Journal of Machine Learning Research 25 (2024) 1-56 Submitted 7/23; Revised 9/24; Published 12/24

Towards Unbiased Exploration in Partial Label Learning

Zsolt Zombori ZOMBORI@RENYI.HU
Alfréd Rényi Institute of Mathematics
Eötvös Loránd University
Budapest, Hungary

Agapi Rissaki RISSAKI.AGAPI@GMAIL.COM
Khoury College of Computer Sciences
Northeastern University
Boston, USA

Kristóf Szabó KRIST.SZ13@GMAIL.COM
Alfréd Rényi Institute of Mathematics
Budapest, Hungary

Wolfgang Gatterbauer WGATTERBAUER@NORTHEASTERN.EDU
Khoury College of Computer Sciences
Northeastern University
Boston, USA

Michael Benedikt MICHAEL.BENEDIKT@CS.OX.AC.UK
Department of Computer Science
University of Oxford
Oxford, UK

Editor: Luc De Raedt

Abstract

We consider learning a probabilistic classifier from partially-labelled supervision (inputs denoted with multiple possibilities) using standard neural architectures with a softmax as the final layer. We identify a bias phenomenon that can arise from the softmax layer in even simple architectures that prevents proper exploration of alternative options, making the dynamics of gradient descent overly sensitive to initialization. We introduce a novel loss function that allows for unbiased exploration within the space of alternative outputs. We give a theoretical justification for our loss function, and provide an extensive evaluation of its impact on synthetic data, on standard partially labelled benchmarks and on a contributed novel benchmark related to an existing rule learning challenge.

Keywords: partial label learning, disjunctive supervision, rule learning

1. Introduction

Partial Label Learning (PLL) (Cour et al., 2011; Nguyen and Caruana, 2008; Jin and Ghahramani, 2002; Feng and An, 2019; Feng et al., 2020; Wen et al., 2021; Yao et al., 2020; Tian et al., 2023) deals with learning in the presence of imperfect supervision, where training data has a set of labels, one of which is the true label. The framework of PLL is very general, and a number of well-studied problems, including learning in the presence of partially-observable variables, can be seen as particular instances with certain specialized

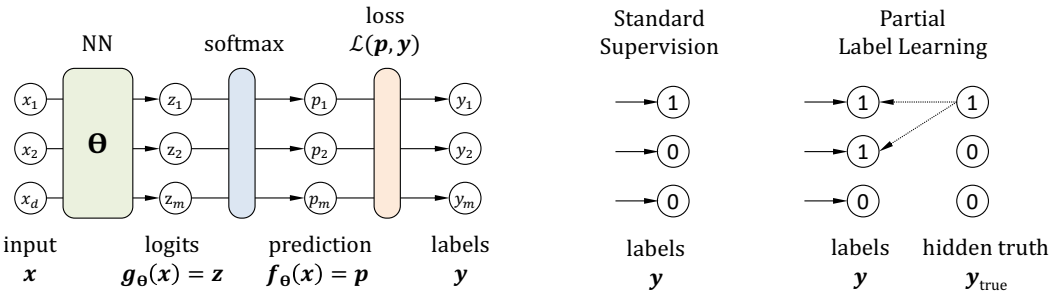
assumptions (e.g. that one has a probabilistic model that constrains the generation of disjunctive outputs). Over the last decade a multitude of proposals for PLL have emerged: for example, methods that treat the set of labels as an ensemble and average over them (Cour et al., 2011), or methods that try to learn patterns that distinguish noisy labels from true labels (Jin and Ghahramani, 2002; Nguyen and Caruana, 2008; Liu and Dietterich, 2012). We are motivated by the setting where no assumptions are made about how the partial supervision is generated, but only on the class of functions being learned.
One motivating application for this scenario of PLL is in applying modern machine learning techniques to *rule learning*, where the goal is to learn rules that can be used to derive target facts from some source facts. When formulated as a supervised machine learning problem, an important feature is that there may be multiple rules that can be used to derive any given target fact.

Example 1 (Rule Learning with Partial Supervision) *Let us assume that we have two database tables,* `Person` *and* `Author`*. A tuple* `Person(x,y,z)` *implies there is a person called x, who is y years old and belongs to group z, and a tuple* `Author(x)` *implies that x is an author. As a simple example of PLL, suppose the source facts include* `Person(alice,45,1)` *and* `Person(bob,34,1)` *and we would like to find mapping rules that derive target facts* `Author(alice)` *and* `Author(bob)`*. Two candidate rules may be*

$$\texttt{Author}(x) \leftarrow \exists a, t.\texttt{Person}(x, a, t)$$
$$\texttt{Author}(x) \leftarrow \exists a.\texttt{Person}(x, a, 1)$$

Above we use Prolog-style syntax, where x is implicitly universally quantified. We are interested in neural models that generate rules of the form above from such source and target facts. Either rule above is equally acceptable as an output for deriving the target facts. Thus the target facts can be associated with the partial label consisting of the set of output rules that can derive them.

Example 2 (Semantic parsing with Partial Supervision) *We consider another example of PLL, a variant of the* semantic parsing *task, inspired by Guu et al. (2017); Curran and Clark (2017): A user issues a sequence of commands in natural language, where each command describes a transformation of a fixed state (e.g. repositioning objects within a scene). The goal is to translate the natural language utterances into commands in some fixed programming language. A human annotator provides supervision on training examples, but only at the level of the observed state sequence. Since several commands can have the same impact along the entire training and test data set, there may be no unique correct answer. As a simple example, suppose that the parser is trying to learn the state transition associated with utterance* `Alice moves to the left of Bob`*. The available supervision only reveals that Alice ends up at position 1 (which is left of Bob), making it impossible to distinguish the intended state transition from the one associated with utterance* `Alice moves to position 1`*.*
In solving this problem it is natural to learn a sequential model, where a network outputs the probabilities of a command for a given utterance, conditioned on the prior sequence of utterances. Notice that in this task we can efficiently check whether a command sequence matches the supervision, by executing it. But usually we cannot hope to compute an explicit



(a) Neural architectures with softmax (b) Multi-class (c) PLL

Figure 1: We consider standard neural architectures with a softmax layer (i.e. the goal is to *predict one output*) whose parameters $\boldsymbol{\theta}$ are learned via supervision from labeled examples $(\boldsymbol{x}, \boldsymbol{y})$ via a loss function $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y})$ comparing the predictions $\boldsymbol{p}$ against the labels $\boldsymbol{y}$ (a). In the standard *multi-class learning* scenario, exactly one correct label is supplied (b). In *Partial Label Learning* (PLL), several alternative labels are supplied but only one among them is correct (c).

Learning task	**Supervision**	**Prediction**	**Interpretation**
Multi-class	1	1	Single true label
Multi-label	multiple	multiple	Several true labels
PLL	multiple	1	Single (unknown) true label

Table 1: Comparing multi-class classification, multi-label classification and PLL.

list of the acceptable outputs that match the supervision: the number of possible sequences can be enormous. Since we cannot enumerate all acceptable sequences when we want to compute the aggregate loss over examples, the best we can do is to sequentially sample according to our current learned distribution.

With multiple outputs labeled for a given input in the training set, supervision for PLL resembles supervision for multi-label classification. The key difference is that PLL seeks a function that produces a single output as the answer. Table 1 and Figure 1 show a comparison between the tasks, while Example 3 gives an example of each.

Example 3 (Path Learning Scenarios) *Consider a path finding problem in some dangerous environment: given endpoints A and B, we aim to find paths that take us safely from A to B. Standard multi-class learning is when there is a single safe path between A and B and it is provided for each training sample. In PLL too, there is a single safe path for each pair of endpoints, but it is not known for the training samples, only a set of paths that contains the single safe one. In multi-label learning, there are numerous safe paths and we aim to identify all of them.*

We focus on classifiers that output a probability distribution over output space $\mathcal{Y}$, by application of a final *softmax* layer. Consider partially labelled training samples of the form

$(\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{y} \subseteq \mathcal{Y}$ is the set of acceptable labels for input $\boldsymbol{x}$. The output of a classifier $\boldsymbol{p} = \boldsymbol{f}(\boldsymbol{x})$ represents a probability distribution over $\mathcal{Y}$. Much of the literature on PLL (e.g. Feng et al., 2020; Guu et al., 2017), uses a variation of the loss

$$\mathcal{L}_{\text{NLL}}(\boldsymbol{p}, \boldsymbol{y}) = -\log\Big(\sum_{i \in \boldsymbol{y}} p_i\Big).$$

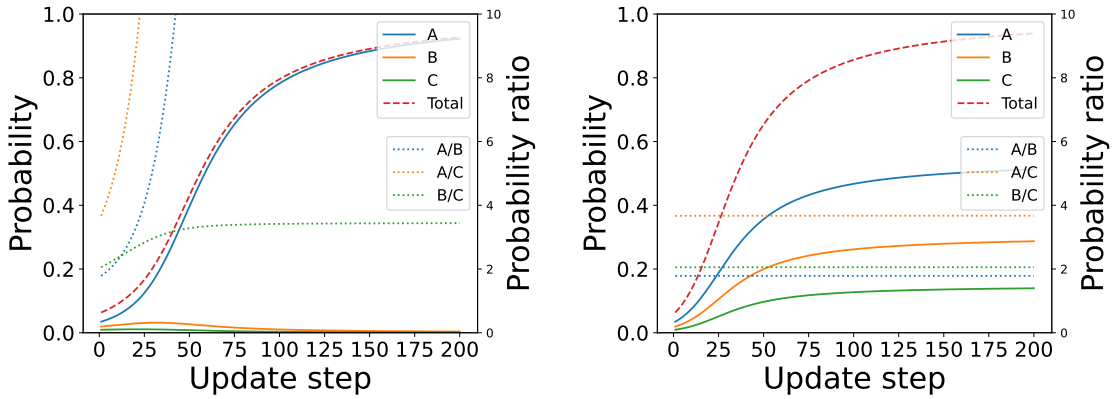
This is simply the negative log likelihood of obtaining a target in $\boldsymbol{y}$ when sampling from $\boldsymbol{p}$, denoted NLL-loss. However, for the softmax architecture, we show that simply training with this loss leads to an undesirable property that some of the *acceptable* labels would be favored over others, when trained using gradient descent. In fact, in the absence of other supervision, this leads to a *winner-take-all* scenario, where all the probability concentrates on only one of the acceptable labels.
As an alternative, we propose a novel loss function, the Libra-loss (Definition 7), whose updates preserve the ratios of the probabilities for the acceptable labels in the absence of other supervision. We show that such a loss function is unique up to composition by a differentiable function under some natural technical conditions. This more balanced loss leads to more stable training and increased success rate in finding a better optimum irrespective of the starting conditions.

