

Neuro-Symbolic Constraint Programming for Structured Prediction

Paolo Dragone¹, Stefano Teso² and Andrea Passerini²

¹Twitter, London, United Kingdom

²University of Trento, Trento, Italy

Abstract

We propose Nester, a method for injecting neural networks into constrained structured predictors. Nester first uses a neural network to compute an initial prediction that may or may not satisfy the constraints, and then applies a constraint-based structured predictor to refine the raw predictions according to hard and soft constraints. Nester combines the advantages of its two components: the network can learn complex representations from low-level data while the constraint program on top reasons about the high-level properties and requirements of the prediction task. An empirical evaluation on handwritten equation recognition shows that Nester achieves better performance than both the either component in isolation, especially when training examples are scarce, while scaling to more complex problems than other neuro-programming approaches. Nester proves especially useful to reduce errors at the *semantic* level of the problem, which is particularly challenging for neural network architectures.

Keywords

Machine Learning, Neuro-Symbolic Integration, Structured Prediction, Constraint Programming

1. Introduction

We study neuro-symbolic integration in the context of structured output prediction under constraints. In structured prediction, one predicts objects – like parse trees, floor plans, and molecules – encoded by multiple interdependent variables [1]. We tackle the more complex problem of predicting structures subject to complex logical-numerical constraints from complex sub-symbolic inputs like images or sensor data.


This problem is beyond the reach of existing methods. Classical frameworks for structured prediction [2, 3] lack support for representation learning whereas most deep structured prediction approaches have no support for constraints [4, 5]. Existing neuro-symbolic approaches are restricted to logical constraints and either rely on fuzzy logic [6, 7] or enforce satisfaction of constraints only in expectation [8], failing to deal with hard constraints. Closest to our work, DeepProbLog [9] supports sub-symbolic data as well as logic and (discrete) numeric constraints, but relies on an inference procedure that becomes prohibitively expensive when dealing with structured prediction problems with a high degree of non-determinism, as highlighted by our experiments.

NeSy'21: 15th International Workshop on Neural-Symbolic Learning and Reasoning

✉ pdragone@twitter.com (P. Dragone); stefano.teso@unitn.it (S. Teso); andrea.passerini@unitn.it (A. Passerini)

ORCID 0000-0002-9147-8514 (P. Dragone); 0000-0002-2340-9461 (S. Teso); 0000-0002-2765-5395 (A. Passerini)

© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

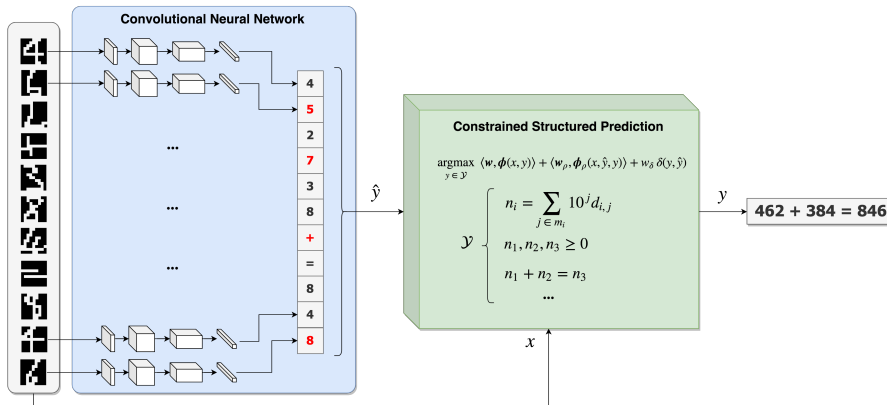


Figure 1: How Nester predicts an equation y from a sequence of images x . Each image is first classified by a CNN (the same network for each image) and the predicted labels are grouped into a sequence \hat{y} that happens to violate both syntactic and semantic constraints. The structured predictor then takes x and \hat{y} and outputs a refined label y that is guaranteed to satisfy all constraints. (Best viewed in color.)

We propose Nester (NEuro-Symbolic consTrainEd structuRed predictor), an approach that addresses constrained structured prediction with sub-symbolic inputs and complex constraints. Nester integrates a neural network and a constrained structured predictor. The former uses the input data alone to produce an initial guess, i.e., a structure that may violate some constraints. The structured predictor takes this initial guess and leverages a constraint satisfaction solver to impose hard and soft constraints on it. Inspired by work on declarative structured prediction [10], Nester leverages MiniZinc [11] to define the constraint program and can therefore deal with both hard and soft constraints as well as categorical and numerical variables. The entire architecture can be trained end-to-end.

