Lifted Relational Neural Networks

Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny

& Ondrej Kuzelka

Outline

- Motivation
  - From Neural Nets point of view (possibly)
  - From Markov Logic point of view
- What are Lifted Relational Neural Networks
  - Short version
  - Long version (possibly)
- Learning latent concepts with LRNNs

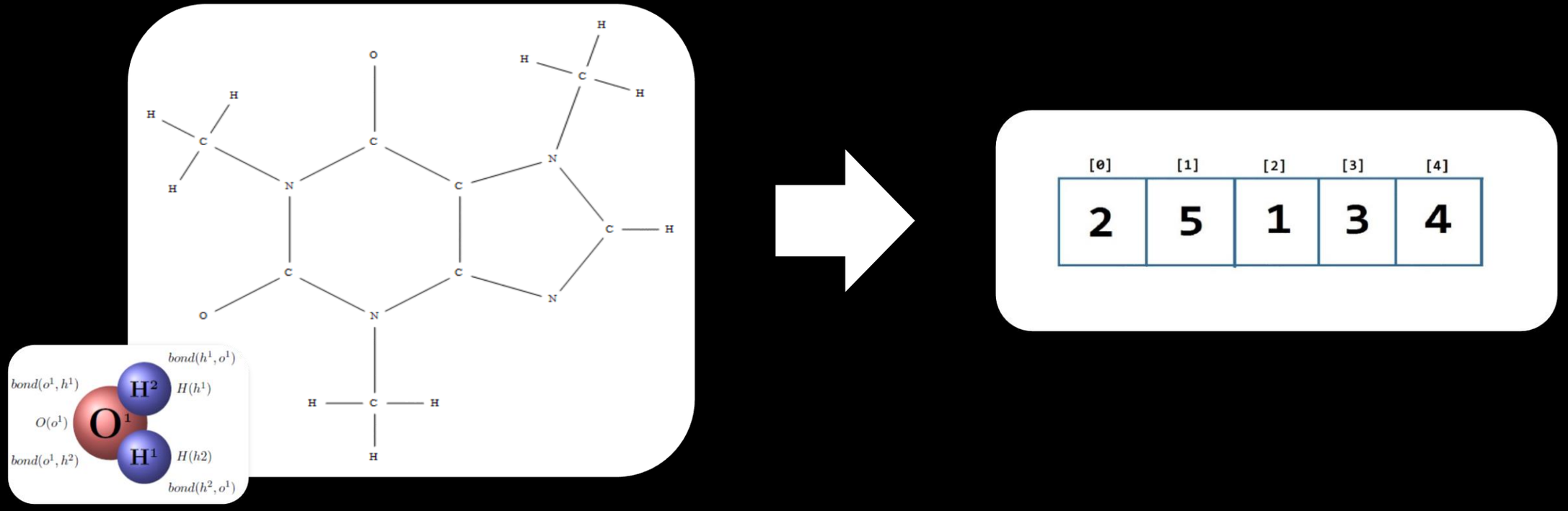
LRNN Motivation from Neural Networks' POV

Motivation (NN POV)

- How to learn with relational or graph-structured data?
  - Examples : molecules (networks, trees, etc.)
- How to represent data samples?
  - Sets of vertices & edges, relational logic clauses
  - Isomorphic samples should be treated the same!
- How to feed them into a classifier, a neural network?

Propositionalization

- Idea : turn arbitrary graph into a fixed-size vector
  - Through a **predefined** aggregation mapping



- Powerful, yet need to predefine all useful patterns

Auxiliary concepts

- There may be useful **sub-structures** present
- For instance, halogen groups in a molecule (mutagenicity) classification problem
  - e.g., **C-Br, C-Cl, C-F** may be indicative
  - i.e., there is a useful pattern **C-(halogen atom)**
- We can **predefine** these in the feature-vector

Latent predicate invention

- What if we do not know any of the useful sub-structures of the problem in advance?
  - e.g., we do not know there is something like halogens or other indicative group of atoms

→ We may design anonymous predicates for these patterns

- And learn these in a way such that they are useful in different contexts (rules) *(Muggleton,1988)*

→ Neural learning of latent (non ground) patterns

- This is beyond the scope of propositionalization

LRNNs

- We propose a framework avoiding the aforementioned limitation of propositionalization
- **Lifted Relation Neural Networks (LRNNs)**
- Inspiration:
  - Lifted (templated) graphical models: Markov Logic Networks*(Richardson, Domingos,2005)*, Bayesian Logic Programs*(Kersting, De Raedt,2000)*
  - Neural-symbolic approaches: KBANN*(Towel, Shavlik,1994)*, CILP*(Franca, Zaverucha, Garcez,1999)*

LRNN Motivation from Markov Logic POV

Motivation (Markov Logic POV)

- How to learn with relational or graph-structured data in the presence of uncertainty?
  - **Lifted** graphical **models**, e.g. Markov Logic
- How to ***efficiently*** learn latent concepts?
  - Neural Networks (propositional concepts)
- How about latent *relational* concept learning?
  - **Lifted Relational Neural Networks**

What is LRNN?

short version

What is LRNN? (short version)

- Syntactically: **Set** of weighted first-order Horn clauses
  - *0.5* : water :- bondOH(X,Y)
  - *1.0* : bondOH(X,Y) :- H(X), O(Y), bond(X,Y)
    - LRNN encoding looks familiar - like a weighted Prolog program…
- Semantically: **Template** for **neural network construction**
  - *We turn the template's Herbrand models into NNs as follows..*

Network Construction

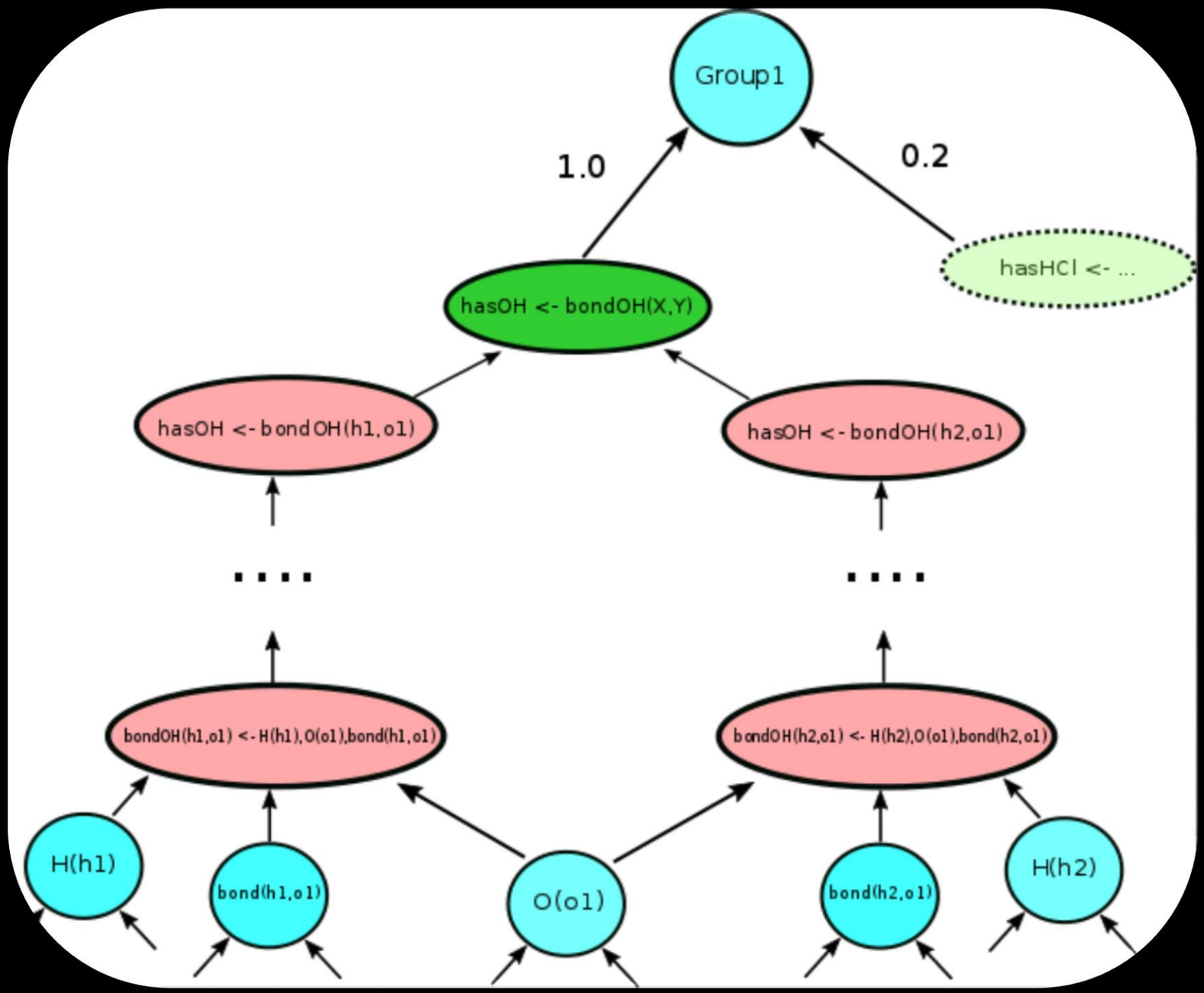
1. Every ground proposition (atom) which can be *derived* * from a given LRNN model corresponds to an **atom neuron**

