r
r ← SplitPatterns(P, X)
rules ← rules ∪ r
for all unused pattern p ∈ P do
    rules ← rules ∪ {∅ → p}
end for
return rules
```

Method	Average rank (↓ better)
RuleNaps	1.1
Grab	2.5
BinaPs	3.5
Arm-ae	3.5
Kingfisher	4.4

Table 4: *Average ranks.* Average rank of each algorithm across all synthetic datasets.

A.3 Ablation of Regularization Terms

We ablate the regularization terms for otherwise the same setting as depicted in Figure 2b. We average across all densities and show the average loss, the average accuracy on 1-valued features, and the average rule length. The results in Table 6 show that the combination of both regularization terms gives the best results.

Algorithm 2 MergeCooccurringPatterns

Description: Given a set of patterns P, find suitable rules $A \to B$ where

Require: Set of patterns P, data X, minSup= 0.008, threshold= 0.5

rules $\leftarrow \emptyset$
for $p_1, p_2 \in P$ with $p_1 \neq p_2$ **do**
  $X_{p_1} \leftarrow$ rows of X where $> 90\%$ of p_1 items are present
  $X_{p_2} \leftarrow$ rows of X where $> 90\%$ of p_2 items are present
  $X_{p_1p_2} \leftarrow X_{p_1} \cap X_{p_2}$
  // skip if full rule is not present at least minSup times
  if $|X_{p_1p_2}| <$ minSup **then**
    continue
  end if
  // If the number of joint samples is greater than 60% of the samples where p_2 is present, p_2 might be a tail. Check additionally, if number of joint samples is smaller than the number of samples where p_1 is present, as otherwise p_2 is not a tail.
  if $|X_{p_1p_2}| \geq 0.6 \cdot |X_{p_2}|$ and $|X_{p_1p_2}| < |X_{p_1}|$ **then**
    rules $\leftarrow$ rules $\cup$ $(p_1 \to p_2)$
  else if $|X_{p_1p_2}| \geq 0.6 \cdot |X_{p_1}|$ and $|X_{p_1p_2}| < |X_{p_2}|$ **then**
    rules $\leftarrow$ rules $\cup$ $(p_2 \to p_1)$
  else
    $o_1 \leftarrow |X_{p_1p_2}|/|X_{p_1}|$
    $o_2 \leftarrow |X_{p_1p_2}|/|X_{p_2}|$
    if $o_1 <$ threshold and $o_2 <$ threshold **then**
      continue
    else if $o_1 ==$ 1 and $o_2 ==$ 1 **then**
      continue
    else if $o_1 <$ threshold and $o_2 >$ threshold **then**
      rules $\leftarrow$ rules $\cup$ $(p_2 \to p_1)$
    else if $o_1 >$ threshold and $o_2 <$ threshold **then**
      rules $\leftarrow$ rules $\cup$ $(p_1 \to p_2)$
    else if $o_1 > o_2$ **then**
      rules $\leftarrow$ rules $\cup$ $(p_1 \to p_2)$
    else
      rules $\leftarrow$ rules $\cup$ $(p_2 \to p_1)$
    end if
  end if
end for
return rules

	Wilcoxon W	p (raw)	p (Holm)
Grab	622.0	7.28×10^{-10}	7.28×10^{-10}
Kingfisher	630.0	2.91×10^{-11}	1.16×10^{-10}
Arm-ae	630.0	2.91×10^{-11}	1.16×10^{-10}
BinaPs	629.0	5.82×10^{-11}	1.16×10^{-10}

Table 5: *Pairwise Wilcoxon tests.* We show the results of pairwise Wilcoxon test between RuleNaps vs. the baselines.

A.4 Variance across Seeds

To show consistency, we run RuleNaps on the data of Experiment 2 (Figure 2b) using random seeds 1, 2, 3, 4, and 5. Per data setting, we average over the seeds and report the average number of rules, the exact core rules (identical rules recovered by at least 3 seeds), near-core rules (rules with at most one item difference, recovered by at least 3 seeds), and soft F1. The results are shown in Table 7. The found number of near-core rules in each setting shows that RuleNaps is consistent. Nevertheless, we see a higher variance when considering exact core rules due to the stochasticity of the weights.

Algorithm 3 SplitPatterns

Description: Given a set of patterns P, find suitable rules $A \to B$ where $A \cap B = \emptyset$ and $A \cup B = p$ for all $p \in P$.

Require: Set of patterns P, data X

rules $\leftarrow \emptyset$
for $p \in P$ **do**
  $X_p \leftarrow$ rows of X where $> 50\%$ of p items are present
  $d \leftarrow$ dictionary ▷ contains for each subset how often it is not present
  for each $s \subset p$ **do**
    $n \leftarrow$ # rows in X_p where s is not present
    $d[s] \leftarrow d[s] + n \cdot |s|$ ▷ weigh importance by length of s
  end for
  if d is not empty **then**
    tail $\leftarrow s \in d$.keys() with maximal d.value
    head $\leftarrow p -$ tail
    rules $\leftarrow$ rules $\cup$ (head $\to$ tail)
  end if
end for
return rules

Variant	Avg. Loss ↓	Avg. Acc ↑	Avg. Rule Length
$\beta = 0$	81.4	0.19	231.8
$\gamma = 0$	40.4	0.67	7.9
Normal	**5.1**	**0.89**	**3.6**

Table 6: *Ablation of regularizers.* We run RuleNaps with three different configurations, without horizontal regularizer ($\beta = 0$), without vertical regularizer ($\gamma = 0$), and with both regularizers (normal). We show loss, accuracy, and rule length averaged over all 25 datasets as used in Figure 2b.

Head frequency	#Rules	Core	Near-core	SoftF1
0.01	420.0	117.0	385.0	0.61
0.05	371.0	183.8	356.4	0.82
0.10	269.2	189.4	265.2	0.92
0.20	277.6	187.4	264.4	0.91
0.30	308.2	188.7	290.8	0.89
Average	329.2	173.3	312.4	0.83

Table 7: *Consistency across seeds.* We run RuleNaps on the 25 datasets in Figure 2b with five different random seeds. We report the average number of rules, identical rules recovered by at least 3 seeds (core), those with at most one item difference (near-core), and soft F1 score.