A Unified Formulation and Analysis of Learning Algorithms with Output Constraints.

Mooho Song[1] and Jay-Yoon Lee[1]

[1] Graduate School of Data Science, Seoul National University, Seoul, South Korea, {anmh9161, lee.jayyoon}@snu.ac.kr

Abstract

Many studies indicate injecting human knowledge by reducing output constraints during training can improve model performance and reduce constraint violations. While there have been several attempts to compare different existing algorithms under the same programming framework, nonetheless, there has been no previous work that categorizes learning algorithms with output constraints in a unified manner. Our contributions are as follows: (1) We categorize the previous studies based on three axes: type of constraint loss used, exploration strategy of constraint-violating examples, and integration mechanism of learning signals from main task and constraint. (2) We propose new algorithms to integrate the information of main task and constraint injection, inspired by continual-learning algorithms. (3) Furthermore, we propose the $H\beta$-score as a metric for considering the main task metric and constraint violation simultaneously. To provide a thorough analysis, we examine all the algorithms on three NLP tasks: natural language inference (NLI), synthetic transduction examples (STE), and semantic role labeling (SRL).

Keywords— *Constraint injection, Weakly supervised learning, Neuro-symbolic AI*

I. Introduction

Neural network (NN) models typically learn from data in the form of (input, output) pairs, which can sometimes conflict with human knowledge. Previous research has shown that incorporating human knowledge into NN models by reducing relevant constraint violations during training can improve model performance and reduce these violations [6, 10, 8, 12, 14]. The relationship between constraints and the task itself can be seen as a relationship between a sub-task and the main task: the main task aims for accurate predictions, while the sub-task focuses on producing constraint-satisfying outputs.

Various studies have investigated constraint injection during training [6, 10, 8], often by adding a constraint-related loss to the existing supervised loss. However, how to formulate this constraint loss and its incorporation into the overall loss function varies across different studies and applications.

The first goal of this work is to provide a unified analysis of existing methods under a single mathematical formulation. Previous efforts to compare different constraint injection methods [12] focused on performance, whereas our study aims to formalize these methods to understand key success factors. For instance, while the primal-dual algorithm [10] showed positive results with dynamic weight updates for constraint-loss, it was only tested with one loss type, making it unclear whether the positive results were due to the loss type or the dynamic weight update mechanism. Investigating the combination of different components of constraint injection under a unified formulation is necessary.

The second goal is to propose new learning algorithms that integrate constraints within the suggested unified formulation. A common approach involves handling a constraint loss term, $\lambda \times \mathscr{C}$, where $\mathscr{C}$ denotes the constraint loss and λ is a fixed scalar representing its weight. [10] introduced an algorithm that dynamically controls λ, starting at 0 and adjusting it during training. This approach, which gradually increases λ based on constraint degrees, stands out from methods using a fixed λ. However, there has been insufficient research on integrating supervised data learning signals and constraint information.

Is a monotonically increasing λ necessary? Can λ be updated considering both supervised learning and constraint injection? Inspired by continual learning, this paper proposes a new approach that considers both supervised and constraint losses during gradient updates. This method takes into account the progress of both tasks, offering a new perspective on simultaneous learning. Experiments demonstrate that our approach outperforms other learning algorithms in various scenarios.

II. Related work & Unified formulation of previous work

In this section, we categorize the previous studies on injecting constraints during training time based on three dimensions: type of mathematical expression used for constraint loss (§A.), exploration strategy of constraint-violating examples (§B.), and mechanism for integrating losses from the main task and the constraint injection task (§C.). A common approach in machine learning is to define a loss function and employ optimization algorithms to update model parameters in the direction of minimizing that loss. When the labeled data $\{(x_i, y_i)\}_{i=1}^{N}$ is given, the goal of typical supervised learning is to solve the following optimization problem:

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \mathscr{L}(x_i, y_i; \theta), \text{ or simply } \min_{\theta} \mathscr{L}(\theta) \tag{1}$$

, where $\mathscr{L}(x, y; \theta)$ is the standard supervised loss function for the task we are learning.

Most of the existing constraint injection methods, while differing in specific formulations, inject the constraint information by expanding the loss function in a following manner:

$$\mathscr{T}(\theta) = \lambda_1 \mathscr{L}(\theta) + \lambda_2 \cdot \mathscr{C}(\theta) \tag{2}$$

, where $\mathscr{C}(\theta)$ is a loss related to the constraint, and λ_i's are fixed weights, We can further generalize the equation (2) as follow:

$$\nabla\mathscr{T}(\theta) = \Lambda^{sup} \cdot \nabla\mathscr{L}(\theta) + \Lambda^{con} \cdot \nabla\mathscr{C}(\theta) \tag{3}$$

, where Λ^{sup}, Λ^{con} are usually scalar matrices. $\mathscr{C}$ reflects the human knowledge the algorithm wants to inject and is typically computed without the true label. To be more precise, for some hard constraints on output labels, $\mathscr{C}(\theta)$ is computed via output $f_\theta(x)$ given some unlabeled input x. Injecting more than one hard constraint is also possible by expanding $\Lambda^{con} \cdot \nabla\mathscr{C}(\theta)$ to $\sum_{i=1}^{K} \Lambda_i^{con} \cdot \nabla\mathscr{C}_i(\theta)$ in equation (3) , where K is a number of constraints.

