Pattern Mining with Deep Learning

α version !

DMV course, M2 SIF

Alexandre Termier

2022/2023

Introduction

- Overaching goal: capture patterns in data
  - Patterns are not limited to itemsets/sequences/…. !
- **All** machine learning methods capture some kind of patterns in data
  - One cannot classify/predict/cluster in chaos
- Patterns found by ML methods
  - Can be seen as a “by-product” of the ML method
  - Are not easy to understand by humans / are not designed for that…
  - …and its often hard to retrieve them from the ML model
  - But they can contain valuable insights about the data

From Machine Learning to Pattern Mining…and back

- **Question of the day:**
  - Can we design a Neural Net where the « internally learnt » patterns are interesting itemsets?

- **Question for another day:**
  - Can pattern mining be used to understand what are the patterns learned by a Deep Neural Network?

  - *Ongoing research*
  - *See the following paper for a first taste:*

    Luca Veyrin-Forrer, Ataollah Kamal, Stefan Duffner, Marc Plantevit, Céline Robardet: What Does My GNN Really Capture? On Exploring Internal GNN Representations. IJCAI 2022: pp 747-752

BinaPs

Jonas Fischer, Jilles Vreeken
« Differentiable Pattern Set Mining »
KDD 2021
pp. 383-392

Motivation

- Really interesting pattern mining problem: mine Single Nucleotide Polymorphism (SNP) data
  - 100k – 1M items
  - dense

- Current state of the art problem in pattern mining: find a pattern set

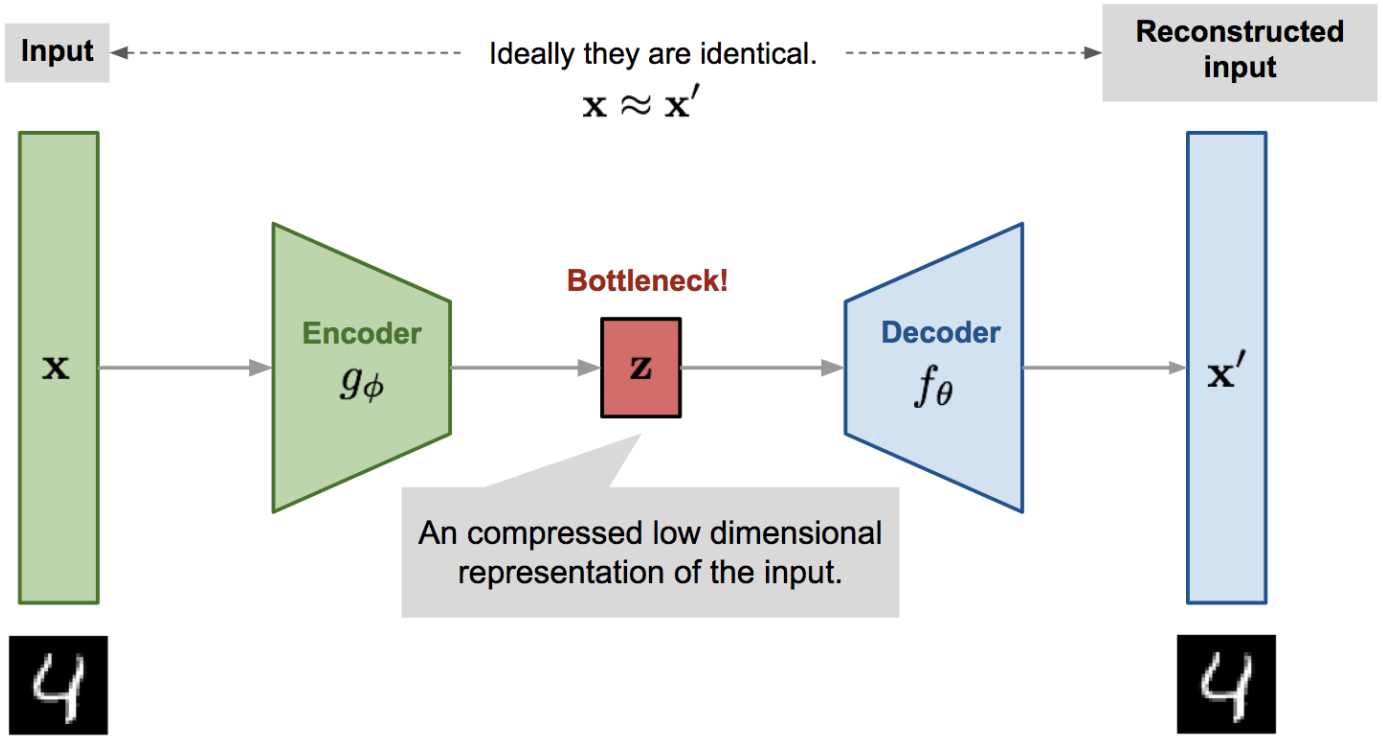
- **Pattern set**
  - (small) set of patterns that captures the main structures existing in the data
  - See Peggy's course at the end of DMV module

