A Scalable Technique for Weak-Supervised Learning with Domain Constraints

Abstract

We propose a novel scalable end-to-end pipeline that uses symbolic domain knowledge as constraints for learning a neural network for classifying unlabeled data in a weak-supervised manner. Our approach is particularly well-suited for settings where the data consists of distinct groups (classes) that lends itself to clustering-friendly representation learning and the domain constraints can be reformulated for use of efficient mathematical optimization techniques by considering multiple training examples at once. We evaluate our approach on a variant of the MNIST image classification problem where a training example consists of image sequences and the sum of the numbers represented by the sequences – e.g., (, , 9), and show that our approach scales significantly better than previous approaches that rely on computing all constraint satisfying combinations for each training example.

1 Introduction

Integrating logical reasoning and machine learning is one of the main challenges of AI. Doing so can lead to systems that can solve more complex problems, learn from less data, comply with domain knowledge, and/or have better performance [1, 2]. In particular, incorporating domain knowledge as symbolic constraints during ML training has shown to improve the accuracy of ML predictions by learning an ML model that tries to enforce the constraints as much as possible [3, 4]. These techniques are especially promising when training data is limited or the constraints are so complex that a pure ML model won't be able to detect or enforce them even from lots of training data. However, several of these techniques compromise on either scalability, accuracy, expressiveness, or ease of use. To the best of our knowledge, there is no single approach that can be used for a multitude of industry use cases, which often have complex domain constraints and require high scalability.

In this work, we develop a scalable weak-supervised learning technique to incorporate symbolic domain knowledge constraints into neural networks. In the same category, some recent approaches handle constraints by incorporating probabilities from neural networks into logic programs. For instance, models that incorporate constraints expressed in ProbLog [5] and Answer Set Programs have been proposed in [6] and [3] respectively. Some other approaches such as [7] and [8] use stochastic grammar based logic programs. However, all these approaches don't scale well when the complexity of the problem increases due to the limitations arising from model grounding. The grounding of a constraint is the computation of all satisfying assignments of the constrained variables, which can lead to intractable combinatorial blow-up in cases with a large number of constrained variables. This problem, known as the grounding bottleneck, rises in logical reasoning domains, such as statistical relational AI (StarAI) and Answer Set Programming (ASP) [9, 10]. While the techniques presented in [6, 3, 7] are highly expressive and accurate, they rely on naively iterating through all possible output combinations for each training example in order to compute the satisfying output combinations. For instance, for (, , 5), they would iterate over 10^2 combinations in order to compute the satisfying combinations $(0, 5), (1, 4), (2, 3), (3, 2), (4, 1), (5, 0)$. While expressiveness can be traded for scalability – DeepStochLog [7] vs. DeepProbLog [6] –, the combinatorial blow-up still poses a major hurdle for more complex problems; with n images per example, the number of possible output combinations grows exponentially (10^n).

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Our key contribution is an end-to-end pipeline that avoids the grounding bottleneck and scales to state-of-the-art results on a challenging neuro-symbolic problem. Our approach is based on the insight that in some cases (a) the data consists of distinct groups (e.g., digit images) that are partially recoverable through clustering-friendly representation learning, and (b) the constraint (e.g., addition) is the same across all training examples and the properties of the constraint are known and can be exploited to our advantage.

In this work, we focus on a variant of the MNIST image classification problem where the image labels are not given, but addition constraints, sequence of images, and their numerical sums are used instead. The dataset consists of a set of n examples. Each example is a sequence of $w \times h$ MNIST images and an integer s where w represents the number of digits in a number and h is the number of numbers in the example whose sum is equal to s. Variables w and h refer to the *width* and *height* of an example. A practical example of such a use case is in information extraction from hand-filled forms such as financial documents (e.g., loan and tax forms) where certain "total" fields are pre-filled in typesetting, while other fields leading up to the total are handwritten information from the user.

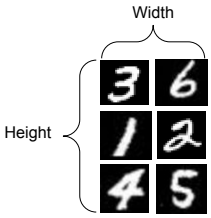

Figure 1: An example with $h = 3, w = 2$, and $s = 93$.

