Symbolic Generative Models for Deep Learning

Anton R. Fuxjaeger

Master of Science by Research
School of Informatics
University of Edinburgh
2019

Abstract

Neural networks (NNs) as an architecture for modelling complex functions and mini-batch stochastic gradient decent as an optimization scheme for NNs have been studied for many years by the research community, producing impressive results in many domains. They provide a modelling language that is very adaptable to individual problems (classification, language modelling, image completion/segmentation, financial trading, etc.), scalable (by using GPU batch processing) and able to handle noisy data sufficiently. However, inherent limitations in their interpretability can lead to a number of problems in real world applications. This thesis challenges these limitations by using well defined methods from mathematical logic and symbolic artificial intelligence (AI). In particular, we use probabilistic sentential decision diagrams, a tractable circuit representation for probability distributions, as a high-level generative modelling language. Furthermore, variational autoencoders (VAEs) constitute a low-level model such that the combination of these two formalisms, in addition to a newly defined interface, make up a novel model capable of learning and representing a joint distribution over unstructured data. Such a formulation then enables us to ask complex queries after training as well as draw conditional samples from the learned distribution.

Acknowledgements

I wish to thank, first and foremost, my main supervisor Vaishak Belle for his continuous support and help on this project. His insights and guidance on problems I encountered proved to be indispensable. Furthermore I want to thank my secondary supervisor John Quinn for many insightful conversations and support throughout this project. I also want to address a special thank you to my family for their unwavering support throughout my academic career. Their financial and emotional backing over the years, not only enabled, but also encouraged me to pursue my interests into new fields and countries, and for that i am deeply grateful. I am grateful for many stimulating and enjoyable conversations with my peers, and in particular to Lewis Hammond and Ionela Gini Mocanu for proofreading earlier versions of this dissertation.

This work was supported by the Engineering and Physical Sciences Research Council (grant EP/L01503X/1), EPSRC Centre for Doctoral Training in Pervasive Parallelism at the University of Edinburgh, School of Informatics.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Anton R. Fuxjaeger*)

Table of Contents

Chapter 1

Introduction and Motivation

Over the past few years we have seen tremendous progress in the wider field of AI. The abundance of data has led the research community to new formalisms and architectures that incorporate and use this resource in order to learn functional representations of systems. Neural networks (NNs) as such an architecture, have been on the forefront of said development producing impressive results in many different domains including image classification [He *et al.* , 2016, Krizhevsky *et al.* , 2012] and language modelling [Mikolov *et al.* , 2010]. On the other hand the discovery of the network polynomial [Darwiche, 2003], and local structure have led the research community to new and increasingly powerful symbolic methods and models [Darwiche, 2011, Liang *et al.* , 2017, Liang & Van den Broeck, 2019]. Additionally, one of the main formalisms for probabilistic inference on propositional knowledge bases, weighted model counting (WMC) [Chavira & Darwiche, 2008] has been extended to hybrid domain [Belle *et al.* , 2015]. Such formalisms are an intricate part of symbolic methods, especially when it comes to learning the structure of a graphical network from data, where inference is used to justify manipulations to the structure.

1.1 Connectionist Artificial Intelligence

NNs as an architecture for modelling functions by means of learning or training from data, are considered to be connectionist systems. A NN is a directed graph. A unit or neuron in this graph describes a function $y_i = \sigma_i(\sum_j W_{ij}x_j + b_i)$, where x_j is the output from unit j, b_i is a bias variable, $\mathbf{W}_i$ is a weight vector and σ_i is the activation function of the neuron. Such a unit can also be graphically depicted as in Figure 1.1. All such neurons are represented as nodes in a graph where an edge from unit x to

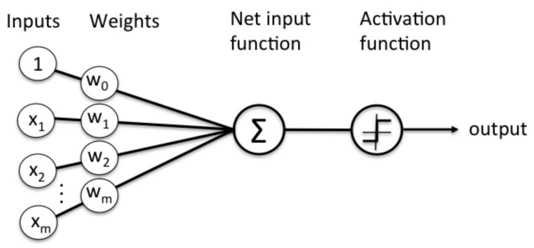


Figure 1.1: A unit or perceptron of a neural network. Here $1 * w_0 = b$

unit y indicates that $W_{yx} > 0$. In feed-forward network architectures, the computational graph is also acyclic, such that neurons can be arranged in so called *layers*, indicating the depth of a NN. This makes it possible to use matrix notation in order to express a given layer l as a function $\mathbf{h}^l = \sigma^l(\mathbf{W}^l h^{l-1} + \mathbf{b}^l)$ and the whole NN with L layers as:

$$\mathbf{h}^{L-1} = \sigma^{L-1}(\mathbf{W}^{L-1}(\sigma^{L-2}(\mathbf{W}^{L-2}(\ldots\sigma^{0}(\mathbf{W}^{0}\mathbf{X} + \mathbf{b}^{0})\ldots) + \mathbf{b}^{L-2})) + \mathbf{b}^{L-1})$$

Formally, [Hornik *et al.* , 1989] showed that standard multi-layer feed-forward networks are capable of approximating any measurable function to any desired degree of accuracy, also termed the universal approximation theorem.

Generally speaking the strength of NN lies in their strict specification of a loss or error function indicating the performance of the given NN. Minimizing said function for the given data and network architecture then constitutes the learning procedure. Here stochastic optimization procedures such as stochastic gradient decent are used in conjunction with back-propagation [Rumelhart *et al.* , 1985] to fit the weights of the network to the data and are responsible for their impressive results. Furthermore, the general formulation of NNs has allowed the research community to adapt the architecture for specific domains such as convolutional neural networks, used for image classification since they incorporate spatial information of the input data. Another example is given by recursive neural networks, which capture the recursive nature of language and are thus usually applied to such problems. Yet another reason why NNs as a modelling formalism are so interesting for the AI community is their ability to deal with incomplete and noisy data thanks to their connectionist nature. This makes them especially useful for many real-world problems.

Due to these properties, NNs have proven very useful in many areas, however they come with strong limitations as well. Firstly, it should be noted that NNs can only be used to learn functions. This means that a given problem must be specified by two disjoint sets, inputs ($\mathbf{X}$) and outputs ($\mathbf{Y}$) such that $\mathbf{X} \sqcup \mathbf{Y} = \mathbf{D}$ where $\mathbf{D}$ is the observed data. In terms of probability theory the network is then used to learn the conditional probability distribution of the output given the input $Pr(\mathbf{Y}|\mathbf{X})$. In a more practical sense this means that after training, the network can only be queried in the same way it was trained, with some input $\mathbf{x}$ to predict $\mathbf{y}$, other queries cannot be computed. Furthermore, the connectionist nature of the model gives no conceptional understanding how a given output is chosen for some input $\mathbf{x}$. Considering for example a classification task with n categories, we can use the model to classify a given ('unseen') input $\mathbf{x}$ and retrieve the corresponding predicted label/class y. While the correctness of the classification seems to be the main focus, understanding how a given output was predicted, that is how the NN reasons about the decision it is making, is often neglected. This inability to understand how decisions in a NN are made has recently led to some notable problems in the industry, where an example is given by an AI recruiting tool used by Amazon that showed a bias against women [1]. Other examples are given by individuals incorrectly being denied parole based on an algorithmic decision [Wexler, 2017].

1.2 Symbolic Artificial Intelligence

Models and learning architectures that fall in the category of symbolic AI are concerned with the mathematical manipulation and meaning of symbols representing variables. Incorporating logical notation, rules and methods, such symbols are put in relation to each other in order to express a given system or data. One noteworthy problem in this field is concerned with determining if a given propositional sentence (or knowledge base) is *satisfiable* (SAT). Although only considering propositional logic as it's underlying theory, the SAT problem has been shown to be translatable to many problems (e.g. planing, induction,...) such that exact and approximate solvers are still used in many applications today. Expert systems and inductive logic programming (ILP) are other examples of *purely* symbolic research areas within this field. The need for modelling uncertainty and emerging methods in probability theory lead the research community to a multitude of formalisms combining declarative logic with probability theory such as statistical relational learning (SRL). One class of this research area are probabilistic graphical models where Bayesian belief networks (BN) are a well known example. A BN is a directed acyclic graph (DAC), where vertices represent variables and edges their conditional dependencies, additionally to a conditional probability ta-

[1] https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G

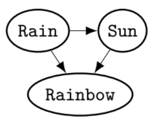
ble, see Figure 1.2a. Such a model can be learned from data and then used to reason about unknown variables given some evidence as well as infer new knowledge.

Another interesting example, more relevant to the proposed project are tractable circuit representations. Tractable circuits are graphical models representing probability distributions where querying is efficient (polytime). While historically tractable learning focused on low-tree-width graphical models [Zhang, 2004] [Bach & Jordan, 2002], the discovery of local structure in Bayesian networks led to arithmetic circuits [Darwiche, 2003], sum-product networks [Poon & Domingos, 2011], and finally to probabilistic sentential decision diagrams (PSDDs) [Kisa *et al.* , 2014], which are a much more powerful representation of tractable probability distributions. PSDDs are a probabilistic extension of sentential decision diagrams [Darwiche, 2011], which represent Boolean functions as logical directed acyclic circuits. Owing to their intricate structural properties, PSDDs support closed-form parameter learning, MAP inference, complex queries [Bekker *et al.* , 2015], and even efficient multiplication of distributions [Shen *et al.* , 2016], which are all increasingly rare. These strong properties permit the learning of PSDDs in probability spaces that are subject to complex logical constraints disallowing large numbers of possible worlds [Kisa *et al.* , 2014]. In this context, knowledge compilation algorithms can build PSDD structures without looking at the data. PSDDs can be learned from data [Liang *et al.* , 2017], possibly with the inclusion of logical constraints standing for background knowledge. The ability to encode logical constraints into the model directly enforces sparsity which in turn can lead to increased accuracy and decreased size. Furthermore learning a PSDD from data, approximates the complete joint probability distribution $Pr(D_1, D_2, \ldots, D_n)$ where $\mathbf{D}$ denotes the data. This enables us to reason about *any* variable subset given some data after learning.

1.3 Comparison

When it comes to comparing the two types of approaches to AI (connectionist/symbolic), there are three major aspects we would like to highlight in this report;

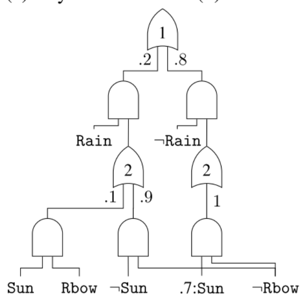
- Firstly, symbolic AI learning systems are mostly generative, such that they try to learn the given system as a whole. In probability theory for example we assume that all data ($\mathbf{D}$) is drawn from some unknown joint probability distribution over all variables ($Pr(D_1, D_2, ..., D_n)$). Symbolic methods such as PSDDs then try to learn this joint distribution, which allows us to reason about *any* variable subset



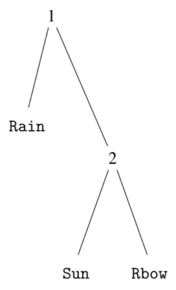
(a) Bayes net

$$\Pr(\texttt{Rain}) = 0.2,$$
$$\Pr(\texttt{Sun} \mid \texttt{Rain}) = \begin{cases} 0.1 \text{ if } & \texttt{Rain} \\ 0.7 \text{ if } & \neg\texttt{Rain} \end{cases}$$
$$\Pr(\texttt{Rbow} \mid \texttt{R}, \texttt{S}) = \begin{cases} 1 \text{ if } \texttt{Rain} \wedge \texttt{Sun} \\ 0 \text{ otherwise} \end{cases}$$

(b) Conditional probabilities



(c) Equivalent PSDD circuit



(d) PSDD's vtree

Figure 1.2: A Bayesian network and its equivalent PSDD [Liang *et al.* , 2017]

given some data after learning. NNs as a connectionist system however, do not attempt to learn the complete distribution, but rather only learn the distribution $Pr(\mathbf{Y}|\mathbf{X})$, where $\mathbf{X} \sqcup \mathbf{Y} = \mathbf{D}$. This of course means that at a later stage (after learning) we can only reason about Y given some $\mathbf{X}$. In other words; we cannot ask new questions.

- Secondly, the performance of connectionist systems (NN) over symbolic ones is as of now dramatically better. This is due to their powerful numeric learning methods (stochastic gradient decent) as well as their adaptability to a specific data format (image/text/sound).

- Thirdly, we see that symbolic architectures such as SSDs are highly parallelizable due to their structural decomposability property [Darwiche & Marquis, 2002]. NNs on the other hand are optimized for batch processing which can be viewed as local parallelization, such that one batch is computed in parallel (using GPUs and vector/matrix operations) but multiple batches have to be run iteratively.

- And finally we would like to emphasize the understandability we as humans have of the reasoning of a given AI system. As already mentioned, this is very difficult for connectionist systems, and is usually done by a complicated ad-hoc procedure that only gives an indication rather than a full understanding. Symbolic systems on the other hand describe the reasoning they perform by design.

Since it is possible to translate a system into mathematical logic, a given system can sometimes even be represented graphically in a meaningful (and human readable) format (Bayesian/Markov Networks).

1.4 Neuro-Symbolic Systems

As we have seen so far, symbolic and connectionist AI complement each other in the sense that the strength of the one is the weakness of the other. The idea of combining the two types of systems (connectionist/symbolic) into one architecture is therefore by no means new. Neuro-symbolic learning systems have been studied for some time, producing interesting results [Yi *et al.* , 2018, Fischer *et al.* , 2018, Hu *et al.* , 2016], but are usually very intricate and can not compete with the simpler models in terms of accuracy and performance.

One such system particularly interesting to our research is a probabilistic logic programming language that incorporates neural predicates [Manhaeve *et al.* , 2018] and as such extends ProbLog [De Raedt *et al.* , 2007]. Here, a neural predicate is the output of a neural network (for classification), to be used as a propositional predicate with a corresponding probability. Interestingly enough, a given probabilistic logic program is first specified, including the neural predicates (by specifying the NN architecture). In the next instance the neural predicates (parameters) are learned from data, using grounding, knowledge compilation via SDDs and stochastic gradient decent based optimization. Considering for example the predicate $addition(X,Y,Z)$, where X and Y are images of digits and Z is the natural number corresponding to the sum of these digits. After training, DeepProbLog allows us to make a probabilistic estimate on the validity of, e.g. $addition(img(3),img(5),8)$, where $img(3)$ is used as a placeholder for an image of a handwritten 3. While such a predicate can be directly learned by a standard neural classifier, such a method would have a hard time considering background knowledge such as the definition of the addition of two natural numbers. In DeepProbLog such knowledge can easily be encoded in rules in terms of logic. All that needs to be learned in this case is the neural predicate digit which maps an image of a digit to the corresponding natural number.

1.5 Explainable Artificial Intelligence and Generative Models

Another research area in computer science that has received more attention over the last few years is termed *explainable AI* [Gunning, 2017]. As the name suggests this field is concerned with devising models as well methods for learning and analysis that are inherently explainable. Explainable in this context refers to systems that are not only interpretable but are also capable of explaining the decisions they make to some extend. Generally speaking there seems to be trade-off between explainability and accuracy, such that more explainable models are less accurate and vice-versa [Murdoch *et al.* , 2019]. Therefore, we have seen a trend in post-hoc explanations for inherently explainable models such a neural networks [Rudin, 2018].

Generative models as a research area in the wide field of AI, is concerned with learning the the complete joint probability distribution over the data, and as such ties in quite well with explainable AI. While not all generative models are fully explainable they are usually at least sufficiently inerpreteable such that we can understand the working of the model in some meaningful sense. Examples of generative models are then given by linear classifiers, Gaussian mixture models, variational autoencoders, Bayesian methods and restricted Boltzmann machines (RBM).

RBMs constitute a two layer network structure where the first layer represents the (binary and/or real-valued) visible units $\mathbf{v}$, and the second one the binary hidden ones $\mathbf{h}$. Both layers are put in relation to each other by a symmetric weight matrix W. The probability over hidden and visible units is then defined as $P(\mathbf{v},\mathbf{h}) = \frac{1}{Z}\exp -E(\mathbf{v},\mathbf{h})$, where E is the energy function and Z the partition function. Depending on the the visible units being Binary the energy function is then defined as $E(\mathbf{v},\mathbf{h}) = -\sum_{i,j} v_i W_{ij} h_j - \sum_j b_j h_j - \sum_i c_i v_i$, where b_j are hidden units biases and c_i are visible unit biases. The partition function on the other hand is simply the sum of $\exp -E(\mathbf{v},\mathbf{h})$ for all possible configurations. For RMSs [Hinton, 2002] proposed a training algorithm, that uses contrastive divergence approximation of the intractable gradient of the log-likelihood, in combination with stochastic gradient ascent. This method does work sufficiently well, but does not circumvent the inherent limitation of the representational power of this two layer architecture. Therefore RBMs are usually stacked in many layers to form the much more powerful model termed deep believe network (DBN). DBNs are fully generative and come with a fast training algorithm [Hinton *et al.* , 2006] that trains each layer greedily bottom-up as a RBM using the previous layers hidden units as the inputs for the next one. This formulation has then been adapted to various domains (e.g. images, audio) showcasing impressive results, such as learning hierarchical image representations [Lee *et al.* , 2009] or learning acoustic models [Mohamed *et al.* , 2011].

Chapter 2

Background - Technical

2.1 Graphical Models & Tractable Circuit Representations

Throughout this paper we will refer to Boolean, discrete and continuous random variables as B_j, C_l and X_i respectively for some $i, j, l \in \mathbb{N}$. Lower case letters, $b_j \in \{0, 1\}$, $c_j \in \Delta_j$ and $x_i \in \mathbb{R}$, will represent the instantiations of these variables. Here Δ is the simplex of variable C_j, such that it spans the finite space of all possible instantiations of said variable. Furthermore, bold upper case letters will denote sets of variables and bold lower case letters will denote their instantiation.