- Problem: doubly exponential time complexity + algo rely on heuristics
  - Do not scale *at all* for the mining of SNPs

On the other side of the fence…

- (Deep) Neural Nets methods can handles millions of features
- Some approaches can handle binary data
- So…could NN be made to find a good pattern set on SNP data?
  - What architecture ?
  - What loss ?

Auto-encoders in a nutshell



Auto-encoder:

- Learns low-dim representation of input
- Learns to encode and decode from this low-dim representation

- The low-dim representation should capture « patterns » of the input data

Figure: https://lilianweng.github.io/posts/2018-08-12-vae/

General idea of BinaPs

- Autoencoder with 1 hidden layer
- Binary input/output

- Hidden layer captures the patterns

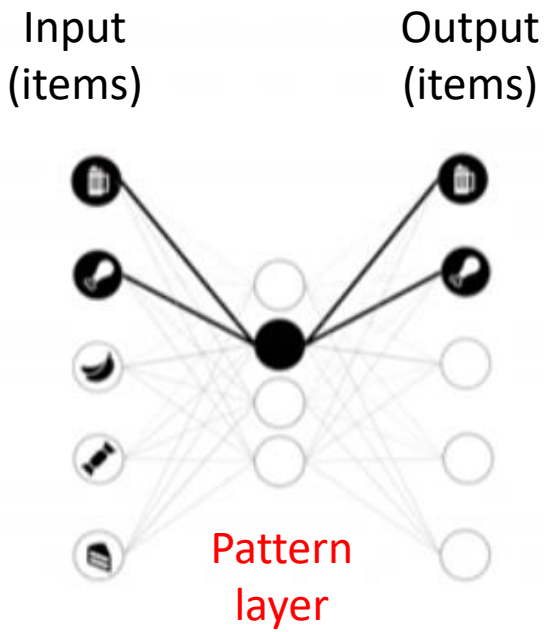
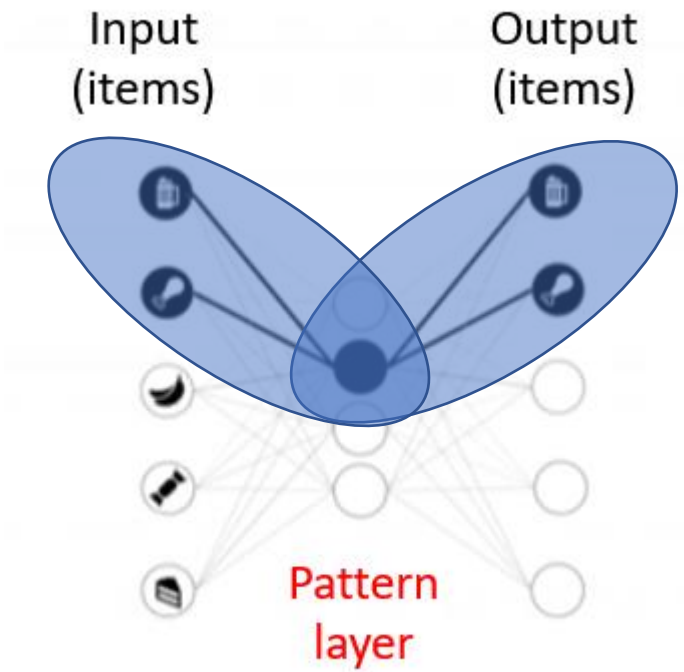


