

Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Visual Sudoku Solver

Tomáš Kadlec

**Supervisor: Ing. Vojtěch Franc, Ph.D.
May 2022**

I. Personal and study details

Student's name: **Kadlec Tomáš** Personal ID number: **483448**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Cybernetics**
Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Visual Sudoku Solver

Bachelor's thesis title in Czech:

Vizuální Sudoku solver

Guidelines:

The goal of the thesis is to design and implement a visual Sudoku solver. The input of the solver is an image capturing assignment of the Sudoku puzzle. The solver converts the image into a symbolic representation and solves the puzzle. Image-to-symbol conversion and solving the puzzle will be done by a single neural network trained from examples in end-to-end fashion. The accuracy of the trained solver will be statistically evaluated and compared to existing solutions.

Introduction:

- Design and implement a neural network architecture suitable for the solver and an algorithm for training its parameters from examples.
- Create a database of Sudoku puzzle examples that will be used for training the solver and testing its performance.
- Compare the results with the existing solutions.

Bibliography / sources:

- [1] V. Franc, A. Yermakov. Learning Maximum Margin Markov Networks from examples with missing labels. ACML 2021.
- [2] V. Franc, B. Savchynskyy. Discriminative Learning of Max-Sum Classifiers. Journal of Machine Learning Research, 2008.
- [3] Wang et al. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. ICML 2019.
- [4] Amos, Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. ICML 2017.

Name and workplace of bachelor's thesis supervisor:

Ing. Vojtěch Franc, Ph.D. Machine Learning FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **07.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Vojtěch Franc, Ph.D.
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I would like to thank to my supervisor Ing. Vojtěch Franc, Ph.D. for his help and guidance during my work on this thesis.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date

Signature

Abstract

We address the problem of learning a visual Sudoku solver from examples. We see the solver as an instance of Markov Network (MN) based structured output classifier. The recently proposed extension of the Maximum Margin Markov Network (M3N) algorithm can learn the linear Markov Network classifier with an arbitrary neighborhood structure using completely annotated and partially annotated training examples. In this thesis, we propose to integrate the MN classifier with neural networks. We show how to use the M3N algorithm to learn the parameters of the MN classifier simultaneously with a neural network to extract the features of the classifier. We show experimentally that the visual Sudoku solver learned by the proposed method outperforms all baselines, achieving a test accuracy of 97%.

Keywords: Markov networks, neural networks, learning, Sudoku

Supervisor: Ing. Vojtěch Franc, Ph.D.

Abstrakt

Problém, kterým se zabýváme, je učení vizuálního Sudoku solveru z příkladů. Na solver se díváme jako na typ klasifikátoru se strukturovaným výstupem, založeném na Markovově Síti (MS). Nedávno navržené rozšíření algoritmu Maximum Margin Markov Network (M3N) je schopné učit lineární MS klasifikátor s libovolnou sousedskou strukturou s využitím kompletně i částečně anotovaných dat. V této práci navrhujeme propojení MS klasifikátoru s neuronovými sítěmi. Ukážeme, jakým způsobem využít M3N algoritmus k souběžnému učení parametrů MS klasifikátoru a neuronové sítě, sloužící k extrakci příznaků klasifikátoru. Experimentálně ukážeme, že vizuální Sudoku solver, naučený navrženou metodou překoná všechny srovnávané metody a dosáhne 97% přesnosti.

Klíčová slova: Markovovy sítě, neuronové sítě, učení, Sudoku

Překlad názvu: Vizuální Sudoku Solver

Contents

1 Introduction	1
1.1 Contributions of the thesis	1
1.2 Structure of the thesis	2
2 State of the art	3
3 Method	5
3.1 MN classifier	5
3.2 Maximum Margin Markov Network (M3N) learning algorithm	6
3.2.1 Learning from partial annotations	7
3.3 LP relaxation of the partial margin-rescaling loss	8
3.4 Learning MN classifier on top of the Neural Network	8
3.4.1 Symbolic PyTorch implementation	9
3.4.2 Optimizing weights	9
3.4.3 Neural network architecture used as a backbone	9
4 Results	11
4.1 Datasets	11
4.1.1 HMC	11
4.1.2 Symbolic Sudoku	12
4.1.3 Visual Sudoku	13
4.2 Compared algorithms	15
4.3 Verification of the PyTorch implementation on the HMC dataset	15
4.4 Verification of the PyTorch implementation on the symbolic Sudoku dataset	17
4.5 Evaluation of the proposed method on the Visual Sudoku dataset	19
5 Conclusions	23
Bibliography	25

Figures

3.1 LeNet5 architecture [2]	10
4.1 An example Sudoku puzzle assignment from the Visual Sudoku Data Set.	14
4.2 Train and test error of MIL-M3N and NN-MIL-M3N models	16
4.3 Weights comparison of MIL-M3N and NN-MIL-M3N models	17
4.4 Performance of MN classifiers learned by the MIL-M3N-LP and NN-MIL-M3N-LP algorithms on the symbolic Sudoku dataset.	18
4.5 Weights comparison of MN classifiers learned by MIL-M3N-LP and NN-MIL-M3N-LP algorithms on the symbolic Sudoku dataset.	18
4.6 The evolution of the training 0/1-loss when learning the Visual Sudoku solver.	19
4.7 An example of a single row of the Visual Sudoku assignment.	20
4.8 Visualization of the learned score functions of the visual Sudoku solver.	20
4.9 Average 0/1-loss of the Visual Sudoku solver in test examples with respect to the number of training examples used.	21

Tables

4.1 Symbolic Sudoku input example	13
4.2 Symbolic Sudoku labels example	13
4.3 The performance of the Visual Sudoku solver learned by the proposed NN-MIL-M3N-LP algorithm and comparison to various baselines.	22

Chapter 1

Introduction

Solving the Sudoku puzzle can be seen as an instance of a structured output prediction problem, where the puzzle assignment is the classifier input and the puzzle solution is represented by the output labels to be predicted. In this thesis, we employ this idea and use the structured output classifier based on Markov networks, which allows us to efficiently and transparently model relations between the output labels. We consider a family of so called Maximum Margin Markov Network (M3N) algorithms [3, 4, 16, 14, 15, 6, 8], which have been designed to learn the Markov Network (MN) classifier from fully annotated and partially annotated examples. The recent extensions of the M3N algorithms [6, 8], using the framework of linear programming relaxation, allow us to learn MN classifiers with a generic neighborhood structure between the output labels, which is necessary to efficiently represent the rules of the Sudoku puzzle. The existing M3N algorithms, however, learn MN classifiers with a linear score only, that is, a score made up of fixed features and learnable weights. The linear model constitutes the main restriction of the existing methods because designing the features manually can be hard, which is the case, for example, in computer vision applications like the visual Sudoku considered in this thesis.

1.1 Contributions of the thesis

In this thesis, we extend the existing M3N algorithms by combining them with convolutional neural networks. In particular, we append the MN classifier as the last layer of a convolutional neural network. We employ the fact that the loss function of the M3N algorithms is differentiable, which allows us to train the parameters of the NN and the MN classifier simultaneously by the standard back-propagation.

We implement the algorithm learning the MN classifier on top of the neural network (NN) using PyTorch. We use the algorithm to learn the Visual Sudoku solver from examples of Sudoku puzzles and their solutions. As the backbone of the MN classifier, we use the LeNet5 [10] architecture. We perform an empirical evaluation that shows that the Visual Sudoku solver learned by our method outperforms all baselines and achieves 97 % accuracy in its best configuration. In addition, we show that the Solver can be learned on

partially annotated data, when some labels of training examples are missing, and still maintain a competitive performance.