To unify and distinguish different algorithms that learn with constraints, we focus on how $\mathscr{C}(\theta)$ is formulated (§A.), how constraint-violating examples are explored (§B.), and how Λ^{sup}, Λ^{con} (in equation (3)) are determined(§C.).

A. *Type of constraint loss*

Type of constraint loss is related to how the violation of constraints can be transformed into the form of a differentiable loss function $\mathscr{C}(\theta)$ in equation (2), which we will refer to as the *constraint loss*. How to convert symbolic constraints into a differentiable loss function can be broadly categorized into two approaches: Probabilistic Soft Logic (PSL) and REINFORCE.

Probabilistic Soft Logic (PSL) PSL [3] is associated with expressing logic in terms of probabilities, and research utilizing PSL measures the degree of constraint violation in the logic itself, employing it as a loss. Gödel, product, Łukasiewicz logics can be primarily used to soften logic [9, 10, 6]. Generally, PSL is not suitable for representing all types of hard constraints, since it must be converted to linear constraints before they can be directly applied [12]. Section §B. is an example task illustraing the challenges in applying PSL.

REINFORCE In contrast, studies employing the REINFORCE [13] evaluate whether (or to what degree) the model's output violates constraints. Constraint injection research during training time using REINFORCE can be further classified into two ways depending how the reward is formulated. A simple method is to assign binary reward (e.g.: $\{1, 0\}$) when the model satisfies or violates the constraints [1]. This simple method with binary reward only considers whether the constraint is satisfied or not. On the other hand, one could make the reward more fine-grained by measuring the degree of constraint violation and assigning real-valued rewards related to it [8]. A significant feature of REINFORCE is that the determination of constraint loss relies solely on the rule of assigning rewards based on the presence or absence of constraint violation in sampled examples, regardless of the specific constraint. This differs from PSL in that it does not require intricate implementations for generating constraint loss. However, due to the need for sampling procedures, the computational cost is generally higher than when using PSL [12].

To summarize, PSL and REINFORCE are mainly used approach to generate $\mathscr{C}(\theta)$ in eq.(2) to reduce expected constraint violation with following differences. PSL defines constraint violation as a continuous measure, while REINFORCE relies on the reinforcement learning paradigm to guide the model towards satisfying constraints. Specifically, the REINFORCE method is divided into two types based on the method of setting rewards: binary rewards and real rewards.

B. *Exploration of constraint-violating examples*

Exploration of constraint-violating examples can significantly impact the effectiveness and efficiency of constraint learning. The possible questions we have are as follows: Would it be better to explore the model's approximate output space? Would it be best to examine the model's best possible effort? Or would it be better to explore by considering all possible probability distributions? Theses are considered to determine the magnitude of the constraint loss $\mathscr{C}$. For example, in REINFORCE with $\{1, 0\}$ reward, the reward will be 0 if we only visit constraint-violating examples.

According to the questions posed above, exploration strategies are divided by three, each explained below: *sampling*, *argmax*, and *exhaustive*.

Sampling Sampling strategy involves drawing samples from the forward propagation results of the model f to examine different instances that violate constraints. As demonstrated by [1], this method commonly employs the REINFORCE algorithm to incorporate constraint violations into the loss function for the identified examples. The sampling strategy can be applied independently to all combinations for our other analysis axes, specifically concerning the type of constraint loss (§A.) and the integration mechanism of learning signals from main task and constraint (§C.).

Argmax (Top-1) Argmax (Top-1) strategy, constraint violation is assesed by choosing the combination with the highest probability from $f(x)$. Following greedy decoding process such as beam search or Vitrerbi decoding [8], it evaluates constraint violation for the decoded example. Similar to sampling, it evaluates constraint violation for the decoded example, but distinguishes itself by considering the most probable prediction at that moment without multiple samplings. Like the sampling strategy, the argmax strategy can also be applied independently to all combinations under our other axes of anlysis.

Exhaustive Exhaustive strategy considers probabilities of all output class and its combinations. It is prominently employed in research related to PSL [10, 6]. Since there is no sampling involved, it is computationally cost-effective rather than sampling strategy. When considering the type of constraint loss (§A.), our performance evaluation is exclusively conducted using PSL for the exhaustive strategy, excluding the REINFORCE in the constraint loss, as it would be impossible to consider all possible combinations in REINFORCE. Since the exhaustive strategy can only be applied for constraint loss type of PSL, exhaustive strategy cannot handle all of general type of constraints.

C. *Integration mechanism of learning signals from main task and constraint*

This section is related to the integration of main task and the constraint information. We categorize integration mechanisms of prior studies into *static* and *monotone* ($\lambda \uparrow$). Additionally, we introduce three new integration mechanisms based on the linear projection: *projection-sup*, *projection-con*, and *projection-both*, which will be discussed in section §iii.. We provide detailed explanations of these mechanisms below.

Static For constraint loss $\mathscr{C}$, a widely used approach incorporating $\mathscr{C}$ into the existing supervised loss term is to add $\lambda \cdot \mathscr{C}$ to the previous existing loss, where λ is a fixed positive real number [1, 6, 8, 9]. In this approach, the value of λ remains unchanged throughout the training process, serving as a constant multiplier that determines the relative influence of $\mathscr{C}$ in comparison to the main task loss $\mathscr{L}$ in eq.(2).

Monotone ($\lambda \uparrow$) On the other hand, the study by [10] deviates from this by not using a fixed λ. Instead, it initiates training with λ starting from 0 and progressively adjusting its value during the learning process. This concept emerged from the transformation of the constrained optimization problem into a max-min problem, employing alternative updates. In this method, the value of λ steadily grows throughout the training, signifying a progressive emphasis on the constraint loss.