Figure from KDD21 video presentation of Jonas Fischer https://dl.acm.org/doi/10.1145/3447548.3467348

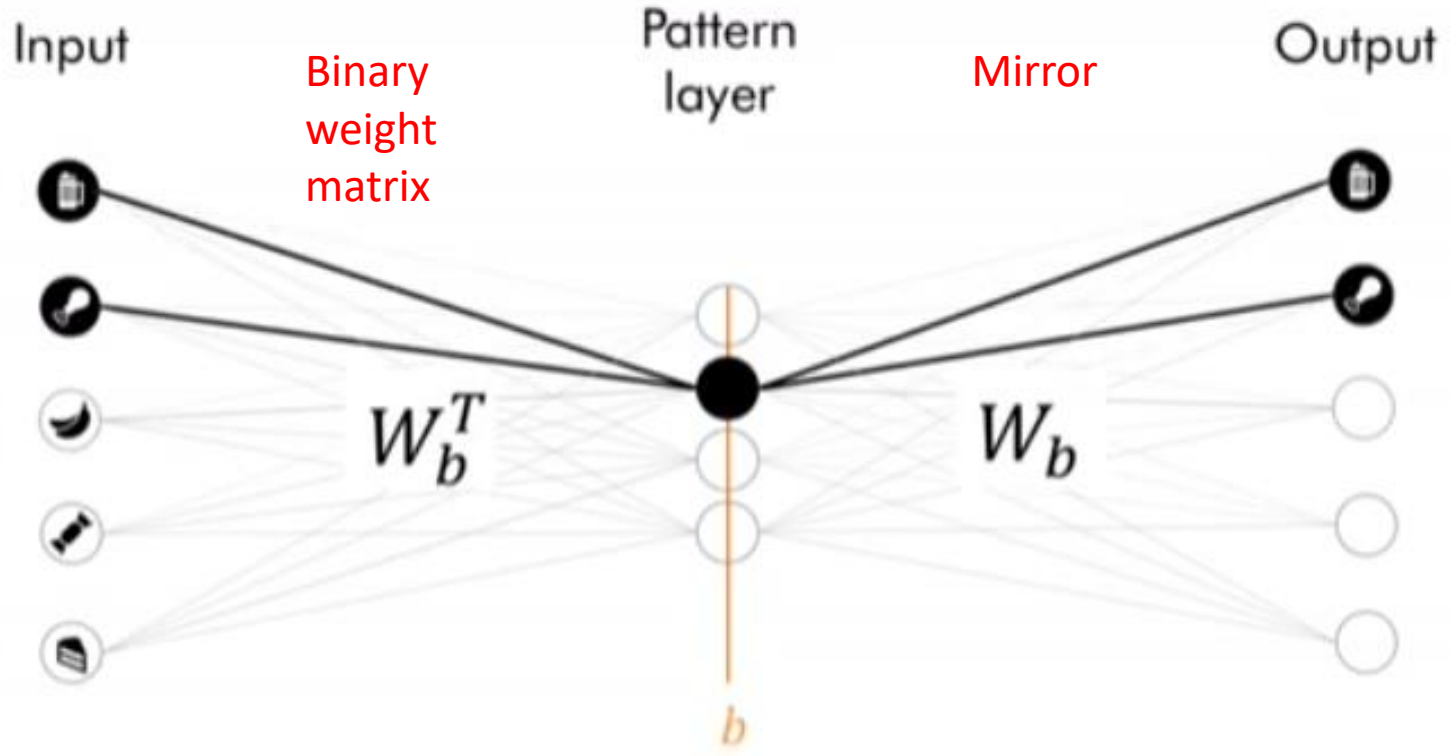
Difficulty

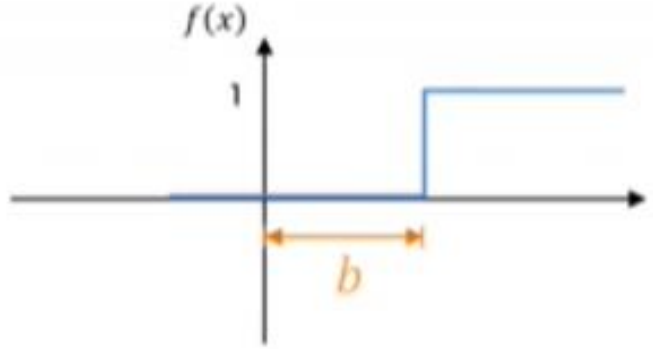


- For interpretable results, weigths must be binary
- But for proper loss optimization, weights should be continuous (differentiable)…
- Solution:
  - Binary weights **in the forward pass**
  - Continuous weights **in the backward pass**