Example 4 *Let us examine a toy problem with* $d = 10$ *inputs and* $m = 100$ *outputs. We assume a single training sample* $(\boldsymbol{x}, \{A, B, C\})$*, i.e., having* $k = 3$ *allowed outputs. We train a neural network that consists of a single dense layer with* 100 *neurons and softmax nonlinearity, having* 1100 *parameters altogether. Figure 2a shows the behavior of the standard* NLL-loss*, and Figure 2b our* Libra-loss*, both starting from the same initial condition.* NLL-loss *results in a distribution where the allowed output A with the highest initial probability accumulates all the probability mass. In contrast,* Libra-loss *yields a balanced update and the ratio of the allowed outputs does not change.*

1.1 Contributions

The paper's contributions are as follows:

- We describe a bias phenomenon for architectures ending in a softmax layer when learning from partially labelled data sets and using NLL-loss. We show (Theorem 4) that it prevents proper exploration of alternatives when optimizing NLL-loss.
- We formulate a property to avoid the observed bias and derive from it the Libra-loss function, whose updates maintain the ratios of probabilities for acceptable labels produced by the softmax. We show that when loss functions are restricted to depend only on the predicted probabilities of *acceptable outputs*, Libra-loss is uniquely defined (up to composition by differentiable functions).
- We consider a stronger property that aims to avoid bias not only among acceptable labels, but *also among unacceptable ones* and derive from it the Sag-loss function. We show that among all loss functions that can depend on both acceptable and unacceptable probabilities, Sag-loss is uniquely defined (again, up to composition by differentiable functions).



(a) NLL-loss (b) Libra-loss

Figure 2: Example 4: Learning curves using a small classifier neural network for NLL-loss (a) and Libra-loss (b). Solid lines represent probabilities and dotted lines represent probability ratios of allowed outputs. The dashed Total line is the sum of allowed probabilities, i.e., $A+B+C$. Notice how the relative ratio between allowed labels remain constant in (b), while the 'winner-takes-all' in (a).

- We compare several methods from the PLL literature experimentally, on synthetic and real-world data sets. These experiments demonstrate the performance and accuracy benefits of Libra-loss, while results related to Sag-loss are not conclusive. In particular, we find that Libra-loss is more robust than other variants when the learning task becomes harder, either because there are more labels in the label sets or because some distractor labels co-occur very often with the true label.
- We provide novel PLL data sets appropriate for *rule learning* in a supervised context.
- The entire codebase is available from the project webpage (BESS project'23).

1.2 Organization

We overview related work in Section 2 and define our problem in Section 3. Section 4 provides our key technical contributions: the formalization of the bias problem, and our solution using probability-preserving loss functions. Section 6 is concerned with testing our approach experimentally. We close with conclusion in Section 7. All proofs, as well as some details of the experimental set up are in the Appendix.

2. Related work

In this section we summarize the relevant literature.

2.1 Partial Label Learning (PLL)

Partial Label Learning has by now an extensive literature. See, for example Tian et al. (2023) for a recent survey. One common approach is to dampen the loss proportionally to

an average of the overlap between the output probability distribution and each acceptable label, possibly also including a component that enhances the loss proportionally to an average overlap with the unacceptable outputs. This approach has many variations and goes under the heading of "average-based methods" (Wen et al., 2021; Cour et al., 2011). Another family of approaches attempts to learn the noise model in combination with learning the prediction. These are sometimes referred to as "identification-based" methods (Feng and An, 2019; Liu and Dietterich, 2012). They might use a strategy similar to *expectation maximization* to alternate between refining the model of the most likely true labels and exploiting the model to make predictions.

The semantic parsing literature also contains scenarios which can be interpreted as PLL (e.g. Guu et al., 2017). There, neither the allowed nor the disallowed labels can explicitly be enumerated, requiring some sampling strategy to compute the loss.

Many problems in the machine learning literature can be recast as special constrained cases of PLL. For example, if one has a model with latent variables, such as a *Hidden Markov Model* (Baum and Petrie, 1966), any value of the output can be generated by multiple valuations of the hidden variables, thus the output can be considered partial supervision over the possible latent variable values. The underlying probabilistic model constrains how partial supervision can be generated. In contrast, here we will have a model on the underlying function class being learned, but no assumption on how the partial supervision is generated: thus prior techniques from the latent variable literature will not be applicable.

2.2 Optimization

In terms of optimization, our Libra-loss function can be viewed as a form of *entropic regularization* (Jagatap et al., 2022), with the notable difference that we apply regularization to a truncated distribution of the output that is different for each data point.

2.3 Rule Learning

One of the applications of our loss function is in the setting of a neural approach to rule learning. Rule learning has been studied from both a theoretical and practical perspective for many decades. The theory includes complexity bounds within a number of learning models. An example is the complexity of finding a Horn sentence that entails a given set of statements, while contradicting (or merely failing to entail) another set of sentences (De Raedt and Džeroski, 1994). This problem has also been considered in the presence of a background theory Σ: thus entailment is with respect to Σ. Our setting is of this form, where the background theory consists of ground facts. Like most variations of the problem, this is known to be intractable even when the size of the rule bodies is fixed. Intuitively, one has to guess a rule or rules that fit the data, and then verify via evaluating the body of the guessed clause. For a formalization of this intuition, see the Σ_2^p-completeness results in Gottlob et al. (1999).

2.4 ML for Rule Learning

One response to the combinatorial hardness of rule learning is to consider a smooth semantics for logical rules, aiming to make the loss amenable to neural methods. An example of

this approach is Neural Theorem Proving (Rocktäschel and Riedel, 2017; Minervini et al., 2020), which looks for candidate rules of a shape constrained by a template. Atoms are scored using a smooth variant of unification, based on a parameterized embedding of facts in Euclidean space. Scores are aggregated using the MIN function within a rule and MAX across rules. The score of a rule and the parameters of the embedding are then optimized via gradient descent. The MIN/MAX aggregation results in extremely sparse gradients, leading to computational difficulties. In addition, the sharpness of MIN/MAX boundaries makes it difficult to move between alternatives, resulting in a "closest-take-all" behavior, not unlike the "winner-take-all" behavior of the NLL-loss, presented in our paper. In Evans and Grefenstette (2018) each possible rule is associated with a weight, and k-step forward reasoning is performed to compute a score for supervised facts. When aggregating scores of alternative proofs, the authors note that MAX aggregation adversely affects gradient flow and use the probabilistic sum $f_{\text{agg}}(x, y) = x + y - xy$ instead. This makes the gradients denser, but does not guarantee balanced gradients among alternatives. The Libra-loss function presented in this paper is designed specifically to make the transition between alternative derivations as smooth as possible, allowing better exploration.

2.5 Symbolic Supervision in ML

Learning logical rules represents one application of our framework, but there is a broader connection between PLL and logic, in that partial supervision can be thought of as a special case of symbolic supervision, where the supervision is given wholly or in part by constraints. The set up contrasts with much prior work on neuro-symbolic methods (Xu et al., 2018; Ahmed et al., 2022; Hu et al., 2016; Xie et al., 2019), which focus on enforcing semantic information given by logical constraints that are known to hold globally across all inputs, including those outside the training set. This prior work deals with logical constraints that are more complex than disjunctions, and the loss functions that are introduced (e.g. in Xu et al., 2018; Ahmed et al., 2022) are themselves hard to compute in the worst case. Ahmed et al. (2022) deals with a regularization term which is constraint-aware, analogous to our loss function. But entropy is being minimized to achieve sharper decision boundaries, while in our case it is being maximized to enhance exploration.

When a model is trained to predict a structured output involving several atomic predictions and we have some background knowledge constraining the output, it is a common assumption—due mostly to computational reasons—that all predictions are independent from each other. However, van Krieken et al. (2024) shows that this seriously limits the distributions that the model can represent, and can hinder learning dynamics even when only point mass distributions (which can clearly be succinctly representable) are considered. Our work goes further, and shows that even if a fully expressive model is employed—such as the sequential model described below—a standard loss function can still introduce a bias that hinders optimisation. While van Krieken et al. (2024) focuses on identifying a winner-take-all phenomenon, we provide theoretical and empirical justification for a loss function that can assist in addressing the problem.

A partially labelled problem class is discussed in Marconato et al. (2024) where the model predicts concepts while the supervision provides labels that can be indirectly derived from the concept predictions via background knowledge. They raise the problem of *reason-*

ing shortcuts, where the model learns to satisfy the supervision in incorrect ways, which can be explained by the winner-take-all behaviour that we identify. As a solution to the winner-take-all problem, they propose two techniques: entropy regularization and ensembling several models that are trained to be maximally different. Our approach implements a special, input-dependent form of entropy regularization: see Section 4.6 for a comparison. Ensembling is orthogonal to our work, and can easily be combined with it.

2.6 Rule Learning for Aligning Heterogeneous Data Sources

Our rule learning experiments are based on the RODI benchmark (Pinkel et al., 2015), aimed at comparing systems for aligning relational sources with a target schema. Several such systems are evaluated in Pinkel et al. (2015). However, the systems do not make use of supervision, looking only at textual and structural similarities between source and target. In contrast, we focus on learning the alignment from supervision. Nevertheless, we note that the success percentage of all examined systems on the RODI challenges ranges between $3-50\%$, much lower than ours (see Section 6.4). This highlights the benefit of approaching the alignment problem via supervised machine learning.