■ 1.2 Structure of the thesis

- In Chapter 2, we give a brief review of existing algorithms to learn Markov Network classifiers from examples.
- In Chapter 3, we give a formal definition of the MN classifier and briefly describe an algorithm recently proposed in [8] which can learn MN classifier with a generic neighborhood structured using both completely annotated and partially annotated examples. Finally, in Section 3.4, we describe our contribution which shows how to apply the algorithm of [8] to learn the MN classifier simultaneously with neural networks extracting features for the classifier.
- In Chapter 4, we empirically verify the functionality of our method and compare the results of our approach with existing solutions.



Chapter 2

State of the art

The main tool, we will use throughout this thesis is the Markov network (MN) classifier, which is a structured output classifier, that allows us to represent dependencies between specified label pairs, so called neighboring labels. In this chapter, we introduce an essential group of methods used to learn the MN classifier from examples which are the maximum-margin methods.

The approach to learn the MN classifier with an acyclic neighboring structure using Perceptron algorithm was suggested in [13, 5]. This approach, however, requires linearly separable data. Consequently, the method can be applied only for noisy-less examples. To resolve the deficiency, binary Support Vector Machines (SVM) algorithm was extended, to fit the structured output context. Therefore, to learn the MN classifiers with an acyclic neighboring structure, Hidden Markov Support Vector Machines [3, 4] and Max-Margin Markov networks [15] were proposed. The approach described in those papers is to transform the learning into solving a quadratic programming (QP) task with immense amount of linear constraints. The number of the constraints depends on the classifier, as it is proportional to the cardinality of its output. The QP problem is efficiently tractable using the optimization methods which rely on the fact that the MN classifier inference is tractable by dynamic programming.

The approach to learn the Associative Markov Networks (AMN) classifier structure was proposed by Taskar et al. [14]. In the AMN classifier, the neighboring structure is unrestricted, however, the pairwise quality functions are restricted in a similar fashion as in the Potts model. A new compact QP task can be composed by integrating a Linear Programming (LP) relaxation into the SVM QP task as described in [14]. The resulting task is tractable as it contains only a polynomial number of constraints.

The approach to learn a completely unrestricted MN classifier, that is, without any neighboring structure or pairwise quality scores structure restrictions is tackled in [7, 6]. In [7], an oracle is required to solve the inference problem efficiently. Therefore, in the case of an unrestricted MN classifier, the convergence is not ensured. In [6], the LP relaxation is used to transform the learning of the MN classifier into a QP task which contains a polynomial number of variables and constraints. This method will be further denoted as the M3N-LP algorithm.

So far, all the introduced methods required a fully annotated set of the training examples. Recently, in [8], the M3N-LP algorithm was extended to be able to learn from partially annotated (with missing labels) data. The adjustment preserves the learning as a tractable QP task with the same complexity as the one used to learn M3N-LP algorithm. This method will be further denoted as the MIL-M3N-LP algorithm (where MIL stands for MIssing Labels). In the case of fully annotated data, the MIL-M3N-LP can be used as well and becomes an M3N-LP equivalent.

All introduced algorithms are applicable to learn the linear version of the MN classifier. For linear MN classifier, the features are fixed and only the weights are the subject of learning. In this thesis, we extend the MIL-M3N-LP algorithm to learn the features simultaneously with the weights. To achieve this, features will be firstly extracted from the input using neural network and then forwarded to its last layer, which is the linear MN classifier. In this approach, we utilise that the loss function of the MIL-M3N-LP algorithm is differentiable, which means it can be used to train the neural network that extracts the features.

Chapter 3

Method

In this chapter, we formally describe the MN classifier which can be used as a Sudoku solver and then algorithms to learn its parameters from examples. Namely, we define the MN classifier in Section 3.1 and introduce the M3N algorithms in Section 3.2. We also describe the missing labels (MIL) extension, which allows us to learn the classifier from partially annotated data as well. Afterwards, in Section 3.3, we describe the extended algorithm using an LP relaxation, which allows us to learn the MN classifier for unrestricted graphs. The notation and methods described in Sections 3.1, 3.2 and 3.3 are adopted from [8]. Finally, in Section 3.4 which is dedicated to the contribution of this thesis, we describe how to append the MN classifier to the last layer of a neural network (NN), and how to learn the parameters of the neural network simultaneously with the parameters of the MN classifier using the PyTorch framework.

3.1 MN classifier

We treat the Sudoku puzzle as an instance of a structured output classification task. The input of the classifier is the puzzle assignment, and its output is a grid of labels representing the solution of the puzzle. The rules of Sudoku make the predicted labels interdependent. To model the dependencies between the labels, we use the framework of Markov Networks. The label dependency structure is defined a priori while the scores measuring match between the dependent labels are learned from examples by algorithms described in the follow-up sections.

A general MN classifier is defined as follows. Given an undirected graph $(\mathcal{V}, \mathcal{E})$ with a finite set of objects \mathcal{V} and a set of edges connecting these objects $\mathcal{E} \subseteq \binom{\mathcal{V}}{2}$. These objects are assigned labels $\mathbf{y} = (y_v \in \mathcal{Y} | v \in \mathcal{V})$, where \mathcal{Y} is a finite set, that depend on observation $\mathbf{x} \in \mathcal{X}$. Let $f_v(\mathbf{x}, y_v)$ be a scoring function that determines the match between an observation \mathbf{x} and a label y_v . Let $f_{vv'}(y_v, y_{v'})$ be a scoring function that determines the match between the labels $(y_v, y_{v'})$ on the edge $(v, v') \in \mathcal{E}$. The MN classifier is a function $\mathbf{h} : \mathcal{X} \rightarrow \mathcal{Y}^{\mathcal{V}}$ that for a given observation $\mathbf{x} \in \mathcal{X}$ outputs labeling, i.e. a

sequence of labels, that yields maximum score of function

$$\mathbf{h}(\mathbf{x}) \in \arg \max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{V}}} \sum_{v \in \mathcal{V}} f_v(\mathbf{x}, y_v) + \sum_{\{v, v'\} \in \mathcal{E}} f_{vv'}(y_v, y_{v'}). \quad (3.1)$$

The evaluation of the MN classifier leads to a max-sum problem which is essentially an NP-hard task, however there are some instances that can be solved more easily. One of those instances is an MN represented by a tree graph, for which the classifier can be evaluated using dynamic programming. However, in case of the Sudoku puzzle, the neighborhood structure $(\mathcal{V}, \mathcal{E})$ is a general graph which makes the prediction an NP hard problem. Nevertheless, we can resort to methods based on the Linear Programming relaxation of the max-sum problem described in [18]. The LP based max-sum solvers often find the optimal solution but there is no guarantee it will be the case. We will show experimentally, however, that in case of the Sudoku puzzle, the LP max-sum solves work reasonably well. The LP relaxation approach to solve the max-sum problem used is described in [18, 12].