We evaluated Nester on handwritten equation recognition [10], a sequence prediction task in which the prediction must obey both syntactic and semantic constraints, cf. Figure 1. Our results show that the neural network and the constrained structured predictor work in synergy, achieving better results than either model taken in isolation. Importantly, the neural network alone struggles to achieve low error at the semantic level, whereas Nester predicts structures that are both syntactically and semantically sound.

2. Structured Prediction under Constraints

The goal of structured prediction is to output a structured object $y \in \mathcal{Y}$ that best matches a given input $x \in \mathcal{X}$. Learning a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ directly is non-trivial. One strategy is to acquire a scoring function $F : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ that measures the compatibility between inputs and outputs and then use it to identify a highly compatible output $y = f(x) := \operatorname{argmax}_{y' \in \mathcal{Y}} F(x, y')$ [12]. A principled approach for estimating such scoring functions are structured SVMs [3], which model $F(x, y)$ as a linear function of the form $\langle \mathbf{w}, \phi(x, y) \rangle$, where $\phi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$ is a feature map that transforms input-output pairs into a joint, d -dimensional feature space, and $\mathbf{w} \in \mathbb{R}^d$ are weights learned from data. Given

a training set of input-output pairs $\{(x_i, y_i)\}_{i=1}^n$, where y_i is a high-quality output for x_i , structured SVMs learn \mathbf{w} by solving the following quadratic problem:

$$\min_{\mathbf{w}, \xi} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \xi_i \quad \text{s.t.} \quad \forall i, \forall y \in \mathcal{Y} \setminus \{y_i\} . \langle \mathbf{w}, \phi(x_i, y_i) \rangle - \langle \mathbf{w}, \phi(x_i, y) \rangle \geq \Delta(y_i, y) - \xi_i$$

For each example $i \in [n]$, the constraints encourage the correct label $\langle \mathbf{w}, \phi(x_i, y_i) \rangle$ to score above any alternative label y by a task-dependent margin $\Delta(y_i, y)$ that measures how much y differs from y_i . The objective minimizes the slacks ξ_i , which grow when a constraint is not satisfied, and an ℓ_2 regularization term $\frac{\lambda}{2} \|\mathbf{w}\|^2$ that penalizes overly complex functions.

Following recent work on structured prediction under constraints [10], we solve the inference problem $\operatorname{argmax}_{y'} F(x, y')$ using an off-the-shelf constraint programming solution. Specifically, we leverage MiniZinc [11], a constraint programming language that interfaces several state-of-the-art solvers and that allows to inject complex background knowledge – in the form of logical and algebraic constraints – into the inference (and learning) steps. For instance, in our equation recognition experiment, we encode equation validity by transforming the sequence of predicted labels into integers and then imposing a constraint that the left and right hand sides match, see Section 4.

3. Neural Constrained Structured Prediction

We propose Nester, a neuro-symbolic approach that upgrades the structured SVM pipeline with representation learning while offering support for complex hard and soft constraints. Nester combines two components: a neural network and a general-purpose constrained structured predictor [13] that acts as a “refinement” layer on top of the network. The network is not directly aware of the constraints: given an input $x \in \mathcal{X}$, its task is to guess an initial, raw prediction $\hat{y} \in \mathcal{Y}$. Furthermore, the network is free to predict distinct output variables (e.g., individual characters in an equation) independently. The constrained structured predictor then takes the input $x \in \mathcal{X}$, the network’s prediction $\hat{y} \in \mathcal{Y}$, and a set of hard and soft constraints, and outputs a final prediction $y \in \mathcal{Y}$, as illustrated in Figure 1 for handwritten equation recognition. The hard constraints encode validity requirements that the output has to satisfy (e.g., the output equation must be algebraically valid). The soft constraints are weighted constraints that measure how good the output structure is based on the input, network output and refined output, and whose weights are learned during training. The constrained structured predictor is in charge of grouping the neural predictions while enforcing the constraints, thus correcting potential mistakes and inconsistencies.