Now consider a probabilistic model, defined on $\mathbf{B}$ and $\mathbf{X}$ and let

$$(\mathbf{b}, \mathbf{x}) = (b_1, b_2, ..., b_m, x_1, x_2, ...x_n)$$

be one element in the probability space $\{0, 1\}^m * \mathbb{R}^n$, denoting a particular assignment to the values in the respective domains. A graphical model can then be used to describe dependencies between the variables and define a joint density function of those variables compactly. The graphical models we will consider in this paper are Markov networks, which are undirected models. (Encoding a directed graphical model as a weighted propositional theory is discussed in [Chavira & Darwiche, 2008].) Roughly, the nodes of the graph are Boolean functions taking real and Boolean variables and the edges represent dependencies. We can then compactly factorize the joint density function in terms of the cliques of the graph [Koller & Friedman, 2009]:

$$Pr(\mathbf{b}, \mathbf{x}) = \frac{1}{Z} * \prod_k \phi_k(\mathbf{b}_k, \mathbf{x}_k)$$

where $\mathbf{b}_k$ and $\mathbf{x}_k$ are those random variables participating in the k^{th} clique and $\phi_k(.,.)$ is a non-negative, real-valued potential function. Here ϕ_k is not necessarily denoting a probability and so Z is used as a normalizing constant, also called the partition function defined as [Belle *et al.* , 2015]:

$$Z = \sum_{B_1} \cdots \sum_{B_m} \int_{X_1} \cdots \int_{X_n} \left[\prod_k \phi_k(\mathbf{b}_k, \mathbf{x}_k) \right] d\mathcal{X}.$$

2.2 Probabilistic Inference by Weighted Model Counting

Section has been taken from [Fuxjaeger & Belle, 2018]:
Weighed model counting (WMC) [Chavira & Darwiche, 2008] is a strict generalization of model counting [Biere *et al.* , 2009]. In WMC, each model of a given propositional knowledge base (PKB) Γ has an associated weight and we are interested in computing the sum of the weights that correspond to models that satisfy Γ. (As is convention, the underlying propositional language and propositional letters are left implicit. We often refer to the set of literals $\mathcal{L}$ to mean the set of all propositional atoms as well as their negations constructed from the propositions mentioned in Γ.)

In order to create an instance of the WMC problem given a PKB Γ and literals $\mathcal{L}$, we define a weight function $wf : \mathcal{L} \to \mathbb{R}^{\geq 0}$ mapping the literals to non-negative, numeric weights. We can then use the literals of a given model m to define the weight of that model as well as the weighted model count as follows:

Definition 1.: Given a PKB Γ over literals $\mathcal{L}$ and weight function $wf : \mathcal{L} \to \mathbb{R}^{\geq 0}$, we define the weight of a model m as:

$$\text{WEIGHT}(m, wf) = \prod_{l \in m} wf(l)$$

Further we define the weighted model count (WMC) as:

$$\text{WMC}(\Gamma, wf) = \sum_{m \models \Gamma} \text{WEIGHT}(m, wf)$$

Based on the above definition, it can be observed that if $wf(l) = 1$ for all l, the weight of a model m obtained via WEIGHT(m, wf) is always 1, and thus we are just counting the number of satisfying models, in other words, computing the model count.

2.2.1 Probabilistic Inference in Markov Networks by Weighted Model Counting

WMC can be used to calculate probabilities of a given graphical model [Chavira & Darwiche, 2008]. As discussed earlier, the simplest instance, an undirected model can be represented as a weighted propositional theory in that the nodes of the graph are Boolean functions taking real and boolean variables, and the edges represent dependencies, from which one gets:

Theorem 2.: *[Belle et al. , 2015] Let $\mathcal{N}$ be a Markov network over Boolean random variables $\mathbf{B}$ and potentials $\phi_1, .., \phi_k$. Now let Γ be the PKB that encodes the network structure and wf be the weight function. Then*

$$Pr_{\mathcal{N}}(q|\mathbf{e}) = \frac{\mathrm{WMC}(\Gamma \wedge q \wedge \mathbf{e}, wf)}{\mathrm{WMC}(\Gamma \wedge \mathbf{e}, wf)}$$

for some evidence $\mathbf{e}$ *and query q, where* $\mathbf{e}, q$ *are PKBs as well, defined for* $\mathbf{B}$.

2.3 Sentential Decision Diagram

Sentential decision diagrams (SDDs) were first introduced in [Darwiche, 2011] and are graphical representations of propositional knowledge bases. SDDs are shown to be a strict subset of deterministic decomposable negation normal form (d-DNNF), a popular representation for probabilistic reasoning applications [Chavira & Darwiche, 2008] due to their desirable properties. Decomposability and determinism ensure tractable probabilistic (and logical) inference, as they enable MAP queries in Markov networks. SDDs however satisfy two even stronger properties found in ordinary binary decision diagrams (OBDD), namely structured decomposability and strong determinism. Thus, they are strict supersets of OBDDs as well, inheriting their key properties; canonicity and a polynomial time support for Boolean combination. Finally SDDs also come with an upper bound on their size in terms of tree-width.

2.3.1 Structure

This section has been taken from [Fuxjaeger, 2018]

2.3.1.1 Structured Decomposability, Strong Determinism and Vtrees

Consider the Boolean function $f(\mathbf{B_Z})$ such that $\mathbf{B_Z} = \mathbf{B_X} \sqcup \mathbf{B_Y}, \mathbf{B_X} \cap \mathbf{B_Y} = \emptyset$. If p_i and s_i are further Boolean functions and $f = (p_1(\mathbf{B_X}) \wedge s_1(\mathbf{B_Y})) \vee ... \vee (p_n(\mathbf{B_X}) \wedge s_n(\mathbf{B_Y}))$, then

$\{(p_1, s_1), ..., (p_n, s_n)\}$ is called a $(\mathbf{B_X}, \mathbf{B_Y}) - decomposition$ of f since it allows us to express f purely in terms of functions on $\mathbf{B_X}$ and $\mathbf{B_Y}$ [Pipatsrisawat & Darwiche, 2010]. Formally, a conjunction is decomposable if each pair of its conjuncts share no variables. Now if $p_i \wedge p_j \equiv false$ for $i \neq j$ the decomposition is considered to be strongly deterministic. In such a case the pair (p_i, s_i) is called an *element* of the decomposition and p_i, s_i the element's prime and sub respectively [Darwiche, 2011]. However the decomposition used by SDDs has structural properties as well, that build on the notion of the vtrees [Pipatsrisawat & Darwiche, 2010].

Definition 3.: A vtree for a set of variables $\mathbf{B_Z}$ is a full, rooted binary tree whose leaves are in one-to-one correspondence with the variables in $\mathbf{B_Z}$.

Figure 2.1a represents a possible vtree for the function: $f = (B_A \wedge B_B) \vee (B_B \wedge B_C) \vee (B_C \wedge B_D)$. In this paper v is used to denote a vtree node. v^l and v^r are used to represent the left and right child respectively. Furthermore, each vtree induces a total variable order that is obtained by a left-right traversal of the tree.

2.3.1.2 The Syntax and Semantics of SDDs

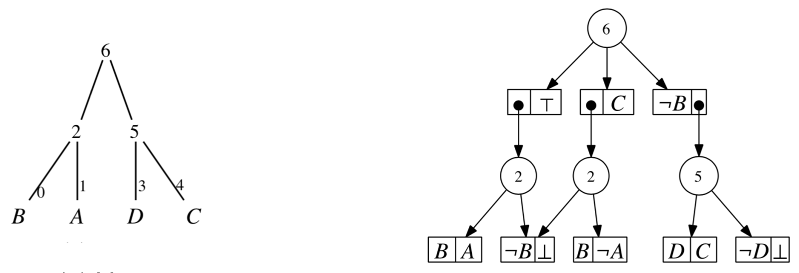
Here $\langle . \rangle$ is used to specify a mapping from an SDD to a Boolean function.

Definition 4: *α is an SDD that respects vtree v iff:*

- $alpha = \bot$ or $\alpha = \top$
  Semantics:$\langle \bot \rangle = false$ and $\langle \top \rangle = true$
- $\alpha = B_X$ or $\alpha = \neg B_X$ and v is a leaf with variable B_X
  Semantics:$\langle B_X \rangle = B_X$ and $\langle \neg B_X \rangle = \neg B_X$
- $\alpha = \{(p_1, s_1), ..., (p_n, s_n)\}, v$ is internal,
  $p_1, ..., p_n$ are SDDs that respect subtrees of v^l,
  $s_1, ..., s_n$ are SDDs that respect subtrees of v^r,
  and $\langle p_1 \rangle, ..., \langle p_n \rangle$ is a partition
  Semantics: $\langle \alpha \rangle = \bigvee_{i=1}^{n} \langle p_i \rangle \wedge \langle s_i \rangle$

The size of an SDD α, denoted $|\alpha|$, is obtained by summing the sizes of all its decompositions.

Boolean and literal SDD nodes are called *terminal nodes* and *decomposition/decision nodes* otherwise. Graphically, we represent a decision node by a circle with a number indicating the vtree node it respects, and elements of the decision node by boxes.



(a) Vtree

(b) SDD

Figure 2.1: Function: $f = (B_A \wedge B_B) \vee (B_B \wedge B_C) \vee (B_C \wedge B_D)$ [Darwiche, 2011]

Figure 3.3 depicts an SDD and the vtree it respects for the specified Boolean function. SDDs can therefore be used to represent a Boolean function in compact form.

2.3.1.3 Canonicity

Canonicity of SDDs is an especially interesting property introduced and proved in [Darwiche, 2011]. This will ultimately provide a bound on the SDDs we will be constructing in our framework.

Theorem 5.: *[Darwiche, 2011] Let α and β be compressed and trimmed SDDs respecting nodes in the same vtree. Then $\alpha \equiv \beta$ iff $\alpha = \beta$.*

2.3.1.4 Polytime Apply Operation

The *Apply* operation allows us to construct an SDD from any propositional KB in CNF by simply converting each clause into an SDD and then recursively combining all of the SDDs. The details can be found in [Darwiche, 2011] and the technique follows the same ideas as for the *Apply* operation of OBDDs. Any two SDDs can be conjoined as well as disjoined in polynomial time with respect to the size of the SDD. Furthermore, we can also negate an SDD in polytime, by doing an exclusive-or with $\top$ [Darwiche, 2011]. Formally, the *Apply* operation for a given Boolean connective and SDDs α_1 and α_2 returns the combined SDD in $O(|\alpha_1| * |\alpha_2|)$ time, where $|\alpha_x|$ denotes the size of an SDD.

2.3.1.5 Upper Bound on SDDs

Finally, the upper bound of SDDs can be expressed as a function of an arbitrary CNF formula:

Theorem 6.: *[Darwiche, 2011] A CNF with n variables and treewidth w has a compressed and trimmed SDD of size $O(n \cdot 2^w)$.*

While this means that the size of the SDD is still exponential in w, it is linear in the number of variables n. This means that SDDs come with a tighter bound on their size than binary decision diagrams (BDD), which is squared exponential in treewidth.

2.3.2 Inference

When it comes to inference in SDDs, the WMC formulation is used, allowing us to compute a given probabilistic query $Pr_{\mathcal{N}}(q|\mathbf{e})$ as a factor of two WMC 's by Theorem 2. $Pr_{\mathcal{N}}(q|\mathbf{e}) = \frac{\text{WMC}(\Gamma \wedge q \wedge \mathbf{e}, wf)}{\text{WMC}(\Gamma \wedge \mathbf{e}, wf)}$. As each WMC of a given SDD can be computed in one pass up the tree, the inference time is polynomial (tractable) in the size of the SDD.

2.3.3 Learning

The first algorithm for learning the structure of an SDD from data was proposed by [Bekker *et al.* , 2015] and is depicted in 1. The algorithm can start off with an initial SDD set to True, and then performs a greedy, general-to-specific search to simultaneously learn a Markov network and its underlying SDD. In the algorithms **D** stands for the dataset, e for the maximum number of edges and the term α is indicating the relative importance of fitting the data over the cost of inference (complexity of the SDD). For more information on how individual features are generated please refer to the original paper [Bekker *et al.* , 2015].

The score function used here is then defined as:

$$score(M, \mathbf{D}, \alpha) = [log Pr(\mathbf{D} \mid M') - log Pr(\mathbf{D} \mid M)] - \alpha \frac{|SDD_{M'}| - |SDD_M|}{|SDD_M|} \tag{2.1}$$

where M' is the model extended with feature f, M is the old model and $|SDD|$ return the number of edges in the SDD representation. Here we see that the first terms measures the improvement in the log-likelihood of the model due to adding feature f, while the second term represents the relative growth w.r.t. adding feature f. As previously mentioned, α is then defined by the user giving a relative importance of fitting the data vs. building a compact model.

Algorithm 1 LearnSDD($\mathbf{D}, e, \alpha$)

```
1:  initialize model $M$ with variables as features
2:  $M_{best} \leftarrow M$
3:  **while** number of edges $|SDD_M| < e$ **and not** timeout **do**
4:      $best\_score = -\infty$
5:      $D \leftarrow$ generateFeature(M,D)
6:      **for** each feature $f \in F$ **do**
7:          $M' \leftarrow M.add(f)$
8:          **if** score($M', \mathbf{D}, \alpha$) > best_score **then**
9:              best_score = score($M', \mathbf{D}, \alpha$)
10:             $M_{best} \leftarrow M'$
11:         **end if**
12:     **end for**
13:     $M \leftarrow M_{best}$
14: **end while**
```

2.4 Probabilistic Sentential Decision Diagram

Building on SDDs [Kisa *et al.* , 2014] proposed Probabilistic Sentential Decision Diagrams (PSDDs), a complete and canonical representation of a joint probability distribution over the variables of the network. Here each parameter in the tree structure can be viewed as a conditional probability of making a decision in the underlying SDD. In practice PSDDs have proven to represent real-world data much better than SDDs due to their probabilistic nature. Thus using PSDDs, we are able to relax the strict propositional rules of SDDs such that a given rule or propositional formula (represented within the underlying SDD) is annotated with a probability, allowing (outlying) examples to 'contradict' the propositional rules of the SDD. While this might seem undesirable at first glance, it much better models real-world scenarios where examples contradicting rules is not uncommon, due to noise in the measurements (the nature of our universe) and the influence of hidden variables that can not be observed by the measurements of data. Furthermore, [Liang & Van den Broeck, 2019] proposed a learning regime for such tractable representations that is capable of learning a PSDD as well as the underlying SDD and vtree [Pipatsrisawat & Darwiche, 2010] from data directly. The proposed algorithm is also much more scalable in terms of the number of variables in comparison to the learnSDD algorithm.

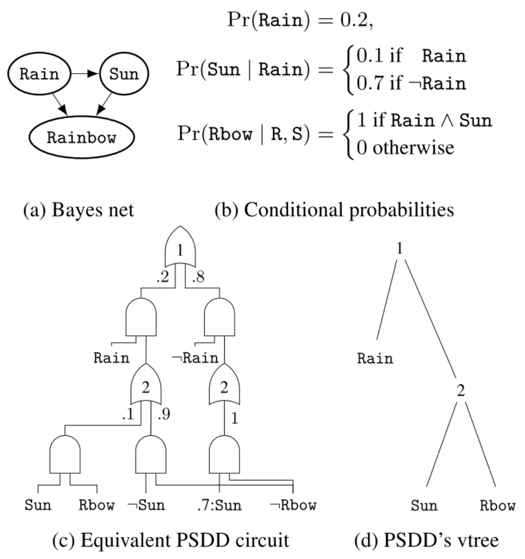

Rain
Sun
Rainbow
Pr(Rain) = 0.2,
Pr(Sun | Rain) = {0.1 if Rain; 0.7 if ¬Rain}
Pr(Rbow | R, S) = {1 if Rain ∧ Sun; 0 otherwise}
(a) Bayes net
(b) Conditional probabilities
1
.2
.8
Rain
¬Rain
2
.1
.9
2
1
Sun
Rbow
¬Sun
.7:Sun
¬Rbow
(c) Equivalent PSDD circuit
1
Rain
2
Sun
Rbow
(d) PSDD's vtree

Figure 2.2: A Bayesian network and its equivalent PSDD [Liang *et al.* , 2017]

2.4.1 Structure

When it comes to the structure of PSDDs, they are represented as parametrized directed acyclic graphs (DAGs). As previously mentioned they are probabilistic extensions of SDDs [Darwiche, 2011] which are directed acyclic graphs representing boolean formulas or a propositional knowledge base. In both cases we are dealing with strictly Boolean variables, such that any variables can either be true or false ($b_j \in \{true, false\}$ for variable B_j). However in PSDDs each terminal node represents a univariate (Bernoulli) distribution over a variable e.g. B_j, such that b_j is ether always true, always false ($-b_j$) or true with a probability θ_j represented by the tuple $(\theta_j : B_j)$. Within the tree, each note is either an AND or OR node. As in SDDs a given AND node has two inputs termed prime p for the left one and sub s for the right one. The OR node on the other hand, can have an arbitrary number of inputs, where each of the n input wires is annotated by a probability $\theta_1, ..., \theta_n$ together making up a normalised distribution over the variables represented by the corresponding vtree. Now OR and AND gates always alternate, such that a given OR node can also be represented as a set of AND nodes or decisions; $\{(p_1, s_1, \theta_1), ..., (p_n, s_n, \theta_n)\}$.

Now, in order to retain the desirable properties of SDDs for inference (tractability) and canonicity, similar syntactic restrictions hold here as well. Firstly each of the AND gates has to be *decomposable*, meaning that the vtree nodes represented by prime and sub share no variables. In other words the prime and sub have to represent probability distributions over disjoint sets of variables. As explained in more detail in

Section 2.3.1.1, a vtree is a full binary tree where each leaf is labelled with a variable. Internal nodes of the vtree then split the variables into two sets, the ones appearing in the left subtree $\mathbf{B_p}$ and the ones appearing in the right one $\mathbf{B_s}$. This partition of variables at a given vtree node then corresponds to the partition of variables at a decision node within the PSDD respected by this vtree node. Thus the primes of the decision nodes within the tree range over the variables in $\mathbf{B_p}$ and the subs over the variables in $\mathbf{B_s}$ [Kisa *et al.* , 2014]. This is also depicted in Figure 2.2 where each decision node is labelled with the vtree node it 'respects'.