3. Preliminaries and Problem Statement

Supervised classification is the task of learning a function that conforms to a given set of samples $D = \left\{\left(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)}\right)\right\}_{j=1}^{n}$ where $\boldsymbol{x} \in \mathbb{R}^d$ is the input and $\boldsymbol{y} \in \{0,1\}^m$ the one-hot encoded desired output (i.e. exactly one entry is 1). In *Partial Label Learning* (PLL), however, there can be more than one allowed output, represented as $\boldsymbol{y}$ having multiple entries being 1. PLL assumes one single correct output among the given 1 entries that is unknown at training time (thus the labels are uncertain). We at times overload the notation and define $\boldsymbol{y}$ as the set of allowed labels as indexed by the binary vector.[1] We use k to denote the number of acceptable outputs associated with label $\boldsymbol{y}$ in supervision. We use $\boldsymbol{y}_{\text{true}}$ to denote the one-hot encoded unknown correct output.
PLL assumes a joint data generating distribution $\mathcal{P}(\boldsymbol{x}, \boldsymbol{y}_{\text{true}}, \boldsymbol{y})$ on inputs $\boldsymbol{x} \in \mathbb{R}^d$, true one-hot outputs $\boldsymbol{y}_{\text{true}} \in \{0,1\}^m$, and partial supervision $\boldsymbol{y} \in \{0,1\}^m$. In other words, the observed labels $\boldsymbol{y}$ are a distorted representation of the true labels $\boldsymbol{y}_{\text{true}}$ and the former always includes the latter. The goal is to learn a function $\boldsymbol{f}$ in a given target class that maximizes:

$$\mathbb{E}_{\mathcal{P}(\boldsymbol{x}, \boldsymbol{y}_{\text{true}}, \boldsymbol{y})}[P(\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}_{\text{true}})].$$

3.1 Disjunctive Supervision (DS)

We briefly note that there exists a variant of PLL in which no assumption is made about a single true output $\boldsymbol{y}_{\text{true}}$. We refer to it as *Disjunctive Supervision (DS)*. It assumes only a joint data generating distribution $\mathcal{P}(\boldsymbol{x}, \boldsymbol{y})$ on inputs $\boldsymbol{x}$ and partial supervision $\boldsymbol{y}$. The target is to learn a function $\boldsymbol{f}$ that maximizes:

$$\mathbb{E}_{\mathcal{P}(\boldsymbol{x}, \boldsymbol{y})}[P(\boldsymbol{f}(\boldsymbol{x}) \in \boldsymbol{y})].$$

1. We use 1-indexing: For example $\boldsymbol{y} = (1,0,1)$ has 2 acceptable outputs and could equally be written as $\boldsymbol{y} = \{1,3\}$.

The two setups differ in their problem statement but are identical in the format of the supervision and require very similar treatment. Even though some of the problems discussed below could arguably be considered DS instead of PLL (because there is no clear true label), for simplicity, we will remain within the PLL framework for this paper.

3.2 Learning Setup

As in supervised learning, we do not know the PLL distribution $\mathcal{P}$: instead, we assume a finite $D = \{(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)})\}_{j=1}^{n}$ sampled uniformly from $\mathcal{P}$ and we focus on optimizing performance on this set. Our learning target class will be a statistical model $\boldsymbol{p} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathrm{softmax}(\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{x}))$, with input $\boldsymbol{x}$ and parameters $\boldsymbol{\theta} \in \mathbb{R}^t$, interpreting its output as a probability distribution over the output space. The function $\boldsymbol{g}$ gives the unnormalized output, called *logits*, which we denote as $\boldsymbol{z}$.
We aim to find the *Maximum Likelihood Estimate (MLE)*, which maximizes the joint probability of the observed data, which is the conditional probability given $\boldsymbol{x}$ of observing an o with $o \in \boldsymbol{y}_{\text{true}}$. For computational reasons, one usually minimizes the negative logarithm of this value

$$-\log\left(\prod_{j=1}^{n} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(j)}) \cdot \boldsymbol{y}_{\text{true}}^{(j)}\right) = -\sum_{j=1}^{n} \log\left(\boldsymbol{p}^{(j)} \cdot \boldsymbol{y}_{\text{true}}^{(j)}\right). \qquad \text{(PLL)} \qquad (1)$$

Equation (1) cannot be optimized directly as $\boldsymbol{y}_{\text{true}}$ is not known for training samples. In the absence of any prior preference over acceptable labels, maximum likelihood methods use the the candidate label set rather than the unknown true label to define the likelihood (e.g. Côme et al., 2009; Grandvalet and Bengio, 2004; Jin and Ghahramani, 2002) which corresponds to the MLE of the DS problem

$$-\log\left(\prod_{j=1}^{n} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(j)}) \cdot \boldsymbol{y}^{(j)}\right) = -\sum_{j=1}^{n} \log\left(\boldsymbol{p}^{(j)} \cdot \boldsymbol{y}^{(j)}\right) \qquad \text{(DS)}$$

which yields the following samplewise NLL-loss:

$$\mathcal{L}_{\text{NLL}}(\boldsymbol{p}, \boldsymbol{y}) = -\log\left(\boldsymbol{p} \cdot \boldsymbol{y}\right) = -\log\left(\sum_{i=1}^{m} p_i y_i\right) = -\log(P_{\text{acc}}). \qquad (2)$$

The above formulation of the NLL-loss is a direct generalization of the classical case with a single allowed output to multiple allowed outputs and taking their probability mass together as $P_{\text{acc}} = \sum_{i=1}^{m} p_i y_i$. We use the NLL-loss as a baseline for optimization and argue that it is not an ideal choice for PLL due to its sensitivity to initial configuration. The same applies to most identification-based methods, such as LWS-loss and RC-loss (defined later in Section 6.1).

3.3 Extension to Sequential Outputs

PLL has important applications in which the output space cannot be effectively modeled as a set of unstructured objects. For instance, in the path finding problem of Example 3,

there can be a huge number of paths (even unbounded) and we may want our model to generalize to unseen paths, not just to unseen endpoints. In such scenarios, it is not tractable to explicitly compute a distribution over the entire output space, as a standard classifier model would do. *Autoregressive models* provide a solution for such problems: instead of producing the output one-shot, they build it incrementally: given an input and a partially constructed output, an autoregressive model predicts the next component of the output. Consequently, one has to repeatedly evaluate such models to obtain the final prediction.

Example 5 (Example 3 continued) *Returning to the path finding problem, each output can naturally be modeled as a sequence of atomic choices coming from a small fixed set, e.g.* {*"north", "west", "south", "east"*}.

We extend PLL for problems where outputs are represented as sequences over a finite alphabet Σ of m elements. Following the terminology of language modeling, we refer to the elements of the alphabet as *tokens*. Our learning target class will be a statistical model $\boldsymbol{p} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{s}_{\text{prefix}}) = \text{softmax}(\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{s}_{\text{prefix}}))$. Thus, besides input $\boldsymbol{x}$, the model receives an extra argument $\boldsymbol{s}_{\text{prefix}} \in \Sigma^*$ which is a sequence of tokens from Σ. Notice that while the size of alphabet m is finite, the length of the sequences is not necessarily so. The output is a probability distribution over Σ, interpreted as the distribution of the next token of the output following $\boldsymbol{s}_{\text{prefix}}$. Given input $\boldsymbol{x}$ and sequence $\mathbf{s} = s_1 \dots s_\ell \in \Sigma^*$ the model $\boldsymbol{f}$ can be used to compute the predicted probability of $\mathbf{s}$ as

$$P_{\boldsymbol{\theta}}(\mathbf{s}|\boldsymbol{x}) = \prod_{i=1}^{\ell} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}, (s_1 \dots s_{i-1}))_{I(s_i)} \tag{3}$$

where $I(s_i)$ refers to the index in the output of $\boldsymbol{f}$ that corresponds to token s_i. Let $\mathbf{s}^{(1)}, \mathbf{s}^{(2)} \dots$ be an arbitrary, fixed ordering of all sequences over Σ. Given input $\boldsymbol{x}$, let $\boldsymbol{p}$ represent the (possibly infinite) vector of model predicted probabilities, i.e.,

$$p_i = P_{\boldsymbol{\theta}}(\mathbf{s}^{(i)}|\boldsymbol{x})$$

In dataset $D = \{(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)})\}_{j=1}^{n}$ with sequential output space, $\boldsymbol{y}^{(j)}$ is an indicator vector over finite sequences. Note that even if Σ is finite, the set of all sequences may be infinite. We restrict $\boldsymbol{y}$ to have only finitely many 1's, so that it is finitely representable. This way, the predicted probabilities of allowed sequences can be computed according to Equation (3). Hence, any method that directly optimizes only the probabilities of allowed outputs generalizes directly to the sequential case. In particular, minimizing the NLL-loss is also applicable and yields the maximum likelihood estimate for DS and a natural proxy loss for PLL. Note, however, that the probabilities of all the disallowed outputs cannot be effectively computed, ruling out some optimization methods.
Table 2 summarizes the notation used throughout the paper.

4. Addressing Bias in Partial Label Learning

We prove that, even for simple architectures, standard optimization based on a direct generalization of the MLE (i.e. the NLL-loss in Equation (2)) leads to biased "winner-take-all" learning. We then introduce our main contribution, a novel property of loss functions,

Symbol	Definition
$(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)})$	Example training point with $\boldsymbol{x} \in \mathbb{R}^d$ and $\boldsymbol{y} \in \{0,1\}^m$
$\boldsymbol{y}^{(j)}_{\text{true}}$	Unknown one-hot true label $\boldsymbol{y}^{(j)}_{\text{true}} \subseteq \boldsymbol{y}^{(j)}$ with $\lvert\boldsymbol{y}^{(j)}_{\text{true}}\rvert = 1$
$\boldsymbol{\theta}$	Parameters to learn
n	Number of samples
d	Input dimension
m	Number of outputs
k	Number of 1's in label $\boldsymbol{y}$
ℓ	Length of sequential output
$\mathcal{P}$	Data generating distribution
D	Finite dataset, sampled uniformly from $\mathcal{P}$
$\mathcal{L}$	Loss function
$\boldsymbol{g} : \mathbb{R}^d \to \mathbb{R}^m$	Logit function
$\boldsymbol{f} : \mathbb{R}^d \to [0,1]^m$	Probabilistic classifier function
$\boldsymbol{z}$	Unnormalized model prediction ("logits"): output of $\boldsymbol{g}$
$\boldsymbol{p}$	Normalized model prediction ("probabilities"): output of $\boldsymbol{f}$
P_{acc}	Model predicted probability of all acceptable labels $\sum_{i=1}^{m} p_i \boldsymbol{y}_i$
o	Element from the output space
Σ	finite alphabet of m elements for in the sequential setup

Table 2: Summary of the notation used in the paper

the PRP property, which formalizes the absence of learning bias. We provide a loss function, the Libra-loss, that possesses this property, and also show that it is the unique loss function satisfying the PRP property, up to composition by differentiable functions. Next, we relate the Libra-loss to entropy regularization (Pereyra et al., 2017) and the NLL-loss. Then, we introduce the bi-PRP property, an extension of the PRP property and provide an analogous characterization theorem based on a loss function called Sag-loss. We end the section with practical considerations.