2. Every **ground rule** $h \leftarrow (b_1, \ldots, b_k)$ such that $(b_1, \ldots, b_k)$ can be *derived* * from a given LRNN corresponds to a **rule neuron**

3. To aggregate different groundings derived with the same rule's ground head $\{h \leftarrow (b^1_1, \ldots, b^1_k), \ldots, h \leftarrow (b^n_1, \ldots, b^n_k)\}$ there is an **aggregation neuron**

* meaning it is present in the least Herbrand model

Putting it all together…



Weight Learning

- LRNN model := grounding of {*sample*, *template*} clauses
  - ➢ Different samples result in different ground networks
  - ➢ This induces **weight sharing** across ground networks as their neurons are *tied* to the same template rules

- Different aggregation functions are used as neurons' activations so as to reflect the (fuzzy) logic of **disjunction**, **conjunction**, and different forms of **aggregative reasoning** over relational patterns

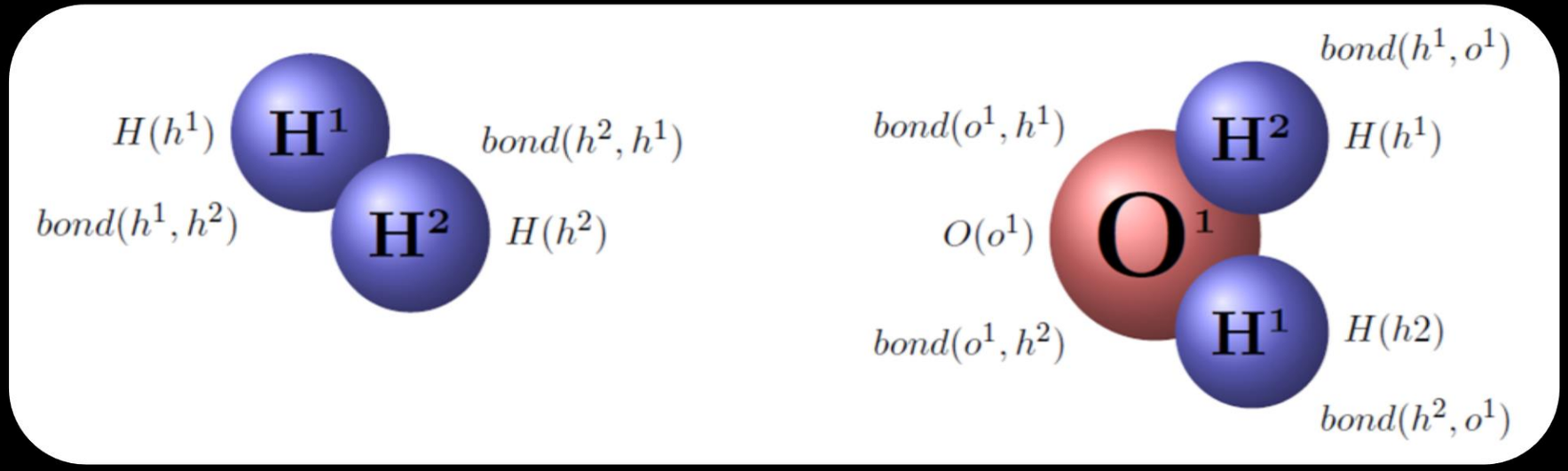
- **Stochastic Gradient Descend** can be used for training

What is LRNN?

Long version

Data representation

- No propositionalization or feature vector transformation
- Similarly to LRNNs, we represent **samples** simply as raw **sets of** corresponding **facts** (typically ground unit clauses)



- A simple **set union {}** of a LRNN template with a relational sample can thus be though of simply as another LRNN

LRNN construction

- LRNN := union of a sample and template **clauses**

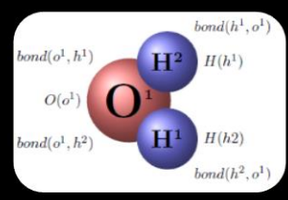
  →Different samples result in different LRNNs

  - Template remains the same

- We introduce building blocks of LRNN construction, these are 3 different types of neurons :
  atom neurons, **rule neurons**, **aggregation neurons**

Atom Neurons

- Every ground proposition (atom) which can be *derived** from a given LRNN corresponds to an **atom neuron**
- Example LRNN:
  - Template : 1.0 : bondOH(X,Y) :- H(X), O(Y), bond(X,Y).
  - Sample : H(h1), H(h2), O(o1), bond(h1,o1), bond(h2,o1)



→ Set of all atom neurons:

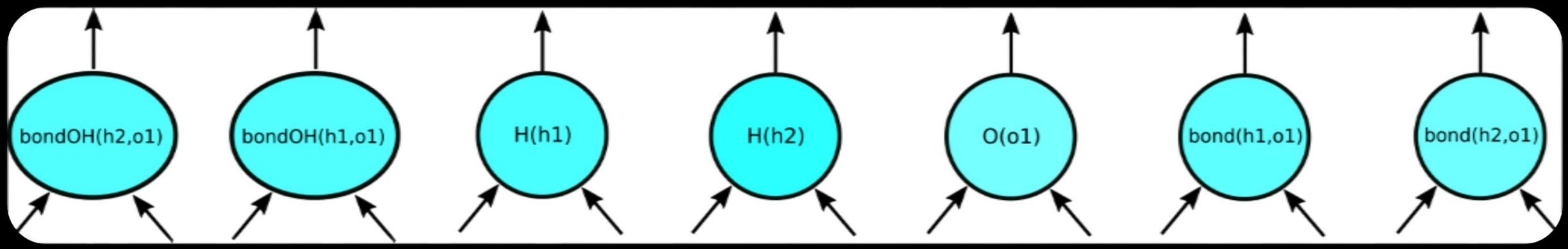
- $\{ N_{H(h1)}, N_{H(h2)}, N_{O(o1)}, N_{bond(h1,o1)}, N_{bond(h2,o1)}, N_{bondOH(h1,o1)}, N_{bondOH(h2,o1)} \}$

(* Meaning present in the least Herbrand model of it)

Atom Neurons

- Every ground proposition (atom) which can be *derived** from a given LRNN corresponds to an **atom neuron**
- Example LRNN:
  - Template : 1.0 : bondOH(X,Y) :- H(X), O(Y), bond(X,Y).
  - Sample : H(h1), H(h2), O(o1), bond(h1,o1), bond(h2,o1)

→ Set of all atom neurons:



Rule neurons

- Every **ground rule** h ← (b_1, …, b_k) such that (b_1, …, b_k) can be *derived** from a given LRNN corresponds to a **rule neuron**
- Example LRNN:
  - Template : 1.0 : bondOH(X,Y) :- H(X), O(Y), bond(X,Y)
  - Sample : H(h1), H(h2), O(o1), bond(h1,o1), bond(h2,o1)