Projection Unlike previous methods, projection methods perform gradient updates considering the gradients of two losses: $\mathscr{L}$ and $\mathscr{C}$. For both *static* and *monotone* ($\lambda \uparrow$), Λ's are all diagonal matrices in equation (3). However, the projection method results in non-diagonal matrices depending on the gradients of both loss functions. The detailed formulation will be introduced in section §iii..

It is important to note that the decision on how to integrate two losses ($\mathscr{L}$, $\mathscr{C}$) is entirely separate from the process of formulating the constraint loss $\mathscr{C}$ (§A.), and the exploring strategy of constraint-violation examples (§B.). Therefore, adjusting Λ's (or, λ's) mentioned in this section can be independently combined with other analysis axes.

III. Further Exploration on Integration of Main Task and Constraint Information

In this section, we propose new methods for integrating the losses of main task and constraint injection task. Departing from categorized methods used in previous research, 'static' and 'monotone($\lambda \uparrow$)', we introduce three new integration mechanism for the two losses: 'projection-sup', 'projection-con', and 'projection-both'.

Motivation Gradient Episodic Memory (GEM) [7] model is designed for continual learning for positive backward transfer, aiming to store memories of previous tasks in such a way that the loss does not increase when learning from new data. It introduces constraints to prevent an increase in loss for previous tasks stored in memory when learning from new data and presents a new minimization problem. A-GEM [4] is a variant of GEM that is designed for effective memory and computational cost, by storing the averaged episodic memory across the all tasks. Motivated by these works, we propose a new method for integrating losses – $\mathscr{L}(\theta)$ and $\mathscr{C}(\theta)$ – for two tasks. In

GEM/A-GEM, whenever new data was deemed to violate positive backward transfer, it applies a projection operation for the gradients to adjust them. We adapt the concept from GEM/A-GEM and utilize it in designing the integration mechanism of main task and constraint information

Method Recall that the derivative of loss function with constraint has form of:

$$\nabla\mathscr{T}(\theta) = \Lambda^{sup} \cdot \nabla\mathscr{L}(\theta) + \Lambda^{con} \cdot \nabla\mathscr{C}(\theta)$$

Our approach is rooted in the idea that supervised learning and constraint injection are two distinct tasks, and during their respective updates, we can prevent negatively effecting each other by executing a projection of gradient for each other. The followings are explanations of three new algorithms.

Projection-sup applies the projection method to the gradient of the constraint loss (namely, adjust Λ^{con}) to prevent it from negatively affecting the supervised learning task, while storing the averaged gradient vector of supervised learning task g_{sup} previously used for training. Mathematically, project $\nabla\mathscr{C}(\theta)$ via:

$$\text{Proj}(\nabla\mathscr{C}(\theta)) = \nabla\mathscr{C}(\theta) - \frac{\nabla\mathscr{C}(\theta)\cdot g_{sup}}{g_{sup}\cdot g_{sup}} g_{sup} \qquad (4)$$

, whenever $\nabla\mathscr{C}(\theta)\cdot g_{sup} < 0$. Then, the vector $\text{Proj}(\nabla\mathscr{C}(\theta))$ satisfies $\text{Proj}(\nabla\mathscr{C}(\theta))\cdot g_{sup} = 0$. This ensures that $\nabla\mathscr{C}$ is transformed orthogonally to g_{sup}, preventing it from providing information that contradicts supervised learning.

Conversely, *projection-con* applies the projection method to the gradient of the supervised loss (namely, adjust Λ^{con}) to prevent it from negatively affecting the constraint injection task, while storing the averaged gradient vector of constraint injection task g_{sup} previously used for training. Mathematically, project $\nabla\mathscr{L}(\theta)$ via:

$$\text{Proj}(\nabla\mathscr{L}(\theta)) = \nabla\mathscr{L}(\theta) - \frac{\nabla\mathscr{L}(\theta)\cdot g_{con}}{g_{con}\cdot g_{con}} g_{con} \qquad (5)$$

, whenever $\nabla\mathscr{L}(\theta)\cdot g_{con} < 0$. Then, the vector $\text{Proj}(\nabla\mathscr{L}(\theta))$ satisfies $\text{Proj}(\nabla\mathscr{L}(\theta))\cdot g_{con} = 0$. This ensures that $\nabla\mathscr{L}$ is transformed orthogonally to g_{con}, preventing it from providing information that contradicts constraint injection.

Projection-both combines both projection-sup and projection-con, applying projection to both gradients (namely, adjust both Λ^{sup} and Λ^{con}) to ensure that neither task negatively impacts the other. It stores two types of gradients separately by each task used for training before, and apply two projections (4) and (5) together.

IV. Tasks

In this section, we introduce the tasks for which we conduct experiments: Natural Language Inference (NLI), Synthetic Transduction Example (STE), and Semantic Role Labeling (SRL).

A. Natural Language Inference (NLI)

NLI is a task that involves understanding the logical relationships between pairs of text. Given a premise (P) and a hypothesis (H), the task is to determine whether P entails H, contradicts H, or maintains a neutral relationship with H. There exists constraints such as if P entails H, then H must not contradict P. We used the five constraints listed in [9]. The dataset used is SNLI [2].