4.1 Stability of Probability Ratios of Allowed Outputs During Training

Given a set of samples D and a function $\boldsymbol{f}$, let $\boldsymbol{f}_{|D}$ denote the function with domain restricted to $\{\boldsymbol{x} \mid (\boldsymbol{x}, \boldsymbol{y}) \in D\}$. When the supervision is *total*, i.e., each $\boldsymbol{x}$ corresponds to a single output, and $\boldsymbol{y} = \boldsymbol{y}_{\text{true}}$ is one-hot, then there is a single optimal function $\boldsymbol{f}^*_{|D}$ that fits perfectly to D, namely when $\boldsymbol{f}^*_{|D}(\boldsymbol{x}) = \boldsymbol{y}$ for each $(\boldsymbol{x}, \boldsymbol{y}) \in D$.[2] This does not hold when supervision is partial: given that we have no direct information (PLL) or preference (DS) about the true label $\boldsymbol{y}_{\text{true}}$, any output distribution that places all the probability mass over the acceptable outputs can be considered as perfect fitting to the training signal. Other constraints—such as regularization, interaction among training points, or task-specific requirements—might restrict this set of optima. However, we argue that it is very important to avoid any prior bias in the learning algorithm towards any of these optimal distributions. Let $p_i = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})_i$ denote the probability of the i^{th} dimension of the output distribution. The unwanted bias that we target in this paper is "winner-take-all". That is if $(\boldsymbol{x}, \boldsymbol{y}) \in D$ with $y_i = y_j = 1$ and $p_i > p_j$ at initialization, then the optimization converges to $p_i = 1$ and $p_j = 0$. To see why such behavior is undesirable, consider Example 6.

2. Note, however, that the same optimal function $\boldsymbol{f}^*_{|D}$ can have multiple realizations in terms of $\boldsymbol{\theta}$.

Example 6 *Consider a problem with $m = 3$ outputs: A, B and C. Assume two samples with the same input $\boldsymbol{x}$: $(\boldsymbol{x}, \{A, B\})$ and $(\boldsymbol{x}, \{A, C\})$.*[3] *Next, assume that at initialization we have $\boldsymbol{f}(\boldsymbol{x})_B > \boldsymbol{f}(\boldsymbol{x})_A$ and $\boldsymbol{f}(\boldsymbol{x})_C > \boldsymbol{f}(\boldsymbol{x})_A$. Then the signals from the two samples work against each other if using* NLL-loss *trying to increase the probability of B and C, respectively, instead of finding the joint optimum in A. This example is analyzed in greater depth in Example 9, as well as in Figures 4 and 5 within Section 4.4.*

In general, any randomized initialization in the parameters can lead to an initial bias among the outputs, which may prevent the expected interaction among different points. Ideally, we would like the model update operation to preserve an *invariance property*: as we increase the aggregate probability of a set of values, the distribution within the set should not change:

Postulate 1 (Ratio preservation) *For one training point with multiple allowed outputs, a single optimization step that updates model parameters $\boldsymbol{\theta}$ should preserve the* ratio of probabilities *of the allowed outputs.*

We formalize this for a parameterized distribution $\boldsymbol{f}_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$. We assume that training is done via *gradient descent*, referred to as *Gradient-update*:

Definition 2 (Gradient-update) *Given a parameter vector $\boldsymbol{\theta}$, an update operation is called a* Gradient-update *if there exists some loss function $\mathcal{L}$ and* learning rate $\lambda > 0$ *such that the update on the i^{th} parameter is*

$$\theta_i' := \theta_i - \lambda \frac{\partial \mathcal{L}}{\partial \theta_i}$$

4.2 Negative Log Likelihood (NLL-loss) and Bias

We now show that gradient descent on the NLL-loss from Equation (2) leads to a "winner-take-all" effect in the presence of partial supervision. The intuitive explanation for this is that the easiest way to decrease the NLL-loss is to increase the greatest probability: the gradient of the NLL-loss that the logits receive (through the softmax layer) is proportional to the output probabilities.
Our formal results apply exactly to a simple class of classifiers called *softmax regression* (Tsoumakas and Katakis, 2007).

Definition 3 (Softmax Regression) *We refer to* softmax regression *as the parametric model $\boldsymbol{p} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{\theta} \cdot \boldsymbol{x})$. This model is also called* multinomial logistic regression *in the literature.*

Theorem 4 (Winner-take-all) *Consider the softmax regression model $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$. Fix a datapoint $(\boldsymbol{x}, \boldsymbol{y})$, and let J be the set of acceptable outputs such that for every $j \in J$, $p_j = \boldsymbol{f}_{\boldsymbol{\theta}}(x)_j$ is maximal among the allowed output probabilities. Then the Gradient-update operation with $\mathcal{L} =$ NLL-loss from Equation (2) yields a limit distribution*

$$p_j = \begin{cases} \frac{1}{|J|} & \text{if } j \in J \\ 0 & \text{otherwise} \end{cases}$$

3. Notice, that for this example we will use the set notation for partial supervision.

Theorem 4 states that the model converges to a distribution in which all the probability mass is evenly distributed among a subset J of allowed outputs that initially had maximal probability. Under any realistic model and random initialization, there is a single allowed output with maximal initial probability, i.e., J is a singleton and all the probability mass converges to a single output. As we have illustrated in Example 6, this "winner-take-all" behavior is harmful, as it can prevent the optimizer from fitting to other points. The proof is provided in Appendix A.

Example 7 *Again, assuming arbitrary input dimension and $m = 3$ outputs: A, B and C, we examine the optimization dynamics of* NLL-loss *with a single sample $(\boldsymbol{x}, \{A, B\})$. Figure 3a visualizes the winner-take-all behavior of* NLL-loss *in this case. We see that the model converges to A or B depending on which one has greater initial probability.*

4.3 Loss Functions with the Probability Ratio Preserving (PRP) Property

Towards correcting this observed systematic bias of the NLL-loss, we present the following formalization of Postulate 1:

Definition 5 (PRP property) *Given a parametric model $\boldsymbol{f_\theta}$, a continuously differentiable function $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y})$ is said to satisfy the* Probability Ratio Preserving *(*PRP*) property for $\boldsymbol{f_\theta}$ if any Gradient-update on $\boldsymbol{f_\theta}$ with loss function $\mathcal{L}$ preserves the ratio of probabilities of all outputs i with $y_i = 1$.*

Given any loss function $\mathcal{L}$, whether it satisfies the PRP property depends on the model architecture. Theorem 4 demonstrates how NLL-loss introduces a strong preferential bias even for a basic softmax regression model. Therefore, we focus on this model class in our formal results:

Definition 6 ($\mathbf{PRP_s}$ property) *A continuously differentiable function $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y})$ is said to satisfy the* $\mathrm{PRP_s}$ *property if it satisfies the* PRP *property for the softmax regression model.*

There is a simple loss function that satisfies the $\mathrm{PRP_s}$ property. Since the loss function "balances" the probabilities of the different outputs, we refer to it as the Libra-loss.

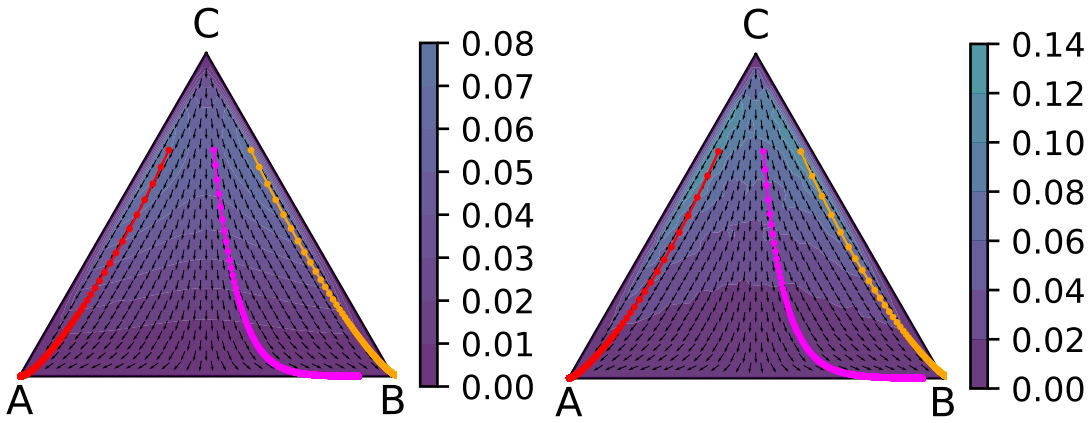
Definition 7 (Libra-loss) *Let* Libra-loss *denote the following function:*

$$\mathcal{L}_{\mathrm{Lib}}(\boldsymbol{p}, \boldsymbol{y}) = \underbrace{-\frac{1}{k}\sum_{i=1}^{m} y_i \log(p_i)}_{\text{Allowed term}} + \underbrace{\log\left(1 - \sum_{i=1}^{m} y_i p_i\right)}_{\text{Disallowed term}}$$

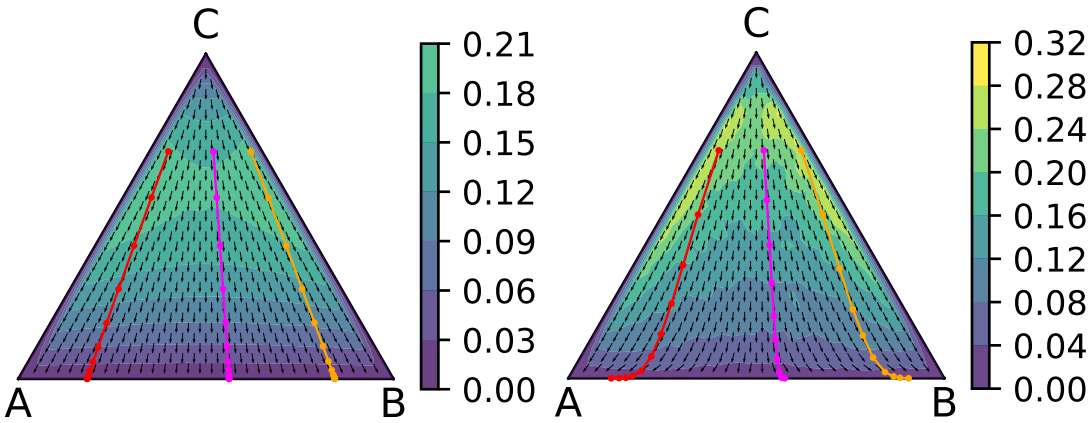
where $k = \sum_i y_i$ is the number of allowed outputs. The first term is the average of the individual negative log likelihood losses for each allowed output, while the second term is the positive log likelihood of selecting a disallowed label.