Another syntactic restriction on the structure of PSDDs is termed *determinism*. This restriction demands that for each possible world, there can be at most one prime that assigns a non-zero probability to this world. This again is a probabilistic version of the *strong determinism* property of SDDs.

When it comes to the semantics of PSDDs, each node represents a probability distribution over the variables, the vtree node it respects, spans. Each decision node q is normalized for a given vtree node with $\mathbf{B_p}$ and $\mathbf{B_s}$ in its left and right sub-trees respectively, represents a distribution over $\mathbf{B_p}\mathbf{B_s}$ as

$$Pr_q(\mathbf{B_p}\mathbf{B_s}) = \sum_i \theta_i Pr_{p_i}(\mathbf{B_p}) * Pr_{s_i}(\mathbf{B_s})$$

Using these semantics, the PSDD in Figure 2.2c represents the same distribution as the Baysian network in Figure 2.2a [Liang *et al.* , 2017].

In addition to the outlined properties above, [Kisa *et al.* , 2014, Liang *et al.* , 2017] also prove that PSDDs are canonical and complete (given a vtree). Meaning that every distribution is induced by a unique, (compressed) PSDD and that every probability distribution can be represented by a PSDD.

2.4.2 Inference

Reasoning and inference are crucial aspects of PSDDs as it has been designed with such applications in mind. As such, it makes full use of the syntactic and semantic properties of the graph structure. Therefore, the probability of any (partial) assignment $\mathbf{b}$ can be computed in time linear in the PSDD size [Kisa *et al.* , 2014] or in other words in one pass over the tree. The proposed algorithms for computing the probability of some evidence e, and the probability of contexts given the tree, are both based on the Theorems 8 and 7. For more details on the individual proofs of said theorems we kindly refer the reader to the original paper [Kisa *et al.* , 2014].

Theorem 7: *[Kisa et al. , 2014] Consider a decision node $n = (p_1, s_1, \theta_1), ..., (p_k, s_k, \theta_k)$ that is normalized for vtree node v, with left child l and right child r. For evidence* **e***, we have*

$$Pr_r(e_v) = \sum_{i=1}^{k} Pr_{p_i}(e_l) * Pr_{s_i}(e_r) * \theta_i$$

Theorem 8.: *[Kisa et al. , 2014] Consider a PSDD r, variable X, and its leaf vtree node v. Let $n_1, ..., n_k$ be all the terminal nodes normalized for v and let $\gamma_{n1}, ..., \gamma_{nk}$ be their corresponding contexts. For evidence e, we have*

$$Pr_r(X, e_v) = \sum_{i=1}^{k} Pr_{ni}(X) * Pr_r(\gamma_{n_i}, e_v)$$

2.4.3 Learning

The proposed algorithm [Liang *et al.* , 2017] for learning the structure of PSDDs works by iteratively updating and improving the structure of the PSDD to better fit the data. It does so by applying specified clone and split operations to PSDD r at each step. Learning is then carried out until a time limit is reached or the defined *score* converges on the validation data (if present, otherwise training data). This 'score' is based on the log-likelikhood of the model given the data, but takes the size of the tree into account as well;

$$score = \frac{ln\mathcal{L}(r'|\mathcal{D}) - ln\mathcal{L}(r|\mathcal{D})}{size(r') - size(r)}$$

where r is the original and r' is the updated PSDD [Liang *et al.* , 2017]. The log-likelihood of PSDD r given data $\mathcal{D}$ is then a sum of log-likelihood contributions per node:

$$ln\mathcal{L}(r|\mathcal{D}) = lnPr_r(\mathcal{D}) = \sum_{q \in r} \sum_{i \in q} ln\theta_{q,i}\mathcal{D}\#(\gamma_q, [p_{q,i}])$$

where $\#(\gamma_q, [p_{q,i}])$ is the number of examples that satisfy the node context of q and the base of $q's$ prime $p_{q,i}$.

Additionally, [Liang *et al.* , 2017] also proposed an algorithm for learning ensembles of PSDDs (EM-LearnPSDD) which is built on the learnPSDD algorithm as well as the soft structural EM algorithm by [Friedman, 1998]. This algorithm consists of two nested learners, where the outer EM is learning the structure and the inner EM is learning the parameters. This is also the algorithm we predominantly use in our experiments.

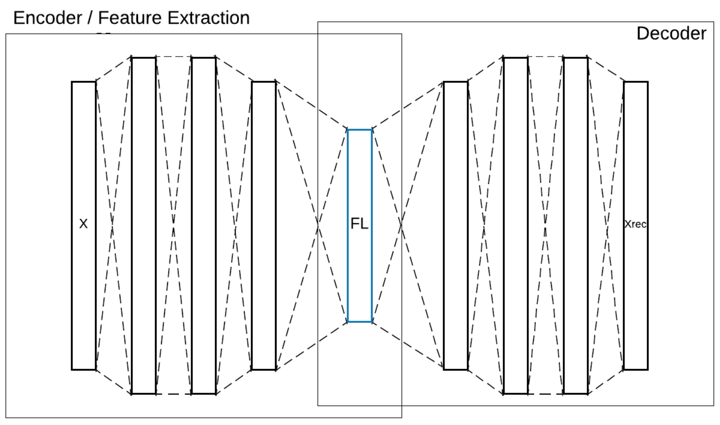


Figure 2.3: Autoencoder structure

2.5 AutoEncoder

An autoencoder (AE) is a specific version of an artificial NN which is intended to reproduce the given input as the output again (that is a copy of the input) [Goodfellow *et al.* , 2016a]. As such, it consists of two parts: the encoder e and decoder d. Internally it has a hidden layer termed the feature layer (FL) such that FL =e(x) for some input data x and x_{rec} =d(FL) where x_{rec} is the reconstructed input and FL $\in \mathbb{R}^d$. A graphical depiction is given in Figure 2.3. The ideal AE would represent the identity mapping d(e(x)) = x and is as such not particularly useful. Thus, restrictions are usually imposed on the structure of the network such as reduced dimentionality in the FL or added noise on the input. By reducing the dimension of the FL relative to the dimension of the input x, the network is intended to learn the 'most valuable' features in the input space, where 'most valuable' should be viewed as 'most valuable for reconstructing the original image'. Such an AE network structure is termed **undercomplete AE** and can be constructed in the simplest case by linear mappings as encoder and decoder. We can see that in such a scenario (undercomplete AE with linear mappings) the learned encoder will span the same subspace as the principal component analysis (PCA) [Goodfellow *et al.* , 2016b] such that the AE will have learned the principal component. However if we relax this property and use non-linear activation functions and multiple layers, the AE can learn a much more powerful generalization of PCA.

The **learning** procedure then constitutes learning the encoder and decoder function simultaneously by minimising the reconstruction loss specified as:

$$\mathcal{L}_{rec}(d(e(x)), x) \tag{2.2}$$

where $\mathcal{L}_{rec}$ is a loss function penalising $d(e(x))$ for being dissimilar to x. An example for such a function is given by the mean-squared-error (mse):

$$\mathcal{L}_{mse}(d(e(x)), x) = \frac{1}{|x|} * \sum_{i=1}^{|x|} (d(e(x_i))) - x_i)^2 \tag{2.3}$$

where $x \in \mathbb{R}^d$, $d \in \mathbb{N}$ for example.

In general, AEs can be viewed as a special case of classical feed-forward neural networks, and as such, modern machine learning algorithms and methods can be used to learn the parameters of the network. Especially the mini-batch gradient decent optimization algorithm updating the weight by following the gradients through back-propagation. Furthermore regularization terms as well as sparsity enforcing loss terms can be added to the loss function. One important difference to classical neural networks is that this learning regime classifies as *unsupervised learning*, since we do not have to partition our data into disjoint input-output pairs.

2.5.1 Variational Autoencoder

The Variational autoencoder, or VAE [Kingma & Welling, 2013, Rezende *et al.* , 2014] is a version of AE that builds on the stochastic generalisation of the classical AE architecture, where instead of a deterministic function, the encoder and decoder are stochastic mappings $p_{encoder}(\mathrm{FL}|x)$ and $p_{decoder}(x|\mathrm{FL})$. Thus we can also view the encoder and decoder as conditional probability distributions. Utilising this probabilistic interpretation the VAE framework defines a distribution $p(\mathrm{FL})$ such that FL samples can be drawn from said distribution. These samples can then be run through the differentiable generator network (or decoder) d. Finally, the resulting output x_{rec} is then sampled from the distribution $p(x_{rec}|d(\mathrm{FL})) = p_{decoder}(x_{rec}|\mathrm{FL})$. Now since we want to be able to sample from the distribution $p(\mathrm{FL})$, a trick is used that defines this distribution to be a well-known one, such as a Gaussian distribution. Furthermore the approximate inference network (or encoder) $q(\mathrm{FL}|x)$ serves as an approximation network to the true intractable posterior distribution $p(\mathrm{FL}|x) = \frac{p(x|\mathrm{FL}) * p(\mathrm{FL})}{p(x)}$.

Formally we want the approximation $q(\mathrm{FL}|x)$ to be as similar as possible to the true distribution $p(\mathrm{FL} \mid x)$. Thus using the KullbackLeibler divergence (D_{KL}) we can specify the optimization problem as follows:

$$min D_{KL}(q(\mathrm{FL} \mid x) || p(\mathrm{FL} \mid x)) \tag{2.4}$$

which is equivalent by [Goodfellow *et al.* , 2016a] to optimizing:

$$logp(x) - D_{KL}(q(\mathrm{FL} \mid x) \| p(\mathrm{FL})) \tag{2.5}$$
$$= \mathbb{E}_{\mathrm{FL} \sim q(\mathrm{FL}|x)} logp(x \mid \mathrm{FL}) - D_{KL}(q(\mathrm{FL} \mid x) \| p(\mathrm{FL})) \tag{2.6}$$
$$\approx \mathbb{E}_{x \sim \mathcal{D}}[\mathbb{E}_{\mathrm{FL} \sim q(\mathrm{FL}|x)}[logp(x \mid \mathrm{FL})] - D_{KL}(q(\mathrm{FL} \mid x) \| p(\mathrm{FL}))] \tag{2.7}$$

where $\mathcal{D}$ is the data we are working on.
In equation 2.6 we recognise the first term as the reconstruction loss ensuring that the output of the decoder is as close as possible to the input of the encoder. The second term is then the KullbackLeibler divergence enforcing the decoder network to be as similar as possible to our chosen distribution $q(\mathrm{FL})$.

Thus we can define the loss of the VAE network as:

$$\begin{aligned}\mathcal{L}_{VAE} &= \mathbb{E}_{\mathrm{FL} \sim q(\mathrm{FL}|x)} logp(x \mid \mathrm{FL}) - D_{KL}(q(\mathrm{FL} \mid x) \| p(\mathrm{FL})) \\ &= \mathcal{L}_{rec}(d(e(x)), x) - D_{KL}(q(\mathrm{FL} \mid x) \| p(\mathrm{FL}))\end{aligned} \tag{2.8}$$

An example for the reconstruction loss ($\mathcal{L}_{rec}$) in this framework is defined as the cross-entropy between the distribution over the input and output space rather than a measure of distance such as the mse (Equation 2.3). When it comes to the actual implementation, one possible example is the Binary-Cross-Entropy loss function assuming that input and output values are such that $x_i \in [0, 1]$:

$$\mathcal{L}_{rec-BCE}(d(e(x)), x) = \frac{1}{|x|} \sum_{i=1}^{|x|} -[d(e(x_i))) \cdot \log x_1 + (1 - d(e(x_i)))) \cdot \log(1 - x_i)] \tag{2.9}$$

Now when it comes to **learning** the parameters of the encoder and decoder network we want to be able to compute the gradient of Equation 2.7. Doing so we can move the gradient into the expectation, such that we sample a single value x before sampling a value FL from $q(\mathrm{FL} \mid x)$. We can then simply use those samples to compute the gradient for:

$$logp(x \mid \mathrm{FL}) - D_{KL}(q(\mathrm{FL} \mid x) \| p(\mathrm{FL})) \tag{2.10}$$

Averaging the gradient of Equation 2.10 for arbitrary many samples, the result will converge to Equation 2.7, see [Doersch, 2016]. This however raises a serious problem as it does not give us the gradient with respect to the encoder network ($q(\mathrm{FL}, x)$). In other words, we are able to compute the forward pass through the whole network

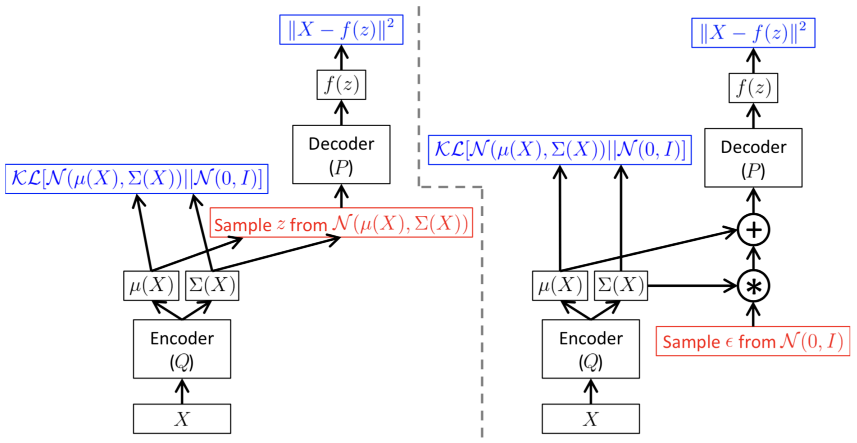

$\|X - f(z)\|^2$
$f(z)$
Decoder (P)
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
Sample z from $\mathcal{N}(\mu(X), \Sigma(X))$
$\mu(X)$
$\Sigma(X)$
Encoder (Q)
X
$\|X - f(z)\|^2$
$f(z)$
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
Decoder (P)
$+$
$*$
$\mu(X)$
$\Sigma(X)$
Encoder (Q)
Sample ϵ from $\mathcal{N}(0, I)$
X

Figure 2.4: A training-time variational autoencoder implemented as a feed- forward neural network, where $P(X \mid z)$ is Gaussian. Left is without the reparameterization trick, and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behaviour of these networks is identical, but backpropagation can be applied only to the right network. Taken from [Doersch, 2016]

without a problem but are unable to back-propagate the gradient through the sampling layers, which are non-continuous and have no gradient. More formally we observe that stochastic gradient descent can handle stochastic input layers, in contrast to stochastic units within the network. This is visually depicted in Figure 2.4. In order to circumvent this problem the so called *reparameterization trick* [Kingma & Welling, 2013] is used here to move the sampling to an input layer. Thus instead of sampling from $q(\text{FL}, x)$ (defined to be Gaussian) by drawing $\text{FL} \sim \mathcal{N}(\mu(x), \Sigma(x))$, we first sample $\epsilon \sim \mathcal{N}(0, 1)$ and then compute $\text{FL} = \mu(x) + \Sigma^{1/2}(x) * \epsilon$, where $\mu(x), \Sigma(x)$ represent the mean and covariance of $q(\text{FL}, x)$ respectively. Graphically this is depicted in right side of Figure 2.4. Applying this *trick* to equation 2.7, the optimization goal is now defined as:

$$\mathbb{E}_{x \sim \mathcal{D}}[\mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}[log p(x \mid \text{FL} = \mu(x) + \Sigma^{1/2}(x) * \epsilon)] - D_{KL}(q(\text{FL} \mid x) || p(\text{FL}))] \qquad (2.11)$$

2.5.2 Categorical Variational Autoencoder

The VAE setup as we have discussed it so far (see Section 2.5.1) constitutes a very powerful framework for unsupervised learning and feature learning. However it necessitates continuous variables in all layers (including the FL) and assumes that the distribution over the FL $p(\text{FL} \mid x)$ is Gaussian. In practice however, we find that discrete random variables arise naturally in many domains (e.g. mathematical logic) and applications (e.g. classification). As such they have been used to learn probabilistic latent representations corresponding to semantic classes [Kingma & Welling, 2013], image regions [Xu *et al.* , 2015] or memory locations [Graves *et al.* , 2014]. Moreover discrete random variables are computationally more efficient [Rae *et al.* , 2016] and maybe most importantly (to us); much more interpretable than their continuous counterparts in many cases [Chen *et al.* , 2016]. To mitigate this problem of non-differentiability, the Gumbel-Softmax distribution was proposed by [Jang *et al.* , 2016].

2.5.2.1 The Gumbel-Softmax Distribution

The Gumbel-Softmax distribution was defined as a continuous distribution over the simplex that can approximate samples from a categorical distribution [Jang *et al.* , 2016]. Let us first define a categorical distribution for the discrete variables C such that the class probabilities for the k different values of C are given by $\pi_1, \pi_2 ..., \pi_k \in [0,1]$. Samples c from a categorical distribution are then represented as k-dimensional 'one-hot' encoded vectors, such that $c \in \Delta^{k-1}$. Firstly the Gumbel-Max trick [Gumbel, 1954, Maddison *et al.* , 2014] enables us to sample a variable c from a categorical distribution $p(C)$ with class probabilities π as follows:

$$z = one\_hot(arg_i max[g_i + log \pi_i]) \tag{2.12}$$

where $g_1, g_2, ..., g_k$ are i.i.d. samples drawn from $Gumbel(0,1)$ by sampling $u \sim Uniform(0,1)$ and computing $g_i = -log(-log(u))$. [Jang *et al.* , 2016] then propose the softmax function as a continuous approximation of the argmax function such that the individual values $\hat{c}_i$ of a sample $\hat{\mathbf{c}}$ ($\{\hat{c}_1, \hat{c}_2, ..., \hat{c}_k\} = \hat{\mathbf{c}} \in \hat{\Delta}_C$) are defined as:

$$\hat{c}_i = \frac{exp((log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} exp((log(\pi_j) + g_j)/\tau)} \text{ for } i = 1, ..., k \tag{2.13}$$

Now we observe that as the softmax temperature τ approaches 0, samples from the Gumbel-Softmax become identical to the categorical distribution $p(C)$. Furthermore we see that for $\tau > 0$ the distribution is smooth and thus has a well-defined gradient with respect to the parameters π allowing us to back-propagate through the network. In practice we see that there is a trade-off between small temperature, where samples are close to on-hot but the variance of the gradients is large, and large temperatures, where samples are smooth but the variance of the gradients is small ([Jang *et al.* , 2016]). Thus annealing can be applied to τ such that we start off at a higher value and decrease it steadily with respect to the training epochs.