3.2 Maximum Margin Markov Network (M3N) learning algorithm

Assuming the MN functions $f_v(\mathbf{x}, y_v)$ and $f_{vv'}(y_v, y_{v'})$ are linear in parameters, the MN classifier is an instance of a linear classifier defined as follows

$$\mathbf{h}(\mathbf{x}, \mathbf{w}) \in \operatorname{Argmax}_{\mathbf{y} \in \mathcal{Y}^{\mathcal{V}}} \langle \mathbf{w}, \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) \rangle \quad (3.2)$$

where $\mathbf{w} \in \mathbb{R}^d$ are weights to be learned and joint feature map $\boldsymbol{\psi}: \mathcal{X} \times \mathcal{Y}^{\mathcal{V}} \rightarrow \mathbb{R}^d$ is described as

$$\boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) = \sum_{v \in \mathcal{V}} \boldsymbol{\psi}_v(\mathbf{x}, y_v) + \sum_{(v, v') \in \mathcal{E}} \boldsymbol{\psi}_{vv'}(y_v, y_{v'}) \quad (3.3)$$

with individual feature maps $\boldsymbol{\psi}_v: \mathcal{X} \times \mathcal{Y}^{\mathcal{V}} \rightarrow \mathbb{R}^d$ and $\boldsymbol{\psi}_{vv'}: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^d$.

Given fully annotated training examples $\{(\mathbf{x}^i, \mathbf{y}^i) \in \mathcal{X} \times \mathcal{Y}^{\mathcal{V}} \mid i = 1, \dots, m\}$, M3N algorithm converts learning of the parameters \mathbf{w} into a convex unconstrained problem which reads

$$\mathbf{w}^* = \operatorname{Argmax}_{\mathbf{w} \in \mathbb{R}^n} \left[\frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \Delta(\mathbf{x}^i, \mathbf{y}^i, \mathbf{w}) \right] \quad (3.4)$$

where the margin-rescaling loss $\Delta: \mathcal{Y}^{\mathcal{V}} \times \mathcal{Y}^{\mathcal{V}} \times \mathbb{R}^d \rightarrow \mathbb{R}_+$ is defined as

$$\Delta(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \max_{\mathbf{y}' \in \mathcal{Y}^{\mathcal{V}}} [\ell(\mathbf{y}, \mathbf{y}') + \langle \mathbf{w}, \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}') \rangle] - \langle \mathbf{w}, \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) \rangle \quad (3.5)$$

and for $\ell: \mathcal{Y}^{\mathcal{V}} \times \mathcal{Y}^{\mathcal{V}} \rightarrow \mathbb{R}$ we use Hamming loss

$$\ell(\mathbf{y}, \mathbf{y}') = \sum_{v \in \mathcal{V}} \mathbb{1}[y_v \neq y'_v]. \quad (3.6)$$

In the case of an acyclic graph, the loss function 3.5 can be evaluated using dynamic programming, namely, the Viterbi algorithm was used. For graphs with cycles, we use linear programming (LP) relaxation described in Section 3.3 to transform the problem, so the loss is tractable.

3.2.1 Learning from partial annotations

Before proceeding to the LP relaxation, we first replace the margin-rescaling loss with its partial version. The reason behind the adjustment is that it allows us to train the MN classifier from partially annotated examples, i.e. when values of some labels are missing. This setup is particularly useful in the case of learning a Sudoku solver, as we can use only partial solutions of the assignments. Note that the original M3N algorithm relies on fully annotated examples where each value of the label in the training set has to be known. As gathering fully annotated data can be very resource demanding, it will be a significant advantage if our classifier can learn from partially annotated data while maintaining similar performance.

We define the set of partially annotated examples as

$\mathcal{D} = \{(\mathbf{x}^i, \mathbf{a}^i) \in \mathcal{X} \times \mathcal{A}^\mathcal{V} \mid i = 1, \dots, m\}$ where $\mathcal{A} = \{\mathcal{Y} \cup \{?\}\}$ is a new set of annotations and symbol ? represents a missing label.

To this end, we replace $\Delta(\mathbf{x}, \mathbf{y}, \mathbf{w})$ with $\Delta^p(\mathbf{x}, \mathbf{a}, \mathbf{w})$ defined as

$$\Delta^p(\mathbf{x}, \mathbf{a}, \mathbf{w}) = \max_{\mathbf{y}' \in \mathcal{Y}^\mathcal{V}} [\ell^p(\mathbf{a}, \mathbf{y}') + \langle \mathbf{w}, \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}') \rangle - \langle \mathbf{w}, \boldsymbol{\psi}^p(\mathbf{x}, \mathbf{a}) \rangle] \quad (3.7)$$

where $\ell^p: \mathcal{Y}^\mathcal{V} \times \mathcal{Y}^\mathcal{V} \rightarrow \mathbb{R}$ is the Hamming loss, with only annotated labels counted, that is,

$$\ell^p(\mathbf{a}, \mathbf{y}) = \sum_{v \in \mathcal{V}} \mathbb{I}[a_v \neq ?] \mathbb{I}[a_v \neq y_v] \quad (3.8)$$

and the rescaled feature maps $\boldsymbol{\psi}^p: \mathcal{X} \times \mathcal{Y}^\mathcal{V} \rightarrow \mathbb{R}^d$ are calculated as

$$\boldsymbol{\psi}^p(\mathbf{x}, \mathbf{a}) = \sum_{v \in \mathcal{V}} \frac{\mathbb{I}[a_v \neq ?]}{p(a_v \neq ? | x)} \boldsymbol{\psi}_v(\mathbf{x}, y_v) + \sum_{v, v' \in \mathcal{E}} \frac{\mathbb{I}[a_v \neq ? \wedge a_{v'} \neq ?]}{p(a_v \neq ?, a_{v'} \neq ? | x)} \boldsymbol{\psi}_{vv'}(y_v, y_{v'}). \quad (3.9)$$

Terms $p(a_v \neq ? | x)$ and $p(a_v \neq ?, a_{v'} \neq ? | x)$ describe the probability that a label is not missing for a given object and the probability that both labels are not missing for objects connected by a given edge, respectively. As the labels in our case are missing uniformly, we estimate these probabilities as $p(a_v \neq ? | x) = 1 - \pi$ and $p(a_v \neq ?, a_{v'} \neq ? | x) = (1 - \pi)^2$ where π is a probability that a single label is missing calculated as a share of missing labels in proportion to the amount of all labels. The gradient of 3.7 with respect to \mathbf{w} can be calculated directly. The formula for the gradient can be found in [8]. We will also refer to $\Delta^p(\mathbf{x}, \mathbf{a}, \mathbf{w})$ as the partial margin-rescaling loss and to this learning algorithm as MIL-M3N.

3.3 LP relaxation of the partial margin-rescaling loss

As mentioned in [8], LP relaxation is necessary to learn the classifier for a general graph, since it will provide a tractable loss even for graphs containing cycles, such as the one describing label correlations following from the Sudoku rules. To achieve that, we first rewrite the partial margin-rescaling loss as

$$\Delta^p(\mathbf{x}, \mathbf{a}, \mathbf{w}) = \max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{V}}} f(\mathbf{x}, \mathbf{y}, \mathbf{a}, \mathbf{w}) - \langle \mathbf{w}, \boldsymbol{\psi}^p(\mathbf{x}, \mathbf{a}) \rangle. \quad (3.10)$$

Afterwards, we replace the first term with its upper bound defined as

$$\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{V}}} f(\mathbf{x}, \mathbf{y}, \mathbf{a}, \mathbf{w}) \leq \min_{\boldsymbol{\varphi} \in \mathbb{R}^{2|\mathcal{E}||\mathcal{Y}|}} u(\mathbf{x}, \mathbf{a}, \boldsymbol{\varphi}, \mathbf{w}) \quad (3.11)$$

where

$$u(\mathbf{x}, \mathbf{a}, \boldsymbol{\varphi}, \mathbf{w}) = \sum_{v \in \mathcal{V}} \max_{y \in \mathcal{Y}} u_v(\mathbf{x}, a_v, y, \boldsymbol{\varphi}, \mathbf{w}) + \sum_{v, v' \in \mathcal{E}} \max_{(y, y') \in \mathcal{Y}^2} u_{vv'}(y, y', \boldsymbol{\varphi}, \mathbf{w}). \quad (3.12)$$