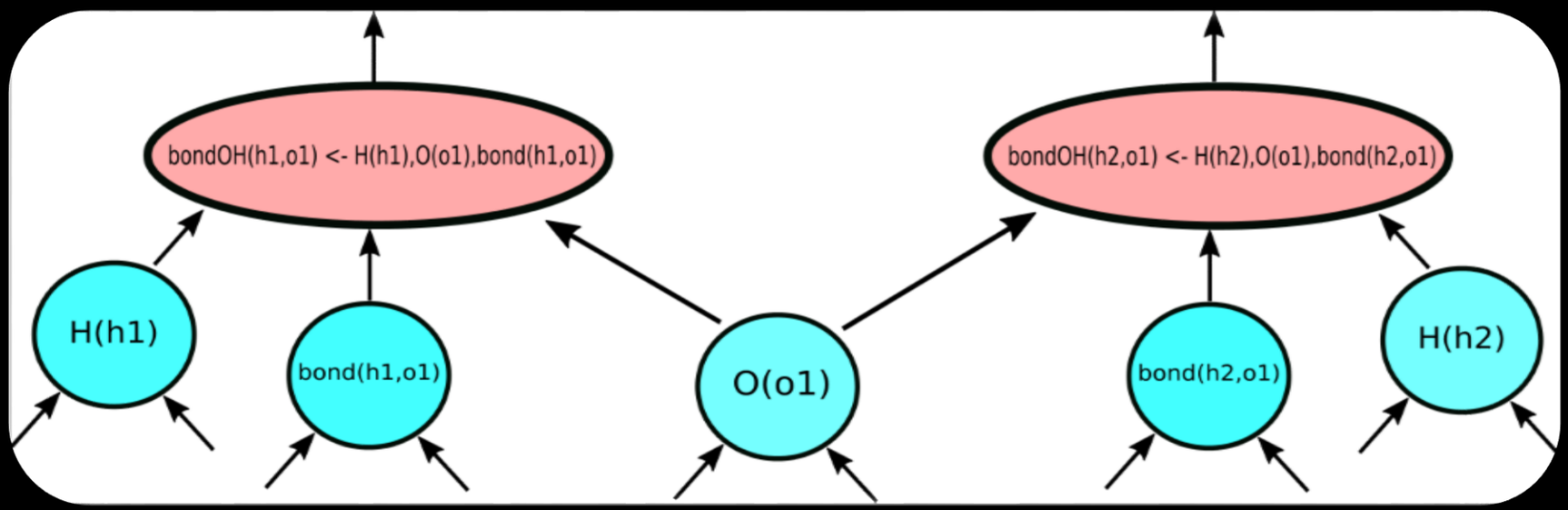
→ Set of all rule neurons:

$N_{bondOH(h1,o1)}$← H(h1), O(o1), bond(h1,o1) , $N_{bondOH(h2,o1)}$← H(h2), O(o1), bond(h2,o1)

(*Meaning the atoms are true in the least Herbrand model)

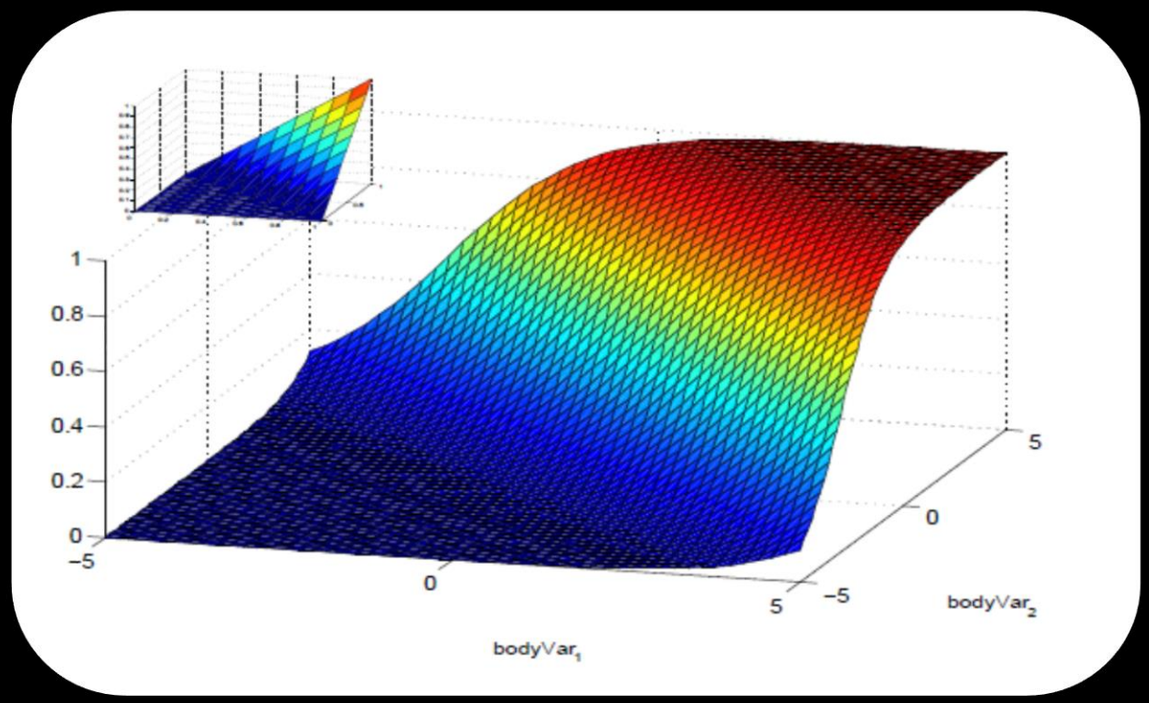
Rule neurons

- Every **ground rule** h ← (b_1, …, b_k) such that (b_1, …, b_k) can be *derived** from a given LRNN corresponds to a **rule neuron**
- Example LRNN:
  - Template : 1.0 : bondOH(X,Y) :- H(X), O(Y), bond(X,Y)
  - Sample : H(h1), H(h2), O(o1), bond(h1,o1), bond(h2,o1)
- → Set of all rule neurons:



Rule neuron activation

- Rule neuron basically represents **conjunctive** If-Then rule
  - This should be reflected in its activation function
- → Rule neuron has high output if and only if **all** the input atom neurons (rule's body) have high outputs
- Fuzzy logic inspiration :