When it comes to categorical variational autoencoders, that is learning a categorical FL , we simply use the Gumbel-Softmax as our (defined) distribution p(FL), where $\pi_1, \pi_2, ..., \pi_k$ are the output of the encoder network e.

Chapter 3

The Model

Within this section we will propose a novel model for representing and learning the joint probability distribution over unstructured data $\mathcal{D}$. Said model is termed Neuro-Symbolic Domain Generative Model (NSDGM), and is a deep network consisting of a connectionist and symbolic network. The bridge between these two sub-networks is made by an intermediate feature layer (FL) consisting of n discrete variables. As the name suggests NSDGM s are domain generative, meaning that we can perform conditional sampling for a given domain with respect to the other domains. Domains, denoted by $D_A, D_B, D_C...$, represent disjoint subsets of the data we are interested in. An example is given by an image dataset where the images are denoted as domain D_A (e.g. $D_A \subset \mathbb{R}^{28x28}_{[0,1,...,255]}$) and the corresponding labels as domain D_Y (e.g. $D_Y \subset \mathbb{N}$) such that $D_A \sqcup D_Y = \mathcal{D}$. The model is then able to approximate the distributions $p(D_A, D_Y)$ sufficiently and can sample from $p(D_A \mid D_Y)$ and $p(D_Y \mid D_A)$. After explaining the architecture in more detail we will go on to discuss the learning of the system as well as querying and sampling.

3.1 Architecture

The model consists of three main parts; the lowlevel connectionist sub-model, the highlevel symbolic sub-model and the intermediate feature layer FL . As such we will discuss them each individually starting with the FL .

3.1.1 Feature layer

The FL is the intermediate discrete representation of the data we are learning on. This layer is intended to build the gap between the connectionist model (NN) and the generative model (tractable circuit), and is mathematically defined as:

Definition 9.: The feature layer FL is a discrete finite set of variables such that $\mathrm{FL} = \{C_0, C_1, ..., C_l\}$ for some $l \in \mathbb{N}$. Conceptually FL represents an encoded discretized version of the original data $\mathcal{D}$. If the data is split into different domains (disjoint sets) $\{D_A, D_B, ..\} = \mathcal{D}$, then the FL can be split into disjoint sets as well; $FL = FL^A \oplus FL^B \oplus ...$ The size of FL is then defined to be the number of variables $\mid \mathrm{FL} \mid = l$ and $\mid FL \mid = \mid FL^A \mid + \mid FL^B \mid + ...$

In the case where FL is binary encoded, it can be written as $FL \subset \{0,1\}^l$. For the discrete case, each variable in the FL is a discrete variable that can take one of the k values. In practice however, samples are encoded as k-dimensional one-hot vectors lying on the corners of the $(k-1)$-dimensional simplex, Δ^{k-1}. Thus we have $one\_hot(FL^X) = FL'^X \subset \{0,1\}^{l \times k}$, where $\sum_{j=1}^{k} \mathrm{FL}'^X_{i,j} = 1$ holds for all $i \in \{1, 2, .., l\}$.

3.1.2 Encoders and Decoders - Connectionist Model

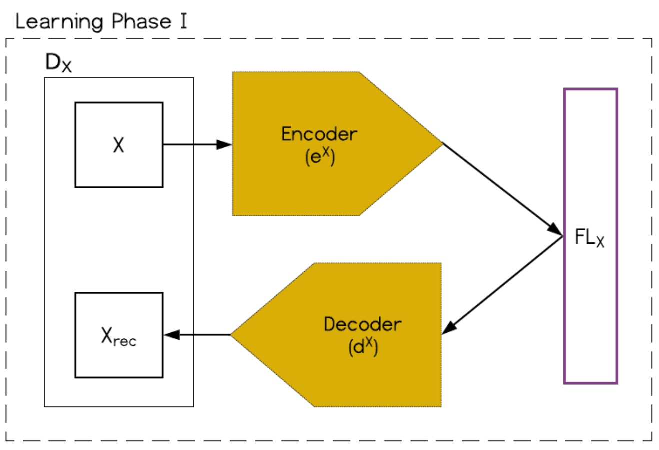


Figure 3.1: Domain specific learning - phase one

Now for a given domain $D_X \in \{D_A, D_B, D_C, ..\}$, we have **independent** encoder e^X and decoder d^X such that $FL^X = e^X(D_X)$ and $d^X(FL^X) = d^X(e^X(D_X)) \subset D_X$. This constitutes the usual autoencoder setup, where we map from the input domain (e.g. D_X) to an intermediate representation (e.g. FL^X) using the encoder and then mapping back to the original domain using the decoder (see Section 2.5). This formulation also works for stochastic encoder/decoder networks with $p_{encoder}(\mathrm{FL}^X \mid D_X)$ and $p_{decoder}(D_X \mid \mathrm{FL}^X)$.

Encoders and decoders are then functions of varying complexity depending on the domain. While some encoder-decoder pairs will have to be learned from data, others can be defined deterministically. Considering again the MNIST dataset as an example; here we define domain D_A to be the images (e.g. $a \in D_A \subset \mathbb{R}^{28 \text{ x } 28}_{\{0,1,..,255\}}$) and domain D_Y to represent the labels (e.g. $y \in D_Y = \{0, 1, .., 9\}$) corresponding to the images. The encoder and decoder for domain D_A (e^A, d^A) are then defined as a deep convolutional neural network (e.g. similar to ResNet [He *et al.* , 2016], AlexNet [Krizhevsky *et al.* , 2012]). The encoder/decoder for the domain D_Y on the other hand can be defined as the onehot encoding ($e^Y(y) \mapsto one\_hot(y)$, $d^Y(fl^Y_i) \mapsto arg_j max(fl^Y_{ij})$.

The learning of these functions (if applicable) is done in an unsupervised manner using VAEs in our setup and is referred to as *learning phase 1* throughout this report. This will be discussed in greater detail in Section 3.2.

Defining the encoder-decoder pairs in such a general way, that is as general functions and being domain specific, brings two advantages to the formulation. One the one hand, this allows us to define **independent** encoder-decoder mapping for each domain of the data, as showcased with the MNIST example above. Furthermore, this also makes it possible to add an additional domain, for example the audio feed of a person speaking the given number (e.g. 'one'). We could then define a domain specific NN architecture for feature extraction, train it independently and then add it to the overall model. On the other hand this formulation is **invariant** to the actual dimensionality reduction the encoder performs on a given domain. Consider again the MNIST example, where domain D_A are the images. The dimensionality reduction that we perform with the encoder can vary between no reduction (e.g. simply flattening the image [Liang & Van den Broeck, 2019]) and the *maximal* reduction (e.g. to one categorical variables representing the label of the image). The complexity of the symbolic generative model is then in an inverse relation to the complexity of the connectionist model (i.e. the more complex the connectionist model (higher dimensionality reduction) the less complex the symbolic model).

3.1.3 Generative - Symbolic Model

The generative model represents the dependencies between the individual variables of the FL, or in other words represents the joint probability distribution over the variables of the FL; $Pr(FL)$. In this project we chose to use Probabilistic Sentential Decision Diagrams (PSSD) [Kisa *et al.* , 2014] though other models such as sum product networks [Poon & Domingos, 2011] could have been used as well. Due to various desirable properties this graphical representation (PSDDs) allows querying, as well as computing partial derivatives, in polynomial time (see Section 2.4). Additionally, due to the symbolic nature of this representation we are able to add prior knowledge (hard constraints) to the graph before we start learning. An example of such constraints is given by specifying the onehot encoded categorical variables of the FL.
Furthermore, a given PSDD can be translated directly to a Markov network, giving a graphical representation of the dependencies between the individual features (variables in the feature layer). Learning the PSDD is then considered as learning phase 2.

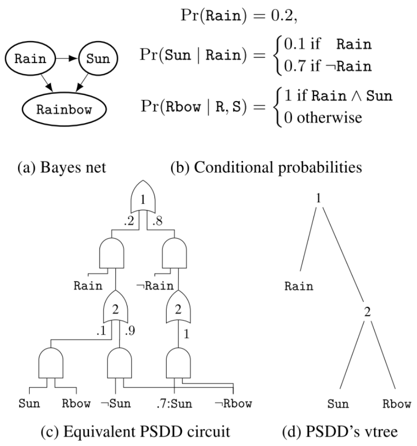

Rain
Sun
Rainbow
(a) Bayes net
Pr(Rain) = 0.2,
Pr(Sun | Rain) = 0.1 if Rain; 0.7 if ¬Rain
Pr(Rbow | R, S) = 1 if Rain ∧ Sun; 0 otherwise
(b) Conditional probabilities
1
.2
.8
Rain
¬Rain
2
.1
.9
2
1
Sun
Rbow
¬Sun
.7:Sun
¬Rbow
(c) Equivalent PSDD circuit
1
Rain
2
Sun
Rbow
(d) PSDD's vtree

Figure 3.2: Example of a PSDD

3.2 Learning

The learning of the system is done in two phases. Learning phase 1 constitutes the learning of the encoder-decoder function pairs for each domain (independently and unsupervised). Once learning phase 1 is completed we can use the encoders to map the data to the FL and learn the generative model where the predicates are the individual variables of the FL .

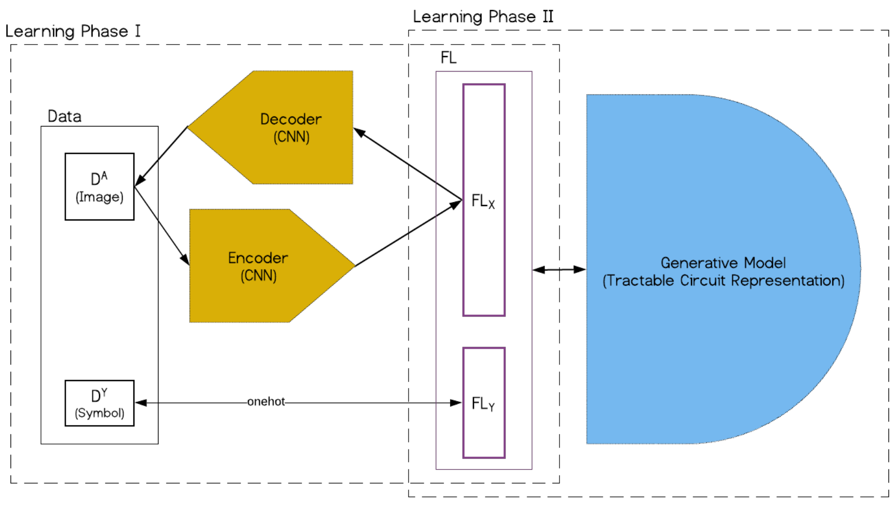


Figure 3.3: The two phases of the proposed neural-symbolic learning system for example1:MNIST

3.2.1 Encoders and Decoders - Connectionist Model

Learning the encoder and decoder is done using the formulation of AE learning. Here we have so far explored two versions, namely the classical one (AE) as well as the variational one (VAE).

The first experiments were conducted using a straight forward AE setup only considering a binary FL. Furthermore we define the final activation layer of the encoder network to either be a *sigmoid* or a *tanh* function during training. When actually using, evaluating or testing the network, an additional heaviside step function is applied, such that values in the FL are binary ($\{0, 1\}$ or $\{-1, 1\}$). This of course is a very approximate solution to the problem of learning the distribution over the data and should only be considered as an initial example and baseline for future work. However running such experiments gave us a better understanding of the architecture and the problem we are dealing with. As such experiments showed that using a *tanh* function in the final layer (such that FL $\subset \{-1, 1\}$) produced much better results in comparison to using a sigmoid.

For most experiments however we used the Gumbel-Softmax formulation described in [Jang *et al.* , 2016] in a variation AE setup to learn discrete variables in the FL . As described in more detail in Section 2.5.1, the Gubmel-Softmax formulation is well defined, capable of learning discrete variables (instead of only binary ones) and as our experiments confirmed much more accurate when it comes to learning.

3.2.2 Generative - Symbolic Model

To learn the probability distribution over the FL we are using the learnPSDD structure learning algorithm by [Liang *et al.* , 2017]. Starting off with an initial PSDD structure (usually just $True$), at each iteration the algorithm changes the structure of the graphical model by executing an operation (that is ether spit or clone). Learning is performed until the log-likelihood of the validation data converges or a given time or size limit is reached. The operation to execute is greedily chosen based on the best likelihood improvement per size increment:

$$score = \frac{ln\mathcal{L}(r'|D) - ln\mathcal{L}(r|D)}{size(r')size(r)}$$

where r is the original and r' the updated PSDD.

The algorithm can also be provided with hard constraints, which are converted into a PSDD and used as a starting point for learning. For more details on the structure and learning of PSDDs please refer to Section 2.4.

3.3 Querying

Due to the generative property of PSDDs, we are able to perform any query of the form;

$$Pr(q|e) = \frac{Pr(q \wedge e)}{Pr(e)}$$

where q is the query and e is the evidence, both boolean functions over the variables of the FL . Furthermore by Theorem 7 of [Kisa *et al.* , 2014] one such probability can be computed in one pass through the tree and thus in polynomial time w.r.t. the size of the graph.

Since the FL is not a human readable (or necessarily understandable) representation, queries and evidence can not easily be formulated in this space directly. However using the encoder and decoder functions we are able to map between the FL and the actual data domains such that we can formulate interesting queries. Coming back to our MNIST example, such queries would then be formulated as: $Pr(e^X(q) \mid e^Y(e))$ where X and Y correspond to two domains of the data ($D_X, D_Y \subset \mathcal{D}$ and $q \in D_X, e \in D_Y$).

3.3.1 Generative Query

A generative query in this case constitutes a query that samples values for all variables in the FL not assigned in the query. This is possible due the discrete limitation of the FL and is outlined in Algorithm 2. As such the algorithm then returns a sample from the joint distribution over the FL conditional on the evidence given. That is:

$$fl = generativeQuery(\Delta, fl_{evidence})$$
$$fl \sim Pr(FL \mid fl_{evidence}))$$

Algorithm 2 generativeQuery Algorithm

```
1:  procedure GENERATIVEQUERY(psdd $\Delta$, query $q$, categorical dimension $k$)
2:      $assigned \leftarrow$ variables_appearing_in($q$)
3:      $not\_assigned \leftarrow$ variables_appearing_in($\Delta$) $- assigned$
4:      $generated \leftarrow dict()$
5:      while not empty(not_assigned) do
6:          $var \leftarrow pop\_random(not\_assigned)$
7:          $probs \leftarrow zeros(dim = k)$
8:          for $j \in range(k)$ do
9:              $probs_j \leftarrow \frac{Pr_\Delta(var=onehot(j)|q)}{Pr_\Delta(q)}$
10:         end for
11:         $inst \leftarrow sample(probs)$
12:
13:         $q \leftarrow (q \wedge (var = inst))$
14:         $generated[var] \leftarrow inst$
15:         $assigned \leftarrow assigned + var$
16:     end while
17:     return $generated$
18: end procedure
```

As mentioned before the resulting assignments of variables returned from the algorithm can then be decoded using the decoder d.

3.4 Evaluation

In order to evaluate our model we focus on two main aspects, namely classification accuracy on image data sets as well as the interpretability of the model. Classification accuracy be used as a quantifiable score that is easily comparable to similar learning systems. As such we train the model on multiple image data sets before asking the model to classify unseen images ($a \in D_A$) into one of the possible categories ($y \in D_Y$) using the following formulation:

$$\begin{aligned} y' &= d^Y(gernerativeQuery(\Delta, e^A(a))) \\ e^Y(y') &\sim Pr(FL^Y \mid e^A(a)) \\ y' &= argmax(fl^y \sim Pr(FL^Y \mid e^A(a))) \\ y' &= argmax(fl^y \sim Pr(FL^y \mid [fl^a \sim p_{encoder}(FL^A \mid a)])) \end{aligned}$$

Furthermore we investigate the interpretability of the model by visually inspecting samples drawn from the distribution for some evidence. Considering again the MNIST case, we would sample images for each category or class and check if the images correspond to the class. Such samples will be computed as follows (again $a \in D_A$ are the images, and $y \in D_Y$ are the class labels):

$$\begin{aligned} a' &= d^A(generativeQuery(\Delta, e^Y(y)) \\ e^A(a') &\sim Pr(FL^A \mid e^Y(y)) \\ a' &\sim p_{decoder}(D_A \mid [fl^a \sim Pr(FL^A \mid e^Y(y))]) \end{aligned}$$

Finally we analyse the individual variables of the binary FL . This should shed some light on the inner workings of the model, and give us an insight into what the individual variables represent. By approximating the expectation of decoded FL samples, where samples are drawn conditional on a specific variables being true or false:

$$\begin{aligned} diff_{\mathrm{FL}_i} &= \mathbb{E}_{fl \sim p(\mathrm{FL}|fl_i)} d^A(fl) - \mathbb{E}_{fl \sim p(\mathrm{FL}|\neg fl_i)} d^A(fl) \\ &\approx \frac{1}{N} * \sum_{fl \sim p(\mathrm{FL}|fl_i)}^{N} d^A(fl) * p(fl) - \frac{1}{N} * \sum_{fl \sim p(\mathrm{FL}|\neg fl_i)} d^A(fl) * p(fl) \end{aligned}$$

Here we define the decoded image to be w pixels in width and h pixels in height. Then each greyscale image a is normalized to be an element of $a \in [0,1]^{w*h}$. What follows is that $diff_{\mathrm{FL}_i} \in [-1,1]^{w*h}$ and as such has to be normalized accordingly in order to produce an image again.

$$visual_{fl_i} = \left[\frac{diff_{\mathrm{FL}_i}+1}{2}, \frac{-diff_{\mathrm{FL}_i}+1}{2}\right]$$

Here $visual_{fl_i}$ is then depicted as a tuple of images corresponding to the variables i being true vs. false and vice versa.