The functions $u_v(\mathbf{x}, a_v, y, \boldsymbol{\varphi}, \mathbf{w})$ and $u_{vv'}(y, y', \boldsymbol{\varphi}, \mathbf{w})$ are described as

$$u_v(\mathbf{x}, a_v, y, \boldsymbol{\varphi}, \mathbf{w}) = \llbracket a_v \neq ? \rrbracket \llbracket a_v \neq y_v \rrbracket + \langle \mathbf{w}, \boldsymbol{\psi}_v(\mathbf{x}, y) \rangle \quad (3.13)$$

$$- \sum_{v' \in N(v)} \varphi_{vv'}(y)$$

$$u_{vv'}(y, y', \boldsymbol{\varphi}, \mathbf{w}) = \langle \mathbf{w}, \boldsymbol{\psi}_{vv'}(y, y') \rangle + \varphi_{vv'}(y) + \varphi_{v'v}(y') \quad (3.14)$$

where $\boldsymbol{\varphi} \in \mathbb{R}^{m \times 2|\mathcal{E}||\mathcal{Y}|}$ is a set of auxiliary parameters that are unique for each training example. These parameters are learned in conjunction with the weights \mathbf{w} and are used only for training, to reparametrize the loss. The gradients of the LP-relaxed partial margin-rescaling loss with respect to \mathbf{w} and $\boldsymbol{\varphi}$ can be directly calculated as described in [8]. In the following sections of the thesis, we refer to this learning algorithm as MIL-M3N-LP.

3.4 Learning MN classifier on top of the Neural Network

In Section 3.3 we described the MIL-M3N-LP learning algorithm based on the minimization of a convex and tractable loss, which can be used for an unrestricted MN problem. As the loss function in this algorithm is differentiable, we can take advantage of it and append it to a neural network (NN). This will result in a neural network that learns the features simultaneously with the weights of the MN classifier.

3.4.1 Symbolic PyTorch implementation

Until this point, the gradients of the loss function could be calculated directly, and therefore there was no need to use any specific framework. However, after prepending the NN, the formulas for calculating gradients become absurdly complicated, as the resulting prediction function is composed of many components. Instead of computing the gradient manually, we use PyTorch framework for automated differentiation, which allows us to implement the NN alongside with the loss function with automatically tracked gradient of their weights and parameters provided by the PyTorch Autograd functionality.

Firstly, to verify the expected behavior, we implement learning of the linear MN classifier which involves only the loss function and NN with just a single (linear) layer. That is, the layer takes the symbolic input and outputs the unary functions represented by the matrix $Q \in \mathbb{R}^{|\mathcal{Y}| \times |\mathcal{Y}|}$, where each column corresponds to the unary scores $f_v(x, y)$, and the binary functions represented by the matrix $G \in \mathbb{R}^{|\mathcal{Y}| \times |\mathcal{Y}|}$, which corresponds to the pairwise scores $f_{vv'}(y, y')$. In the context of our loss function, Q represents the unary scores $\langle \mathbf{w}, \boldsymbol{\psi}_v(\mathbf{x}, y) \rangle$ in Equation 3.13 and G represents the pairwise scores $\langle \mathbf{w}, \boldsymbol{\psi}_{vv'}(y, y') \rangle$ in Equation 3.14. The Q and G matrices are then forwarded to the last layer, which computes the loss in the same way as the MIL-M3N-LP algorithm.

3.4.2 Optimizing weights

To update the weights and parameters, we use Adam optimizer [9]. As there are several sets of parameters, all of them have to be registered when we initialize the optimizer. The tracking of the gradients of the parameters $\boldsymbol{\varphi}$ must be manually turned on and off. This is done by setting the property *requires_grad* of the currently selected training example parameter vector $\boldsymbol{\varphi}^i$ to *True* when performing the forward pass, calculating the gradient in the backward pass, and then setting it back to *False* when the update is performed. This approach prevents us from receiving zero gradients for parameters $\{\boldsymbol{\varphi}^j \in \boldsymbol{\varphi} \mid j = 1 \dots m, j \neq i\}$, which are not currently used, and helps to reduce computational complexity. Furthermore, we had to set all gradients to *None* after performing an optimization step. The reason behind this is again to avoid zero gradients for $\boldsymbol{\varphi}$ parameters, since they are only calculated once every epoch. Without averting zero gradients, the $\boldsymbol{\varphi}$ parameters will continue to update during each step as a result of momentum in the Adam optimizer.

3.4.3 Neural network architecture used as a backbone

After tuning and verifying the desired behavior of our implementation, we proceed to prepend a convolutional neural network with multiple layers instead of using a single layer as described in the previous Section. As the NN backbone, we use the LeNet5 architecture described in [10]. The LeNet5 architecture is shown in Figure 3.1.

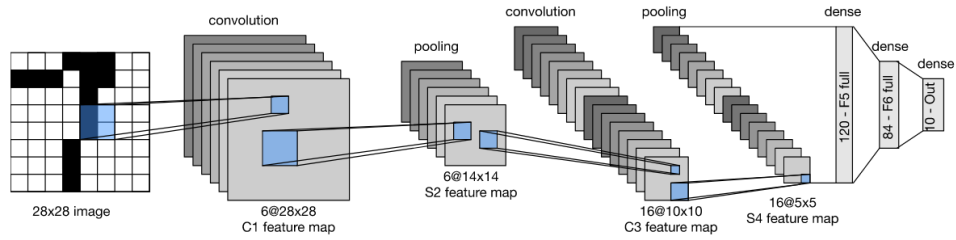


Figure 3.1: LeNet5 architecture [2]

We use the ReLU function as an activation function for the first 4 layers and the softmax function for the last one that enters the MN classifier. Additionally, the number of outputs of the last layer is adjusted to 9, as we want the network to output scores for labels 1 to 9.

Experiments with the resulting network architecture showed that the unary scores of the MN classifiers are prone to overfitting. Therefore, some form of weight regularization had to be used. Firstly, we tried to regularize the unary functions directly using the L2 penalty of the Q matrix, that is, adding the quadratic $\lambda \|Q\|^2$ to the loss function. For this regularization to be effective, the softmax function had to be removed from the last layer, because the softmax already rescales the Q matrix before the penalty is calculated, rendering the regularization ineffective. However, the performance without softmax turned out to be underwhelming, so we resorted to a different approach. We kept the softmax function and instead we regularized all the weights and parameters in the NN using *weight_decay* option of the Adam optimizer which automatically applies L2 penalty to all the optimized tensors. This solution turned out to be sufficient to prevent the model from being overfitted. We will further refer to this learning algorithm as NN-MIL-M3N-LP.

Chapter 4

Results

In this chapter, we describe the experimental evaluation of algorithms for learning MN classifier which were described in Chapter 3.

4.1 Datasets

We used 3 prediction tasks to evaluate the implemented algorithms:

1. Prediction of a sequence of symbols generated by a Hidden Markov Chain (HMC). In this case, we know the data generating process, and hence we can compare the performance of the learned MN classifier to the optimal Bayes predictor.
2. Prediction of the solution of the Sudoku puzzle with symbolically assigned input. That is, the input cells are one-hot-encoded digits.
3. Prediction of the solution of the Sudoku puzzle with visual input. That is, the input cells are images of handwritten digits taken from the MNIST dataset [11].