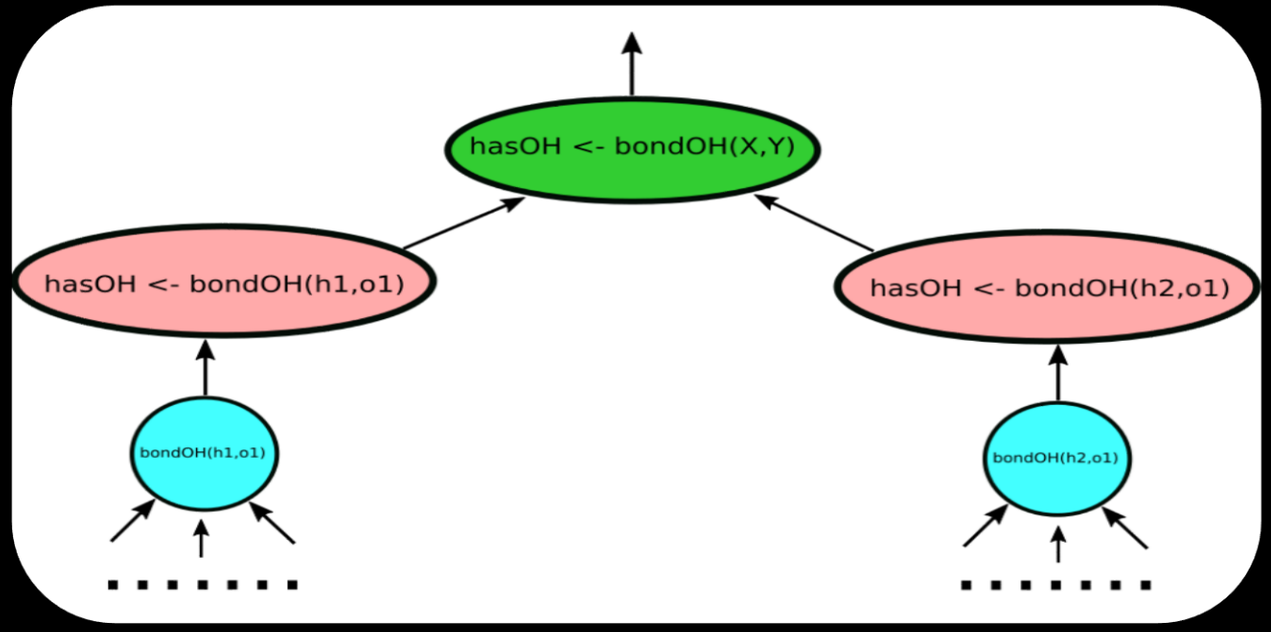


Aggregation neurons

- We need to aggregate different groundings of the same non-ground rule having the same ground literal in the head. For each such aggregation there is an **aggregation neuron.**
- Example LRNN:
  - Template : 1.0 : hasOH :- bondOH(X,Y)
    1.0 : bondOH(X,Y) :- H(X), O(Y), bond(X,Y)
  - Sample : H(h1), H(h2), O(o1), bond(h1,o1), bond(h2,o1)
- Set of different ground rules for hasOH :- bondOH(X,Y) corresponds to neurons:
  - $N_{hasOH \leftarrow bondOH(h1,o1)}$, $N_{hasOH \leftarrow bondOH(h2,o1)}$
- Aggregation neuron $N_{hasOH \leftarrow bondOH(X,Y)}$ **aggregates** over these

Aggregation functions

- Different aggregation functions might be used for different logic of aggregation neurons
- MAX – corresponds to “best pattern” matching



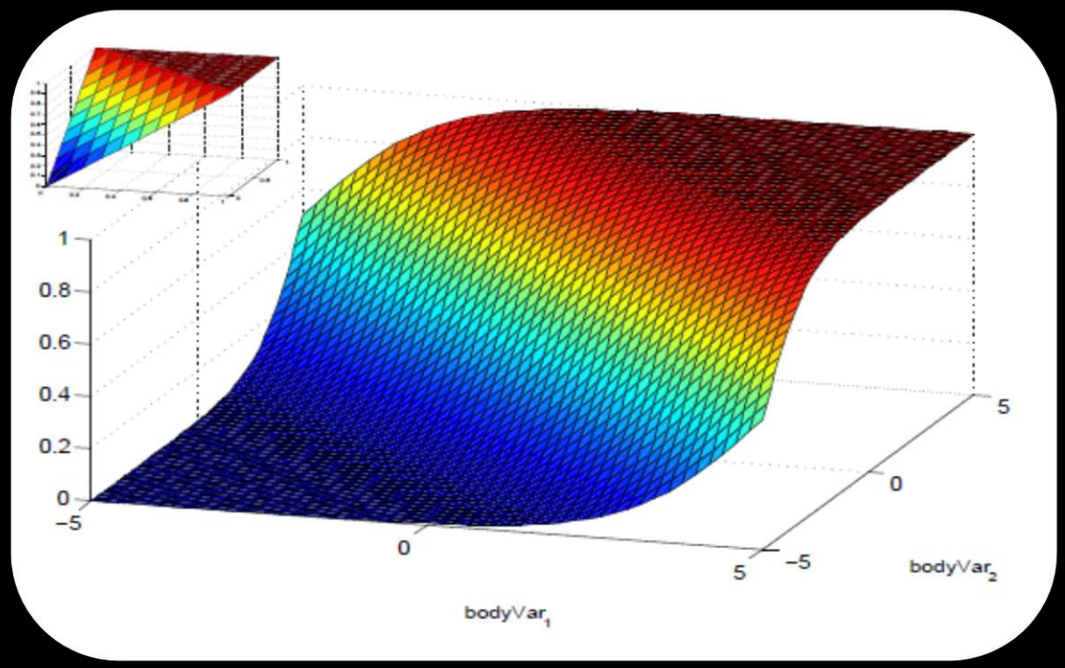
- Possibilities in other contexts include, e.g., AVG

Atom neuron inputs

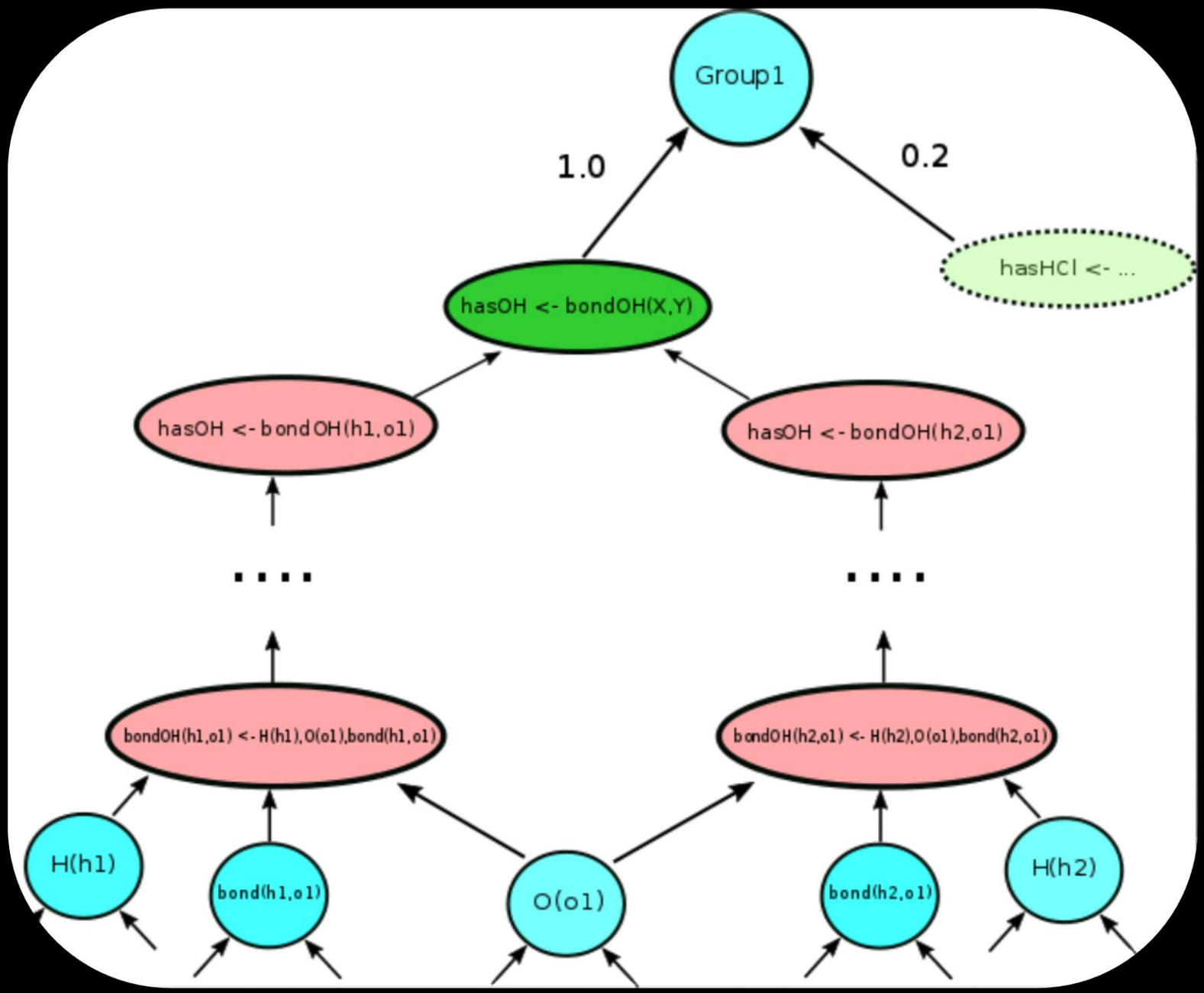
- There may be multiple weighted rules with the same ground head, yet with **different weights**
- Example template:
  - 1.0 : Group1 :- hasOH
  - 0.2 : Group1 :- hasHCl
- I.e. we end up with two different aggregation neurons with different weights:
  - $1.0 : N_{\text{Group1 :- hasOH}}$ and $0.2 : N_{\text{Group1 :- hasHCl}}$
- These finally form the **inputs of atom neuron N_{Group1}**