B. Synthetic Transduction Example (STE)

We also present an artificial task utilized in [5]. A sequence transducer $T : \mathscr{L}_S \to \mathscr{L}_T$ converts the source language $\mathscr{L}_S = (az|bz)^*$ to the target language $\mathscr{L}_T = (za|bbb)^*$, for example, $T(azbzbz) = zabbbbbb$. The constraint imposed involves the relationship between the number of 'b' in the source and the target. Specifically, the count of 'b' in the target must be exactly three times that in the source.

C. Semantic Role Labeling (SRL)

SRL is a natural language processing task that predicts the semantic roles of each word in a sentence with respect to a given verb or predicate. The method of our work employed for this purpose is BIO tagging.

The Unique Core Roles constraint from [6] is applied as a constraint, which means that there can be no more than one occurrence of each core argument. For a predicate u, if the model predicts the i-th word as `B-X`, then other words in the same prediction should not be predicted as `B-X`. This can be expressed as follow.

$$\forall\, u, i \in s, \mathtt{X} \in \mathscr{A}_{core},$$
$$B_{\mathtt{X}}(u,i) \to \bigwedge_{j\in s, j\neq i} \neg B_{\mathtt{X}}(u,j). \qquad (6)$$

The dataset we used is English Ontonotes v5, with the CoNLL-2012 shared task format [11].

V. Experiments

Our experiment is composed of NLI, STE, and SRL tasks, with accuracy, token accuracy, and F1 score are used as the main task metrics, respectively. Our goal is to first observe the performance trends of algorithms according to our three classification criteria. Then, we will explore combinations that show particularly strong performance.

Experiment environment We used RTX 3090 GPU, and Adam optimizer for all of trainings. We conducted training for each case 10 times, and the results are displayed as the mean (in the larger font above) and standard

deviation (in the smaller font below) for both the main task metric (denoted by Perf) and constraint violation (denoted by Const.Vio)[1] rate.

Metric Comparing the superiority of experimental results considering two different metrics simultaneously is very challenging, especially when there is no occurrence of Pareto-improvement. The $H\beta$-*score* (Harmonic β Score) we propose is an indicator that allows for a quick and clear evaluation of experimental outcomes based on two metrics. Assume we have two metrics to consider, and denote the scores for each metric as m_1 and m_2, respectively. Both metrics are assumed to have values ranging from 0 to 1, with higher values indicating better performance[2]. The $H\beta$-score is similar in form to the Fβ-score and is defined as follow:

$$H_\beta(m_1, m_2) = \frac{1+\beta^2}{\frac{1}{m_1} + \frac{\beta^2}{m_2}}.$$

The $H\beta$-score is exactly the same in form as the Fβ-score. It is simply an extension of the Fβ-score, which uses precision and recall as arguments, to be a score for any two arbitrary metrics. If the magnitude of β increases, the evaluation significantly considers the weight of m_2. Conversely, as the value of β approaches zero, the weight of m_1 is significantly considered in the evaluation.

Experiment results Table 1 in appendix §A shows the experiment results for all combinations possible in our analysis axes which consist of previous methods and our newly proposed methods. For each task, we present the experimental results based on our three analysis axes proposed in section §ii.: type of constraint loss (soft, binary, real), exploration strategy of constraint-violating examples (top-1, sampling, exhaustive), and mechanism for integrating the main and the constraint information (static, monotone, proj-sup, proj-con, proj-both).

As the sheer number of experiments is too large to interpret in table 1, we try to examine key factors for diverse $H\beta$-scores by dissecting the table 1 from different perspectives.

Trends per analysis axes Figure 1 illustrates the top 5 experimental results with the highest $H\beta$-scores, considering each of the five exploring strategies described in section §B.. Among the five strategies, one clear observation is that the sampling method consistently demonstrates superior performance across all tasks. Although there are variations, performance tends to improve as the sample size increases. However, the overall performance of the full strategy is not favorable, especially in SRL task. In the full strategy, the model generates errors that significantly different from those expected for realistic output, resulting in suboptimal performance due to the associated loss. We hypothesize that the full strategy's performance of SRL is even worse than that observed in NLI, due to the significantly larger output space.

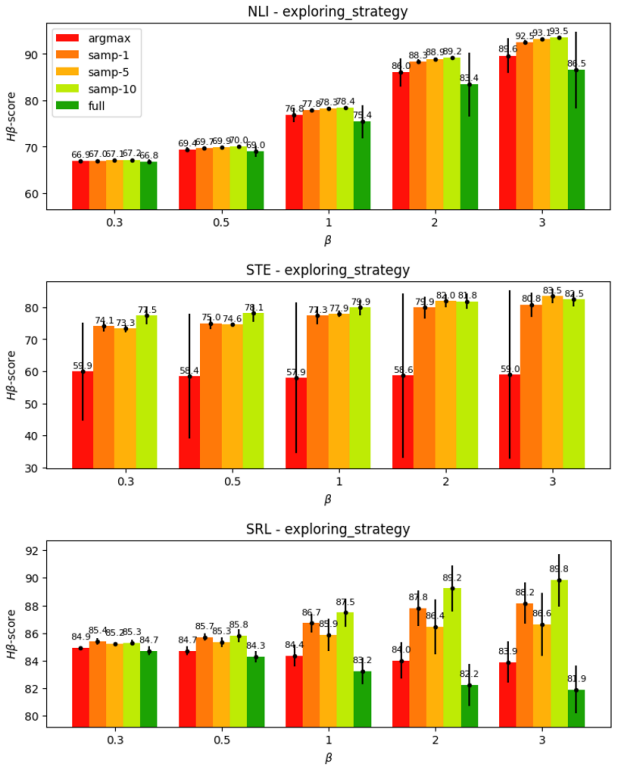

Fig. 1. The $H\beta$-score values for different values of β for three tasks: NLI, STE and SRL. For each β, the top 5 experimental results with the highest $H\beta$-scores are presented for each of the exploring strategy described in section §B..