The refinement step is performed by solving the following constraint program:

$$\operatorname{argmax}_{y \in \mathcal{Y}} \langle \mathbf{w}, \phi(x, y) \rangle + \langle \mathbf{w}_\rho, \phi_\rho(x, \hat{y}, y) \rangle + w_\delta \delta(y, \hat{y}) \quad \text{s.t.} \quad \hat{y} = g(x; W) \quad (1)$$

where the set of candidates \mathcal{Y} is defined by the hard constraints; $g(x; W)$ is a neural network; $(\mathbf{w}, \mathbf{w}_\rho, w_\delta)$ and W are the parameters of the constrained structured predictor and of the network, respectively; and $\phi(x, y)$, $\phi_\rho(x, \hat{y}, y)$ and $\delta(y, \hat{y})$ are features. More

specifically, $\phi(x, y)$ are the prediction features that, analogously to the ones used in standard structured prediction models [2], help predicting the right output from the input. $\phi_\rho(x, \hat{y}, y)$ are the refinement features and are designed to help the structured predict to spot the mistakes made by the network, by relating input, output and network predictions. Finally $\delta(y, \hat{y})$ measures the difference between the raw and refined predictions and (depending on w_δ) it keeps the refined prediction close to the network’s guess.

In Nester, learning can very intuitively be broken up into three steps: i) Bootstrapping the model by pre-training the neural network or by loading a pre-trained model, ii) Freezing the network and training the constrained structured prediction model based on its predictions, and iii) Fine tuning the two components end-to-end. This last step is worth discussing in detail. Structured SVMs are typically learned with cutting planes [14] or block-coordinate Frank-Wolfe [15]. In contrast, Nester uses stochastic sub-gradient descent [16], which works by minimizing the structured hinge loss $L(\mathbf{w}; x_i, y_i, y_i^*) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \langle \mathbf{w}, \phi(x_i, y_i^*) - \phi(x_i, y_i) \rangle$, where $y_i^* = \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y_i, y) - \langle \mathbf{w}, \phi(x_i, y_i) - \phi(x_i, y) \rangle$. Given y_i^* , the algorithm simply computes a sub-gradient $\nabla_{\mathbf{w}} L$ and then performs a descent step. In contrast to the alternatives, stochastic sub-gradient descent allows Nester to back-propagate the gradient of the loss through the whole model. This is achieved by chaining the gradients of L with respect to \hat{y} and the gradient of the neural network output with respect to its weights. For the last layer of the network with weights W_K : $\nabla L = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_K}$. Gradients for the other layers can be obtained by standard backpropagation through the network.

3.1. Neural Constrained Sequence Prediction

In the following we focus on sequence prediction, a very common structured output task. Here, outputs $y = (y_1, \dots, y_m)$ are variable length sequences of elements from an alphabet of q possible symbols $\{s_k\}_{k=1}^q$. For each element, a vector of l input features is typically available. We indicate with $x_{e,i}$ the i -th feature of the e -th element of the input sequence. As prediction features we employ standard features for correlating input and output variables commonly used in e.g. conditional random fields [2] for sequence prediction:

$$\phi_{i,k}(x, y) = \sum_{e=1}^m x_{e,i} \cdot \llbracket y_e = s_k \rrbracket \quad \phi_{k_1, k_2}(x, y) = \sum_{e=1}^{m-1} \llbracket y_e = s_{k_1} \wedge y_{e+1} = s_{k_2} \rrbracket \quad (2)$$

where $\llbracket \cdot \rrbracket$ evaluates to 1 if the argument is true and 0 otherwise. $\phi_{i,k}(x, y)$ encodes emission features, correlating the appearance of a specific input feature (e.g., a pixel inside input images) with each of the emitted symbols. Here i ranges over input features and k over output symbols. $\phi_{k_1, k_2}(x, y)$ encodes transition features, correlating the appearance of consecutive symbols inside the output sequence. Here k_1, k_2 ranges over all the possible combinations of couples of symbols. We define refinement features so as to correlate the value of each input feature with the overriding of neural network predictions by the refinement layer: $\phi_{i,k}(x, \hat{y}, y) = \sum_{e=1}^m x_{e,i} \cdot \llbracket y_e = s_k \wedge \hat{y}_e \neq s_k \rrbracket$, where i ranges over input features and k over output symbols. By learning appropriate weights for them, the structured predictor can learn to fix the most common mistakes made by the neural network. Finally, we define $\delta(y, \hat{y})$ as the Hamming distance between y and \hat{y} , that is, $\delta(y, \hat{y}) = \sum_{e=1}^m \llbracket y_e \neq \hat{y}_e \rrbracket$.