Atom neuron activation

- Combining different rules implying the same atom naturally corresponds to **disjunction**
- Atom neuron output should be high if and only if **at least one** of the rule neurons has high output
- Fuzzy logic inspiration :



Putting it all together…



Weight Learning

- The constructed **ground LRNN** can be thought of as a regular neural network with **shared weights**
- The shared weights come from grounding of same template's clause and exploit sample regularities
  - Similarly to convolutional neural networks this does not pose any problem to weight learning
- **Stochastic Gradient Descend** (SGD) with mild adaptions can be efficiently used for training

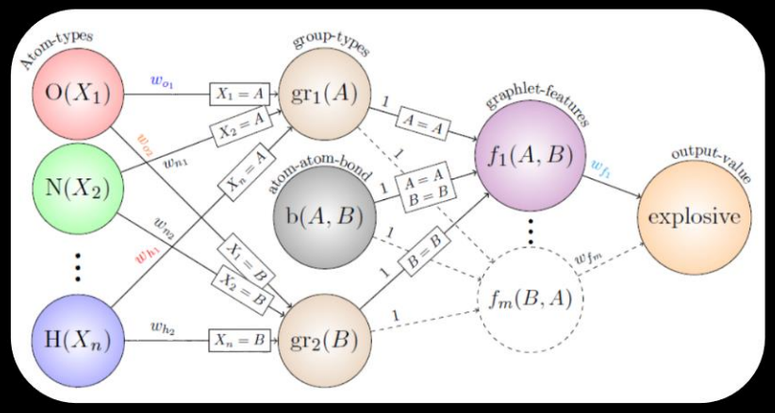
Experiments

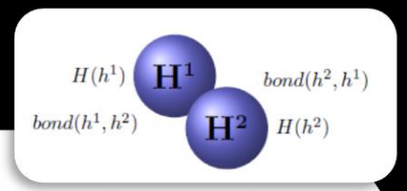
Experiment template

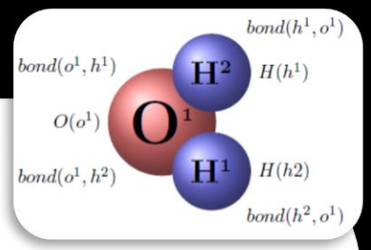
0.0 atomGroup1(X) :- o(X).
0.0 atomGroup1(X) :- cl(X).
....
0.0 atomGroup3(X) :- cl(X).

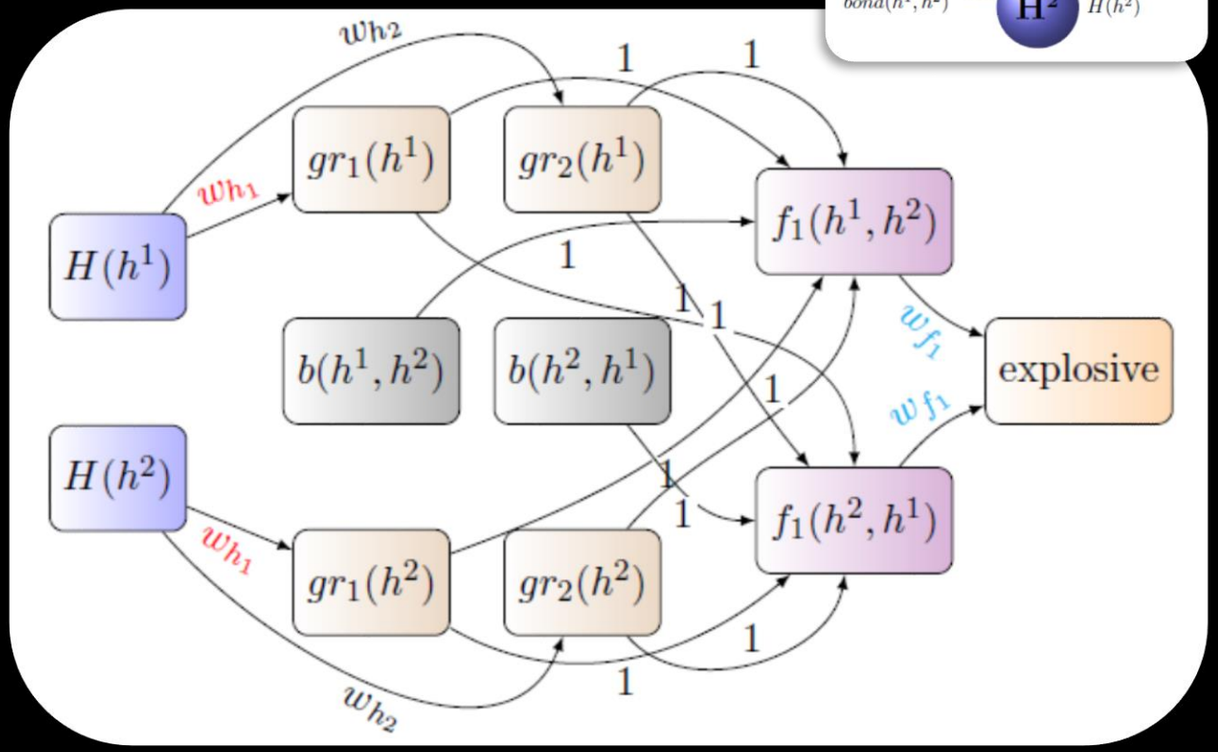
0.0 bondGroup3(X) :- 2=(X).
….
graphlet0 :- atomGroup2(X), bond(X,Y,B1),
bondGroup1(B1), atomGroup3(Y)…
….
0.0 class1 :- graphlet0.
….
0.0 class1 :- graphlet242.

Samples


Atom-types
group-types
graphlet-features
atom-atom-bond
output-value
O(X_1)
N(X_2)
H(X_n)
gr$_1$(A)
gr$_2$(B)
b(A, B)
$f_1(A, B)$
$f_m(B, A)$
explosive