4.1.1 HMC

To create the HMC dataset, we used a setup similar to that in [8]. We define the observed sequence as $\mathbf{x} = (x_0, \dots, x_{99}) \in \{0, \dots, 29\}^{100}$ and the sequence of labels to be predicted as $\mathbf{y} = (y_0, \dots, y_{99}) \in \{0, \dots, 29\}^{100}$.

The HMC used to generate the sequences is defined as

$$p(\mathbf{x}, \mathbf{y}) = p(y_0) \prod_{i=1}^{99} p(y_i | y_{i-1}) p(x_i | y_i). \quad (4.1)$$

We used emission probability $p(x_i | y_i) = 7/10$ if $x_i = y_i$ and $p(x_i | y_i) = 3/290$ otherwise and transition probability $p(y_i | y_{i-1}) = 7/10$ if $y_i = y_{i-1}$ and $p(y_i | y_{i-1}) = 3/290$ otherwise. The prior probability is $p(y_0) = 1/30$. The graph $(\mathcal{V}, \mathcal{E})$ representing the HMC sequence is a chain $\mathcal{V} = \{v | v = 0, 1, \dots, 99\}$, $\mathcal{E} = \{(v, v') | v \in \mathcal{V}, v' \in \mathcal{V}, v = v' - 1\}$. We used the Viterbi algorithm to evaluate the MN classifier for the HMC graph.

Using this setup, we generated 10,000 test sequences, 5,000 validation sequences, and 1,000 training sequences.

We compared the performance of the learned MN classifier to the baseline predictors. First, we used the Maximum A Posteriori (MAP) predictor, which is inferred from the generating distribution (4.1), and predicts the most likely sequence of labels. The MAP predictor is known to be the optimal solution when the goal is to minimize the expectation of 0/1-loss

$$\ell_{0/1}(\mathbf{y}, \mathbf{y}') = \bigvee_{v \in \mathcal{V}} \llbracket y_v \neq y'_v \rrbracket. \quad (4.2)$$

Second, we used the predictor based on the Forward-Backward (FB) algorithm, which predicts the sequence of the most likely labels. The FB predictor is the optimal solution when the goal is to minimize the expectation of Hamming loss,

$$\ell_h(\mathbf{y}, \mathbf{y}') = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \llbracket y_v \neq y'_v \rrbracket. \quad (4.3)$$

which is at the same time the target objective in this experiment. Hence, to evaluate the accuracy of the predictors compared, we used the average of the Hamming loss computed on the test samples.

We evaluated MAP and FB predictors on the 10,000 test HMC sequences and they achieve approximately 75 % and 80 % accuracy, respectively, in terms of the average Hamming loss (4.3).

4.1.2 Symbolic Sudoku

A Sudoku puzzle is defined by a 9×9 grid, where each cell is either empty or contains a number ranging from 1 to 9. The goal of solving a Sudoku puzzle is to fill the empty cells with numbers ranging from 1 to 9 while fulfilling the rule that each row, column and each of the 9 non-overlapping 3×3 sub-squares contain each number from 1 to 9 exactly once. To treat solving the Sudoku puzzle as a prediction problem, we denote the observations as $\mathbf{x} = (x_0, \dots, x_{80}) \in \{0, \dots, 9\}^{81}$ and the labeling as $\mathbf{y} = (y_0, \dots, y_{80}) \in \{1, \dots, 9\}^{81}$. The graph $(\mathcal{V}, \mathcal{E})$ of the Sudoku puzzle consists of the objects $\mathcal{V} = \{v | v = 0, 1, \dots, 80\}$ representing each of the 9×9 cells and edges $\mathcal{E} = \{(v, v') | v \in \mathcal{V}, v' \in \mathcal{V}, v < v', (v/9 = v'/9) \vee (v \bmod 9 = v' \bmod 9) \vee (v/3 = v'/3 \wedge v \bmod 3/3 = v' \bmod 3/3)\}$ representing the label relationships inferred from the rules of the puzzle. To evaluate the MN classifier for the Sudoku graph, we used the augmenting DAG algorithm described in [18].

Symbolic Sudoku dataset containing 1,000,000 puzzles with solution was downloaded from [1]. Each example is represented by string of digits ranging from 0 to 9, where 0 represents an empty slot in the Sudoku assignment.

An example Sudoku input and labels are shown in Tables 4.1 and 4.2

0	0	4	3	0	0	2	0	9
0	0	5	0	0	9	0	0	1
0	7	0	0	6	0	0	4	3
0	0	6	0	0	2	0	8	7
1	9	0	0	0	7	4	0	0
0	5	0	0	8	3	0	0	0
6	0	0	0	0	0	1	0	5
0	0	3	5	0	8	6	9	0
0	4	2	9	1	0	3	0	0

Table 4.1: Symbolic Sudoku input example

8	6	4	3	7	1	2	5	9
3	2	5	8	4	9	7	6	1
9	7	1	2	6	5	8	4	3
4	3	6	1	9	2	5	8	7
1	9	8	6	5	7	4	3	2
2	5	7	4	8	3	9	1	6
6	8	9	7	3	4	1	2	5
7	1	3	5	2	8	6	9	4
5	4	2	9	1	6	3	7	8

Table 4.2: Symbolic Sudoku labels example

4.1.3 Visual Sudoku

Visual Sudoku Data Set was created from the symbolic Sudoku dataset by replacing the input symbols with the 28×28 images of handwritten digits, which were obtained from the MNIST dataset [11]. The Visual Sudoku Data Set contains 6 000 samples divided into 5 sets, each set containing 1 000 training examples, 100 validation examples, and 100 testing examples. We ensured that there is no overlap between images used in validation and testing sets; however, some digits in the training part are used multiple times due to insufficient number of images in the MNIST dataset. An example from the Visual Sudoku Data Set is shown in Figure 4.1.

	8		5	3		7		6
		9	1					
5		2		7				
		3		6		5	2	1
		5	9			8		
6		7		1			3	4
9			7			2		3
	4	1			8		7	
			2	5	6		1	

Figure 4.1: An example Sudoku puzzle assignment from the Visual Sudoku Data Set.

4.2 Compared algorithms

We compare the following algorithms:

MIL-M3N algorithm. It learns the linear MN classifier (3.2) with an acyclic neighborhood structure. It is described in Section 3.2.

MIL-M3N-LP algorithm. It learns the linear MN classifier (3.2) with a generic neighborhood structure. It is described in Section 3.3.

NN-MIL-M3N/NN-MIL-M3N-LP algorithms proposed in this thesis. It learns the MN classifier (3.1) with a generic neighborhood structure and the features of the unary scores extracted by a neural network. In particular, we use the LeNet5 [10] architecture to extract the features. This extension of the MIL-M3N and MIL-M3N-LP algorithms is described in Section 3.4. Note, that we use the NN prefix to refer to the PyTorch implementation. In the case of the symbolic input, however, there is not an actual NN employed. Instead, only one linear layer to convert the input is used, as described in Section 3.4.1.