In Appendix B, we show that the Libra-loss has the desired property:

Theorem 8 *The* Libra-loss *function has the* $\mathrm{PRP_s}$ *property.*



(a) NLL-loss



(b) Libra-loss

Figure 3: Example 7: Given $m = 3$ outputs A, B, C and one sample $(\boldsymbol{x}, \{A, B\})$, we show the direction (arrows) and magnitude (colors) of gradient updates with (a) the NLL-loss, (b) the Libra-loss. We show a Softmax regression model (Left) and an MLP with 10 hidden layers (right). We show real trajectories from three fixed starting points: $(p_A, p_B, p_C) \in \{(0.25, 0.05, 0.7), (0.13, 0.17, 0.7), (0.05, 0.25, 0.7)\}$, terminated when $p_C < 0.0001$. NLL-loss makes the model converge to either A or B, while Libra-loss only slightly distorts the initial probability ratios between A and B. Also notice the increased speed: while NLL-loss with softmax regression took $(8374, 8671, 9906)$ steps to converge with fixed learning rate, Libra-loss took only $(14, 14, 14)$ steps.

Libra-loss only depends on the probabilities of allowed outputs and is invariant under permutation of the output vector. We formalize this property as:

Definition 9 (acceptable-dependent) *A loss function $\mathcal{L}$ is* acceptable-dependent *if its value only depends on the predicted probabilities p_i for allowed labels, i.e., for which label*

$y_i = 1$ and it is invariant under any permutation $\pi \in S_m$ of the coordinates of the arguments of $\mathcal{L}$. (i.e., $\forall \pi \in S_m, \mathcal{L}(\pi \circ \boldsymbol{p}, \pi \circ \boldsymbol{y}) = \mathcal{L}(\boldsymbol{p}, \boldsymbol{y})$).

In fact, when we restrict attention to acceptable-dependent functions, we do not have that much choice about how to satisfy the $\mathrm{PRP_s}$ property. We show that any acceptable-dependent loss function satisfying the property can be obtained from the Libra-loss.

Theorem 10 *Let $\mathcal{L}$ be an acceptable-dependent function on m outputs that has the $\mathrm{PRP_s}$ property. Then there exists a function $h : \mathbb{R} \times [m] \to \mathbb{R}$ that is continuously differentiable in its first argument such that $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y}) = h(\mathcal{L}_{\mathrm{Lib}}(\boldsymbol{p}, \boldsymbol{y}), k)$ where $k = \sum_i y_i$.*

Theorem 10 is a central result of our work. It gives a characterization of all acceptable-dependent functions that have the $\mathrm{PRP_s}$ property, showing that any such function can be constructed as $\mathcal{L}_{\mathrm{Lib}}$ composed with a continuously differentiable function $h_k : \mathbb{R} \to \mathbb{R}$, possibly using different functions for different number of allowed inputs. Because this theorem is one core of our work, we add a quick intuitive proof sketch.

Proof [Proof sketch] The core of the argument considers an arbitrary loss function $\mathcal{L}$ with the $\mathrm{PRP_s}$ property and a real value z, and shows that on the set H_z of values where $\mathcal{L}_{\mathrm{Lib}} = z$, $\mathcal{L}$ is constant: thus $\mathcal{L}$ is a function of $\mathcal{L}_{\mathrm{Lib}}$, and once this is proven it is easy to show that the function is smooth. To prove that the function is constant, we first argue that H_z is path connected. We then fix two points a and b in H_z let γ be a path in H_z connecting them, and write $\mathcal{L}(b) - \mathcal{L}(a)$ as a line integral of the gradient of $\mathcal{L}$ over that path. We show that the gradient of $\mathcal{L}$ is a constant multiple of the gradient of $\mathcal{L}_{\mathrm{Lib}}$. But since $\mathcal{L}_{\mathrm{Lib}}$ is constant on H_z, hence constant on γ, its gradient must be 0. We have thus shown that $\mathcal{L}(b) - \mathcal{L}(a)$ is 0 as required. Details are in Appendix B. ■

Example 8 (Example 7 continued) *We see in Fig. 3b that in our simple example of 3 possible outputs and a single sample $(\boldsymbol{x}, \{A, B\})$ the model does not necessarily converge to either A or B, as it did for the* NLL-loss. *In fact, in the case of softmax regression (leftmost plot), the update operations strictly preserve the initial probability ratios between A and B. In the more general case, the output with the greater initial probability increases only moderately faster and only at the very end of training.*

4.4 Increasing Model Complexity

Our results about the winner-take-all property of the NLL-loss and the PRP property of Libra-loss assume a softmax regression model. This can be seen in Fig. 2, where we show learning curves during training of a single layer: NLL-loss makes the model prediction collapse into a single output, while Libra-loss guarantees to keep to the initial probability ratios of allowed outputs. For more complicated networks with hidden layers, our results become approximations. However, any model that ends with a softmax will have to deal with a biased gradient flow for NLL-loss. The backwards propagated error will be greatest for the preferred output, i.e., weights deeper in the network will change more if they affect that output than for other outputs. This is corrected by Libra-loss, even if no exact statement can be made about how the probability ratios change after the update. In Fig. 3 we

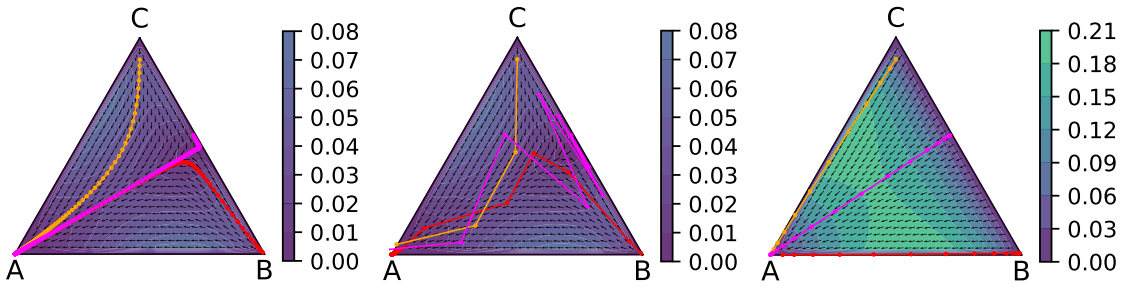
experimentally observe how probability ratios change in a larger network. We find that the update dynamics remain mostly unchanged as we increase the model complexity and the training trajectories indeed do not converge towards one or the other side.

Example 9 (Example 6 continued) *Let us return to the slightly more complex example where the same input is associated with two consistent label sets:* $(\boldsymbol{x}, \{A, B\})$ *and* $(\boldsymbol{x}, \{A, C\})$*. The update dynamics of this setting for a softmax regression model is depicted in Fig. 4. For both* NLL-loss *and* Libra-loss*, label A constitutes the single attractor. However, in the case of* NLL-loss*, reaching A can take a long time when we start with very low probability assigned to A and it can even lead to oscillation between B and C if the learning rate is not sufficiently small. On the other hand,* Libra-loss *yields a smooth trajectory to A from any starting configuration.*

Example 10 (Example 9 continued) *We run simulations of the entire training process on a 2-layer MLP with random adversarial starting configurations on the same problem from Example 9: we train the model for 20 steps to approach the B-C line (using adversarial sample* $(\boldsymbol{x}, \{B, C\})$ *and then train it for another 200 steps with the two correct samples* $(\boldsymbol{x}, \{A, B\})$ *and* $(\boldsymbol{x}, \{A, C\})$*. With only three outputs* ($m = 3$) *as before, we find that both losses still make the model converge to A (although the* NLL-loss *also still takes longer as before (Fig. 4). However, as we increase the output size m while keeping the samples (i.e., we add unrelated disallowed outputs) bad local optima emerge and it becomes harder for* NLL-loss *to find the optimum. With* $m = 10$ *possible outputs,* NLL-loss *misses the optimum* 20% *of the time (based on 30 trials) and when there are 100 outputs, it misses the output* 100% *of the time (based on 10 trials). In the meantime, the* Libra-loss *robustly learns to select output A that satisfies both points. Figure 5 shows typical learning curves for 100 outputs.*

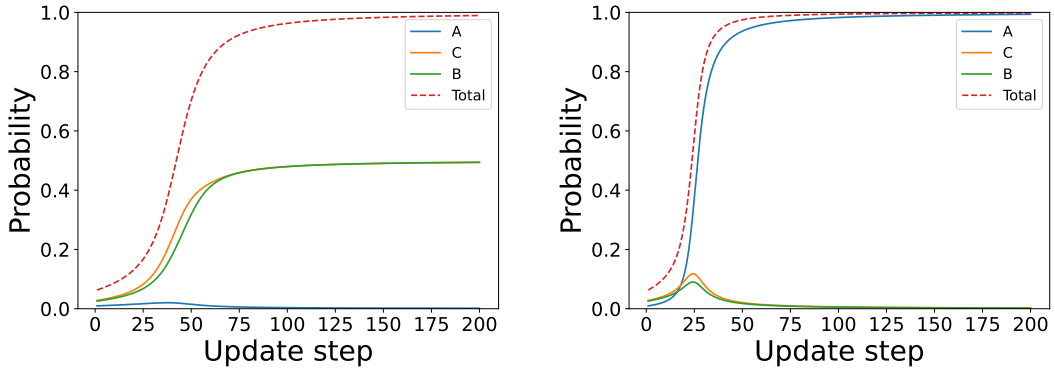
4.5 Connection with Negative Log Likelihood Loss

A closer inspection of the Libra-loss Definition 7 reveals that it is a combination of several different log likelihood losses. The disallowed term $\log(1 - P_{\text{acc}})$ is the *positive* log likelihood of selecting a disallowed output. It has the same monotonicity and optimum as the NLL-loss for PLL (Equation (2)). However, what is different is its convexity: the more the model fits to a sample (i.e. the higher the sum of allowed probabilities P_{acc}), the flatter the NLL-loss becomes. On the other hand, $\log(1 - P_{\text{acc}})$ becomes steeper as we start fitting the sample. This curvature, however, is compensated by the allowed term $-\frac{1}{k}\sum_i y_i \log(p_i)$, which is the average of the individual negative log likelihood losses for each allowed output.
In general, loss components that reward the log probability of allowed outputs (such as NLL-loss or the allowed term of Libra-loss) will have vanishing gradients when we are close to fitting the allowed labels. Analogously, loss components that penalize the log probability of disallowed outputs (such as the disallowed term in Libra-loss) will have vanishing gradients when we are far from fitting the allowed labels. Libra-loss provides a "perfect" balance between these two kinds of components. The gradients are stable throughout the optimization, and we show in Appendix B that they are always $-\frac{1}{k}$ for the allowed outputs. Figure 6 visualizes the balancing effect of the Libra-loss for the classical supervised case with $k = 1$ and a single allowed output. In this case Libra-loss reduces to the *log inverse odds*



(a) NLL-loss, $LR = 1$ (b) NLL-loss, $LR = 15$ (c) Libra-loss, $LR = 1$

Figure 4: Example 9: Given $m = 3$ outputs A, B, C and two samples $(\boldsymbol{x}, \{A, B\})$, $(\boldsymbol{x}, \{A, C\})$, we show how softmax regression updates the probabilities. **(a):** NLL-loss with small learning rate may take a long time to reach the attractor. **(b):** NLL-loss with large learning rate can lead to oscillation. **(c):** Libra-loss, in contrast, heads directly towards the attractor. We show three real trajectories from fixed starting points: $(p_A, p_B, p_C) \in \{(0.003, 0.99, 0.007), (0.05, 0.05, 0.9), (0.01, 0.44, 0.55)\}$. The trajectories are terminated when $p_A > 0.9999$.