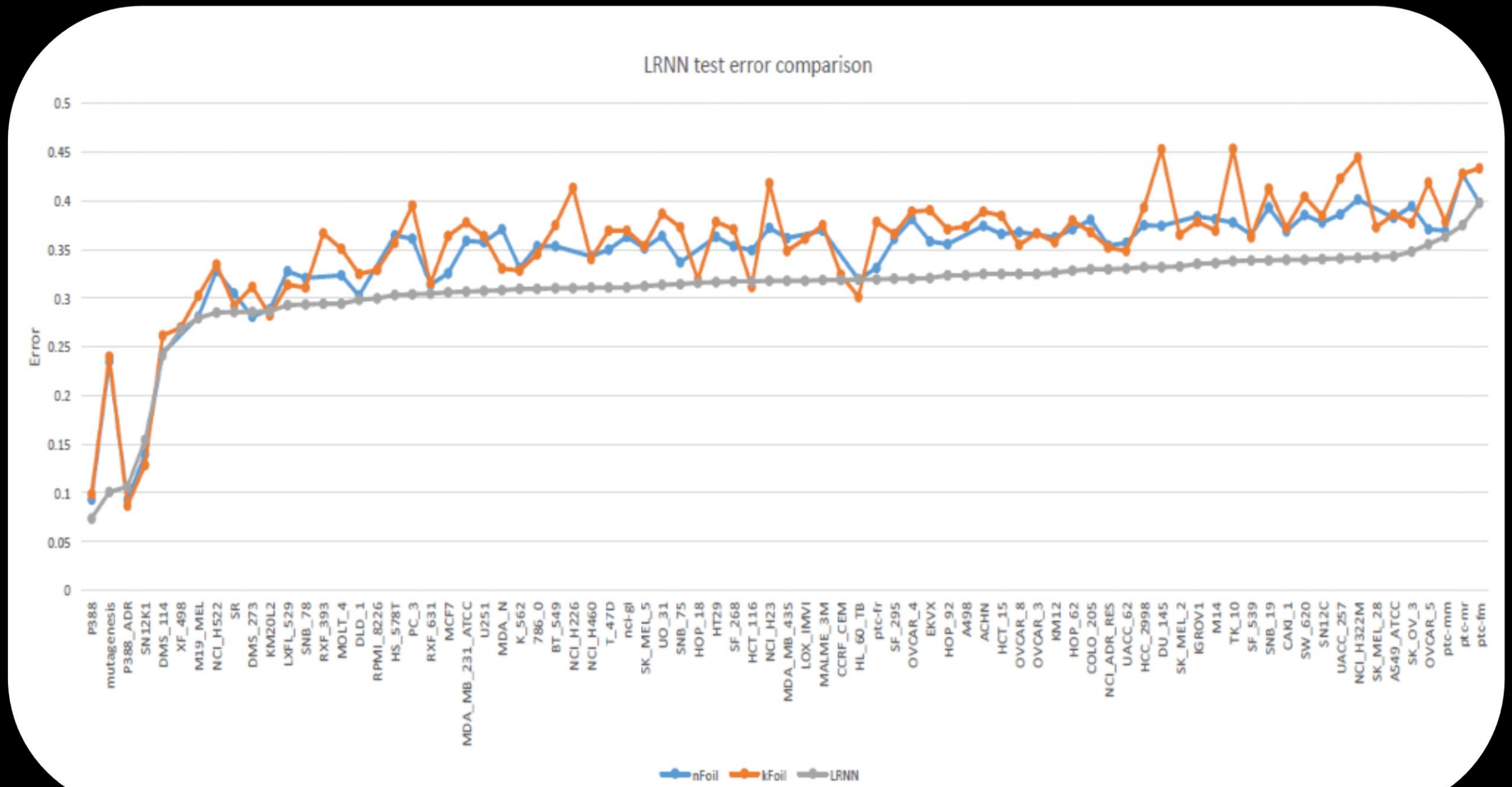

$H(h^1)$
$bond(h^1, h^2)$
$bond(h^2, h^1)$
$H(h^2)$


$bond(h^1, o^1)$
$bond(o^1, h^1)$
$O(o^1)$
$bond(o^1, h^2)$
$H(h^1)$
$H(h2)$
$bond(h^2, o^1)$


wh_2
wh_1
w_{h_2}
$H(h^1)$
$H(h^2)$
$gr_1(h^1)$
$gr_2(h^1)$
$gr_1(h^2)$
$gr_2(h^2)$
$b(h^1, h^2)$
$b(h^2, h^1)$
$f_1(h^1, h^2)$
$f_1(h^2, h^1)$
w_{f_1}
explosive

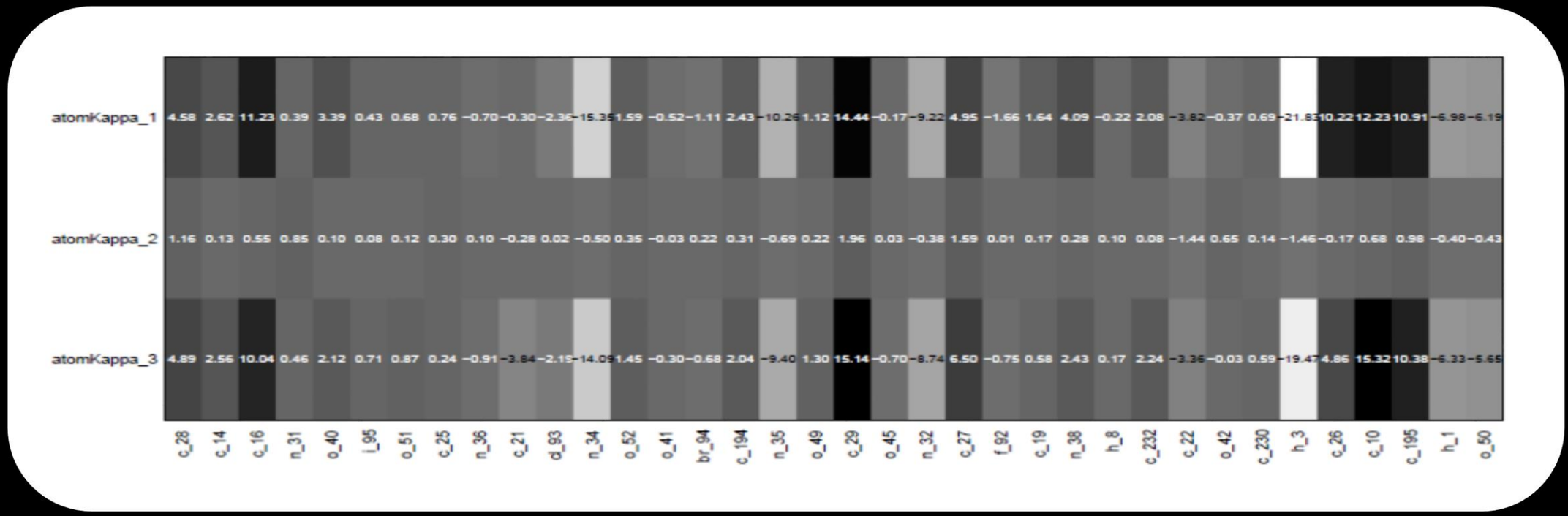
wh_2
wh_1
w_{o_1}
w_{o_2}
$H(h^1)$
$H(h^2)$
$O(o^1)$
$gr_1(h^1)$
$gr_2(h^1)$
$gr_1(h^2)$
$gr_2(h^2)$
$gr_1(o^1)$
$gr_2(o^1)$
$b(h^1, o^1)$
$b(o^1, h^1)$
$b(o^1, h^2)$
$b(h^2, o^1)$
$f_1(h^1, o^1)$
$f_1(o^1, h^2)$
$f_1(h^2, o^1)$
w_{f_1}
explosive

Results


LRNN test error comparison
Error
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
P388
mutagenesis
P388_ADR
SN12K1
DMS_114
XF_498
M19_MEL
NCI_H522
SR
DMS_273
KM20L2
LXFL_529
SNB_78
RXF_393
MOLT_4
DLD_1
RPMI_8226
HS_578T
PC_3
RXF_631
MCF7
MDA_MB_231_ATCC
U251
MDA_N
K_562
786_0
BT_549
NCI_H226
NCI_H460
T_47D
nci-gl
SK_MEL_5
UO_31
SNB_75
HOP_18
HT29
SF_268
HCT_116
NCI_H23
MDA_MB_435
LOX_IMVI
MALME_3M
CCRF_CEM
HL_60_TB
ptc-fr
SF_295
OVCAR_4
EKVX
HOP_92
A498
ACHN
HCT_15
OVCAR_8
OVCAR_3
KM12
HOP_62
COLO_205
NCI_ADR_RES
UACC_62
HCC_2998
DU_145
SK_MEL_2
IGROV1
M14
TK_10
SF_539
SNB_19
CAKI_1
SW_620
SN12C
UACC_257
NCI_H322M
SK_MEL_28
A549_ATCC
SK_OV_3
OVCAR_5
ptc-mm
ptc-mr
ptc-fm
nFoil
kFoil
LRNN

Where was latent predicate Invention?