Figure from KDD21 video presentation of Jonas Fischer https://dl.acm.org/doi/10.1145/3447548.3467348

Forward pass



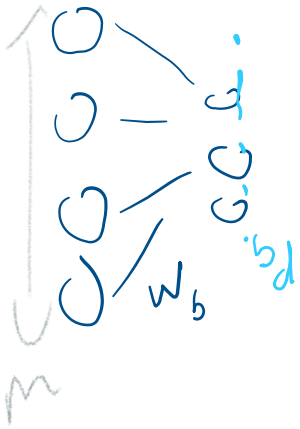


Negative bias = minus size of pattern:
pattern neuron activate ONLY if all items of the pattern are found

Figure from KDD21 video presentation of Jonas Fischer https://dl.acm.org/doi/10.1145/3447548.3467348

Forward pass

Encoding



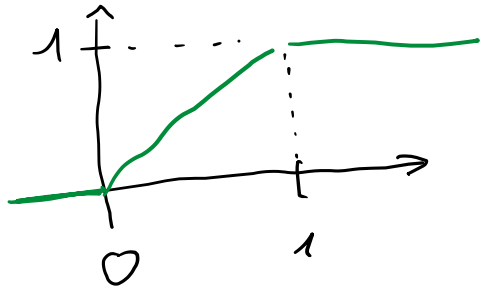
$$f_E(s) = \lambda_E(s\, W_b^T)$$

$s \in \{0,1\}^m$

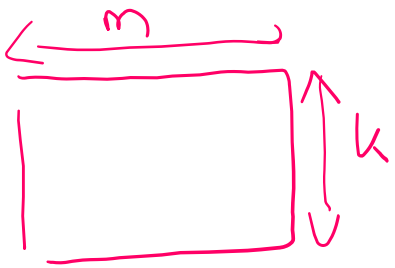
activation

$$\lambda_E(x) = \text{round}(\text{clamp}(x + b_d))$$

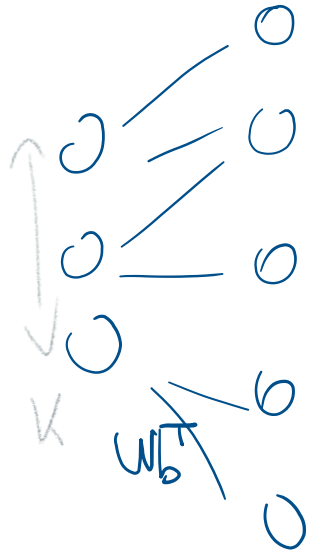
clamp :



W_b :



Decoding



$$f_D(y) = \lambda_D(y\, W_b)$$

$k \in \{0,1\}^k$

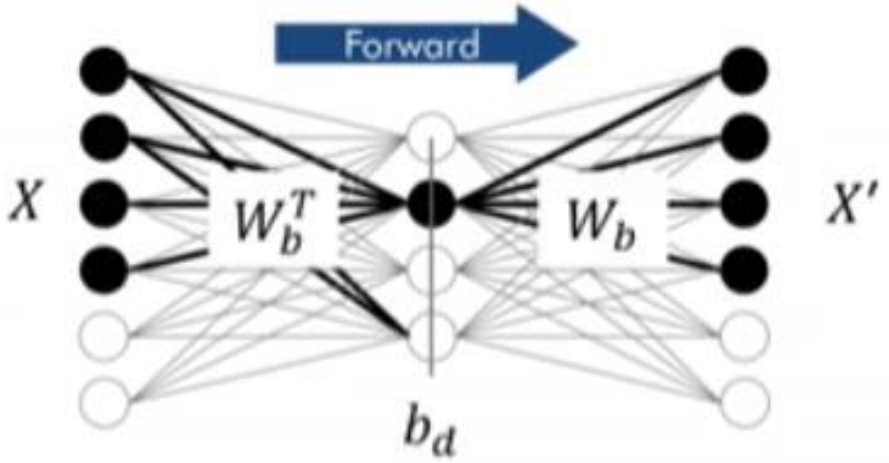
$$\lambda_D(x) = \text{round}(\text{clamp}(x))$$