(a) NLL-loss (b) Libra-loss

Figure 5: Example 10: Learning curves using a classifier MLP with two layers and $m = 100$ outputs on a training set that consists of two samples: $(\boldsymbol{x}, \{A, B\})$, $(\boldsymbol{x}, \{A, C\})$, as described in Example 6. While A is the optimal output, initially, B and C have higher probabilities. *Total* shows $A + B + C$, i.e., the sum of probabilities of outputs in occurring in some label set. Starting from identical initialization, NLL-loss (a) gets stuck in a bad local optimum, while Libra-loss (b) quickly recovers the global optimum.

ratio. Let p_{true} denote the probability of the single allowed output, and thus $p_{\text{true}} = \sum_i y_i p_i$.

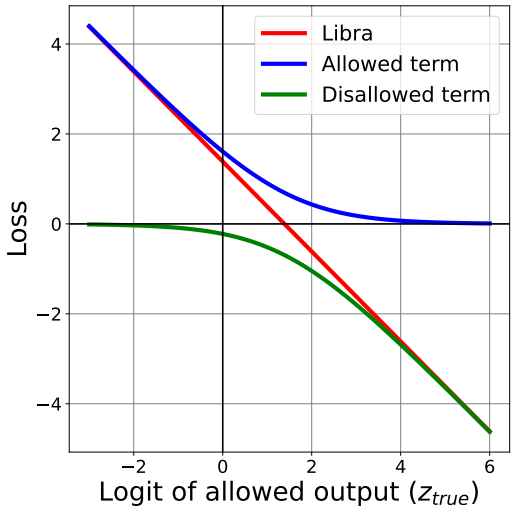


Figure 6: Section 4.5: Libra-loss and its two terms when there is a single allowed output, plotted against the single allowed logit z_{true}. The derivatives of the two terms add up to -1 (i.e., the gradient of Libra-loss is independent from the model prediction). This implies the PRP property. **Blue line (Allowed term)**: negative log likelihood of the allowed output ($-\log(p_{\text{true}})$, NLL-loss), **Green line (Disallowed term)**: positive log likelihood of the disallowed outputs ($\log(1-p_0)$), **Red line**: Libra-loss

Then the Libra-loss becomes

$$\mathcal{L}_{\text{Lib}} = \underbrace{-\log(p_{\text{true}})}_{\text{allowed}} + \underbrace{\log(1-p_{\text{true}})}_{\text{disallowed}} = \log\left(\frac{1-p_{\text{true}}}{p_{\text{true}}}\right)$$

The derivative of the loss with respect to the single allowed logit z_{true} is:

$$\begin{aligned}\frac{\partial \mathcal{L}_{\text{Lib}}}{\partial z_{\text{true}}} &= \frac{\partial \log(1-p_{\text{true}})}{\partial z_{\text{true}}} + \frac{\partial -\log(p_{\text{true}})}{\partial z_{\text{true}}} = \left(\frac{-1}{1-p_{\text{true}}} + \frac{-1}{p_{\text{true}}}\right)\frac{\partial p_{\text{true}}}{\partial z_{\text{true}}} \\ &= \frac{-p_{\text{true}} - 1 + p_{\text{true}}}{p_{\text{true}}(1-p_{\text{true}})} p_{\text{true}}(1-p_{\text{true}}) = -p_{\text{true}} - 1 + p_{\text{true}} = -1\end{aligned}$$

We obtain that the Libra-loss is linear in z_{true}, with constant derivative -1.

4.6 Connection with Entropy Regularization

Libra-loss implements a special, input-dependent form of entropy regularization (Pereyra et al., 2017), whose intuitive goal is to penalize distributions with low entropy. However, Libra-loss applies entropy regularization *only over the allowed outputs*. As we have already seen, the disallowed term $\log(1-P_{\text{acc}})$ aims to minimize the likelihood of disallowed labels. The allowed loss term can be rewritten as

$$-\frac{1}{k}\sum_i y_i \log(p_i) = -\sum_{\{i|y_i=1\}} \frac{1}{k}\log(p_i) = H(U_{\boldsymbol{y}}, \boldsymbol{p})$$

where $U_{\boldsymbol{y}}$ is the uniform distribution over the k allowed outputs and $H(U_{\boldsymbol{y}}, \boldsymbol{p})$ is the cross entropy of $U_{\boldsymbol{y}}$ relative to $\boldsymbol{p}$. This rewriting shows that the allowed term is a cross entropy loss, measuring the distance between $U_{\boldsymbol{y}}$ and the model output distribution $\boldsymbol{p}$. Minimizing this term is equivalent to entropy regularization (i.e. maximizing entropy), restricted to the allowed outputs. In other words, it is minimal when $\boldsymbol{p}$ is uniform on the allowed outputs and zero elsewhere.

4.7 Preserving both Acceptable and Unacceptable Inputs

The reader may have noticed that the PRP property requires that the loss is acceptable-dependent (Definition 9). It enforces constraints which concern preservation of ratios between outputs, but it does this only on the acceptable outputs. It is natural to drop the first requirement, allowing dependence on all outputs, but replacing the constraints with a stronger property that is symmetric in acceptable and unacceptable outputs. We give this analog of the PRP property below:

Definition 11 (bi-PRP property) *Given a parametric model $\boldsymbol{f_\theta}$, a continuously differentiable function $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y})$ is said to satisfy the* bi-PRP property *for $\mathcal{M}$ if any Gradient-update with loss function $\mathcal{L}$ preserves the ratio of probabilities of all outputs i with $y_i = 1$, and also the ratio of probabilities of outputs with $y_j = 0$.*

As before, we focus on the bi-PRP property for a softmax regression model:

Definition 12 (bi-PRP$_\mathbf{s}$ property) *A continuously differentiable function $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y})$ is said to satisfy the* bi-PRP$_\mathrm{s}$ property *if it satisfies the* bi-PRP *property for the softmax regression model.*

Again, we demonstrate that the property is not vacuous. We define a loss function that performs "balancing" on both the acceptable and unacceptable loss. Contrasting with the Libra-loss, we call this the *Sagittarius loss*, abbreviated Sag-loss.

Definition 13 (Sag-loss)

$$\mathcal{L}_{\mathrm{Sag}}(\boldsymbol{p}, \boldsymbol{y}) = \underbrace{-\frac{1}{k}\sum_i y_i \log(p_i)}_{\text{Allowed term}} + \underbrace{\frac{1}{n-k}\sum_i (1-y_i)\log(p_i)}_{\text{Disallowed term}}$$

The allowed term is identical to that of Libra-loss*, i.e., it is the average of the individual negative log likelihood losses for each allowed output. The disallowed term is the average of the individual positive log likelihood losses for each disallowed output. Also notice that both terms can be seen as cross entropies of $\boldsymbol{p}$ relative to uniform distributions on the 1) allowed outputs (allowed term) and 2) disallowed outputs (disallowed term).*

We can show that $\mathcal{L}_{\mathrm{Sag}}$ has the bi-PRP$_\mathrm{s}$ property:

Theorem 14 *The* Sag-loss *function has the* bi-PRP$_s$ *property and for any continuously differentiable family of* $h_i : \mathbb{R} \to \mathbb{R}$ *functions* $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y}) = h_k(\mathcal{L}_{\text{Sag}}(\boldsymbol{p}, \boldsymbol{y}))$ *also satisfies the* bi-PRP$_s$ *property, where* $k = \sum_i y_i$.

Furthermore, we get a characterization analogous to the one of Theorem 8.

Theorem 15 *Let* $\mathcal{L}$ *be a function that has the* bi-PRP$_s$ *property, invariant under the permutation of the input (i.e.,* $\forall \pi \in S_n, \mathcal{L}(\pi \circ \boldsymbol{p}, \pi \circ \boldsymbol{y}) = \mathcal{L}(\boldsymbol{p}, \boldsymbol{y})$*). Then there exist* $h_i : \mathbb{R} \to \mathbb{R}$ *continuously differentiable functions such that* $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y}) = h_k(\mathcal{L}_{\text{Sag}}(\boldsymbol{p}, \boldsymbol{y}))$.

The proofs are provided in Appendix C.

4.8 Comparing Libra-loss and Sag-loss

The Libra-loss and the Sag-loss have many similarities and are strongly related to NLL-loss. They both factorize into an allowed and a disallowed term, and the allowed terms are identically $H(U_{\boldsymbol{y}}, \boldsymbol{p}) = -\frac{1}{k} \sum_i y_i \log(p_i)$: the cross entropy of $\boldsymbol{p}$ relative to the uniform distribution on the allowed outputs, which is also the average of the individual negative log likelihood losses for each allowed output. They differ in the disallowed term. For Libra-loss it is $\log(1 - P_{\text{acc}})$ the positive log likelihood of selecting a disallowed output, while for Sag-loss it is $\frac{1}{n-k} \sum_i (1 - y_i) \log(p_i) = H(U_{\bar{\boldsymbol{y}}}, \boldsymbol{p})$ the cross entropy of $\boldsymbol{p}$ relative to the uniform distribution on the disallowed outputs $U_{\bar{\boldsymbol{y}}}$, or equally the average of the individual positive log likelihood losses for each disallowed output.
The bi-PRP property implies the PRP property, hence the Sag-loss satisfies the PRP$_s$ property. At this point the reader may expect that the Sag-loss, having a stronger property, should be superior to Libra-loss. Surprisingly, we will explain in Section 6 that this is not the case: the need to retain balance on both acceptable and unacceptable outputs leads to some undesirable practical effects which affect learning. In particular, the magnitude of the logit vector increases rapidly during learning, leading to numerical instability.