- Different modeling concepts exploiting predicate invention
- Particularly, implicit soft clustering:



- Other concepts include soft-matching, hypergraph approximation, relational autoencoders,…

Learning Predictive Categories Using *Lifted Relational Neural Networks*

Gustav Sourek[1] , Suresh Manandhar[2] , Filip Zelezny[1] , Steven Schockaert[3] , and Ondrej Kuzelka[3]

1) Czech Technical University in Prague, Czech Republic
{souregus, zelezny}@fel.cvut.cz
2) Department of Computer Science, University of York, UK
suresh.manandhar@york.ac.uk
3) School of CS & Informatics, Cardiff University, UK
{SchockaertS1, KuzelkaO}@cardiff.ac.uk

Learning Predictive Categories with LRNNs

Learning Predictive Categories

We consider a following (learning) scenario with latent categories:

1. Entities
   - a) Have properties, b) Belong to **categories**
     - **Categories** largely determine belonging entities' properties
2. Properties
   - a) Belong to entities, b) Belong to **categories**
     - **Categories** largely determine entities satisfying a property

Learning Predictive Categories

1. *Given* : a set of entities and corresponding lists of their properties

2. *Assumption* : there exists some latent hierarchy of categories that are predictive of their corresponding object's properties

   - The hierarchy should allow for property inheritance

     - Similarly we induce latent hierarchy on properties

3. *Goal* : Learn suitable category structures from data

Encoding in LRNN

- Given input samples : {1/0 HasProperty(e, p)
- Membership to categories : w_{ec} : IsA(e, c)
- Category hierarchy : w_{c1c2} : IsA(c_1, c_2)
- Category properties : w_{cecp} : HasProperty(c_e, c_p)
- Transitivity : w_{isa} : IsA(A, C) ← IsA(A, B),IsA(B, C)
- Categories determine their entities' properties :
  w'_{cecp} : HasProperty(A, B) ←
  IsA(A, c_e), IsA(B, c_p), HasProperty(c_e, c_p)

Learning Setting

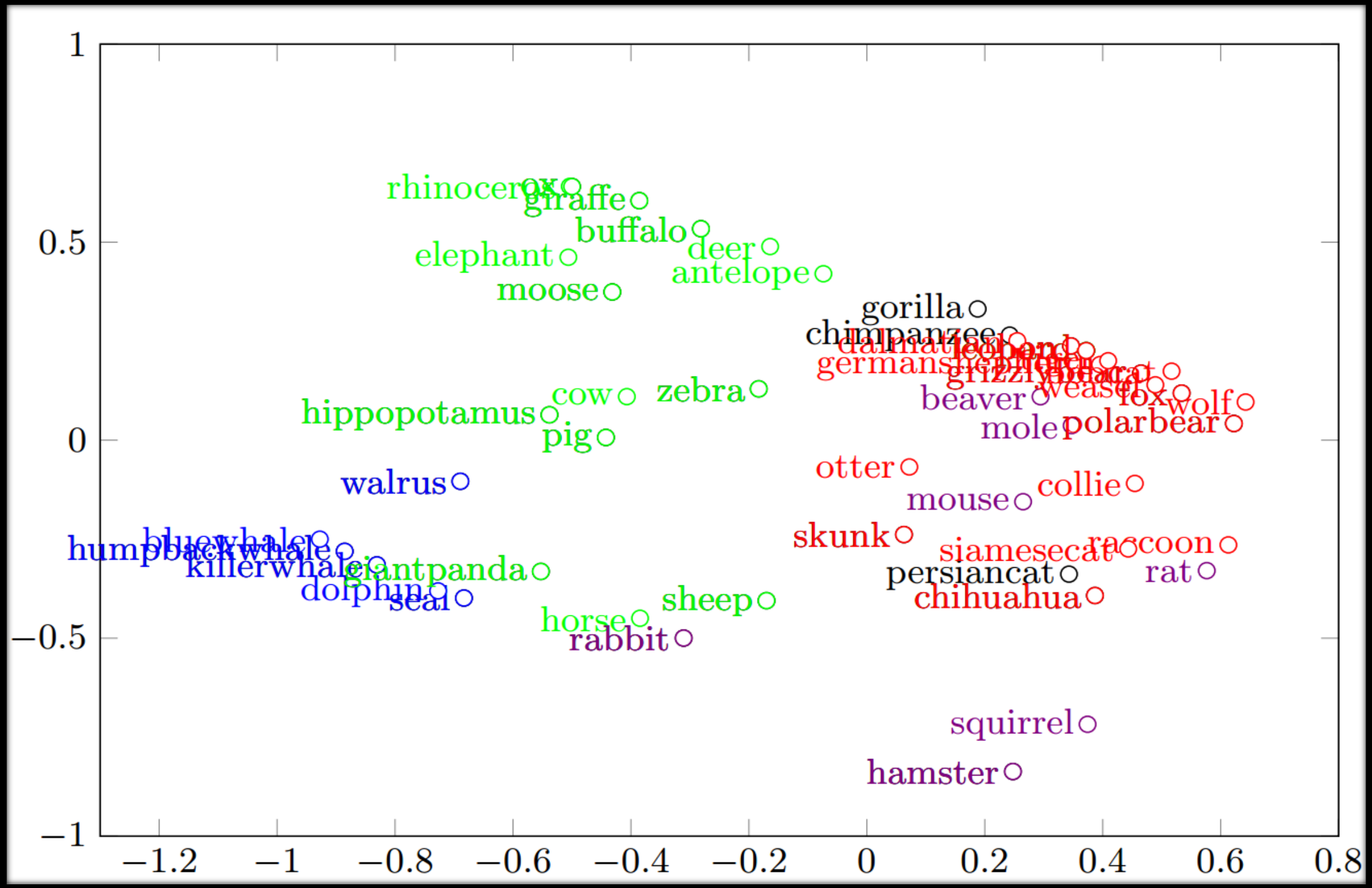
- We minimize MSE of the query atom neuron outputs and their targets {1/0 HasProperty(e, p)} via SGD
- The activation functions used were
  - Conjunction $\wedge_{(b1, \ldots, bk)} = \text{sigm}\left(\sum_{i=1}^{k} b_i - k + b_0\right)$
  - Disjunction $\vee_{(b1, \ldots, bk)} = \text{sigm}\left(\sum_{i=1}^{k} b_i + b_0\right)$
  - Aggregation $*_{(b1, \ldots, bm)} = \max_i b_i$
- We set up 2 level hierarchy with [3, 2] hidden categories for both objects and properties

Evaluation

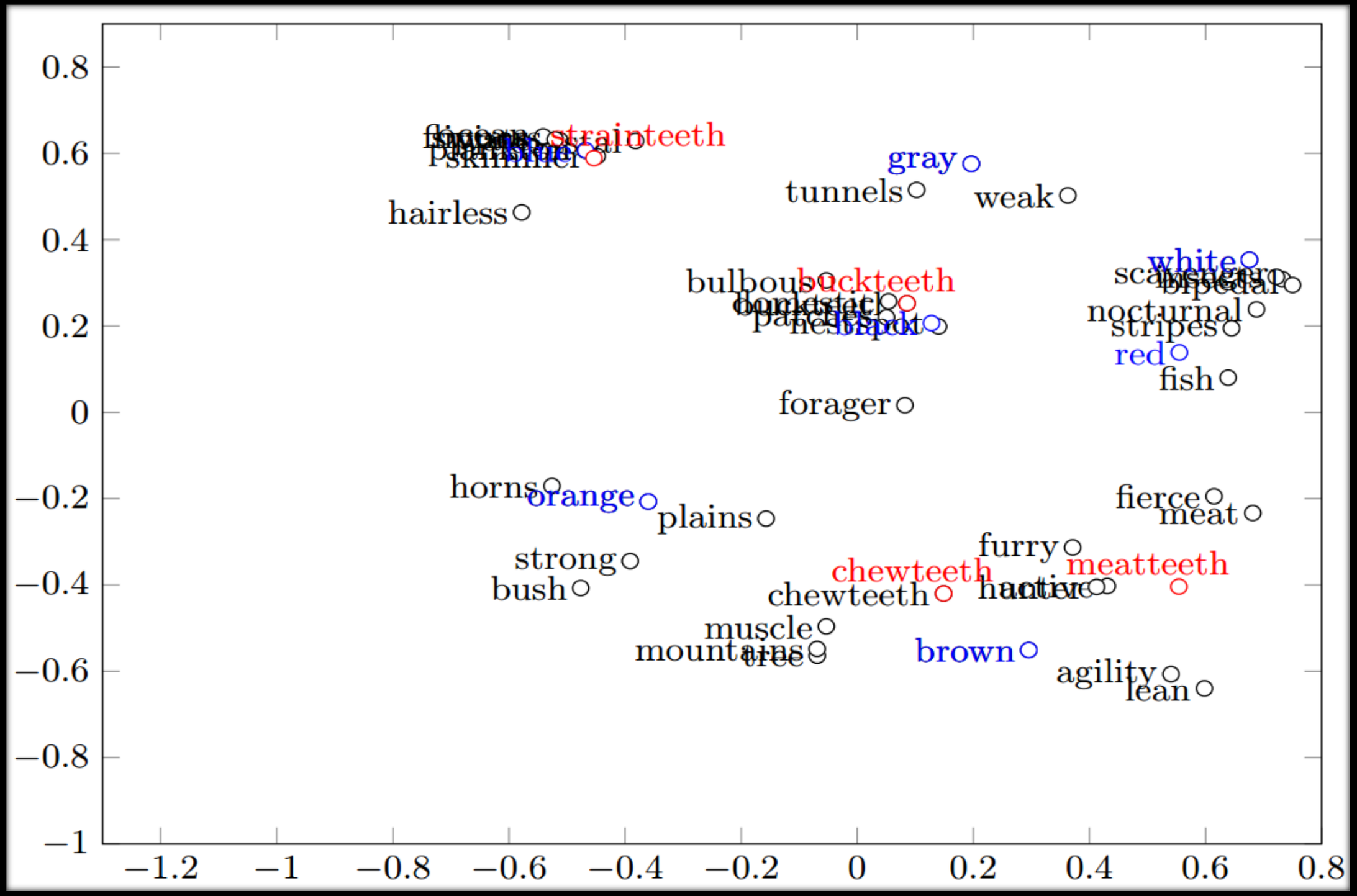
- Animals dataset (https://alchemy.cs.washington.edu/data/animals)
  - 50 animals + 65 properties (e.g., large, smelly, strong,…)