Specific combinations of axes outperforming others Figures 2, 3, and 4 depict the $H\beta$-scores for different combinations of loss types and integration mechanisms when the sampling strategy is fixed as samp-10. For visibility, we consider values of 0.3, 1, and 3.

Notably, our newly proposed projection-based algorithms, projection-con and projection-both, exhibit the highest-level performance across most situations. We found that for the soft or real types of loss, the projection-both mechanism shows the best-level performance than other mechanisms for most combinations. In the case of the SRL task, there are instances where the monotone mechanism performs well. Particularly, when used in conjunction with a soft type of loss, the monotone mechanism exhibits higher performance, which is inconsistent with other experimental results. The reason for this discrepancy has not been clearly identified yet, but the specific characteristics of weight updates in constraint loss combined with a

[1] For example, $84.72_{\pm 00.77}$ means that the average is 84.72, and the standard deviation is 00.77 from 10 experiments.

[2] Constraint violation rate is used for table 1. However, when we consider $H\beta$-score, we convert it to the constraint satisfaction rate, which is 1−(constraint violation rate).

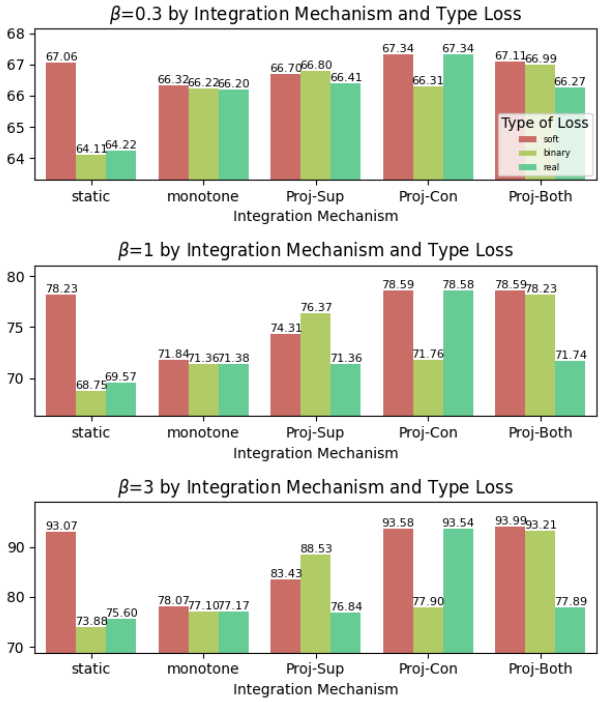

Fig. 2. Experiment result from NLI task with samp-10. Three bar plots represents the $H\beta$-scores with respect to the integration mechanism (separated by the x-axis) and type of constraint losses (separated by the color). From top to bottom, the corresponding values of β's are 0.3, 1, 3, respectively.

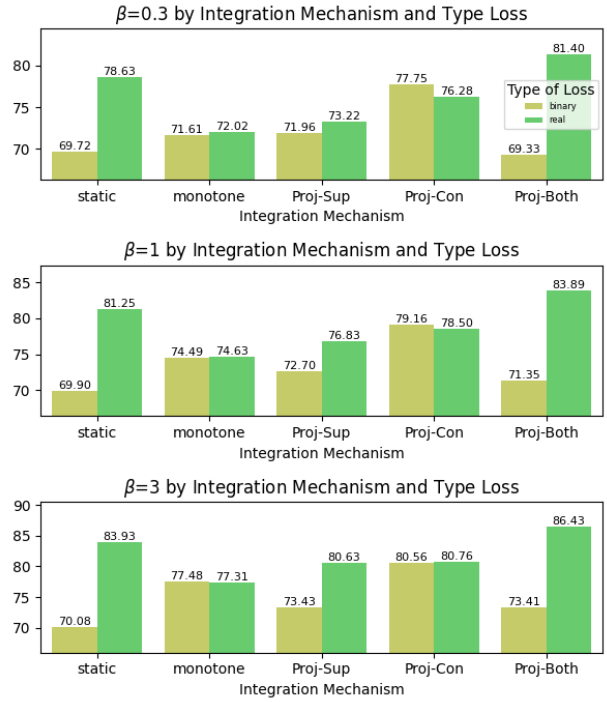

Fig. 3. Experiment result from STE task with samp-10. Three bar plots represents the $H\beta$-scores with respect to the integration mechanism (separated by the x-axis) and type of constraint losses (separated by the color). From top to bottom, the corresponding values of β's are 0.3, 1, 3, respectively.

soft type of loss for achieving higher performance remain a subject for future work.

Another noteworthy observation is that under samp-10, the soft type of loss exhibits the highest performance in most cases. Results from figures 2 and 4 show that, except for the projection-sup instance in SRL, soft type of loss generally outperforms real type of loss. It indicates that soft type of loss performs exceptionally well when combined with the sampling strategy.