For this problem, the state-of-the art solution [7] has been shown to work for $w \leq 4, h = 2$. We show that our approach scales in both width and height as our model achieves between 92% and 97% accuracy for $w \leq 10, h \leq 6$ with the training time being linear.

2 Method

Overall, our solution can be broken down into the four main steps as presented in Figure 2.

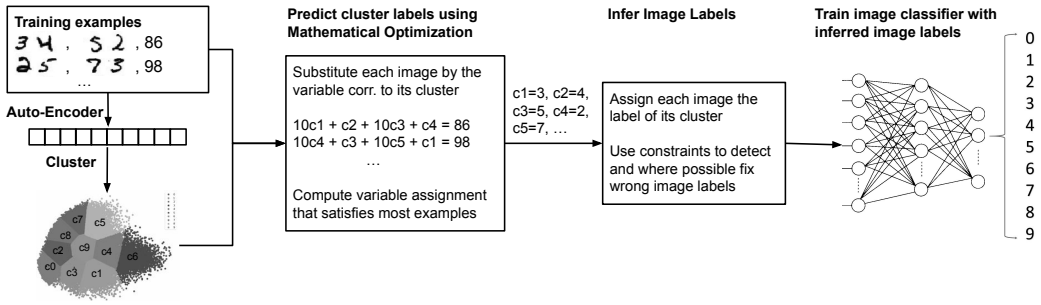

Figure 2: The main steps of our proposed method.

1. Autoencoder based clustering: We first pre-train a fully connected symmetric autoencoder with dense layer dimensions $[500, 500, 2000, 10]$ for 300 epochs, similar to the one in [11], then use the weights of the encoder to cluster with k-means using k-means++ initialization and 10 clusters.

2. Cluster label prediction: An example of width w, height h, and sum s can be represented as $\sum_{i=1}^{h} \sum_{j=1}^{w} v(img_{i,j}) \times 10^{w-1-j} = s$, where $v(img_{i,j})$ represents the variable assigned to the cluster that contains the jth image in the ith number of the training example. A 100% accurate clustering would assign each cluster $c_i, i \in [0, 9]$ a variable v_i, from which, one could solve a system of linear equations to determine the values of each v_i. Since we don't start with a clustering of 100% purity, in this step, we formulate the problem as an integer linear program using *CVXPY* [12, 13]. We split the training examples into batches of size 100, solve the optimization problem for each batch using the L^1-norm as the objective function, and determine the overall result as the batch result that satisfies the most training examples over all batches. As a result, this step assigns each cluster an integer from $[0, 9]$.

3. Image label inference: The cluster labels computed in the previous step give the initial labeling for each image in the training dataset. The labeling can be at most as accurate as the clustering accuracy. The goal of this step is to improve image labeling based on rule-based inference. Currently, this step consists of only one rule based on the fact that a variable in a linear equation can be resolved if all

$Correct \leftarrow \{\}$
for $1 \leq radius \leq 5$ **do**
  $NewCorrect \leftarrow fromCentroid(radius)$
  $Correct \leftarrow Correct \cup NewCorrect$
  $Changed \leftarrow True$
  while $Changed == True$ **do**
    $Changed \leftarrow fixLabels(Correct)$
  end while
end for

(a) The main loop

$Changed \leftarrow False$
for all $Ex \in TrainingExamples$ **do**
  $Unres \leftarrow getUnres(Correct, Ex)$
  if $|Unres| = 1$ **then**
    $label \leftarrow resolve(img)$
    $Correct \leftarrow Correct \cup \{img\}$
    $Changed \leftarrow True$
  end if
end for
return $Changed$

(b) fixLabels Algorithm

Figure 3: The image label improvement algorithm.

other variables are known. We use an iterative process by which we increasingly resolve variables in the system of equations with every iteration. Since we do not know which images are correctly clustered, we start with the assumption that images closer to their respective cluster's centroid have a higher chance of being correctly clustered, and thus, are correctly labeled.