FORWARD PASS = $f_D(f_E(s))$

Forward + backward

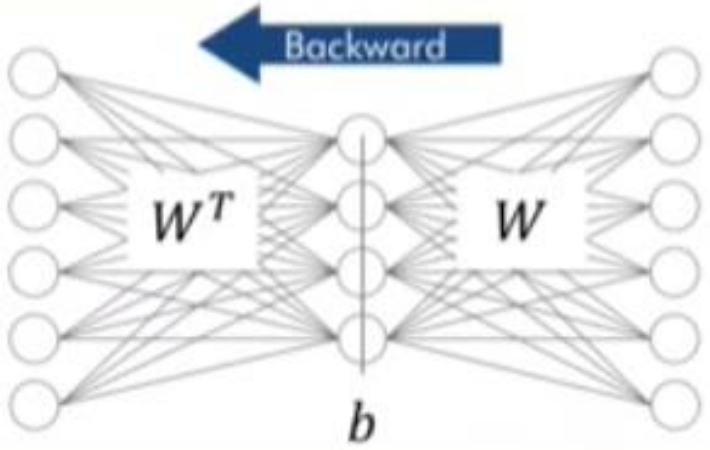
For $X \in D$

1. reconstruct



2. compute loss $L_{W,b}(X)$

3. backpropagate $\frac{dL_{W,b}(X)}{dW}$ $\frac{dL_{W,b}(X)}{db}$



4. Clamp W, b and get binarized W_b, discretized b_d

Figure from KDD21 video presentation of Jonas Fischer https://dl.acm.org/doi/10.1145/3447548.3467348

Backpropagation

- Reconstruction loss needs to take into account the sparsity of the data

$$L_\alpha(D[i]; W, b) = \sum_{j \in [1,m]} \big((1 - D[i,j])\alpha + D[i,j](1-\alpha)\big)|z_{ij} - D[i,j]|$$

$$\alpha = \frac{\#1s}{\#1s + \#0s}$$ -> sparsity of data

- Definition of relevant derivatives for chain rule: *see paper*

- At the end of the backpropagation :
  - Weights are binarized (W -> W_b)
  - Bias is discretized (max = -1 -> patterns will be of size > 1)