Chapter 4

Experiments

Within this section we discuss the experiment conducted in order to evaluate and test the model proposed in Chapter 3. The experiments are then evaluated based along two main axes. Firstly we are interested in the accuracy we achieve with the fully trained model on a held out test set. Accuracy within this setting refers to the classification accuracy on images, that is the number of correctly labelled images normalised by the total number of images. This 'score' is used as a basic measure in the machine learning community and will serve as means of comparing the model to other networks and systems for learning. Secondly, the understand-ability and generative properties of the model are evaluated by visual inspection of generated images. Taking the MNIST image set as an example again, we generate k samples or images for a given label (e.g. 3) and then assess if these images correspond to our understanding of the given number. Furthermore, we compute likelihood of a given generated image conditioned on the evidence. This shows to be a useful indicator of how certain the model is of a generated image.

The following sections will cover the experiments we ran in a roughly chronological order. This should help demonstrate and justify the parameters we chose and used throughout the experiment phase. Thus firstly we will talk about the connectionist model (the deep-convolutional-variation-autoencoder), before moving on the symbolic learning (learnPSDD). Finally we will discuss the experiments conducted on the whole neuro-symbolic model.

4.1 Data

In order to get a good understanding of the capabilities of the proposed model we used three different datasets throughout the experiments. The MNIST data set [LeCun *et al.* , 1998] containing 28x28 (grayscale) images which represent handwritten digits, along with the corresponding class label. Here we have 10 classes in total ranging from 0 to 9. This is a well established data set in the machine learning and artificial intelligence community and thus served as a starting point for our experiments as well. After the first round of experiments on the MNIST data set we used the hyper-parameters for the best performing models and re-ran the experiments on the FASHION data set [Xiao *et al.* , 2017]. The FASHION data set was proposed as an easily implementable, replacement for the MNSIT data set. Again this set contains 10,000 28x28 (grayscale) images and the corresponding labels belong to one for the 10 categories. The images here however represent pieces of clothing for categories such as ”Ankle boot”, ”Sneaker” or ”Dress”. Finally we wanted to investigate the scalability of the model, and used the EMNIST (extended-MNIST) [Cohen *et al.* , 2017] data set in order to ran some final experiments. Here the images correspond to the 10 different (handwritten) digits as well as all the letters of the English alphabet, upper and lower case. Since some letters are very similar in their lower case and capitalised versions (e.g. l and L or o and O), we used the merged version of the data set as suggested by the authors [Cohen *et al.* , 2017]. Thus the total number of different classes within this set is 47 with some labels being quite similar in their writing (e.g. ”3” and ”B” or ”I” and ”L”).

4.2 Hardware

All experiments conducted where run on University of Edinburgh’s computing clusters. Since most experiments involved two training phases using very different optimization methods, we made use of two different cluster architectures so to improve performance. Learning phase 1, concerned with learning the parameters of a deep NN using mini-batch gradient descent and back propagation were run on the University of Edinburgh’s CDT cluster ’Charles’. The cluster nodes here are a mix of Dell PowerEdge R730 and Dell PowerEdge T630. Each has two 16 core Xeon CPUs. Where the GPUs are a mix of NVIDIA cards Tesla K40m, GeForce GTX Titan X and GeForce Titan X. Learning phase 2 on the other hand, applying a version of the learnPSDD structure learning algorithm were run on the University of Edinburgh's CDT cluster 'James'. This cluster consists of 21 nodes, each being a Dell PowerEdge R815 with four 16 core Opteron CPUs and 256GB of memory.

4.3 Unsupervised Encoder-Decoder Learning

The first set of experiments were concerned with evaluating different methods and architectures for learning encoder-decoder mappings. These mappings correspond to the functions e^X and d^X from Section 3.1, where e^X maps an image to a categorical latent representation (FL^X), and d^X maps the latent representation back to an image. In practice both functions are learned simultaneously by means of unsupervised deep-convolutional-variational-autoencoders. Here the key aspect or constraint is the categorical FL, since neural networks are generally continuous and fully differential. Therefore we considered and evaluated different strategies to circumvent this problem, which are explained in more detail in Section 3.1.

These experiments where all run on the University of Edinburgh CDT cluser using the MNIST digit data-set, where each model was evaluated by the reconstruction loss on a held out test data set.

4.3.1 Vanilla Autoencoder

Our first task was to find an appropriate architecture and tune the basic hyper-parameters. This was done by varying architectural properties of the network such as dropout-layers, number-of-conv-layers, number-of-fcc-layers, num-of-channels as well as the size of the FL itself.

The full architecture we ended up choosing for the experiments is depicted in more detail in Appendix A, but in a nutshell the encoder consists of three convolutional layers, each followed by a activation and dim-reduction layer (max-pool). This is then followed by two fully connected layers where the final activation function before the FL is a tanh during training, and tanh + heaviside during validation and testing. The decoder is then a symmetric (mirrored at the FL) version of the encoder.

Optimising the network with 128 binary variables in the FL for the reconstruction loss, that is the mean-squared-error (mse) on the reconstructed image

$$loss_{rec} = mse = \frac{1}{|D_X|} * \sum_{x \in D_X} (d^X(e^X(x))^2 - x)$$

w.r.t. the computed epochs is then depicted in Figure 4.1.

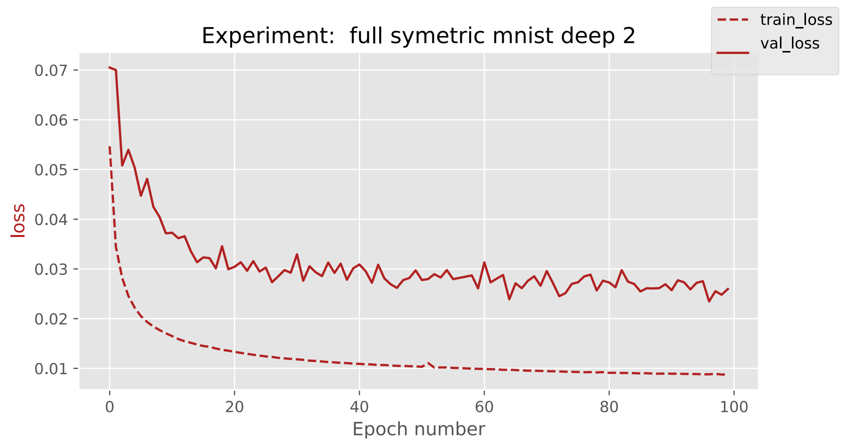
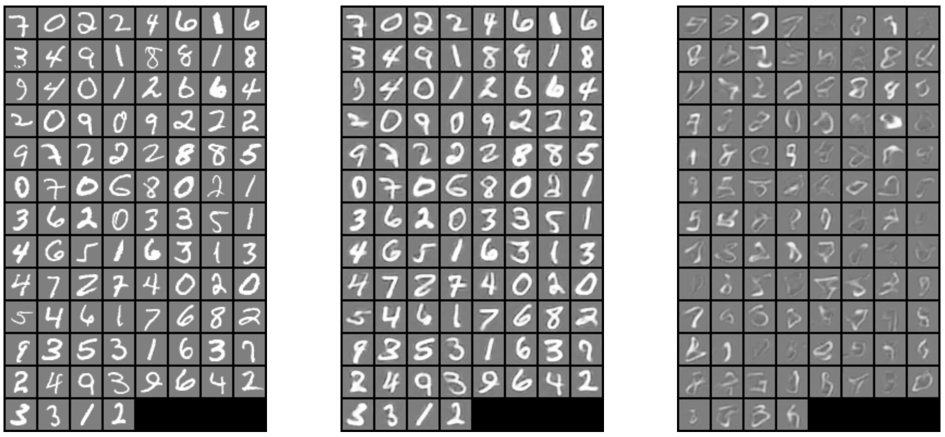


Figure 4.1: The training and validation (reconstruction) loss w.r.t. the epoch for the Vanilla-AE network on the MNIST dataset $|FL^A| = 128$, $|FL_i^A| = 2$

For a random test batch, the original images as well as the reconstructed ones are depicted in Figure 4.2 in addition to the decoding of a random sample of the FL where every element in the FL is drawn independently from a Bernoulli distribution with $p = 0.5$.



(a) In: $a \in A$

(b) Out: $d^A(e^A(a))$

(c) Decoded Random FL Sample

Figure 4.2: (Binary) Encoder-Decoder Visualisation

4.3.2 Variational Autoencoder with Gumbel-Softmax

Next we used a variational autoencoder architecture in addition to the Gumbel-Softmax method in order to learn a discrete FL. The full architecture used here is fairly similar to the architecture used in previous experiments with the additional of the Gumbel-Softmax method as well KullbackLeibler divergence loss term (see 2.5.1). Since we are using a variational setting we use the binary cross entropy function to calculate the reconstruction loss on the network:

$$loss_{rec} = bce = \frac{1}{|D_X|} \sum_{x \in D_X} -\left[d^X(e^X(x))) \cdot \log x + (1 - d^X(e^X(x)))) \cdot \log(1-x)\right]$$

However we also recorded the mean squared reconstruction error in order to compare results obtained with previous experiments.

The experiments we ran using this kind of architecture varied in the FL size (number of categorical variables) the categorical dimension (of each variable in the FL) as well as the weighting of the Kullback-Leibler loss term w.r.t. the reconstruction loss. All experiments were then trained for 200 epochs and evaluated on a held out test set. Investigating the results depicted in Table 4.1 and Figure 4.3 we notice firstly the general trend that a more complex feature space results in a smaller loss as expected. Furthermore we see that

$$error(fl_x c_4) < error(fl_{(x*2)} c_2)$$

holds for all experiments while the size of the model space (of the FL) is the same in both cases.

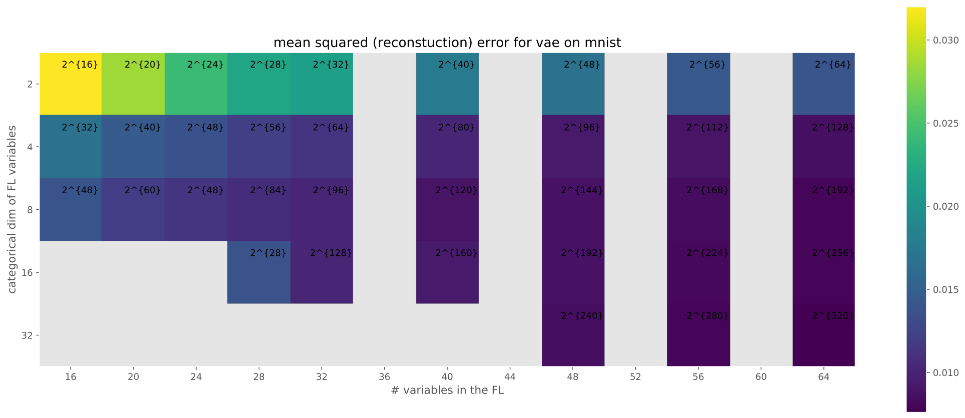


Figure 4.3: The reconstruction loss of the VAE

dataset	$\lvert FL^A \rvert$	$\lvert FL_i^A \rvert$	MSE	model space complexity
mnist	16	8	0.0139	2^{48}
mnist	16	2	0.032	2^{16}
mnist	16	4	0.0168	2^{32}
mnist	20	2	0.0285	2^{20}
mnist	20	8	0.012	2^{60}
mnist	20	4	0.0146	2^{40}
mnist	24	4	0.0137	2^{48}
mnist	24	8	0.0113	2^{72}
mnist	24	2	0.0242	2^{24}
mnist	28	8	0.0108	2^{84}
mnist	28	4	0.0122	2^{56}
mnist	28	2	0.0219	2^{28}
mnist	32	16	0.0102	2^{128}
mnist	32	8	0.0102	2^{96}
mnist	32	2	0.0214	2^{32}
mnist	32	4	0.0113	2^{64}
mnist	40	4	0.0102	2^{80}
mnist	40	2	0.0176	2^{40}
mnist	40	16	0.0094	2^{160}
mnist	40	8	0.0089	2^{120}
mnist	48	8	0.0088	2^{144}
mnist	48	4	0.0094	2^{96}
mnist	48	2	0.0167	2^{48}
mnist	48	32	0.0085	2^{240}
mnist	48	16	0.0088	2^{192}
mnist	56	16	0.0081	2^{224}
mnist	56	2	0.0144	2^{56}
mnist	56	32	0.0079	2^{280}
mnist	56	8	0.0084	2^{168}
mnist	56	4	0.0089	2^{112}
mnist	64	4	0.0086	2^{128}
mnist	64	2	0.014	2^{64}
mnist	64	8	0.0079	2^{192}
mnist	64	16	0.0079	2^{256}
mnist	64	32	0.0076	2^{320}

Table 4.1: Experiment Results MNIST using VAE

Here we can see that the VAE outperforms the previous experiment easily. This superiority in the learning behaviour is also demonstrated by Figure 4.4 demonstrating the mse train and validation error w.r.t to the epoch.

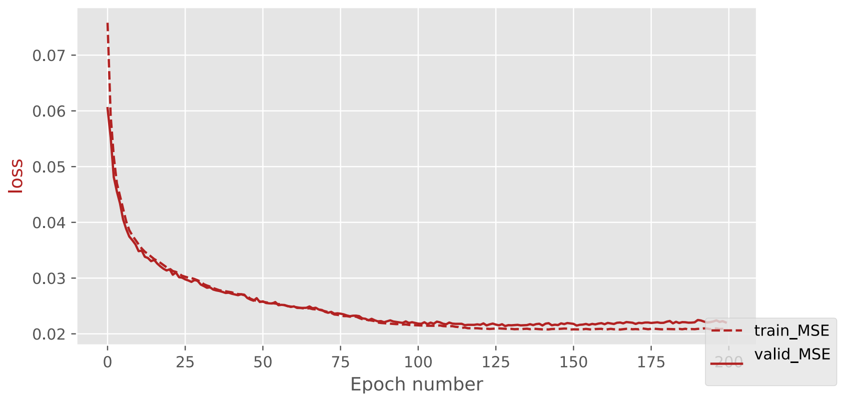


Figure 4.4: The training and validation (reconstruction) mse loss w.r.t. the epoch for the VAE network on the MNIST dataset $|FL^A| = 32$, $|FL_i^A| = 2$

Figure 4.5 then demonstrates the bce and kld loss terms w.r.t. the epoch during the training procedure. The overall loss term used for updating the weighs is computed by a weighted sum of these two terms, where $w_{kld} = .7$ and $w_{BCE} = 1$ proved to be the values resulting in the best overall results.

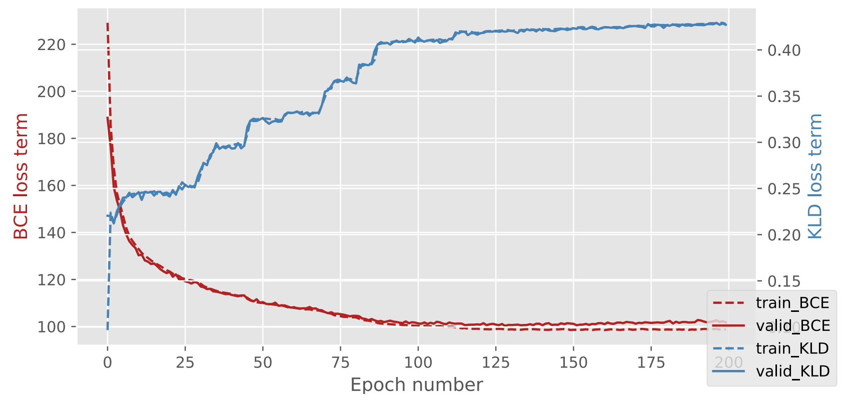
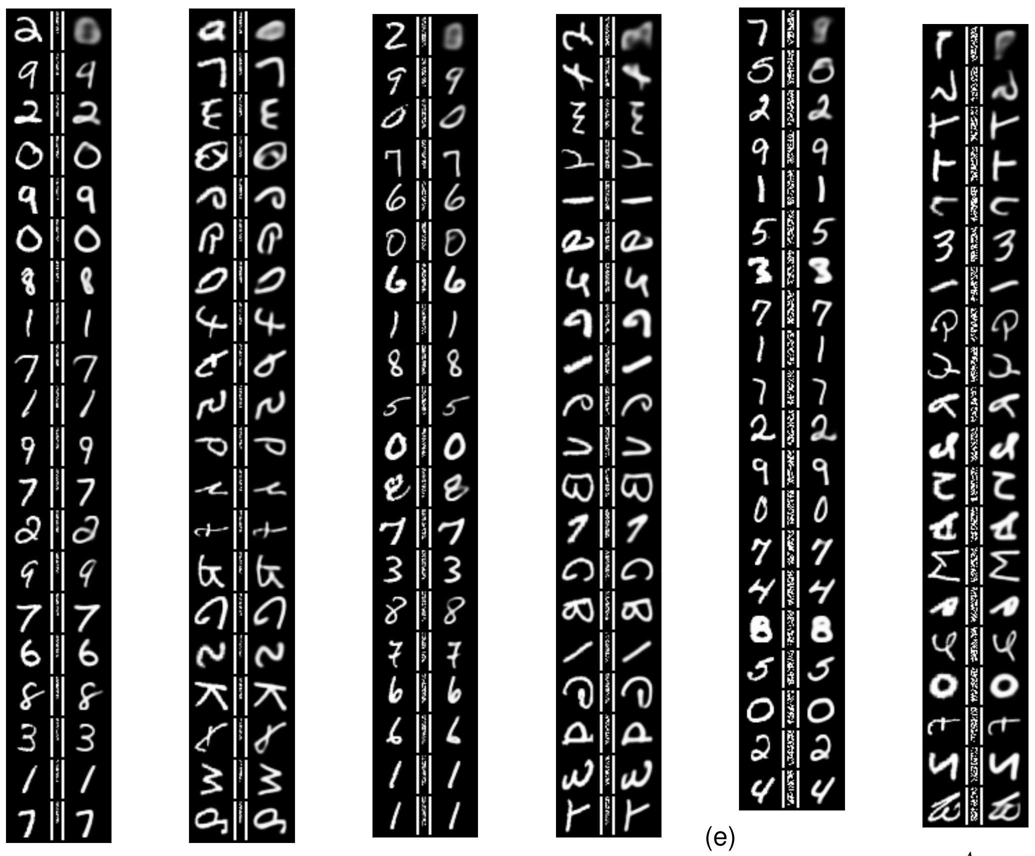


Figure 4.5: The training and validation (reconstruction) bce and kld loss w.r.t. the epoch for the variational-AE network on the MNIST dataset $|FL^A| = 32$, $|FL_i^A| = 2$

Finally, as comparison to the previous experiments, the learning development of the autoencoder network w.r.t. to the training epochs is depicted in Figure 4.6.