VI. Conclusions

We have proposed three axes for classifying and categorizing learning algorithms related to injecting constraints: type of constraint loss, exploring strategy of constraint-violating examples, and integration of main task and constraint information. To the best of our knowledge, this study is the first to systematically classify existing learning algorithms with constraints under a unified formulation. We have analyzed the key factors that affect performance based on our analysis criteria, which helps in understanding learning algorithms with constraints.

Additionally, we have introduced three projection-based mechanisms as a novel approach for the integration mechanism of main task and constraint information. Viewing the main task and constraint injection as two separate tasks, we started with the motivation to prevent negative effects on each other during the gradient update process. This introduces a new perspective on integrating learning signals from main task and constraint, which shows superior performance compared to existing integration mechanisms.

VII. Acknowledgement

This work was supported in part by the National Research Foundation of Korea (NRF) grant (RS-2023-00280883, RS-2023-00222663), by the National Super computing Center with super computing resources including technical support (KSC-2023-CRE-0176), and partially supported by New Faculty Startup Fund from Seoul National University.

References

[1] Kareem Ahmed, Tao Li, Thy Ton, Quan Guo, Kai-Wei Chang, Parisa Kordjamshidi, Vivek Srikumar, Guy Van den Broeck, and Sameer Singh. Pylon: A pytorch framework for learning with constraints. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 319–324. PMLR, 2022.

[2] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal,

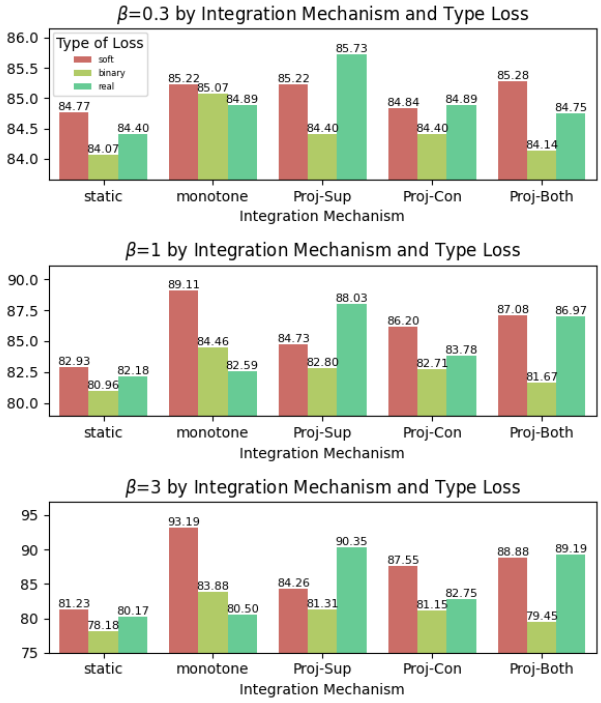

Fig. 4. Experiment result from SRL task with samp-10. Three bar plots represents the $H\beta$-scores with respect to the integration mechanism (separated by the x-axis) and type of constraint losses (separated by the color). From top to bottom, the corresponding values of β's are 0.3, 1, 3, respectively.

September 2015. Association for Computational Linguistics.

[3] Matthias Bröcheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, page 73–82, Arlington, Virginia, USA, 2010. AUAI Press.

[4] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[5] Jay Yoon Lee, Sanket Vaibhav Mehta, Michael Wick, Jean-Baptiste Tristan, and Jaime Carbonell. Gradient-based inference for networks with output constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4147–4154, 2019.

[6] Tao Li, Parth Anand Jawale, Martha Palmer, and Vivek Srikumar. Structured tuning for semantic role labeling. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8402–8412, Online, July 2020. Association for Computational Linguistics.

[7] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[8] Sanket Vaibhav Mehta, Jay Yoon Lee, and Jaime Carbonell. Towards semi-supervised learning for deep semantic role labeling. *arXiv preprint arXiv:1808.09543*, 2018.

[9] Pasquale Minervini and Sebastian Riedel. Adversarially regularising neural NLI models to integrate logical background knowledge. In Anna Korhonen and Ivan Titov, editors, *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 65–74, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[10] Yatin Nandwani, Abhishek Pathak, and Parag Singla. A primal dual formulation for deep learning with constraints. *Advances in Neural Information Processing Systems*, 32, 2019.

[11] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint conference on EMNLP and CoNLL-shared task*, pages 1–40, 2012.

[12] Hossein Rajaby Faghihi, Aliakbar Nafar, Chen Zheng, Roshanak Mirzaee, Yue Zhang, Andrzej Uszok, Alexander Wan, Tanawan Premsri, Dan Roth, and Parisa Kordjamshidi. Gluecons: A generic benchmark for learning under constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9552–9561, Jun. 2023.

[13] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[14] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR, 2018.

A Experiment Result

Table 1 represents the experiment results for all combinations, containing main task metrics(%, denoted as "Perf") and constraint violation rates(%, denoted as "Const.vio"). The gray-colored numbers represent results with main task metrics and constraint violations worse than the baseline. For each type of constraint loss, results showing the highest main task metric and lowest constraint violation are highlighted in bold. For individual task, the highest main task metric and lowest constraint violation results are marked with an asterisk (*). SRL, NLI, and STE tasks used F1 score, accuracy, and token accuracy for main task metric, respectively. For the types of constraint losses, soft, binary, and real respectively represents PSL, REINFORCE method with binary reward, and REINFORCE method with real reward. The term 'Baseline' refers to the experiment results without any constraint injection. To compare various algorithms under a unified formulation, experiments involving constraint loss related to REINFORCE were conducted by generating constraint loss for examples that violated the constraints.