The iterative process (Fig. 3a) goes as follows: Start with an empty set of correctly clustered images. At every iteration $r \in [1, 5]$, get the images within radius r from their centroids as correctly labeled using the method *fromCentroid*. Considering this as the new set of correctly labeled images, infer new (correct) image labels. The main loop runs as long as we can infer new image labels as correct labels. The inference of new correct labels is done in the *fixLabels* algorithm in Fig. 3b. For each training example, *fixLabels* computes the set of unresolved images, i.e., the images that have not been identified as correct thus far. If an example has only one unresolved image, then *resolve* computes its correct label by solving for it, and adds the image to the set of correct images.

4. Classification: The final step is to train a CNN-based classifier using the final inferred image labels from Step 3. We use an architecture similar to the one in [14], and train for 10 epochs.

3 Results

Fig. 4 shows the accuracy and training time for varying widths and heights, where the training examples used for every $w \times h$ combination use all 60K images in the MNIST training dataset exactly once. The reported accuracy is the accuracy of the MNIST classifier (Step 4) using the inferred image labels (Step 2) as described in Section 2. The reported time is the total time for all 4 steps of the pipeline[1]. All our experiments are run on a MacBook Pro 2021 (Apple M1 Max, 64GB).

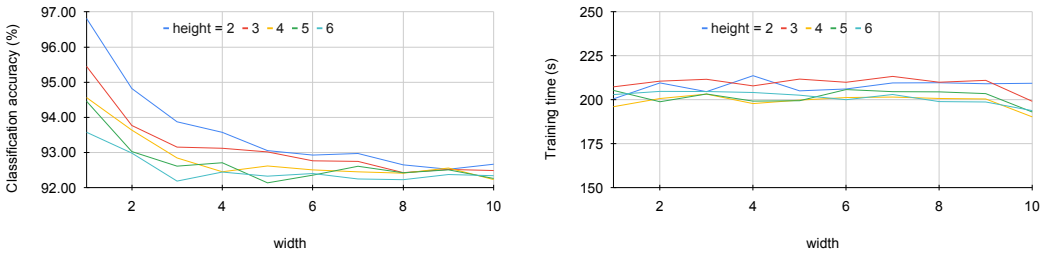

Figure 4: Classification accuracy % (left) and total training time in seconds (right) for varying $w \times h$ combinations.

As previously mentioned, DeepstochLog [7], the state-of-the-art solution so far, can handle examples with $h = 2, w \leq 4$, and scales better than earlier approaches [6, 3], both of which solve $h = 2, w \leq 2$ and timeout for $h = 2, w > 3$. As shown by the results in Fig. 4, our approach not only scales horizontally ($w > 4$), but also vertically ($h > 2$). Furthermore, our approach doesn't blow up in the

[1]The total training time we report doesn't include the autoencoder training time. This step takes roughly 30 minutes. We trained the autoencoder once and used the same encoder weights for all $w \times h$ runs.

total training time as w and h grow. The reason for higher classification accuracy for smaller w, h combinations is that in these cases the image inference algorithm has a higher chance of having an example with only one unresolved image and can infer its label, thus increasing the final image label accuracy. For larger w, h combinations, the classification accuracy between 92% and 93% indicates that the optimizer finds the correct cluster labels as our clustering component has about 91.3% purity.

Note that our classification accuracy not directly comparable to DeepStochLog's addition accuracy reported in [7]. One way to be able to compare the addition accuracy could be by augmenting our pipeline with a component for performing the addition on the output of our classifier. With this architecture we would need approx. 99% classification accuracy to achieve the addition accuracy similar to that of DeepStochLog. Improving the classification accuracy is a planned future work as discussed in Section 4.

One reason why we can avoid the combinatorial blow-up is attributed to the nature of the problem. The addition constraint is linear and can be solved efficiently for a set of examples if they can be formulated using the same set of variables. In our case, this is made possible by the clustering step which essentially reduces the number of variables from 60k (one for each image) to 10. Furthermore, our approach illustrates the trade-off between expressiveness and the scalability of the language. Similar to how DeepStochLog scales better than DeepProbLog by using stochastic logic programs that are less expressive than probabilistic logic programs used by DeepProbLog our approach scales even further, but can only work for problems that fit our "cluster-then-optimize" paradigm.