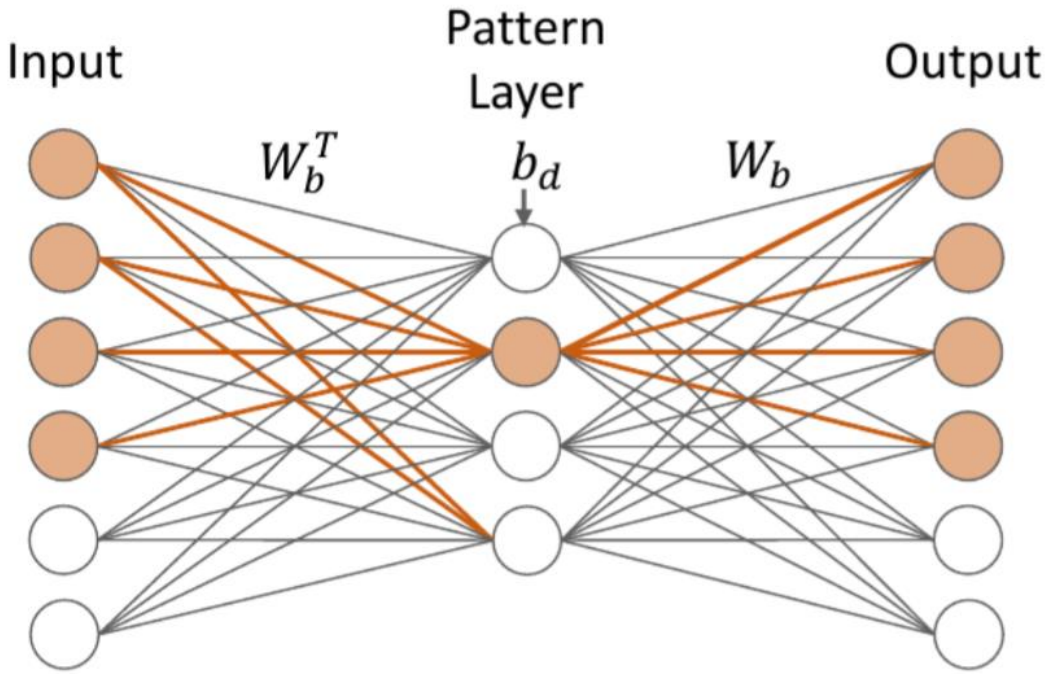
Complete example

Database D

Items I

Samples S

0	0	0	0	1	1
1	1	1	1	0	0
1	1	1	1	1	1
1	1	0	0	0	1
1	1	0	0	0	1
0	0	0	0	1	1
1	1	0	0	0	1



Weight W

0.02	0.01	0.00	0.01	0.95	0.91
0.78	0.81	0.92	0.82	0.03	0.00
0.03	0.04	0.07	0.05	0.01	0.00
0.93	0.90	0.03	0.01	0.07	0.87