Note that we can easily extend the learning algorithms with constraints to semi-supervised learning. For SRL and NLI tasks, we also utilized unlabeled data during the training process. For SRL, we randomly selected 3% of training data for unlabeled data. For NLI, we utilized the unlabeled data used in [1][3].

[3] https://github.com/pylon-lib/pylon/tree/master/examples/nli

Task			Top-1		Sampling-1		Sampling-5		Sampling-10		Exhaustive	
			Perf	Const.Vio	Perf	Const.Vio	Perf	Const.Vio	Perf	Const.Vio	Perf	Const.Vio
NLI	Baseline		Acc: 65.14 ±00.30, Const.Vio: 20.81 ±02.57									
	soft	static	65.18 ±00.35	04.72 ±00.88	65.40 ±00.40	03.60 ±00.60	65.31 ±00.26	02.23 ±00.36	65.22 ±00.40	02.29 ±01.88	65.20 ±00.34	01.95 ±00.22
		monotone ($\lambda \uparrow$)	65.21 ±00.26	21.51 ±03.03	65.20 ±00.27	20.24 ±02.09	65.30 ±00.38	21.83 ±01.74	65.33 ±00.54	20.20 ±01.67	65.20 ±00.55	22.72 ±02.05
		Proj-Sup	65.28 ±00.43	20.14 ±02.99	65.05 ±00.48	20.73 ±01.61	65.26 ±00.40	02.08 ±00.38	65.38 ±00.20	13.93 ±02.55	65.36 ±00.49	01.95 ±00.41
		Proj-Con	65.46 ±00.27	03.05 ±00.42	65.23 ±00.40	03.54 ±00.74	65.05 ±00.26	02.40 ±00.33	65.48* ±00.28	01.73 ±00.17	65.30 ±00.29	02.50 ±00.36
		Proj-Both	65.20 ±00.40	17.45 ±03.13	65.41 ±00.29	06.16 ±01.75	65.39 ±00.21	08.19 ±05.04	65.23 ±00.32	01.17* ±00.84	65.40 ±00.43	21.30 ±03.29
	binary	static	65.20 ±00.38	02.45 ±02.65	64.23 ±00.44	03.24 ±05.29	63.26 ±00.38	17.02 ±11.67	63.26 ±00.49	24.72 ±02.40	-	-
		monotone ($\lambda \uparrow$)	65.36 ±00.31	20.30 ±03.38	65.10 ±00.29	22.66 ±02.66	65.16 ±00.37	22.00 ±01.78	65.29 ±00.36	21.32 ±02.74	-	-
		Proj-Sup	65.21 ±00.35	14.73 ±03.30	65.25 ±00.47	07.06 ±01.66	65.22 ±00.23	02.25 ±00.20	65.18 ±00.25	07.80 ±04.97	-	-
		Proj-Con	65.11 ±00.49	07.49 ±06.08	65.42 ±00.19	11.80 ±02.43	65.25 ±00.35	02.57 ±00.16	65.33 ±00.24	20.40 ±03.01	-	-
		Proj-Both	65.16 ±00.28	19.76 ±02.85	65.05 ±00.34	02.11 ±00.28	65.23 ±00.41	02.10 ±00.37	65.14 ±00.49	02.18 ±00.33	-	-
	real	static	65.26 ±00.24	20.60 ±01.29	64.66 ±00.34	01.92 ±00.90	63.28 ±00.44	23.82 ±01.89	63.26 ±00.30	22.72 ±07.16	-	-
		monotone ($\lambda \uparrow$)	65.42 ±00.21	21.45 ±01.88	65.14 ±00.49	20.97 ±02.25	65.26 ±00.38	21.36 ±02.26	65.26 ±00.38	21.23 ±03.68	-	-
		Proj-Sup	65.26 ±00.26	22.93 ±01.96	65.21 ±00.25	22.70 ±02.23	65.11 ±00.39	20.68 ±02.76	65.51 ±00.48	21.65 ±02.03	-	-
		Proj-Con	65.27 ±00.32	20.98 ±01.67	65.33 ±00.36	08.75 ±01.86	65.04 ±00.24	02.39 ±00.36	65.49 ±00.38	01.78 ±00.20	-	-
		Proj-Both	65.09 ±00.34	23.21 ±01.27	65.15 ±00.31	07.04 ±02.77	65.40 ±00.39	09.33 ±03.42	65.29 ±00.26	20.40 ±03.62	-	-
STE	Baseline		Tok-Acc: 67.26 ±02.26, Const.Vio: 28.89 ±10.07									
	binary	static	69.98 ±00.43	29.35 ±12.11	73.59 ±02.37	18.83 ±12.72	69.35 ±02.20	33.32 ±06.73	69.68 ±03.22	29.87 ±17.58	-	-
		monotone ($\lambda \uparrow$)	67.17 ±06.42	26.93 ±13.09	71.03 ±05.72	33.55 ±11.46	69.43 ±03.79	24.89 ±11.15	71.07 ±04.63	21.74 ±08.34	-	-
		Proj-Sup	43.55 ±04.03	93.87 ±03.65	75.01 ±05.90	10.86 ±07.28	69.19 ±03.67	26.21 ±16.22	71.81 ±02.96	26.38 ±17.69	-	-