(a) $|FL^A| = 32$, $|FL^A_i| = 8$ MNIST

(b) $|FL^A| = 32$, $|FL^A_i| = 8$ EMNIST

(c) $|FL^A| = 64$, $|FL^A_i| = 4$ MNIST

(d) $|FL^A| = 64$, $|FL^A_i| = 4$ EMNIST

(e) $|FL^A| = 128$, $|FL^A_i| = 2$ MNIST

(f) $|FL^A| = 128$, $|FL^A_i| = 2$ EMNIST

Figure 4.6: Learning development over epochs (y-axis) with FL in the middle

4.4 Symbolic psdd learning and evaluation

In order to learn the PSDD on the converted data, we first converted the whole image dataset into the FL representation before starting the learning algorithm (learnPSDD [1]) on the transferred data. The algorithm works by updating the psdd r at given points using specified clone and split operations. Then the 'score' is evaluated to check if the updated PSDD r' fits the data and size requirements better:

$$score = \frac{ln\mathcal{L}(r'|\mathcal{D}) - ln\mathcal{L}(r|\mathcal{D})}{size(r') - size(r)}$$

where r is the original and r' is the updated PSDD [Liang *et al.* , 2017]. The log-likelihood of PSDD r given data $\mathcal{D}$ is then a sum of log-likelihood contributions per

[1] https://github.com/UCLA-StarAI/LearnPSDD

node:

$$ln\mathcal{L}(r|\mathcal{D}) = lnPr_r(\mathcal{D}) = \sum_{q \in r} \sum_{i \in q} ln\theta_{q,i}\mathcal{D}\#(\gamma_q, [p_{q,i}])$$

where $\#(\gamma_q, [p_{q,i}])$ is the number of examples that satisfy the node context of q and the base of $q's$ prime $p_{q,i}$.

In our experiments we used the ensemble-learner, which learns an ensemble of PSDDs that represents the whole distribution (or single PSDD) when combined. Thus our experiments here varied the number of ensembles as well as the *lambdaweight* used to weight the soft-expectation-maximisation formulation. The learning of the PSDD over time is depicted in Figure 4.7.

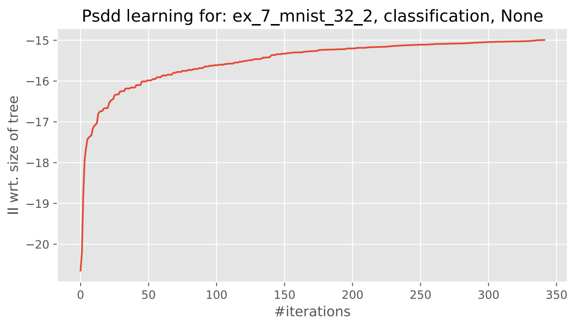


Figure 4.7: The neg log likelihood score (weighed by the size of the tree) for the learnPSDD algorithm over time

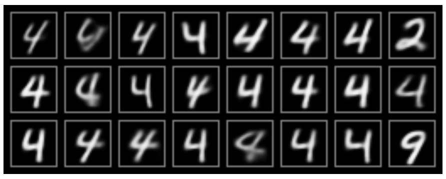
4.5 Complete (Neuro-Symbolic) Model

Within this section we evaluate the whole learning system based on its predictive accuracy on unseen data as well as the generative power of the model. Taking full advantage of the generative and unsupervised properties of the model we explored different families of experiments. Firstly we investigated the standard classification task where one data entry consists of one image and one symbolic value (or label) identifying the class of the image (or the information represented on the image). Secondly we ran experiments where some noise is added to the label of each entry (image), that is each training entry has k additional labels specified (in addition to the correct one). And finally we explored tasks which consist of at least two images. That is, experiments where one data-entry is composes of two images representing numbers in succession and no label for examples. Another experiment for example, composed of two images and the sum of the two numbers represented on the image. All such experiments have been termed 'relational' and are again evaluated by their accuracy and generative power.

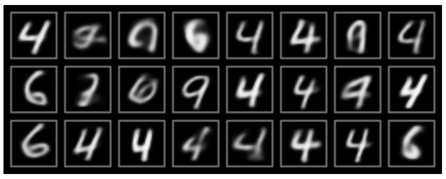
Experiments within this section have been trained and evaluated on three different data-sets, that is the MNIST, EMNIST and FASHION data-set.

4.5.1 Generative Queries

One of the main advantages of the proposed model is the ability to compute so called *generative queries* or *conditional samples*. As explained in more detail in Section 3.3.1, we are able to sample a specified (image) domain conditional on other domains. In practice this means that we can ask the system to produce examples of what it believes the class '4' represents, considering the MNSIT example. To do so we devised an algorithm which iteratively draws the values of the domain in question (e.g. FL^A) Algorithm 2. Figure 4.8a then demonstrates this algorithm for the model trained on the MNIST data-set. Here the left image was produced using the algorithm where each drawn FL_i^A value is added to the assigned variables map and then used to draw then next value FL_{i+1}^A (conditionally dependent). Figure 4.8b in comparison showcases a similar algorithm where drawn values of the FL_A are *not* added to the assigned variables such that each value of FL^A is drawn only conditioned on FL^Y (conditionally independent). Comparing the two images in Figure 4.8 we observe that the algorithm taking the conditional dependency of the latent variables in the FL into account produces much better results. This can be contributed to the fact that all variables are dependent on all other variables, meaning we can not separate them into conditionally independent sets. It should also be noted at this point that there is not difference in cost between the two versions discussed here. Therefore all further sampled images were produced using the fully conditional version of the algorithm.



(a) Sampled images (with border) using fully conditionally dependent algorithm



(b) Sampled images (with border) using conditionally independent algorithm

Figure 4.8: Comparison between fully conditional dependent and conditionally independent generative query algorithms

4.5.2 Classification Task

As previously mentioned, the training experiments within this section consisted of an image and the correct label for that image. We then trained the encoder-decoder pair unsupervised in the first instance and the PSDD on the whole FL (FL^A + FL^Y representing the image and label respectively) in the second instance. The hyper-parameters explored here include the vtree-search algorithm used as well as the option of compressing the label to a binary encoding instead of an one-hot encoding (e.g. $[0,1,0,0]- > [1,1]$). Furthermore we varied the number of variables in FL^X corresponding to the image ($X \in \{A, B\}$) (denoted by $\mid \mathrm{FL}^A \mid$) and the categorical dimension of such variables (denoted by $\mid \mathrm{FL}_i^A \mid$).

4.5.2.1 MNIST

The best classification accuracy on the MNIST dataset was measured at: 89.55% using 32 binary (with categorical dimension 2) variables and a one-hot encoded FL^Y. A more comprehensive overview is given in Figure 4.9 and Table 4.2. In the Figure all images are recorded with the accuracy they achieved w.r.t. the size of the model space the corresponding FL spans. Mathematically the size of the model space is defined as: $\mid \mathrm{FL}_i^X \mid^{|\mathrm{FL}^X|}$. Here we can easily see that there is a clear trade off between expressiveness of the FL and the ability for the PSDD learning to interpret this FL. Meaning that - if the FL is too small, then the connectionist model will not be able to learn a meaningful mapping retaining valuable information in the encoding, and if the FL is too large (or complex), the PSDD learner will not be able to find correlations between the variables. Furthermore, we also see that the PSDD learner records superior performance on experiments with binary FLs although the reconstruction error of the autoencoder is better in the discrete case. This is due to the way in which the learnPSDD algorithm works, by handling each variable independently w.r.t. the vtree it represents. Finally the hyper-parameters indicating the vtree-method as well as the compression of FL^Y do not indicate a clear 'better choice' as experiments with all else being equal achieve very similar performances.

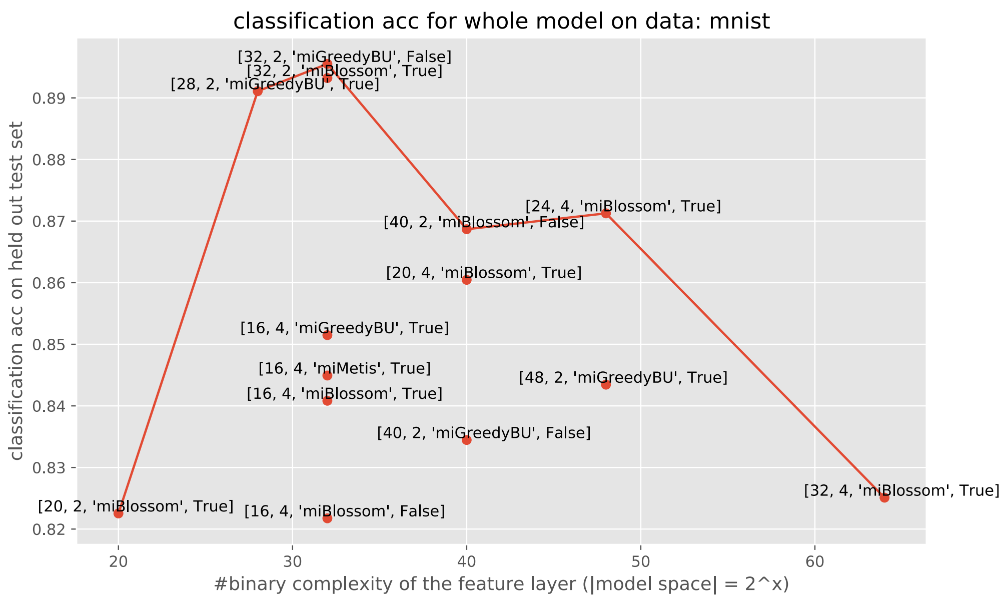
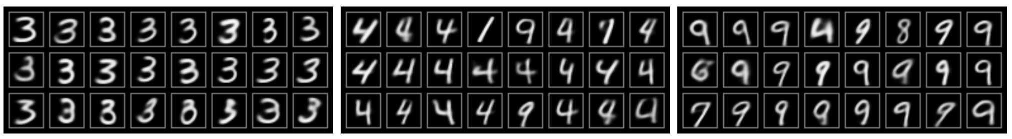


Figure 4.9: Experiment Classification Results MNIST (label format: [| FL^A |,| FL$_i^A$ |, vtree seach algorithm, binary encoded FL^Y])

Using the model from the experiment with the best classification accuracy we sampled images for each of the 10 categories using the proposed conditional-sampling algorithm. These samples are depicted in Figure 4.10 for 3 of the 10 classes. Since these are samples from the joint distribution conditioned on the label or FL^Y, it is normal to see some variation as we do in the depicted Figures. However, overall the system demonstrates some understanding of what a given label represents.



(a) Sampled images for class: 3 (b) Sampled images for class: 4 (c) Sampled images for class: 9

Figure 4.10: Image Generation using the Generative Query (Conditional) for exp$_{mnist}$| FL^A |= 32, | FL_i^A |= 2

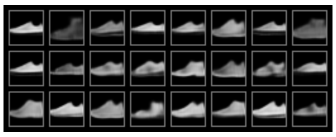
dataset	$\lvert FL^A \rvert$	$\lvert FL_i^A \rvert$	VAE-MSE	best ll	vtree method	compressed y	class acc
mnist	16	4	0.0168	-2e+01	miBlossom	False	0.8217
mnist	16	4	0.0168	-1.9e+01	miBlossom	True	0.8408
mnist	16	4	0.0168	-2e+01	miGreedyBU	True	0.8515
mnist	16	4	0.0168	-1.9e+01	miMetis	True	0.8449
mnist	20	2	0.0285	-9.2	miBlossom	True	0.8225
mnist	20	4	0.0146	-2.4e+01	miBlossom	True	0.8605
mnist	24	4	0.0137	-2.9e+01	miBlossom	True	0.8713
mnist	28	2	0.0219	-1.4e+01	miGreedyBU	True	0.8911
mnist	32	2	0.0214	-1.5e+01	miBlossom	True	0.8932
mnist	32	2	0.0214	-1.6e+01	miGreedyBU	False	0.8955
mnist	32	4	0.0113	-3.9e+01	miBlossom	True	0.8251
mnist	40	2	0.0176	-2e+01	miBlossom	False	0.8687
mnist	40	2	0.0176	-2.1e+01	miGreedyBU	False	0.8345
mnist	48	2	0.0167	-2.5e+01	miGreedyBU	True	0.8434

Table 4.2: Experiment Classification Results MNIST

4.5.2.2 FASHION

For the FASHION dataset on the other hand we achieved a classification accuracy of 75% using 32 binary variables. Interestingly enough, the mse of the connectionist model is smaller (better) in this scenario than in the MNSIT case, and the PSDD score is larger (better) as well. The predictive accuracy on the held out test set is still considerably lower. This can have many causes, most notably of which is probably the more complex images. While the images in the MNIST set are mostly lines bent in a particular way, the FASHION dataset represents more complex objects. This particularly influences the computed reconstruction loss of the autoencoder, as it computes an average over pixel difference between the original image and the reconstructed one.
Again we sampled images for each of the 10 classed depicting them in Figure 4.11 below.

(a) Sampled images for class: Dress


(b) Sampled images for class: Sneaker

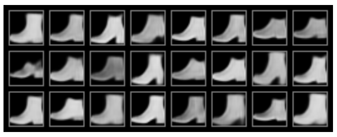

(c) Sampled images for class: Ankle boot

Figure 4.11: Image Generation using the Generative Query (Conditional) for $\exp_{fashion}| FL^A |= 32, | FL_i^A |= 2$

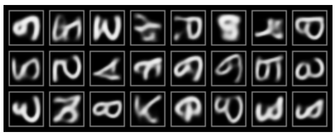
dataset	$\|FL^A\|$	$\|FL_i^A\|$	VAE-MSE	best ll	vtree method	compressed y	class acc
fashion	24	2	0.0198	-1.1e+01	miBlossom	False	0.7052
fashion	24	2	0.0198	-1.1e+01	miBlossom	True	0.6961
fashion	32	2	0.0177	-1.4e+01	miBlossom	False	0.7534
fashion	32	2	0.0177	-1.4e+01	miBlossom	True	0.7193
fashion	40	2	0.0156	-2e+01	miBlossom	True	0.6641

Table 4.3: Experiment Classification Results FASHION

4.5.2.3 EMNIST

Finally running experiments on the EMNIST dataset we found that the system is not quite capable of scaling to such complex problems (47 different classes of objects in this example). Here we recorded an overall best accuracy of 29% (where $1/47 = 0.021$ would be the expected random accuracy). More details on the experiments conducted are recorded in Table 4.4. Examples for samples images are given below again in Figure 4.12, although we see here that a clear category can not be identified looking only at the samples.

(a) Sampled images for class: 0


(b) Sampled images for class: 11

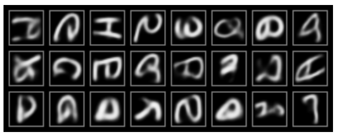

(c) Sampled images for class: 20

Figure 4.12: Image Generation using the Generative Query (Conditional) for $\exp_{EMNIST}| FL^A |= 32, | FL_i^A |= 2$

dataset	$\lvert FL^A \rvert$	$\lvert FL_i^A \rvert$	VAE-MSE	best ll	vtree method	compressed y	class acc
emnist	32	2	0.0274	-2e+01	miBlossom	True	0.2931
emnist	32	2	0.0274	-2.1e+01	miBlossom	False	0.2418
emnist	32	2	0.0274	-2.1e+01	miBlossom	False	0.2704

Table 4.4: Experiment Classification Results EMNIST

4.5.3 Noisy label Task

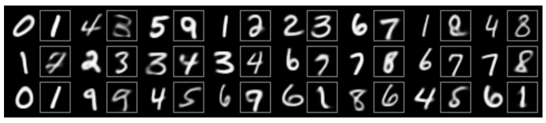
The 'noisy label' task consisted of training the high-level (psdd) model using k additional labels to the correct one. This noise is generated at random, before training. To evaluate the experiment, we computed the accuracy on a held out set only containing the right label (no noise). The results of this task are depicted in Table 4.5 which demonstrates a decreasing accuracy with increasing noise. Furthermore we notice that training on one additional label (noisy-1) decreases accuracy by only .05 considering the task. Even if 2 additional labels are added more than 80% of the images are still classified correctly. Considering that having no noisy labels the overall accuracy was recorded at 89%, this corresponds to a decrease of only 9%.

dataset	$\lvert FL^{A,B} \rvert$	$\lvert FL_i^{A,B} \rvert$	VAE-MSE	task type	best ll	vtree method	class acc
mnist	32	2	0.0214	noisy-1	-1.9e+01	miBlossom	0.8502
mnist	32	2	0.0214	noisy-2	-2.1e+01	miGreedyBU	0.8214
mnist	32	2	0.0214	noisy-3	-2.2e+01	miGreedyBU	0.718

Table 4.5: Experiment Classification Results MNIST for noisy labels

4.5.4 Relational Tasks

So called relational tasks are experiments where one training example consists of at least two images and maybe a symbolic value that stands in a specific relation to each other. The simplest of such task is where each training example consists of two images, where the digit in the second image is the successor to the digit in the first image. This particular experiment can not be evaluated by an accuracy measure, as there is no symbolic value we can generate and compare.