4. Experiments

We tested our technique on the handwritten equation recognition task described in [10]. All our experiments are implemented using Tensorflow and Pyconstruct [10]. MiniZinc [11] was used as constraint programming engine and Gurobi (<https://www.gurobi.com/>) as the underlying constraint solver.

Setting Training examples represent equations are all of the form $a + b = c$, where a, b, c are arbitrary positive integers (of a predefined maximum number of digits). Inputs are variable length sequences of 9×9 black and white images and the corresponding outputs are sequences of symbols that include the digits from 0 to 9, + and =. The dataset contains 10,000 valid equations from the ICFHR'14 CROHME competition data and is used with a 80 : 20 train-test split. Results are reported as learning curves by dividing the training set into 20 chunks of increasing size.

Neural model As highlighted in Figure 1, we use a CNN to predict the symbol of each image in the sequence independently. The CNN is composed of two convolutional layers with 3×3 filters and ReLU activation, each followed by a 2×2 max-pool layers, then a 128 dense layer with dropout regularization (0.5 probability), and finally a softmax layer with 12 outputs, one for each symbol. The CNN is trained using Adam with a cross entropy loss. The left plot in Figure 2 reports the relative misclassification error of the CNN over the test set, i.e. the percentage of sequences for which the network made at least one mistake. We subdivide the errors into syntactic errors, for which the prediction is not properly formatted according to the template $a + b = c$, and semantic errors, i.e. well-formatted predictions for which a plus b does not equal c (errors which are syntactically and semantically correct were a negligible minority) The shaded areas in the figure show how the different types of errors contribute to the total. While the total number of errors decreases with increasing training set size, the proportion between syntactic and semantic errors settles around 33% to 67%, indicating that semantic errors are generally harder to correct. To make sure that these errors are not simply due to the fact that the CNN predicts each output symbol independently, we also experimented with a recurrent architecture stacking an LSTM network over the CNN outputs. Results are reported in the middle plot of Figure 2. The network makes more errors overall, possibly due to the increased number of trainable parameters. Relatively speaking, though, the CNN + LSTM network learns very quickly how to fix syntactic errors, faster than the CNN. Yet, semantic errors prove much more challenging, and end up being the large majority of the errors of the network. These results show that even highly expressive neural architectures can fail to go beyond a syntactic level of understanding, especially when limited data is available.

Neural constrained model As shown in Figure 1, Nester applied to handwritten equation recognition stacks a constrained structured layer on top of the independent CNN predictions. As constrained structured layer (“CST” from now on) we used an enhanced

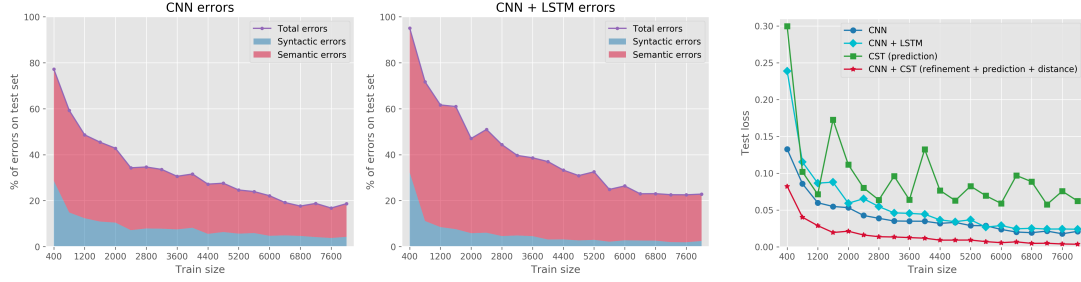


Figure 2: Misclassification error for the CNN (left) and the CNN + LSTM (middle). Right: structural loss (Hamming distance) for the CNN + CST model and the competitors. Best viewed in color.