		Proj-Con	49.64 ±03.27	98.40 ±02.10	73.58 ±03.97	22.18 ±16.16	74.96 ±06.76	22.71 ±16.42	77.48 ±08.92	19.08 ±14.79	-	-
		Proj-Both	51.19 ±01.70	98.77 ±02.08	70.71 ±03.92	23.43 ±12.72	68.51 ±03.34	26.62 ±10.77	68.94 ±02.20	26.06 ±15.75	-	-
	real	static	67.77 ±04.25	30.18 ±12.86	70.16 ±03.03	22.18 ±12.04	70.30 ±04.02	10.86* ±07.53	78.13 ±05.00	15.37 ±11.15	-	-
		monotone ($\lambda \uparrow$)	63.73 ±04.43	22.92 ±13.45	60.74 ±02.93	51.79 ±34.51	69.91 ±03.29	19.07 ±13.41	71.53 ±03.74	21.99 ±13.05	-	-
		Proj-Sup	46.30 ±04.91	94.17 ±04.92	54.32 ±02.47	98.86 ±01.72	73.02 ±04.14	15.04 ±08.75	72.55 ±02.49	18.36 ±10.61	-	-
		Proj-Con	48.20 ±03.88	95.29 ±04.58	52.98 ±01.16	99.74 ±00.73	72.24 ±03.04	14.26 ±07.57	75.86 ±03.16	18.66 ±08.36	-	-
		Proj-Both	50.60 ±02.60	98.68 ±03.16	74.81 ±05.59	17.40 ±16.24	72.00 ±04.31	15.03 ±14.56	80.92* ±04.24	12.91 ±06.07	-	-
SRL	Baseline		F1: 84.72 ±00.77, Const.Vio: 20.43 ±04.09									
	soft	static	85.24 ±01.49	15.17 ±02.04	85.02 ±01.55	14.07 ±03.02	85.21 ±01.13	19.53 ±02.80	85.15 ±00.74	19.18 ±36.95	85.31 ±00.98	21.72 ±04.61
		monotone ($\lambda \uparrow$)	84.42 ±01.07	18.53 ±04.44	85.78* ±01.46	14.40 ±03.66	85.12 ±01.23	16.38 ±03.12	84.49 ±01.16	05.73* ±01.42	85.18 ±00.81	15.97 ±02.97
		Proj-Sup	85.02 ±00.98	20.79 ±04.63	85.19 ±01.24	20.23 ±04.85	85.09 ±01.03	19.59 ±03.93	85.32 ±01.26	15.86 ±04.08	85.18 ±00.97	18.73 ±04.03
		Proj-Con	84.96 ±00.60	15.72 ±01.23	85.24 ±01.53	11.76 ±02.44	85.07 ±00.90	12.80 ±02.85	84.58 ±01.17	12.11 ±03.61	84.31 ±00.57	18.04 ±04.01
		Proj-Both	85.21 ±01.21	21.86 ±03.13	84.71 ±01.06	12.11 ±00.37	85.62 ±01.68	17.68 ±03.57	84.93 ±02.06	10.66 ±01.83	85.03 ±01.11	17.64 ±04.26
	binary	static	85.20 ±01.51	19.84 ±00.66	85.52 ±01.02	19.53 ±02.34	85.19 ±00.91	18.90 ±02.91	84.71 ±01.28	22.48 ±04.34	-	-
		monotone ($\lambda \uparrow$)	84.51 ±00.94	14.25 ±02.93	84.36 ±00.75	20.98 ±05.88	85.23 ±01.44	15.50 ±03.37	85.19 ±00.68	16.26 ±04.40	-	-
		Proj-Sup	84.14 ±01.05	21.20 ±02.12	84.57 ±01.20	19.99 ±02.20	85.70 ±01.01	22.44 ±03.92	84.73 ±00.57	19.05 ±03.03	-	-
		Proj-Con	84.54 ±00.71	21.28 ±03.91	85.56 ±01.23	19.62 ±04.38	85.06 ±01.03	19.79 ±03.58	84.74 ±00.86	19.23 ±06.10	-	-
		Proj-Both	85.23 ±00.77	20.36 ±04.22	84.89 ±01.36	19.14 ±03.33	84.85 ±01.014	19.46 ±03.91	84.61 ±00.63	21.09 ±03.39	-	-
	real	static	85.05 ±00.68	18.26 ±03.21	85.12 ±00.96	09.79 ±03.15	85.33 ±01.09	22.37 ±04.43	84.85 ±01.23	20.32 ±04.10		
		monotone ($\lambda \uparrow$)	84.73 ±00.78	21.99 ±04.21	85.09 ±00.96	23.78 ±03.95	84.94 ±01.29	19.61 ±04.50	85.36 ±01.14	20.00 ±02.65	-	-
		Proj-Sup	85.19 ±01.46	19.19 ±04.13	85.09 ±00.90	17.16 ±04.40	84.87 ±00.94	09.21 ±02.60	85.29 ±01.25	09.05 ±02.20	-	-
		Proj-Con	85.06 ±00.93	18.22 ±04.32	84.31 ±01.67	09.47 ±04.19	85.19 ±01.32	22.55 ±05.09	85.11 ±01.08	17.50 ±03.98	-	-
		Proj-Both	85.05 ±00.94	18.54 ±04.55	85.25 ±00.74	20.36 ±04.28	84.54 ±00.41	11.96 ±02.87	84.33 ±07.19	10.23 ±03.35	-	-

Table 1. Experiment results for all combinations.