5. Learning Mapping Rules via Partial Label Learning

We introduce new sequential data sets with partial supervision. Extending Example 1 in the introduction, these data sets will concern learning rules, a topic that has gained considerable interest in the AI community (e.g. Evans and Grefenstette, 2018; Rocktäschel and Riedel, 2017; Qu et al., 2021). More specifically, we consider learning *mapping rules* which relate data sources in a *source vocabulary* into some *target vocabulary*. This is a common approach in data integration, where the target vocabulary is often standardized (an "ontology" W3C, 2012), optionally equipped with additional logical constraints. Although there are a vast number of tools available for answering queries with known rules, determining the mapping rules by hand is known to be a difficult even with domain expertise (Pinkel et al., 2015). Thus a key challenge is to learn the mapping rules from supervision on the target vocabulary—we know some tuples $\mathbf{t}$ that should or should not be inferred in the target vocabulary, called *positive* and *negative* facts.

Example 11 (Example 1 continued) *Consider source relation* `Person`*, target relation* `Author` *and the following facts:* $Person(alice, 45, 1)$*,* $Person(bob, 34, 1)$*,* $Person(joe, 23, 2)$*,* $Person(lola, 12, 2)$*. Supervision might consist of:*

Positive: `Author(alice)`, `Author(bob)`
Negative: `Author(joe)`, `Author(lola)`

In Example 11, we are looking for mapping rules between source relation `Person` and target relation `Author` that allow for deducing that Alice and Bob are authors, and that cannot be used to prove that Joe and Lola are authors.
The number of possible mapping rules is generally large, much larger than what can be enumerated. In data integration, there are typical patterns in which the source and the target may differ, and when domain experts construct mappings by hand, they tend to try these typical patterns to find the one that fits the task at hand. We formalize these patterns as *mapping templates*, e.g.:

$$T(x) \leftarrow S_1(x) \wedge S_2(x, y) \wedge S_3(y)$$

where variables in the head (the variable x above) are universally quantified and the rest of the variables (the variable y in the example above) are existentially quantified. T and S_1, S_2, S_3 are template variables over predicate names in the target and source language, respectively. Any instantiation of template variables yields a mapping rule. Such templates are assumed in most prior work in the area (e.g. Evans and Grefenstette, 2018; Rocktäschel and Riedel, 2017). In particular, we support mapping templates of the form

$$H(x_1 \dots x_k) \leftarrow \bigwedge_{i \leq b} C_i(\mathbf{y}_i), \bigwedge_{j \leq k} x_i = \tau_i$$

Mapping rules are formed by replacing template variable H by a target predicate, template variables C_i by source predicates and variables y_i by either variables or source constants. The terms τ_i are formed from applying string concatenation to either variables or strings.

Example 12 *To illustrate the usage of string concatenation, we provide a real mapping rule from the NPD challenge (to be described below).*

$\text{Agent}(x) \leftarrow$
$C_1(y_1) \wedge C_2(y_2) \wedge C_3(y_3) \wedge C_4(y_4) \wedge$
$x = CONCAT(\textit{http://sws.ifi.uio.no/data/npd-v2/baa/}, y_1, \textit{/licensee/}, y_2, \textit{/history/}, y_3, /, y_4)$

This mapping rule aligns target concept Agent. In the target language, agents are represented as URL strings. Components of these strings are fixed for all agents, such as the prefix http://sws.ifi.uio.no/data/npd-v2/baa/. Other parts are derived from four source predicates C_1, C_2, C_3, C_4. The source predicates correspond to columns in the database—we omit their description in the example.

We assume that mapping templates $\mathbb{MT}$ are provided by domain experts. Our task is to find a subset $\mathbb{M}$ of the instantiations of the templates such that the source database instance $I_{\mathbb{S}}$ and the mapping rules $\mathbb{M}$ together imply all the positive facts $\mathbb{P}$ and none of the negative facts $\mathbb{N}$. When an exact solution is not achievable, we can also consider a relaxation of the problem, i.e., we want to cover "as many as possible" of the positive facts and "as few as possible" of the negative facts.

The number of possible rules is infinite, due to the number of possible strings in concatenation terms. However, we will only be interested in the rules that can produce a given target fact. For any positive or negative fact F and mapping template $MT \in \mathbb{MT}$, we define the *candidates* of F with respect to MT to be the set of all instantiations M of MT such that M, together with the source instance implies F:

$$\text{candidates}(F, MT) = \{M | M \in MT, (M \wedge I_{\mathbb{S}} \vDash F)\}$$

Taking into account that the source database is finite, candidates(F, MT) is a finite set and typically small enough so that it can be obtained via preprocessing.

Example 13 (Example 1 cont.) *In our example, the rules that derive* `Author(alice)` *are:* $R_0 = Author(x) \leftarrow \exists a, t.Person(x, a, t)$, $R_1 = Author(x) \leftarrow \exists a.Person(x, a, 1)$, $R_2 = Author(x) \leftarrow \exists t.Person(x, 45, t)$. R_0 *and* R_1 *also derive* `Author(bob)` *while* R_2 *does not. Let* R_3 *be some other rule that only derives* `Author(bob)` *and* R_4, R_5, R_6 *some rules that derive negative facts* `Author(joe)` *and* `Author(lola)` *We obtain the following candidate sets:*

`Author(alice)`	$\{R_0, R_1, R_2\}$	`Author(bob)`	$\{R_0, R_1, R_3\}$
`Author(joe)`	$\{R_0, R_4, R_5\}$	`Author(lola)`	$\{R_0, R_4, R_6\}$

In Example 13, each fact has three candidates and we have seven rules in total. R0 proves all facts, R1 proves all positives and none of the negatives, R2 and R3 prove some of the positives, R4 proves all the negatives, R5 and R6 prove some negatives. Clearly, R1 is the optimal choice as a single rule.

Let us consider a function $\boldsymbol{f_\theta}$: fact $\rightarrow$ rule that assigns to each fact a correct mapping rule. Approximating this function via learning can greatly reduce the labor cost of data integration. Given a set of positive facts $\{\mathbb{P}^{(i)}\}_{i=1}^{n_p}$ we can compute the corresponding candidate rule sets $\{\text{candidates}(\mathbb{P}^{(i)}, MT)\}_{i=1}^{n_p}$, which together constitute a partially labelled data set for learning $\boldsymbol{f}$.

Analogously, we can use negative facts $\{\mathbb{N}^{(i)}\}_{i=1}^{n_n}$ to extract a *negative* partially labelled data set for learning $\boldsymbol{f}$. Negative supervision represents global constraints and requires special treatment. Given a negative sample $(\boldsymbol{x}, \boldsymbol{y})$, the labels in $\boldsymbol{y}$ are explicitly forbidden for *any* input. Theoretically, this is equivalent to a partial labelling that excludes globally these outputs, however, producing complementer sets of forbidden label sets can be problematic in practice when the output space is large. Let $A_{neg} = \{\boldsymbol{y} | (\boldsymbol{x}, \boldsymbol{y})$ is a negative example$\}$ be the set of all label sets that appear in some negative example. Given a loss function $\mathcal{L}$ for positive partial supervision, we introduce a new loss term $\mathcal{L}_{neg}(\boldsymbol{p}) = \sum_{\boldsymbol{x} \in A'_{neg}} -\mathcal{L}(\boldsymbol{p}, \boldsymbol{x})$ where $A'_{neg} \subseteq A_{neg}$ is 50 samples selected uniformly at random from A_{neg} for each update step. This term quantifies the extent to which negatives are violated and it is, weighted by a hyperparameter γ[4] added to the loss function:

$$\mathcal{L}'(\boldsymbol{p}, \boldsymbol{x}) = L(\boldsymbol{p}, \boldsymbol{x}) + \gamma \mathcal{L}_{neg}(\boldsymbol{p})$$

Recall that the input space is the set of all possible atoms expressible in the target language, while the output space is all possible mapping rules. Although there are only finitely

4. γ represents the tradeoff between fitting to positive and negative datapoints.

Distortion	Description
renaming	Classes and properties have different names in the ontology and the database
cleaning	Foreign keys in the database are removed, making it harder to join tables.
restructuring	Class hierarchies are represented using attributes indicating subclass membership.
denormalizing	Correlated information is jointly stored in the same table, redundantly.

Table 3: Synthetic distortions applied to make the alignment task harder.

many options when conditioned on the supervision and the source database—when we only consider mapping rules that derive some fact—even in this case the output space remains huge. It can easily reach hundreds of thousands of rules. Directly training a model with so many outputs is challenging and such an approach would neglect similarities across rules. For this reason, we instead represent inputs and outputs as text, i.e., as sequences of tokens, yielding a sequence-to-sequence language modeling task with partial supervision. As discussed in Section 3, autoregressive models can be used to model problems with sequential outputs: model predicted probabilities $\boldsymbol{p}$ can be calculated in a sequence of evaluations. Consequently, any loss function that takes $\boldsymbol{p}$ and label $\boldsymbol{y}$ as input can be applied directly, without modification for optimization, independent of the architecture. In the following we describe the novel data sets that we extracted from the RODI benchmark and that are used in the experiments presented in Section 6.4 to compare various loss functions.

5.1 RODI Challenges

The RODI data set was introduced in Pinkel et al. (2015, 2018) as a benchmark for systems that integrate a set of source relational schemas into a target graph schema. Each challenge provides a target schema consisting of unary and binary relations and a source relational database. The task is to find mapping rules that define concepts in the target using query expressions over the source database.

The challenges are synthetically generated starting from an instance of the target schema, generating a source schema. The target schema consists of binary relations (*properties*) and unary relations (*classes*). The source schema generation involves one or a combination of typical—real life inspired—distortions that make the alignment nontrivial. For competition purposes, RODI provides the target schema (without data), the source data, and a list of *translation pairs* (source query, target query) that can be used for evaluation. In each pair, one is a SPARQL (Prud'hommeaux and Seaborne, 2008) query against the target and the other is an SQL query against the source database. In case of correct mapping, the two queries have to return the same result. The target schemas (*ontologies*) are based on three conference management systems: CMT, SIGKDD and CONFERENCE. RODI uses the distortions described in Table 3 (see Pinkel et al., 2015, for more details):

For each predicate of each challenge, we sample n positive tuples that satisfy the predicate and n negative tuples that do not satisfy it. The positive tuples are sampled uniformly from the tuples returned by the provided SQL query for that predicate. For sampling negatives, we use random constants for each tuple position, selected uniformly from the constants of the database with matching type and ensuring no overlap with the positives.