version of the constrained model from [10]. Our model extends the original one by taking the prediction of the neural network \hat{y} as input and combining the prediction features with the refinement features and the Hamming distance as described in Section 3.1 (see also Figure 1). The constrained structured model is defined in the MiniZinc language. Syntactic constraints over the output sequence enforce a single $+$ and a single $=$ and the fact that $+$ should come before $=$. Semantic constraints are enforced by first combining sequences of consecutive digits into corresponding numbers according to positional rules, and then enforcing $a + b = c$ over the resulting numbers. Prediction and refinement features as well as Hamming distance between neural network output and final output are also encoded in MiniZinc. See [17] for further details. We perform a warm start training of the CNN as in the previous paragraph, and train the CST model using the structured SVMs method with stochastic subgradient descent [16] for one epoch over the training set. The structured loss used in this task is the Hamming distance between the predicted sequence of symbols and the true sequence. Figure 2 (right) shows the test loss of our method (CNN + CST) together to those of the single components, i.e., the CNN and the CST in the original variant [10], and the CNN + LSTM cascade. The general trend is rather clear, the CNN has always an edge over the CST, yet their combination always performs better than both. As expected, the gap between the CNN and the combined model is maximal for the smallest dataset, but it remains clearly evident when the curves start to level off (82.9% relative error reduction at the last iteration). The loss of the CNN + LSTM model is overall slightly higher than the one of the CNN model, consistent with what happens for the misclassification error (see the middle plot of Figure 2). An ablation study shows that the distance feature alone already improves over the plain CNN model (29.6% relative error reduction at the last iteration), and that all features contribute to the overall performance. See [17] for details.

Inference and training time of the CNN + CST model is mostly bounded by the inference time of CST, which in our experiment is below one second using Gurobi. The inference time of the CNN is negligible. While the CNN is much more scalable, the combined model still has low latency, allowing its application to many domains for which sample complexity is more a important aspect than scalability. Nevertheless, we are looking at integrating approximation techniques into Nester as future work [18].

Comparison with DeepProbLog DeepProbLog is an expressive neuro-symbolic framework that combines probabilistic-logic programming with neural predicates and has been used to learn, e.g., to add digits represented as MNIST images [9]. The equation recognition task we address here, however, is more complex as the numbers have a variable number of digits and the position of the operators is not known a priori. Modelling this problem with DeepProbLog requires the use of non-deterministic predicates, which are extremely expensive. Indeed, even after optimizing the encoding¹, equation recognition turned out to be too memory hungry to be even executed by DeepProbLog.

5. Related work

Traditional approaches to structure prediction acquire a scoring function over candidate input-output pairs [2, 3] and admit prior knowledge in the form of ad-hoc constraints on the output [19, 20]. These ideas have been extended to a fully declarative setup in which the constraints are specified using a language like MiniZinc [10, 13, 21]. These approaches rely on manually designed features, which is often sub-optimal compared to representation learning. Nester is a neuro-symbolic upgrade of these techniques.

Deep structured output prediction has been tackled with energy-based and search-based models. Energy-based models like deep value networks [22], structured prediction energy networks (SPENs) [4] and input convex neural networks [5] implement the scoring function using a neural network and use gradient ascent to perform inference, but offer no support for discrete outputs and prior knowledge defined by constraints. Lee et al. [23] address this issue by casting the constrained inference of SPENs into an instance-specific learning problem with a constraint-based loss function. Unlike Nester, this method is not guaranteed to satisfy all hard constraints. Search-based techniques, on the other hand, sequentially build an output structure by repeatedly applying an auto-regressive models to predict the next piece given the already predicted pieces. These approaches however lack support for long-range constraints without an expensive back-tracking procedure.

Most attempts to combine deep networks and reasoning focus on forcing the network to learn weights that ultimately produce predictions satisfying the constraints. A popular line of research integrates constraints into the objective function using fuzzy logic [6, 7]. These approaches cannot guarantee that the predicted outputs satisfy the hard constraints. The same is true for the semantic loss, which encourages the output of a neural network to satisfy given constraints with high probability [8]. Furthermore, these approaches are usually restricted to Boolean variables and logical constraints. Perhaps the most expressive approach in this class is DeepProbLog, which integrates probabilistic-logic programming and neural predicates in a principled manner. In contrast to Nester, DeepProbLog can answer probabilistic queries other than MAP inference, but it does not always scale to complex structured-output predictions tasks, as discussed in Section 4.

¹The main developer of DeepProbLog helped us in trying to optimize the encoding.