(a) Sampled images (with border) for fla

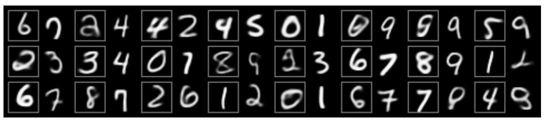

(b) Sampled images (with border) for flb

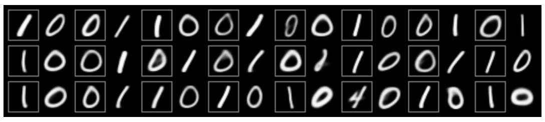
Figure 4.13: Image Generation Successor, image with border was sampled for image with no border

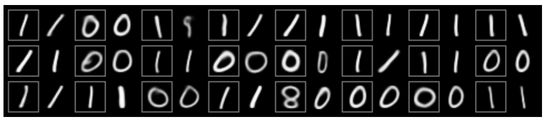
However, we can still judge the performance of generating one of two images given the other one as depicted in Figure 4.13. Here we can see that in Figure 4.13a roughly 16 out of the 24 generated images correspond to a number representing the successor of the one to the left. In Figure 4.13b we generate the predecessor for the image on the right using the same model, where in this example about 14 of the 24 images seem correct. These results indicate that our model is able to pick up on some relational structures between the individual FLs. However it is not quite clear at this point to what extend this can be generalised. Thus we proceeded with a number of experiments similar in nature to this one, but with the addition of a symbolic variable (FL^Y) expressing some relational information over the two images. One example of such a task is given by the binary-logic-and (bland) experiment. Here we first train the unsupervised autoencoder on the whole (e.g. MNIST) dataset, before creating a custom set where each entry contains two images, ether '0' or '1' and the result of applying the logical and operation on the label of these two images (FL^Y). Thus the feature layer in these task is made of three individual parts; FL^A, FL^B representing two images and FL^Y representing a function (or the return value of that function) over the labels of the two images. We can then evaluate the accuracy as usual on the correctness of the predicted symbolic value FL^Y on a held out test set. Furthermore we can sample for one of the two images given the other image and the symbolic value ($FL^Y = 0, 1$). In the pictures below samples have been drawn with the help of the model for the specified FL^Y values and one of the images (no border), while the image surrounded by a border has been generated. The other task we tried is as follows:

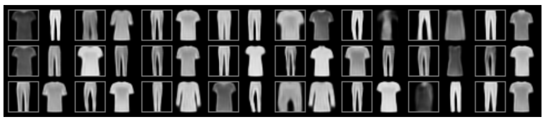
Binary Logical XOR (blxor):
$FL^A = e(imgA),\ FL^B = e(imgB),\ FL^Y = bool(label(imgA))\ XOR\ \ bool(label(imgB))$ where $label(imgA), label(imgA) \in 0, 1$.

When we ran this experiment on the FASHION dataset we set the label with index

'0' ('T-shirt/top') to be 'False' as the output to the bool function and the label with index '1' ('Trouser') to 'True'. All the other entries are disregarded similar to the same experiment on the MNIST dataset. The classification accuracy we measured on a held out test set where .994 and .882 for MNIST and FASHION respectively. Generated samples from the trained model for one given image and a specified FL^Y value can be seen in Figure 4.14.


(a) (MNIST) Sampled images (with border) for fly: 1


(b) (MNIST) Sampled images (with border) for fly: 0


(c) (FASHION) Sampled images (with border) for fly: 1

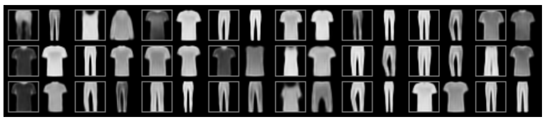
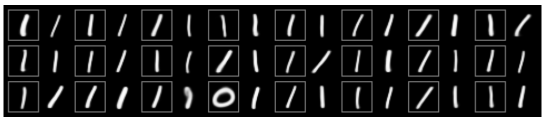

(d) (FASHION) Sampled images (with border) for fly: 0

Figure 4.14: Image Generation binary-logic-**xor** where image with border was sampled for image with no border and fly: 0 or 1

Binary Logical AND (bland):

$FL^A = e(imgA),\ FL^B = e(imgB),\ FL^Y = bool(label(imgA))\ \ AND\ \ bool(label(imgB))$ where $label(imgA), label(imgA) \in 0, 1$.


(a) Sampled images (with border) for fly: 1

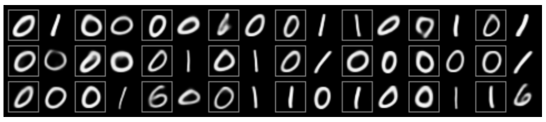
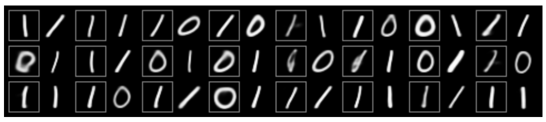

(b) Sampled images (with border) for fly: 0

Figure 4.15: Image Generation binary-logic-and where image with border was sampled for image with no border and fly: 0 or 1

Binary Logical OR (blor):

$FL^A = e(imgA),\ FL^B = e(imgB),\ FL^Y = bool(label(imgA))\ \ OR\ \ bool(label(imgB))$ where

$label(imgA), label(imgA) \in 0, 1$.


(a) Sampled images (with border) for fly: 1

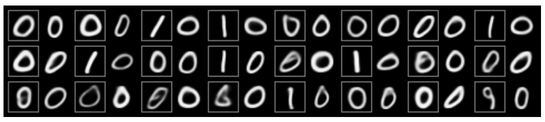
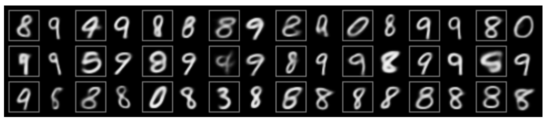

(b) Sampled images (with border) for fly: 0

Figure 4.16: Image Generation binary-logic-or where image with border was sampled for image with no border and fly: 0 or 1

Greater 7 logic And (g7land):
$FL^A = e(imgA)$, $FL^B = e(imgB)$, $FL^Y = (label(imgA) > 7)\ AND\ (label(imgB) > 7)$.


(a) Sampled images (with border) for fly: 1

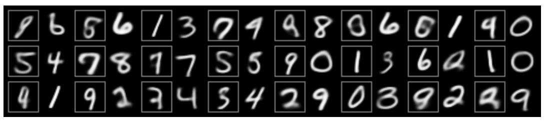
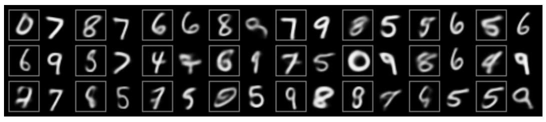

(b) Sampled images (with border) for fly: 0

Figure 4.17: Image Generation greater-7-logic-and where image with border was sampled for image with no border and fly: 0 or 1

Greater 4 logic And (g4land):
$FL^A = e(imgA)$, $FL^B = e(imgB)$, $FL^Y = (label(imgA) > 4)\ AND\ (label(imgB) > 4)$.


(a) Sampled images (with border) for fly: 1

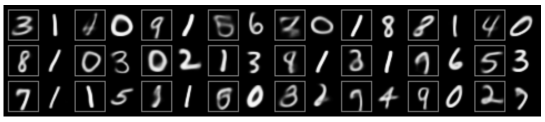

(b) Sampled images (with border) for fly: 0

Figure 4.18: Image Generation greater-4-logic-and where image with border was sampled for image with no border and fly: 0 or 1

PLUS (plus):
$FL^A = e(imgA)$, $FL^B = e(imgB)$, $FL^Y = label(imgA) + label(imgB)$.

PLUS ring 10 (plus-ring-10):
$FL^A = e(imgA)$, $FL^B = e(imgB)$, $FL^Y = (label(imgA) + label(imgB))\ mod\ 10$.

A summary of the experiments and their classification accuracy is depicted in Table 4.6 for the MNIST dataset, and Table 4.7 for the FASHION dataset. When we look at the accuracies in Table 4.6, we see that task where only two kinds of images are considered, that is 0 and 1, achieve a much better accuracy due to being a much simpler problem. The superiority in accuracy of the g7land tasks over the g4land tasks can only be attributed to the larger set of types of images representing a symbolic '1' in the texitg4land tasks. Finally we see that all plus tasks are far more intrecate than all previous ones, where we have much more possible FL^Y values and multiple combination of images can correspond to the same FL^Y value.

dataset	$\|FL^{A,B}\|$	$\|FL_i^{A,B}\|$	VAE-MSE	task type	best ll	vtree method	class acc
mnist	32	2	0.0214	g7land	-3.2e+01	miBlossom	0.8592
mnist	32	2	0.0214	g7land	-3.4e+01	miGreedyBU	0.8284
mnist	32	2	0.0214	plus	-3.5e+01	miBlossom	0.0922
mnist	32	2	0.0214	plus	-3.6e+01	miGreedyBU	0.0992
mnist	32	2	0.0214	g4land	-3.4e+01	miGreedyBU	0.6528
mnist	32	2	0.0214	g4land	-3.2e+01	miBlossom	0.707
mnist	32	2	0.0214	plus-ring-10	-3.5e+01	miBlossom	0.1404
mnist	32	2	0.0214	blxor	-2.5e+01	miBlossom	0.994
mnist	32	2	0.0214	bland	-2.5e+01	miBlossom	0.978
mnist	32	2	0.0214	blor	-2.6e+01	miBlossom	0.966

Table 4.6: Experiment Classification Results MNIST for relational tasks

dataset	$\|FL^{A,B}\|$	$\|FL_i^{A,B}\|$	VAE-MSE	task type	best ll	vtree method	class acc
fashion	32	2	0.0177	blxor	-2.5e+01	miBlossom	0.882
fashion	32	2	0.0177	plus-ring-10	-3.3e+01	miBlossom	0.2134

Table 4.7: Experiment Classification Results FASHION for relational tasks

4.5.5 Answer Confidence

Working on relational tasks and experiments we see that some of the tasks we have tried can produce so called 'impossible' queries. An example is given by the bland task, where we ask the network to generate imageA given that imageB is a 0 and FL^Y is specified to be 1. Such queries can of course be generated for each of the *and* task as well as the *plus* and *or* tasks. Therefore we devised a confidence measure the network records for each query. This measure is computed as the likelihood of the generated

FL conditional on all information given. Thus when we generate FL^A for examples it becomes:

$$\mathcal{L}_{query} = Pr(query_{gen}|evidence) = Pr(FL^A_{gen}|FL^B, FL^Y) = \frac{Pr(FL^A_{gen}, FL^B, FL^Y)}{Pr(FL^B, FL^Y)}$$

Using this score as a basis for our analysis, we again sampled images for each of the tasks and two scenarios, the possible and the impossible one. To get a meaningful and interpretable measure we averaged the likelihood of each query over the possible and impossible cases and normalized it using the softmax function. For each task the possible and impossible confidences are depicted in Table 4.8. Comparing the columns *possible* and *impossible* in Table 4.8, we observe that overall sampled images in a possible scenario are deemed more likely by the model. The exception to this was reordered analysis the *g4land* task. However we see that this task also achieved the lowest classification accuracy. Overall the results indicate that a better performing model (higher classification accuracy) correlates with a more accurate distinction between possible and impossible cases. The difference in the likelihood the model assigns to the individual queries with respect to the possible and impossible scenarios is however much too small to distinguish between them solely based on said score.

task	possible-[y=0]	possible-[y=1]	possible	impossible	classification-acc
g7land	0.3583	0.1953	0.5536	0.4464	0.8592
g4land	0.1996	0.2473	0.4469	0.5531	0.707
bland	0.1582	0.5872	0.7453	0.2547	0.978
blor	0.209	0.576	0.785	0.215	0.966

Table 4.8: Confidence Analysis of generated images

4.5.6 FL Analysis

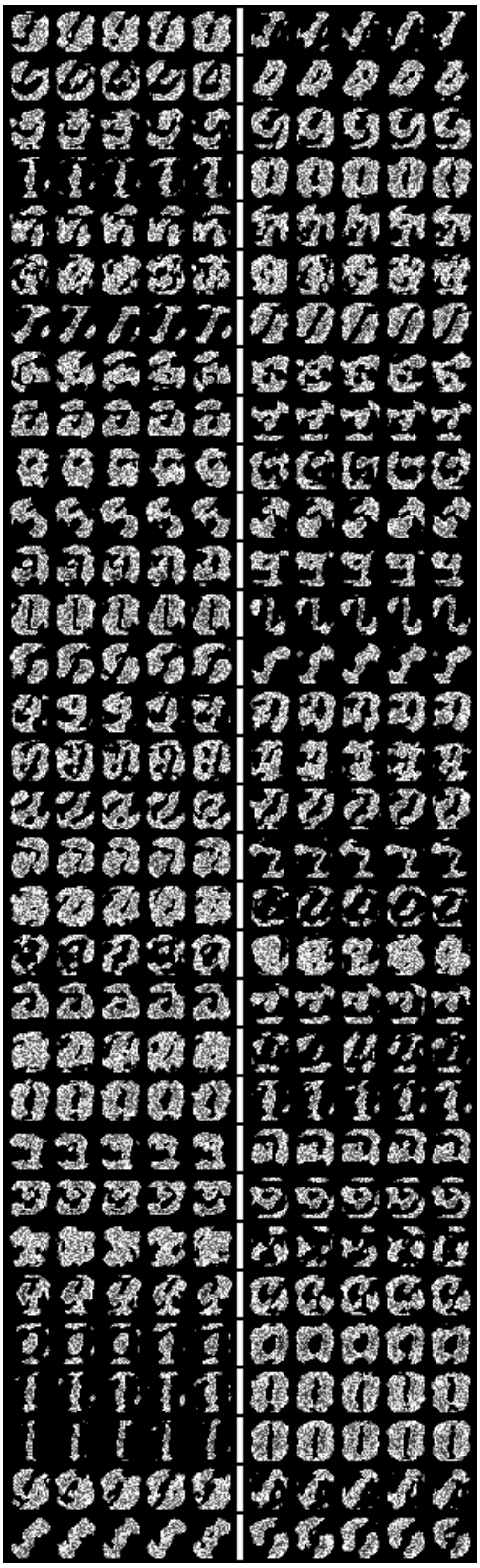
As explained in more detail in Section 3.4 we used the generative query algorithm and the generative properties of the model to analyse the FL in more detail. To be more specific, we are interested here in what a given variable in the FL actually represents, and if we can visually depict this. To do so we samples images for our best performing model on MNIST and FASHION using the following formulation:

$$
\begin{aligned}
diff_{\mathrm{FL}_i} &= \mathbb{E}_{fl\sim p(\mathrm{FL}|fl_i)} d^A(fl) - \mathbb{E}_{fl\sim p(\mathrm{FL}|\neg fl_i)} d^A(fl) \\
&\approx \frac{1}{N} * \sum_{fl\sim p(\mathrm{FL}|fl_i)}^{N} d^A(fl) * p(fl) - \frac{1}{N} * \sum_{fl\sim p(\mathrm{FL}|\neg fl_i)} d^A(fl) * p(fl)
\end{aligned}
$$

Here we define the decoded image to be w pixels in width and h pixels in height. Then each greyscale image a is normalized to be an element of $a \in [0,1]^{w*h}$. What follows is that $diff_{\mathrm{FL}_i} \in [-1,1]^{w*h}$ and as such has to be normalized accordingly in order to produce an image again.

$$
visual_{fl_i} = \left[\frac{diff_{\mathrm{FL}_i} + 1}{2}, \frac{-diff_{\mathrm{FL}_i} + 1}{2}\right]
$$

In the images depicted in Figure 4.19 we computed $visual_{\mathrm{FL}_i}$ five times with $N = 200$ for each variables of FL^A. Here each row corresponds to one variables (in order from 1 to 32). What this Figure then demonstrates is that the model asserts meaningful aspects of the images to each variable of the FL . We can see that individual variables correspond to different shapes such as slightly bend straight lines as in row/variable 1 of Figure 4.19a or circular objects as in rows/variables 4, 28, 29 and 30.


(a) $visual_{fl_i}$ for all i on MNIST

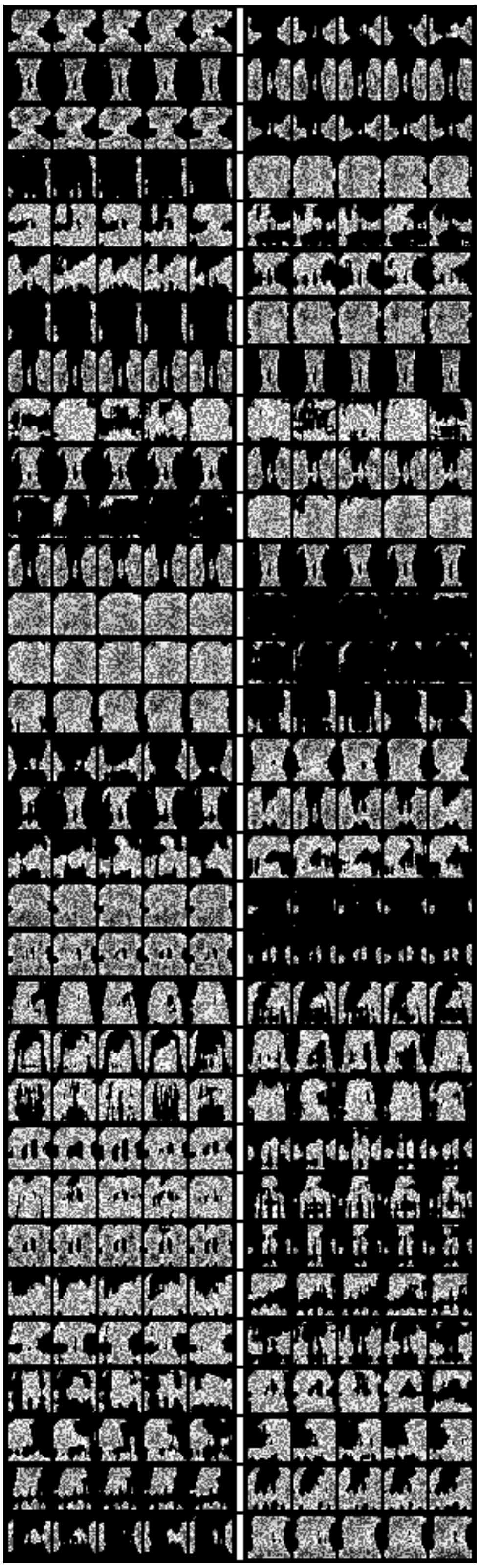

(b) $visual_{fl_i}$ for all i on FASHION

Figure 4.19: Visualisation of the binary variables in the classification models for two datasets (variables are ordered)

Chapter 5

Conclusion and Future Work

As part of this thesis we proposed a novel model for learning the joint distribution over image data. Said model incorporates a connectionist (NN) part as well as a symbolic one, where the lowlevel modelling of the data is distributed for individual domains. After training, the model is able to answer queries as well as generate conditional samples such that we are able to get an in-depth understanding of what the system learned and how it makes decisions. Guided by the methodology formulized in the first instance we conducted a number of experiments on multiple dataset in order to evaluate and test the proposed method. Here the straight forward classification task on the MNIST and FASHION dataset are used as a benchmark for comparison to other systems, while the so called *relational* tasks take full advantage of the modular framework and aiding us in answering more complex questions about the learning behaviour of the proposed model.