W_b

0	0	0	0	1	1
1	1	1	1	0	0
0	0	0	0	0	0
1	1	0	0	0	1

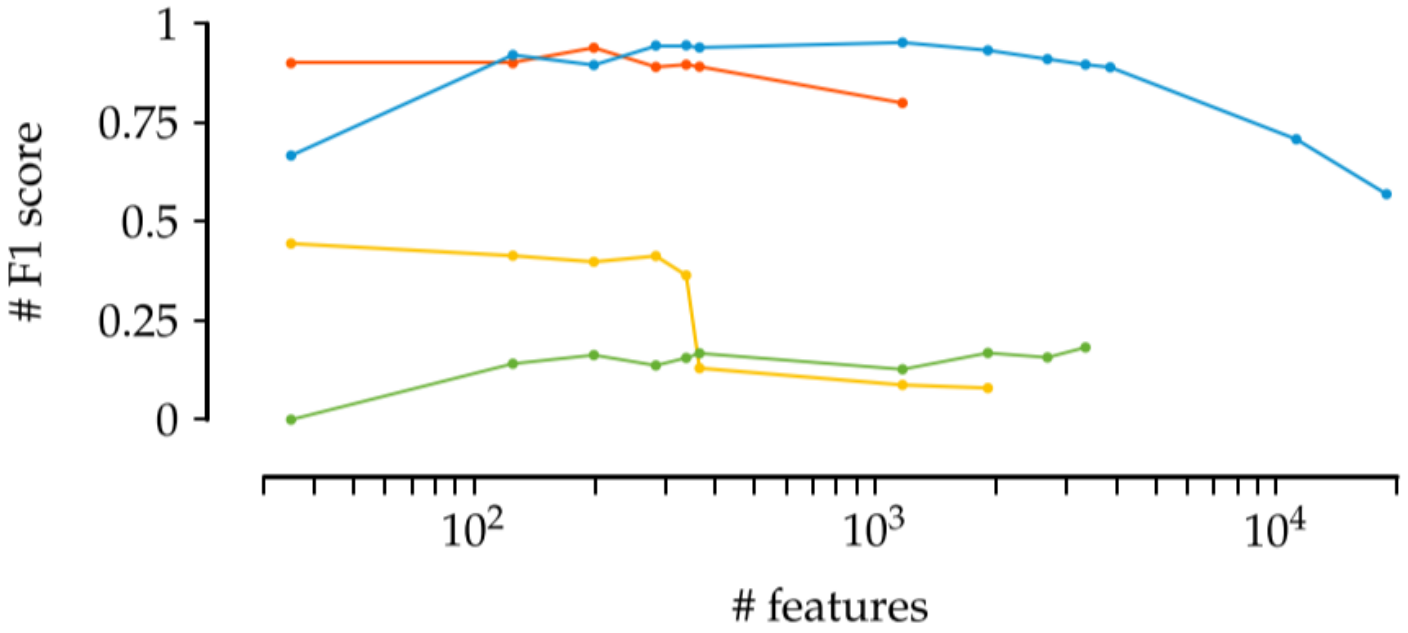
P

{ {5, 6},
{1, 2, 3, 4},
{1, 2, 6} }

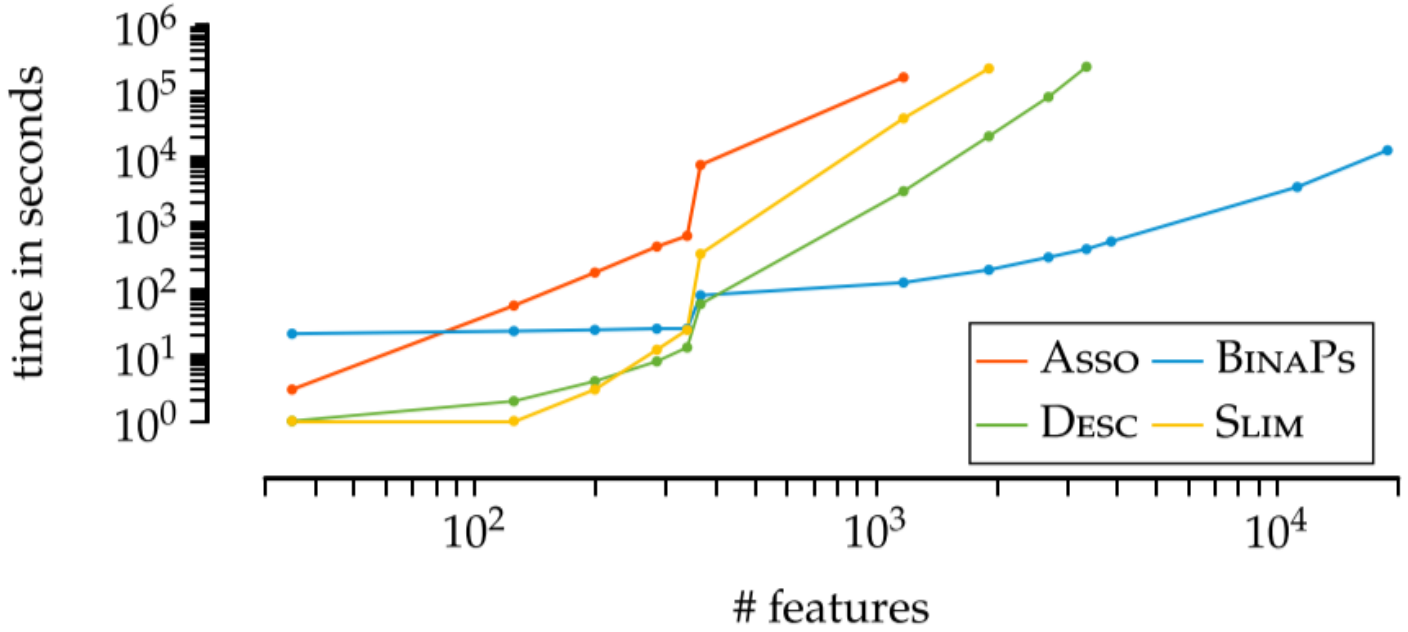
Bias b	b_d
-1.1	-1
-2.9	-2
-1.0	-1
-2.1	-2

Example from original paper

Results on synthetic data



(a) F1 score on scale data (higher is better).



Take home message

- NN can operate in the binary space of transaction matrices, and find find good pattern sets
  - The « main trick » is to switch back and forth between binary space for pattern definition and data reconstruction, and continuous space for optimization

- The approach scales *way better* than state of the art in pattern sets
  - Opens the possibility to analyze realistic SNPs datasets

Where to go from here?

- Extending the approach to other pattern langages
  - Graphs: get help from large body of work on GNN?
  - Sequences?
  - First, simpler variations of itemsets – ex: generalized itemsets?

- BinaPs optimization (gradient descent) works well – MDL's one doesn't: why?
  - Could some ways to optimized be « borrowed » to improve MDL-based approaches heuristics?
  - More generally: have we been exploring the pattern space « wrong » all this time?