6. Conclusion

We presented Nester, an approach to structured output prediction that combines neural networks for input processing and representation learning with constraint programming for high-level reasoning. Our preliminary experiments on a challenging constrained structured prediction task, namely handwritten equation recognition, show that CNN and LSTMs learn quickly to correct syntactic errors but fail to grasp and avoid semantic errors. In contrast, Nester improves recognition performance over both the neural network and the constrained structured model on their own, especially with smaller training sets, while generating predictions that are consistent by design.

A key assumption underlying our method is that the constraints themselves are correct. We plan on investigating settings in which this is not the case, for instance because of modeling mistakes on the designer’s end or because the background knowledge has been acquired semi-automatically [24]. One option is to encode the background knowledge in terms of soft constraints. This is however non-trivial and left to future work.

Acknowledgments

We are grateful to Carlo Nicolò and Edoardo Battocchio for running preliminary experiments and to Robin Manhaeve for support on DeepProbLog. The research of ST and AP was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

References

- [1] G. Bakir, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, S. Vishwanathan, Predicting Structured Data, 2007.
- [2] J. Lafferty, A. McCallum, F. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data, in: ICML, 2001.
- [3] I. Tsochantaridis, T. Hofmann, T. Joachims, Y. Altun, Support vector machine learning for interdependent and structured output spaces, in: ICML, 2004.
- [4] D. Belanger, A. McCallum, Structured prediction energy networks, in: ICML, 2016.
- [5] B. Amos, L. Xu, J. Kolter, Input convex neural networks, in: ICML, 2017.
- [6] M. Diligenti, M. Gori, V. Scoca, Learning efficiently in semantic based regularization, in: ECML-PKDD, 2016.
- [7] I. Donadello, L. Serafini, A. d’Avila Garcez, Logic tensor networks for semantic image interpretation, in: IJCAI, 2017.
- [8] J. Xu, Z. Zhang, T. Friedman, Y. Liang, G. Van den Broeck, A semantic loss function for deep learning with symbolic knowledge, in: ICML, 2017.
- [9] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, L. De Raedt, DeepProbLog: Neural probabilistic logic programming, in: NeurIPS, 2018.
- [10] P. Dragone, S. Teso, A. Passerini, Pyconstruct: Constraint programming meets structured prediction, in: IJCAI, 2018.

- [11] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, G. Tack, Minizinc: Towards a standard cp modelling language, in: CP, 2007.
- [12] Y. LeCun, S. Chopra, R. Hadsell, F. J. Huang, et al., A tutorial on energy-based learning, in: Predicting Structured Data, MIT Press, 2006.
- [13] P. Dragone, S. Teso, A. Passerini, Constructive preference elicitation, *Frontiers in Robotics and AI* 4 (2018).
- [14] T. Joachims, T. Finley, C. Yu, Cutting-plane training of structural svms, *Machine Learning* (2009).
- [15] S. Lacoste-Julien, M. Jaggi, M. Schmidt, P. Pletscher, Block-coordinate frank-wolfe optimization for structural svms, in: ICML, 2013.
- [16] N. Ratliff, J. Bagnell, M. Zinkevich, (approximate) subgradient methods for structured prediction, in: AISTATS, 2007.
- [17] P. Dragone, S. Teso, A. Passerini, Neuro-symbolic constraint programming for structured prediction, *CoRR* abs/2103.17232 (2021).
- [18] J. Mandi, T. Guns, Interior Point Solving for LP-based prediction+ optimisation, in: NeurIPS, 2020.
- [19] T. Kristjansson, A. Culotta, P. Viola, A. McCallum, Interactive information extraction with constrained conditional random fields, in: AAI, 2004.
- [20] E. Fersini, E. Messina, G. Felici, D. Roth, Soft-constrained inference for named entity recognition, *Information Processing & Management* (2014).
- [21] P. Dragone, G. Pellegrini, M. Vescovi, K. Tentori, A. Passerini, No More Ready-Made Deals: Constructive Recommendation for Telco Service Bundling, in: RecSys, 2018.
- [22] M. Gygli, M. Norouzi, A. Angelova, Deep value networks learn to evaluate and iteratively refine structured outputs, in: ICML, 2017.
- [23] J. Lee, M. Wick, J. Tristan, J. Carbonell, Enforcing output constraints via sgd: A step towards neural lagrangian relaxation, in: NIPS, 2017.
- [24] L. De Raedt, A. Passerini, S. Teso, Learning constraints from examples, in: AAI, 2018.