4.3 Verification of the PyTorch implementation on the HMC dataset

We used the MIL-M3N learning algorithm, implemented in plain Python, as a baseline to verify that its PyTorch implementation (further referred to as NN-MIL-M3N) is correct, before proceeding to problems that require relaxation of the LP and the features extracted by the NN network. The MIL-M3N, as well as the NN-MIL-M3N algorithms, were evaluated on the HMC dataset described in Section 4.1.1. In the experiment, we used the following setup:

- Number of training examples: 10, 50, 100, 200, 500, 1 000
- Missing probability: 0 %, 10 %, 20 %, 50 %
- Regularization constant λ : 0.1, 1, 10, 100

To evaluate the test error, we used the average Hamming loss as the metric. To evaluate the training partial error, we used partial average Hamming loss defined as

$$\ell_h^p(\mathbf{y}, \mathbf{y}') = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \llbracket y_v \neq y'_v \rrbracket \llbracket y_v \neq ? \rrbracket. \quad (4.4)$$

For each model, the best λ was determined by evaluation of the 5,000 validation examples. Figure 4.2 shows the performance of trained models for their best lambda. The test error was evaluated on the 10,000 test examples. The training partial error was evaluated on all examples the model was trained on.

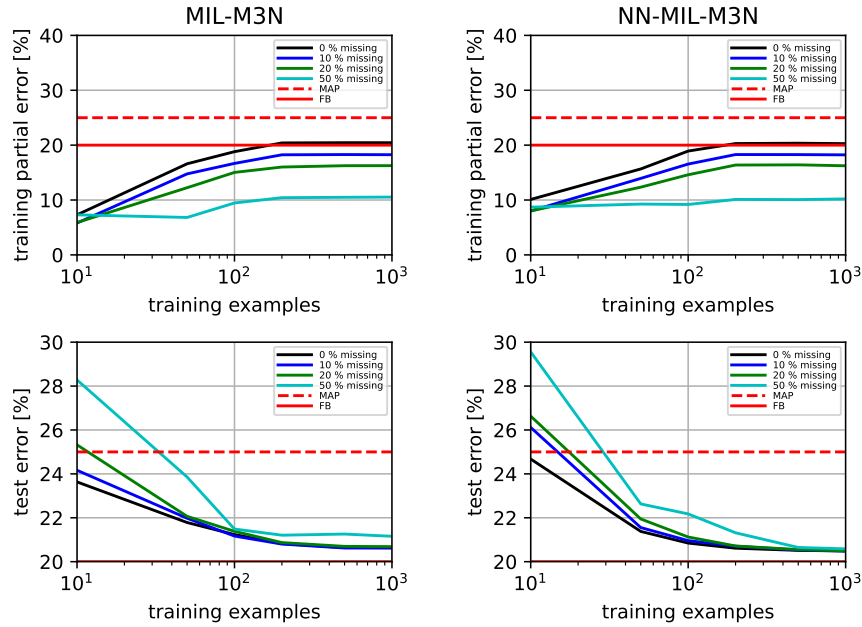


Figure 4.2: Train and test error of MIL-M3N and NN-MIL-M3N models

As we can see in Figure 4.2 the results of the MN classifier learnt by MIL-M3N and NN-MIL-M3N methods converge to similar values and we see that both approach the accuracy of the optimal FB predictor.

In addition, to verify the resulting models, we visualize and compare the learned weights. Figure 4.3 visualizes the weights of the MN classifier learned on the 1,000 training examples, with 0 % missing probability and $\lambda = 0.1$ using the MIL-M3N and NN-MIL-M3N methods. As can be seen, the visualization provides additional confirmation that the learned weights are the same. Additionally, it is clear that the weights represent the HMC transition and emission probabilities defined in 4.1.1 accordingly, as the highest weights are assigned to the same observation-label and label-label pairs.

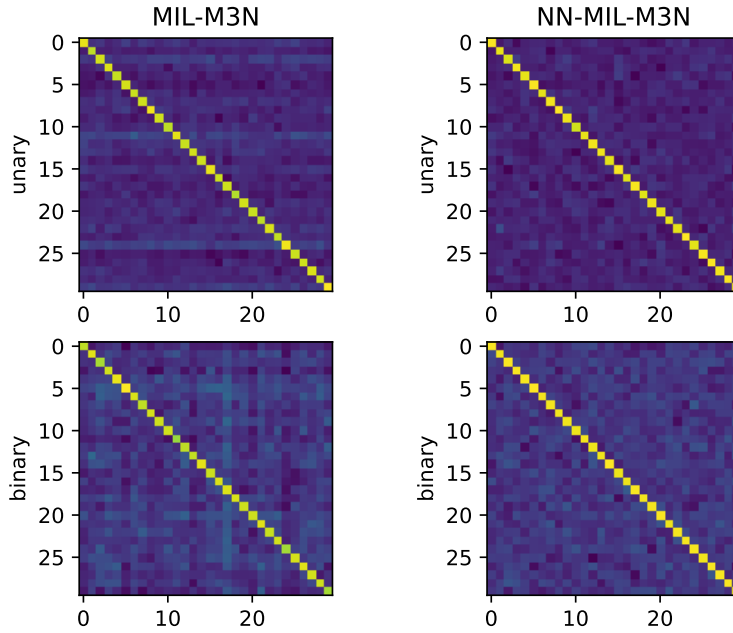


Figure 4.3: Weights comparison of MIL-M3N and NN-MIL-M3N models

4.4 Verification of the PyTorch implementation on the symbolic Sudoku dataset

We used the MIL-M3N-LP algorithm, implemented in a plain Python, as another baseline to verify the functionality of the PyTorch implemented version (further referred to as NN-MIL-M3N-LP algorithm). To evaluate the algorithms we used the symbolic Sudoku dataset.

We tested the classifiers on the symbolic Sudoku dataset using the exact same setup as in Section 4.3, however, regularization was not required; therefore, we used $\lambda = 0$.

The results obtained are shown in Figure 4.4. As we can see, the 2 models produce similar results with only significant deviation for 10 training examples and 50 % missing probability. Other than that, the results show that both methods are capable to learn the MN classifier as nearly perfect symbolic Sudoku solver.

Additionally, we visualize the learned score functions, to receive another performance assurance. The visualized weights are shown in Figure 4.5. As we can see, the learned weights of the two classifiers correspond with each other. Furthermore, the representation of the Sudoku rules is expressed in the learned weights clearly, as they assign high score to the observation with the equivalent label and low score to same label-label pair of any edge.

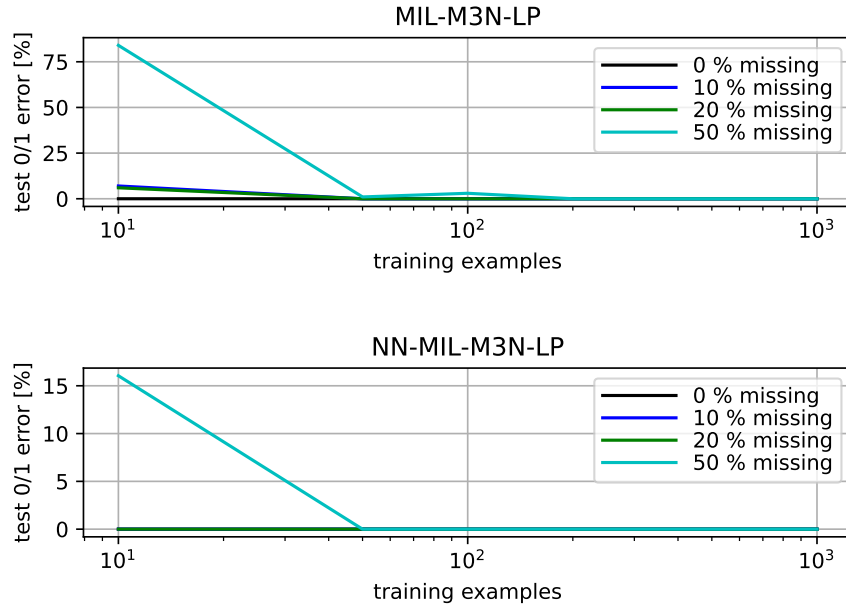


Figure 4.4: Performance of MN classifiers learned by the MIL-M3N-LP and NN-MIL-M3N-LP algorithms on the symbolic Sudoku dataset.