When it comes to the classification tasks we recorded an overall accuracy of about 90% on MNSIT and about 75% on FASHION, which can not compete with discriminative models such as convolutional NN archiving an accuracy of about 99.3% [LeCun *et al.* , 1998] on MNIST. However, classification accuracy was only one of our initial goals, with interpretability being another major one. Therefore we analysed the generative power of the model in great detail using the proposed generative query algorithm to draw conditional samples. Such samples can be drawn after training for a given evidence such as a specific label in the MNIST case. As such we are able to sample images corresponding to a specified label (e.g. ’4’), giving us an insight into what the model *believes* to be a ’4’. The results reported for this task are very promising and closely correspond to what we would consider the label to represent.

After initial experiments on the MNIST and FASHION dataset we wanted to test the scalability of the system on the much more complex EMNIST dataset. In this scenario the number of different classes represented in the images is 47 in comparison to the 10 classes found in MNIST or FASHION. Experiments run on this dataset showcases that the model is not able to scale very well producing classification accuracys of about 30%. Further experiments suggest that this is due to the much more complex image space, confronting the the low-level feature extraction formalism with a much more difficult task. Here we see that individual classes of images (e.g. '1' and 'l') are considerably more difficult to distinguish without incorporating the actual label (since we are using unsupervised methods here). Thus future work might consider different methods of feature learning that are more constrained in the theory they are constructing.

Taking full advantage of the modular framework we conducted a number of experiments we termed *relational*. Such experiments consider a dataset where each data point consists of at least two images and maybe a symbolic value. Generally speaking the datasets used here are created by putting together individual images from MNIST or FASHION in a certain relationship specified by the type of the experiment. The simplest example here is given by the *successor* experiment, where a data point constitutes two images such that the second image is the successor to the first one w.r.t. to the number the images represent (e.g. $[img(4), img(5)]$). On a conceptional level the model is then tasked with learning the relationship between the right and left image, or the distributing over both. While we cannot quantify a score for this particular task, we can visually inspect the results by generating conditional samples, where evidence is given as one of the two images. The resulting samples show a clear correlation to what we would consider the successor or predecessor of the given evidence to look like. However this raises the bigger question of what the model actually learns; *Does it learn abstract representations for each image category, and then puts these in relation to the other image or does it simply correlate models of the* FL *corresponding to visual features of the images, within a finite model space.* Trying to get to the bottom of this particular question we ran a number of experiments, each corresponding to a function f taking two images as inputs and returning a symbolic or numeric value as the output. An example is given by the binary logic XOR task specified by the function $f(imgA, imgB) = bool(label(imgA))\ XOR\ \ bool(label(imgB))$ where $label(imgA), label(imgA) \in 0, 1$. Here a given data point in the set then consists of three parts; *imgA*, *imgB* and $f(imgA, imgB)$. Since our dataset in such tasks includes a numeric value ($f(imgA, imgB)$), we can sample this value conditional on the two images for the elements of a held out test set and record a quantifiable classification accuracy. For the *binary* tasks (or functions), that is *blxor, bland, blor* the model produces very good results, all reporting accuracies of over 97% w.r.t. this score. Again we also inspected the learned model visually by sampling imgA or imgB conditional on the symbolic values and the other image. Similar to the successor task the results here showcase the models capability of learning and capturing the relationship between the three domains (imgA, imgB, and f(imgA, imgB)). While these results are indeed very promising they do not yet give us an answer to the previously posed question concerning the abstraction abilities of the model. Thus we steadily increased the complexity of the function f used to create the dataset. After multiple iterations on versions of binary functions we also investigated the case where f is defined to be the algebraic *plus* operation on the labels of the images given as inputs ($f(imgA, imgB) = label(imgA) + label(imgB)$). This task obviously being much more complex then all previous ones only achieves an accuracy of about 10%. This results in combination with the sampled images here indicate that the model is indeed learning a correlations between FL models corresponding to image parts.

When it comes to future work on the topic, an interesting aspect to explore in relation to this model, are relational graphical models as a high level modelling language. Examples of such representations are given relational SPNs [Nath & Domingos, 2015] and Markov logic networks [Richardson & Domingos, 2006]. Such relational circuits might be able to better capture the relations between individual features by lifting the underlying logic from proportional to first order logic (allowing quantification). Furthermore, another approach could also investigate function learning as we described it by the relational tasks, where the signature of the function is provided for learning. This could aid the system in learning more meaningful latent variables, as it would define the alphabet of an intermediate reasoning language. Probably the most interesting direction going forward will address a third learning phase, where the loss of the symbolic model is back propagated through the connections model. Such a interconnected learning formalisation could help the connectionist model in distinguishing the *most important* features with respect to the task at hand.

Appendices

Appendix A

Architecture details - Vanilla AE

A.1 Encoder

$$(encode-0): Conv2d(1,32, kernel-size=(3,3), stride=(1,1), padding=(1,1))$$

$$(encode-0-activation): ReLU()$$

$$(encode-0-reduction): MaxPool2d(kernel-size=2, stride=2,$$

$$padding=0, dilation=1, ceil-mode=False)$$

$$(encode-1): Conv2d(32,32, kernel-size=(3,3), stride=(1,1), padding=(1,1))$$

$$(encode-1-activation): ReLU()$$

$$(encode-1-reduction): MaxPool2d(kernel-size=2, stride=2,$$

$$padding=0, dilation=1, ceil-mode=False)$$

$$(encode-2): Conv2d(32,32, kernel-size=(4,4), stride=(1,1),$$

$$padding=(1,1))($$

$$(encode-2-activation): ReLU()($$

$$(encode-2-reduction): MaxPool2d(kernel-size=2, stride=2,($$

$$padding=0, dilation=1, ceil-mode=False)($$

$$(encode-3): Linear(in-features=288, out-features=256, bias=True)($$

$$(encode-3-activation): ReLU()($$

$$(encode-4): Linear(in-features=256, out-features=128, bias=True)($$

$$(encode-4-activation): Tanh()($$

$$(encode-5-activation): Heaviside()($$

A.2 Decoder

$$(decode-0) : Linear(in-features = 128, out-features = 256, bias = True)($$

$$(decode-0-activation) : ReLU()($$

$$(decode-1) : Linear(in-features = 256, out-features = 288, bias = True)($$

$$(decode-1-activation) : ReLU()($$

$$(decode-2) : ConvTranspose2d(32, 32, kernel-size = (3, 3), stride = (1, 1), padding = (1, 1))($$

$$(decode-2-activation) : ReLU()($$

$$(decode-2-upsampling) : UpsamplingBilinear2d(scale-factor = 2, mode = bilinear)($$

$$(decode-3) : ConvTranspose2d(32, 32, kernel-size = (4, 4), stride = (1, 1), padding = (1, 1))($$

$$(decode-3-activation) : ReLU()($$

$$(decode-3-upsampling) : UpsamplingBilinear2d(scale-factor = 2, mode = bilinear)($$

$$(decode-4) : ConvTranspose2d(32, 32, kernel-size = (3, 3), stride = (1, 1), padding = (1, 1))($$

$$(decode-4-activation) : ReLU()($$

$$(decode-4-upsampling) : UpsamplingBilinear2d(scale-factor = 2, mode = bilinear)($$

$$(decode-5) : ConvTranspose2d(32, 1, kernel-size = (3, 3), stride = (1, 1), padding = (1, 1))($$

$$(decode-5-activation) : Tanh()))$$

Bibliography

[Bach & Jordan, 2002] Bach, Francis R, & Jordan, Michael I. 2002. Thin junction trees. *Pages 569–576 of: Advances in Neural Information Processing Systems.*

[Bekker *et al.* , 2015] Bekker, Jessa, Davis, Jesse, Choi, Arthur, Darwiche, Adnan, & Van den Broeck, Guy. 2015. Tractable learning for complex probability queries. *Pages 2242–2250 of: Advances in Neural Information Processing Systems.*

[Belle *et al.* , 2015] Belle, Vaishak, Passerini, Andrea, & Van den Broeck, Guy. 2015. Probabilistic inference in hybrid domains by weighted model integration. *Pages 2770–2776 of: Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI).*

[Biere *et al.* , 2009] Biere, Armin, Heule, Marijn, & van Maaren, Hans. 2009. *Handbook of satisfiability*. Vol. 185. IOS press.

[Chavira & Darwiche, 2008] Chavira, Mark, & Darwiche, Adnan. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence*, **172**(6-7), 772–799.

[Chen *et al.* , 2016] Chen, Xi, Duan, Yan, Houthooft, Rein, Schulman, John, Sutskever, Ilya, & Abbeel, Pieter. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Pages 2172–2180 of: Advances in neural information processing systems.*

[Cohen *et al.* , 2017] Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, & van Schaik, André. 2017. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373.*

[Darwiche, 2003] Darwiche, Adnan. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM (JACM)*, **50**(3), 280–305.

[Darwiche, 2011] Darwiche, Adnan. 2011. SDD: A new canonical representation of propositional knowledge bases. *Page 819 of: IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22.

[Darwiche & Marquis, 2002] Darwiche, Adnan, & Marquis, Pierre. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research*, **17**(1), 229–264.

[De Raedt *et al.* , 2007] De Raedt, Luc, Kimmig, Angelika, & Toivonen, Hannu. 2007. ProbLog: A probabilistic Prolog and its application in link discovery.

[Doersch, 2016] Doersch, Carl. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

[Fischer *et al.* , 2018] Fischer, Marc, Balunovic, Mislav, Drachsler-Cohen, Dana, Gehr, Timon, Zhang, Ce, & Vechev, Martin. 2018. DL2: Training and Querying Neural Networks with Logic.

[Friedman, 1998] Friedman, Nir. 1998. The Bayesian structural EM algorithm. *Pages 129–138 of: Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc.

[Fuxjaeger, 2018] Fuxjaeger, Anton. 2018. On Sentential Decision Diagrams, and their use as a query language for Weighted Model Integration.

[Fuxjaeger & Belle, 2018] Fuxjaeger, Anton, & Belle, Vaishak. 2018. Scaling up Probabilistic Inference in Linear and Non-Linear Hybrid Domains by Leveraging Knowledge Compilation. *arXiv preprint arXiv:1811.12127*.

[Goodfellow *et al.* , 2016a] Goodfellow, Ian, Bengio, Yoshua, Courville, Aaron, & Bengio, Yoshua. 2016a. *Deep learning*. Vol. 1. MIT press Cambridge.

[Goodfellow *et al.* , 2016b] Goodfellow, Ian, Bengio, Yoshua, & Courville, Aaron. 2016b. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Graves *et al.* , 2014] Graves, Alex, Wayne, Greg, & Danihelka, Ivo. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[Gumbel, 1954] Gumbel, Emil Julius. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*. Vol. 33. US Government Printing Office.

[Gunning, 2017] Gunning, David. 2017. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web*, **2**.

[He *et al.* , 2016] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, & Sun, Jian. 2016. Deep residual learning for image recognition. *Pages 770–778 of: Proceedings of the IEEE conference on computer vision and pattern recognition.*

[Hinton, 2002] Hinton, Geoffrey E. 2002. Training products of experts by minimizing contrastive divergence. *Neural computation*, **14**(8), 1771–1800.

[Hinton *et al.* , 2006] Hinton, Geoffrey E, Osindero, Simon, & Teh, Yee-Whye. 2006. A fast learning algorithm for deep belief nets. *Neural computation*, **18**(7), 1527–1554.

[Hornik *et al.* , 1989] Hornik, Kurt, Stinchcombe, Maxwell, & White, Halbert. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, **2**(5), 359–366.

[Hu *et al.* , 2016] Hu, Zhiting, Ma, Xuezhe, Liu, Zhengzhong, Hovy, Eduard, & Xing, Eric. 2016. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318.*

[Jang *et al.* , 2016] Jang, Eric, Gu, Shixiang, & Poole, Ben. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144.*

[Kingma & Welling, 2013] Kingma, Diederik P, & Welling, Max. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114.*

[Kisa *et al.* , 2014] Kisa, Doga, Van den Broeck, Guy, Choi, Arthur, & Darwiche, Adnan. 2014. Probabilistic Sentential Decision Diagrams. *In: KR.*

[Koller & Friedman, 2009] Koller, Daphne, & Friedman, Nir. 2009. *Probabilistic graphical models: principles and techniques.* MIT press.

[Krizhevsky *et al.* , 2012] Krizhevsky, Alex, Sutskever, Ilya, & Hinton, Geoffrey E. 2012. Imagenet classification with deep convolutional neural networks. *Pages 1097–1105 of: Advances in neural information processing systems.*

[LeCun *et al.* , 1998] LeCun, Yann, Bottou, Léon, Bengio, Yoshua, Haffner, Patrick, *et al.* . 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

[Lee *et al.* , 2009] Lee, Honglak, Grosse, Roger, Ranganath, Rajesh, & Ng, Andrew Y. 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Pages 609–616 of: Proceedings of the 26th annual international conference on machine learning*. ACM.

[Liang & Van den Broeck, 2019] Liang, Yitao, & Van den Broeck, Guy. 2019. Learning Logistic Circuits. *A¬ A*, **1**, 3.

[Liang *et al.* , 2017] Liang, Yitao, Bekker, Jessa, & Van den Broeck, Guy. 2017. Learning the structure of probabilistic sentential decision diagrams. *In: Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*.

[Maddison *et al.* , 2014] Maddison, Chris J, Tarlow, Daniel, & Minka, Tom. 2014. A* sampling. *Pages 3086–3094 of: Advances in Neural Information Processing Systems*.

[Manhaeve *et al.* , 2018] Manhaeve, Robin, Dumančić, Sebastijan, Kimmig, Angelika, Demeester, Thomas, & De Raedt, Luc. 2018. DeepProbLog: Neural Probabilistic Logic Programming. *arXiv preprint arXiv:1805.10872*.

[Mikolov *et al.* , 2010] Mikolov, Tomáš, Karafiát, Martin, Burget, Lukáš, Černockỳ, Jan, & Khudanpur, Sanjeev. 2010. Recurrent neural network based language model. *In: Eleventh Annual Conference of the International Speech Communication Association*.

[Mohamed *et al.* , 2011] Mohamed, Abdel-rahman, Dahl, George E, & Hinton, Geoffrey. 2011. Acoustic modeling using deep belief networks. *IEEE transactions on audio, speech, and language processing*, **20**(1), 14–22.

[Murdoch *et al.* , 2019] Murdoch, W James, Singh, Chandan, Kumbier, Karl, Abbasi-Asl, Reza, & Yu, Bin. 2019. Interpretable machine learning: definitions, methods, and applications. *arXiv preprint arXiv:1901.04592*.

[Nath & Domingos, 2015] Nath, Aniruddh, & Domingos, Pedro M. 2015. Learning relational sum-product networks. *In: Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[Pipatsrisawat & Darwiche, 2010] Pipatsrisawat, Thammanit, & Darwiche, Adnan. 2010. A Lower Bound on the Size of Decomposable Negation Normal Form. *In: AAAI*.

[Poon & Domingos, 2011] Poon, Hoifung, & Domingos, Pedro. 2011. Sum-product networks: A new deep architecture. *Pages 689–690 of: Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE.

[Rae *et al.* , 2016] Rae, Jack, Hunt, Jonathan J, Danihelka, Ivo, Harley, Timothy, Senior, Andrew W, Wayne, Gregory, Graves, Alex, & Lillicrap, Timothy. 2016. Scaling memory-augmented neural networks with sparse reads and writes. *Pages 3621–3629 of: Advances in Neural Information Processing Systems*.

[Rezende *et al.* , 2014] Rezende, Danilo Jimenez, Mohamed, Shakir, & Wierstra, Daan. 2014. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

[Richardson & Domingos, 2006] Richardson, Matthew, & Domingos, Pedro. 2006. Markov logic networks. *Machine learning*, **62**(1-2), 107–136.

[Rudin, 2018] Rudin, Cynthia. 2018. Please stop explaining black box models for high stakes decisions. *arXiv preprint arXiv:1811.10154*.

[Rumelhart *et al.* , 1985] Rumelhart, David E, Hinton, Geoffrey E, & Williams, Ronald J. 1985. *Learning internal representations by error propagation*. Tech. rept. California Univ San Diego La Jolla Inst for Cognitive Science.

[Shen *et al.* , 2016] Shen, Yujia, Choi, Arthur, & Darwiche, Adnan. 2016. Tractable operations for arithmetic circuits of probabilistic models. *Pages 3936–3944 of: Advances in Neural Information Processing Systems*.

[Wexler, 2017] Wexler, Rebecca. 2017. When a computer program keeps you in jail: How computers are harming criminal justice. *New York Times*, **13**.

[Xiao *et al.* , 2017] Xiao, Han, Rasul, Kashif, & Vollgraf, Roland. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

[Xu *et al.* , 2015] Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron, Salakhudinov, Ruslan, Zemel, Rich, & Bengio, Yoshua. 2015. Show, attend and tell: Neural image caption generation with visual attention. *Pages 2048–2057 of: International conference on machine learning*.

[Yi *et al.* , 2018] Yi, Kexin, Wu, Jiajun, Gan, Chuang, Torralba, Antonio, Kohli, Pushmeet, & Tenenbaum, Josh. 2018. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. *Pages 1039–1050 of: Advances in Neural Information Processing Systems.*

[Zhang, 2004] Zhang, Nevin L. 2004. Hierarchical latent class models for cluster analysis. *Journal of Machine Learning Research*, **5**(6), 697–723.