- Predictive ability : AUC PR 0.8, AUC ROC 0.86
- Same as with second order Markov Logic Networks, reported in (*Statistical Predicate Invention*, Kok and Domingos, 2007)
  - Which is related to the introduced model, while jointly clustering objects and relations

Embeddings of entities



Embeddings of properties



Outlook

- We have learned implicit similarity measure via latent category membership degrees

- We might also incorporate explicit similarities as
  w_l : HasProperty(A, B)
  ← HasProperty(C, B), Similar(A, C, l)
  - Where l denotes some level of similarity, e.g. based on externally obtained embeddings
  - With that we might emulate 1-NN or kernel regression

- Also whole triples of (subject, *predicate*, object) might be considered to learn soft categories of predicates, too

Conclusions

- LRNNs are a flexible framework to easily encode non-trivial SRL scenarios
  - e.g., a joint learning of predictive categories of entities and their properties
- More complicated settings might be easily reached with just mild extensions of the template
  - e.g., semi-supervised learning, embeddings, etc.
- We plan for thorough comparison with MLNs and incorporation of LRNNs into NLP tasks pipelines

General conclusions

- LRNNs may be thought of as a neural analogy to lifted (templated) graphical models (e.g., Markov Logic Networks)
  - Both are template languages for defining and weight tying in the corresponding ground models
- Benefits
  - Latent (deep) relational concepts, Flexible templates (e.g., convolutional NN), Explicit variable binding
- Future work
  - Different modeling concepts, recurrent NNs, ASP, optimization, Structure learning inspired by meta-interpretive learning

Thank you!

See “Lifted Relational Neural Networks” at arXiv.org for more details

Convolutional NN

$$
\begin{array}{lcl}
w_0^{(1)} : f_1 & \leftarrow & left(A), mid(B), right(C),\ next(A,B), next(B,C) \\
w_1^{(2)} : left(X) & \leftarrow & f_0(X) \\
w_2^{(2)} : mid(X) & \leftarrow & f_0(X) \\
w_3^{(2)} : right(X) & \leftarrow & f_0(X)
\end{array}
$$

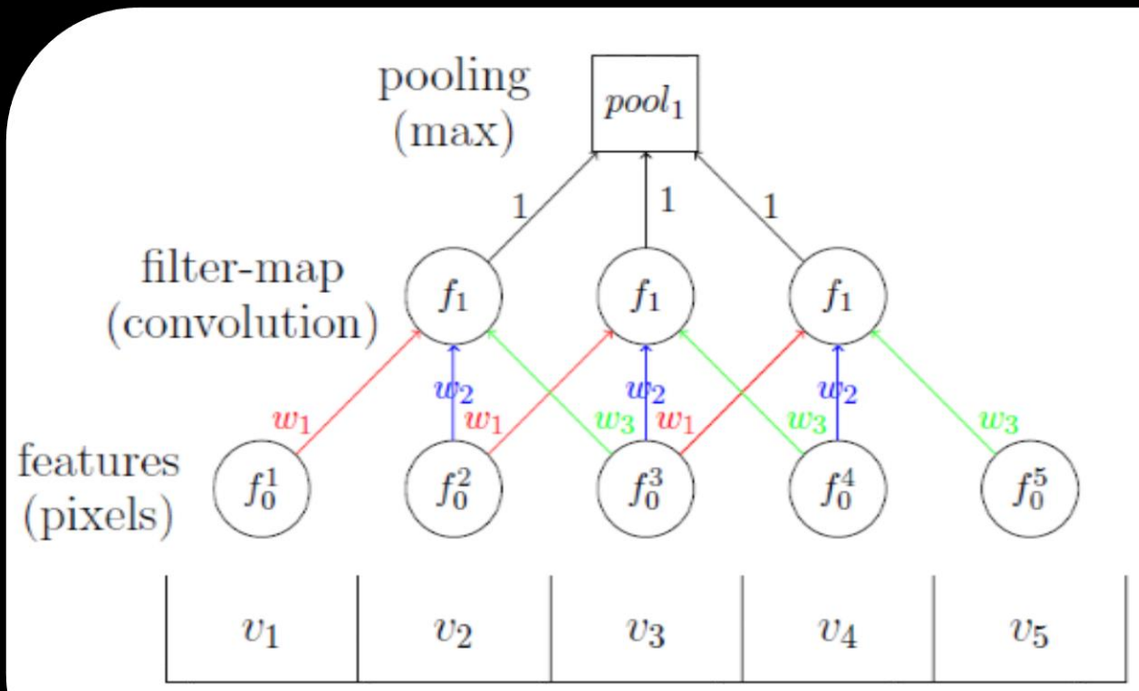


image I as a vector of pixel values

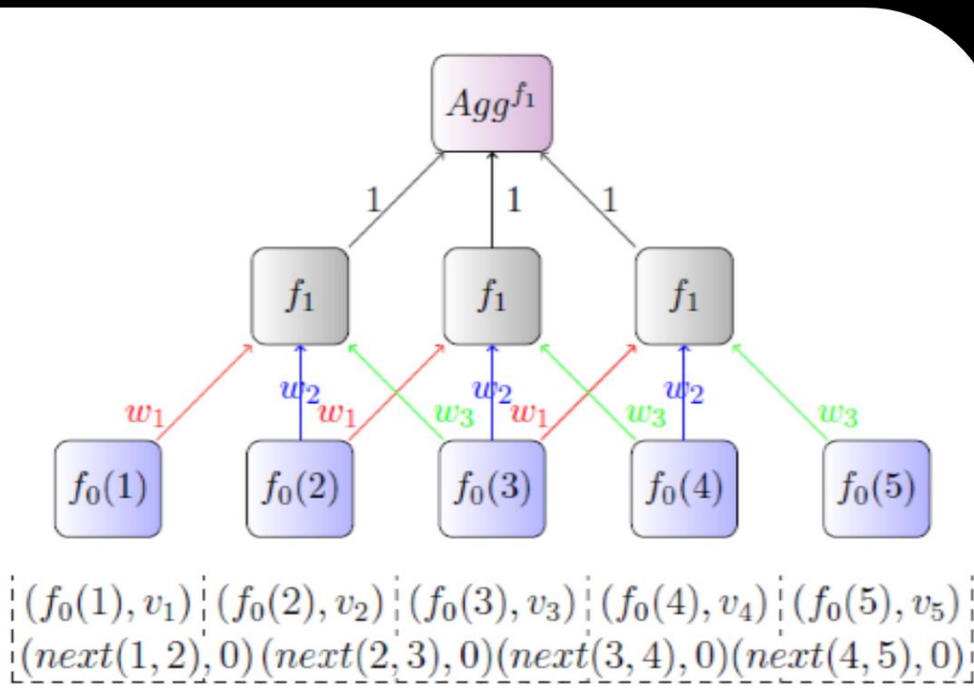


image I as a set of weighted facts