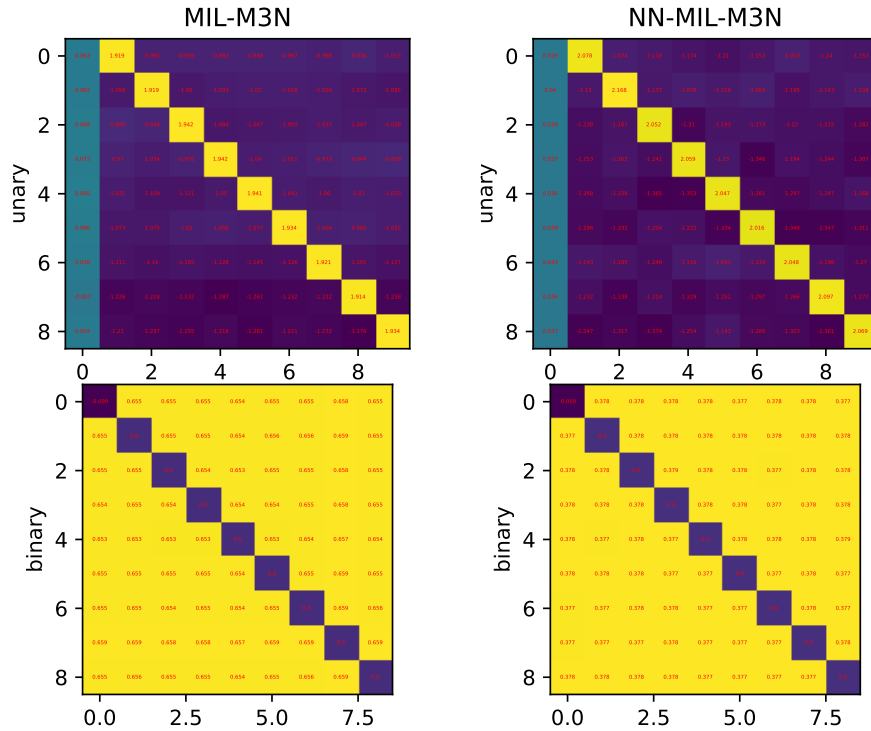


Figure 4.5: Weights comparison of MN classifiers learned by MIL-M3N-LP and NN-MIL-M3N-LP algorithms on the symbolic Sudoku dataset.

4.5 Evaluation of the proposed method on the Visual Sudoku dataset

To learn the visual Sudoku solver, we used the NN-MIL-M3N-LP algorithm learning parameters of the MN classifier on top of the LeNet5 architecture; for more details, see Section 3.4. As the optimizer minimizes a proxy loss function, the first way to verify the functionality of the learning algorithm is to evaluate the actual target loss, in our case the 0/1 loss, on the training set and to observe how it evolves as a function of the number of epochs completed. The training performance of the NN-MIL-M3N-LP model can be seen in Figure 4.6.

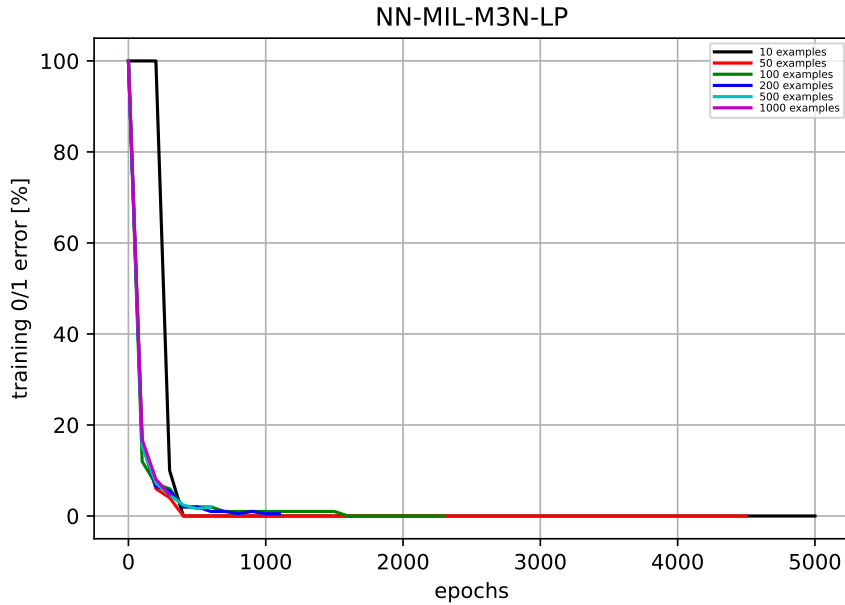
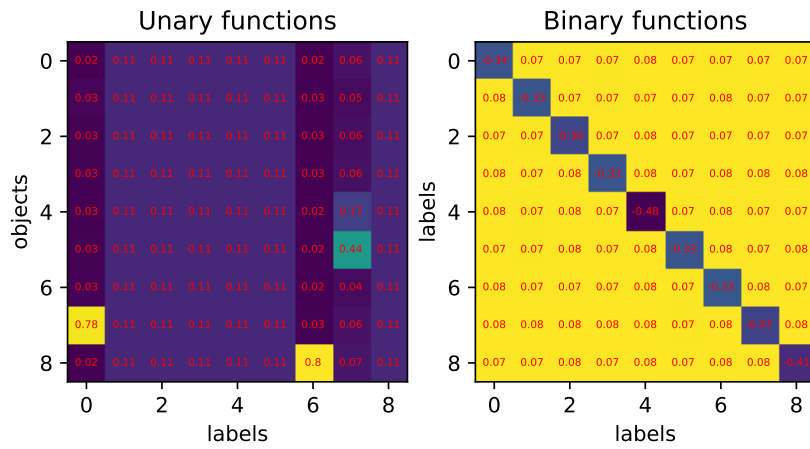


Figure 4.6: The evolution of the training 0/1-loss when learning the Visual Sudoku solver.

We can see that the training 0/1-loss is monotonically decreasing with the number of epochs finished, which provides the assurance that our algorithm works well on the training data at least.

Another way to verify the results is to perform a visual inspection of the learned score functions. As we have already visualized the score functions of the symbolic Sudoku solver, see Figure 4.5, we have an idea of what these functions should look like. Therefore, we visualize the functions only for the first row of the visual Sudoku assignment shown; see Figure 4.7. The visualization of the learned score functions is shown in Figure 4.8. The score functions appear to be similar to those in Figure 4.5 and seem to correctly encode the rules of the Sudoku puzzle.

First row of Sudoku input

**Figure 4.7:** An example of a single row of the Visual Sudoku assignment.**Figure 4.8:** Visualization of the learned score functions of the visual Sudoku solver.

To further evaluate our visual Sudoku classifier, we use a simple baseline. We assume a deterministic Sudoku solver, which always solves Sudoku with correct input and we calculate probability, that LeNet5 model correctly classifies all digits in an example. As stated in [10], LeNet5 achieves around 99.05 % accuracy on the MNIST dataset. Therefore, we can compute the probability of correctly classified Sudoku example simply as

$$p_c = 99.05^N \quad (4.5)$$

where N is the number of nonempty cells in the Sudoku example. After evaluating this accuracy on the 100 test examples, we achieve the average probability of correct classification $p_a = 72.35\%$.