4 Discussion & Future Work

Currently, our classification accuracy depends strongly on the clustering purity since the image label accuracy after the optimization step is upper bounded by it. This is why we currently use the autoencoder as a pre-processing step which gives us approx. 91.3% clustering purity. One major focus for our future work is to make the pipeline more robust wrt. the representation learning step. One way to achieve this may be to keep track of the previous (wrong) labels and the new (correct) ones in the image label inference, and use this information during the training of the classifier or as feedback to prior steps.

Currently, the batch size in the optimization step is fixed to 100. This may not be the best choice for larger combinations, say, $(w = 12, h = 12)$ and the optimizer may not find the correct result. Therefore, we plan to dynamically compute the batches of examples (and their size) to increase the performance. Furthermore, adding more inference rules or heuristics to the image label inference step — currently it relies on one simple rule — would help achieving higher image label accuracy for more cases.

In our experiments, similar to [7, 3], we have used the minimum number of training examples for every $w \times h$ combination such that the MNIST training dataset (60K images) is used entirely where every image appears exactly once across all examples. However, we have preliminary results that show that for small $w \times h$ combinations, increasing the number of training examples (i.e., allowing images to appear more than once in the training dataset) leads to higher classification accuracy. For instance, for $h = 2, w = 5$, with a minimum number of examples ($n = 6000$), we achieve 93.05% classification accuracy, where as increasing n to 18000 reaches 98% accuracy. This is not surprising since having an image appear more than once gives the image label inference algorithm a higher chance of correctly resolving its label.

In this work, we have presented a scalable technique for weak-supervised learning using domain constraints instead of labels, and evaluated it on the MNIST dataset with the addition constraint. We have shown that our approach scales better than the most recent approaches with respect to both width and height, as well as in terms of training time. Our approach is applicable to problems with numerical and logical constraints where the input can be clustered and multiple examples can be used together to efficiently evaluate the constraints. To achieve a more general implementation, we plan to tackle the handwritten formula problem as defined in [15] as our next use case. Furthermore, to illustrate the combination of both numerical and logical constraints, we also plan to tackle the Sudoku problem as defined in [3].

References

[1] Don Monroe. Neurosymbolic ai. Communications of the ACM, 65(10):11–13, 2022.

[2] Pascal Hitzler. Neuro-symbolic artificial intelligence: The state of the art. 2022.

[3] Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In 29th International Joint Conference on Artificial Intelligence (IJCAI 2020), 2020.

[4] Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, et al. Logical neural networks. arXiv preprint arXiv:2006.13155, 2020.

[5] Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. Mach. Learn., 100(1):5–47, 2015.

[6] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. Advances in Neural Information Processing Systems, 31, 2018.

[7] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 10090–10100, 2022.

[8] William W Cohen, Fan Yang, and Kathryn Rivard Mazaitis. Tensorlog: Deep learning meets probabilistic databases. Journal of Artificial Intelligence Research, 1:1–15, 2018.

[9] Efthymia Tsamoura, Víctor Gutiérrez-Basulto, and Angelika Kimmig. Beyond the grounding bottleneck: datalog techniques for inference in probabilistic logic programs. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 10284–10291, 2020.

[10] Bernardo Cuteri, Carmine Dodaro, Francesco Ricca, and Peter Schüller. Overcoming the grounding bottleneck due to constraints in asp solving: Constraints become propagators. In IJCAI, pages 1688–1694, 2020.

[11] Chengwei. How to do unsupervised clustering with keras? `https://www.dlology.com/blog/how-to-do-unsupervised-clustering-with-keras/`, 2018.

[12] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. Journal of Machine Learning Research, 17(83):1–5, 2016.

[13] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. Journal of Control and Decision, 5(1):42–60, 2018.

[14] Jason Brownlee. How to develop a cnn for mnist handwritten digit classification. `https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/`, 2019.

[15] Qing Li, Siyuan Huang, Yining Hong, Yixin Chen, Ying Nian Wu, and Song-Chun Zhu. Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning. In International Conference on Machine Learning, pages 5884–5894. PMLR, 2020.