We trained the solver with the same setup as in 4.3, however we used *weight_decay* parameters $\lambda = 0.1, 0.3, 0.5, 0.9$. The best λ was selected by evaluation on the 100 validation examples. For the best λ we evaluate the

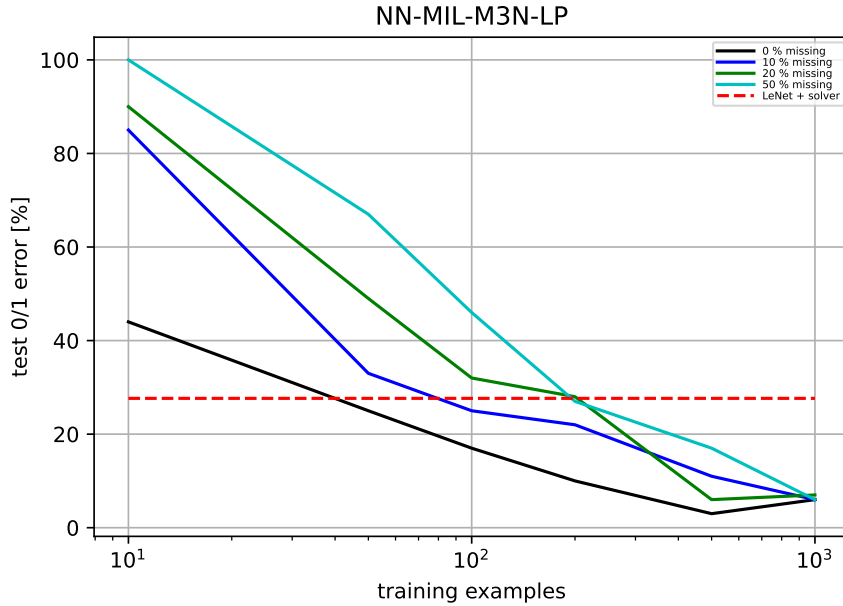


Figure 4.9: Average 0/1-loss of the Visual Sudoku solver in test examples with respect to the number of training examples used.

Visual Sudoku Solver on the 100 test examples. The performance of the Visual Sudoku Solver learned by the proposed NN-MIL-M3N-LP algorithm can be seen in Figure 4.9. It is shown, that our Visual Sudoku Solver outperforms the baseline with just 100 training examples and achieves the peak accuracy of 97 %. Additionally, it seems the accuracy for 1000 training examples might still improve with further training. We only trained the configuration with 1000 training examples for 600 epochs. The configuration with 500 training examples achieved its peak performance around 1000 epochs.

As another baseline, we use SATNet [17]. The SATNet architecture is composed of a convolutional neural network and the maximum satisfiability (MAXSAT) solver as the last layer. SATNet was trained on 9000 training examples without missing labels and evaluated on 1000 examples for which it achieved 63.2 % accuracy on visual Sudoku.


The last baseline that we use is the MN classifier learned by the LP-MIL-M3N algorithm described in [8]. In this paper, they used the Radial Basis Function (RBF) kernels to extract the unary feature maps. They used the same setup as us (see Section 4.3), without weight regularization ($\lambda = 0$). For the missing label probabilities of 0 %, 10 %, 20 % and 50 % they achieved 90 %, 80.4 %, 89.8% and 76.4 % accuracy, respectively.

The comparison of our NN-MIL-M3N-LP algorithm with previously mentioned baselines is summarized in Table 4.3. As can be seen, the NN-MIL-M3N-LP algorithm outperforms all baselines by a large margin and approaches

Method	Annotated labels	Accuracy
NN-MIL-M3N-LP (PROPOSED)	100 %	97.0 %
	90 %	94.0 %
	80 %	94.0 %
	50 %	94.0 %
LP-MIL-M3N + RBF kernels [8]	100 %	90.0 %
	90 %	80.4 %
	80 %	89.8 %
	50 %	76.4 %
LeNet5 [10] + perfect solver	100 %	72.35 %
SATNet [17]	100 %	63.2 %

Table 4.3: The performance of the Visual Sudoku solver learned by the proposed NN-MIL-M3N-LP algorithm and comparison to various baselines.

almost perfect accuracy.



Chapter 5

Conclusions

In this thesis, we have addressed the problem of learning a visual Sudoku solver from examples. We have treated the solver as an instance of a Markov Network (MN) based structured output classifier. We have proposed to integrate the MN classifier with neural networks. We have shown how to use the MIL-M3N-LP algorithm, recently proposed in [8], to learn the parameters of the MN classifier simultaneously with a neural network to extract the features of the classifier. The main challenge was to deal with the complicated loss of the MIL-M3N-LP algorithm, which maintains specific auxiliary variables for each training example. We used the PyTorch library, which turned out to be sufficiently flexible to integrate the custom loss function relatively smoothly. We verified the functionality of our implementation using a series of controlled experiments. We have also experimentally shown that the visual Sudoku solver learned by the proposed method outperforms all baselines, achieving a test precision of 97%. All implemented algorithms and used code can be found in this repository: <https://gitlab.fel.cvut.cz/kadlet14/visual-sudoku-solver>.



Bibliography

- [1] 1 million sudoku games. <https://www.kaggle.com>.
- [2] Convolutional neural networks (lenet). https://d2l.ai/chapter_convolutional-neural-networks/lenet.html#lenet. [Online; accessed 19-May-2022].
- [3] Y. Altun and T. Hofmann. Large margin methods for label sequence learning. In *European Conference on Speech Communication and Technology*, 2003.
- [4] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In *International Conference on Machine Learning*, 2003.
- [5] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Conference on Empirical Methods in Natural Language Processing*, 2002.
- [6] V. Franc and P. Laskov. Learning maximal margin markov networks via tractable convex optimization. *Control Systems and Computers*, (2):25–34, 2011.
- [7] V. Franc and B. Savchynskyy. Discriminative learning of max-sum classifiers. *Journal of Machine Learning Research*, 9(1):67–104, 2008.
- [8] Vojtech Franc and Andrii Yermakov. Learning maximum margin markov networks from examples with missing labels. In Vineeth N. Balasubramanian and Ivor Tsang, editors, *Proceedings of The 13th Asian Conference on Machine Learning*, volume 157 of *Proceedings of Machine Learning Research*, pages 1691–1706. PMLR, November 2021.
- [9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of International Conference on Learning Representations (ICLR)*, 2015.
- [10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

- [11] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.
- [12] M.I. Schlesinger. Syntactic analysis of two-dimensional visual signals in noisy conditions. *Kibernetika*, (4):113–130, 1976. In Russian.
- [13] Michail I. Schlesinger and Václav Hlaváč. *Ten Lectures on Statistical and Structural Pattern Recognition*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.
- [14] B. Taskar, V. Chatalbashev, and D. Koller. Learning associative markov networks. In *International Conference on Machine Learning (ICML)*, 2004.
- [15] B. Taskar, C. Guestrin, and D. Koller. Maximum-margin markov networks. In *Proc. of Neural Information Processing Systems (NIPS)*, 2004.
- [16] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *NIPS*, 2003.
- [17] P.W. Wang, P.L. Donti, B. Wilder, and J.Z. Kolter. SATnet: Bridging deep learning and logical reasoning using a differential satisfiability solver. In *ICML*, 2019.
- [18] T. Werner. A linear programming approach to max-sum problem: A review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, 2007.