We obtain 5 data sets for each domain (one without distortion and four with one of the above distortions) that contain 1500-2000 positive samples and a maximum of 55 candidates for each input. We find that the different domains yield no new insights and preliminary experiments suggest similar performance. Hence, we focus on the CMT system and experiment with the 5 challenges associated with it in Section 6.4. Our distribution contains the extracted CMT data sets, as well as code to generate data sets for any domain.

5.2 NPD Challenge

Besides the synthetically generated challenges, Pinkel et al. (2015) provide a real world data set related to the Norwegian Petroleum Directorate (NPD) FactPages (Skjæveland et al., 2013). The source data and the target schema were constructed from publicly available data and the translation pairs were built from real use cases from end users of the FactPages. The source database contains 40MB data and has a rather complex structure with 70 tables, 1000 columns and 350 foreign keys. The target schema has 300 classes and 350 properties. Existing tools (e.g. Jiménez-Ruiz et al., 2015; Pinkel et al., 2013) for this task rely completely on the structure of the source and target, and are unable to infer any relationships in a challenge like this.

Positive facts are sampled uniformly, just like for RODI. For sampling negatives, however, we find that uniform sampling yields facts that have extremely small probability of being provable by the rules required to prove positives, making it rather easy to avoid negatives. This is because the rules required to align NPD are much more complex than those for RODI. For this reason, negative tuples are sampled uniformly not from the entire database, but only from constants appearing in positive tuples of other predicates. We observe that this way of sampling negatives makes aligning NPD harder, since many of the candidates of positive facts have to be eliminated as they also prove some of the negatives.

We end up with a data set consisting of 34965 positive facts, using 421 target predicates. Over 98% of the facts have less than 1000 candidates and we truncate the set of allowed candidates to 1000 for computational reasons. Our distribution includes the extracted data set, as well as code to generate a new data set.

6. Experiments

Our experiments aim to provide a quantitative overview of how different loss functions perform on learning from partially labelled data, as well as to demonstrate the practical benefit of the newly introduced Libra-loss. We employ three types of data sets:

1. Synthetic inputs, synthetic outputs: These experiments, presented in Section 6.2, examine extremely simple scenarios aimed at highlighting failure cases of various loss functions.
2. Real inputs, synthetic outputs: This is the setup typically used to evaluate PLL methods in the literature. We present two experiments in Section 6.3 based on the CIFAR10 and CIFAR100 data sets.
3. Real inputs, real outputs: This is the most challenging and most important scenario. We experiment with a novel rule learning data set in Section 6.4, as well as a collection of standard benchmarks for PLL in Section 6.5.

Before moving on to the experiments, we provide an overview of the loss functions from the literature that we use as competitors in Section 6.1. We end the section with a discussion of the results in Section 6.6.

6.1 Alternative Methods

We overview the alternative approaches from the literature we compete with in the following experiments.

6.1.1 Negative Log Likelihood Loss (NLL)

The NLL-loss, defined in Equation (2) is the standard example of an average-based loss that appears in the literature on PLL, often under different names. For example, it is called the *maximum marginal likelihood (MML)* loss in Guu et al. (2017) and the *classifier consistent (CC)* loss in Feng et al. (2020). We repeat the definition:

$$\mathcal{L}_{\mathrm{NLL}}\left(\boldsymbol{p}, \boldsymbol{y}\right) = -\log\left(\boldsymbol{p} \cdot \boldsymbol{y}\right) = -\log\left(\sum_i y_i p_i\right) = -\log(P_{\mathrm{acc}})$$

6.1.2 Uniform Loss

A very simple baseline is to compute the negative log likelihood of each allowed output and optimize their sum:

$$\mathcal{L}_u(\boldsymbol{p}, \boldsymbol{y}) = -\sum_i y_i \log(p_i)$$

This is an average-based method and it differs from the allowed term of the Libra-loss only by a multiplicative factor of $\frac{1}{k}$. This loss has a single minimum, when the prediction is uniform on the allowed outputs and zero elsewhere. We refer to this as Uniform-loss.

6.1.3 β-Meritocratic Loss

Recall that Guu et al. (2017) considers the semantic parsing application of PLL, overviewed in Example 2. They propose the β-merit-loss:

$$\mathcal{L}_{\beta\mathrm{M}}(\boldsymbol{p}, \boldsymbol{y}) = -\sum_i w(\beta)_i \log(p_i),$$

where each output i is associated with a weight $w(\beta)_i = \frac{y_i \cdot (p_i / P_{\mathrm{acc}})^\beta}{\sum_q y_q \cdot (p_q / P_{\mathrm{acc}})^\beta}$.
A technical caveat is that the dependence of the $w(\beta)_i$ weights on the model output is disregarded during optimization, i.e., no gradients are propagated through them. This holds for all other w_i weights introduced below.
Notice, that the β parameter provides one possible smooth interpolation between two losses: NLL-loss $= -\log\left(\sum_i y_i p_i\right)$ and Uniform-loss $= -\sum_i y_i \log(p_i)$. More specifically, the β-merit-loss has the same gradient as NLL-loss when $\beta = 1$, since the denominator of $w(\beta)_i$ becomes 1 thus can be ignored. On the other hand, β-merit-loss with $\beta = 0$ is equivalent to Uniform-loss. All three losses focus solely on the probabilities of the acceptable outputs, since $w(\beta)_i = 0$ when $y_i = 0$. Guu et al. (2017) observes that while there is no

universal β across data sets, tuning this hyperparameter can greatly increase convergence speed and slightly improve final accuracy. In our experiments, we report the extreme values as Uniform-loss and NLL-loss and let β-merit-loss refer to the best performing β for the given task from the set $\{0.25, 0.5, 0.75\}$.
Note that this interpolation is similar in spirit to the Libra-loss, which has two terms: one similar to NLL-loss and has the winner-take-all property, while the other is an entropy regularizer and pushes the probabilities towards uniform distribution. What is different is that Libra-loss does not have an extra β parameter: the strength of the two loss terms depends implicitly on how well the model fits to the sample.

6.1.4 Leverage-Weighted Loss (LWS)

Wen et al. (2021) introduces *leverage weighted* loss, as a family of loss functions based on the unnormalized model outputs or logits $\boldsymbol{z}$. They focus on the following loss:

$$\mathcal{L}_{LW}(\boldsymbol{z}, \boldsymbol{y}) = \sum_i y_i w_i \sigma(z_i) + \beta \sum_i (1 - y_i) w_i \sigma(-z_i)$$

where $\sigma(t) = \frac{1}{1+e^t}$. The loss has two terms, one for allowed outputs ($y_i = 1$) and one for disallowed outputs ($y_i = 0$) and the leverage hyperparameter β controls their relative importance. The authors achieve best empirical results with $\beta = 1$ most of the time and sometimes with $\beta = 2$. The results presented in our experiments use the best performing value from $\{0.5, 1, 2\}$, which turns out to be $\beta = 1$ in all cases.
Each output i is associated with an input dependent weight w_i, which is defined as the likelihood assigned to the output by the model, normalized so that weights for allowed and disallowed outputs both add up to one:

$$w_i = \begin{cases} \frac{e^{z_i}}{\sum_j y_j e^{z_j}} & \text{if } y_i = 1 \\ \frac{e^{z_i}}{\sum_j (1-y_j) e^{z_j}} & \text{if } y_i = 0 \end{cases}$$

This is a typical identification-based loss: the model predicted w_i values are used to "identify" how much an allowed/disallowed output should be rewarded/penalized for fitting. We refer to this as LWS-loss.

6.1.5 Risk-Consistent Loss (RC)

A similar identification-based approach is provided in Feng et al. (2020), using loss function

$$\mathcal{L}_{RC}(\boldsymbol{p}, \boldsymbol{y}) = -\frac{1}{2} \sum_i y_i w_i \log(p_i)$$

For each allowed output i the negative log likelihood loss ($-\log(p_i)$) is weighted by

$$w_i = \frac{p_i}{\sum_j y_j p_j}$$

which is the model predicted probability of output i, normalized to the allowed outputs. We refer to this as RC-loss.

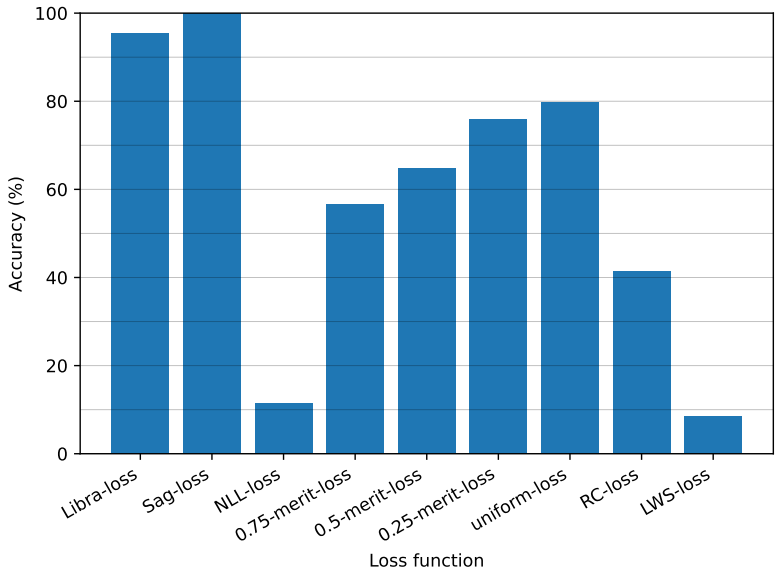


Figure 7: Average training accuracy on the small consistent data set over 1000 random initializations for various loss functions. Libra-loss and Sag-loss almost always find the optimal solution (95.5% and 99.9%), while NLL-loss, RC-loss and LWS-loss perform extremely poorly on this task. β-merit-loss alleviates this weakness and reaches 79.7% in the extreme case of $\beta = 0$ (Uniform-loss).

6.2 Synthetic Experiments

We provide two experiments with synthetic data sets.

6.2.1 Small Consistent Synthetic Data Set

Recall that Example 6 presented an extremely simple situation with $m = 3$ outputs and $n = 2$ samples with the same input $\boldsymbol{x}$: $(\boldsymbol{x}, \{A, B\})$ and $(\boldsymbol{x}, \{A, C\})$, i.e., each sample having $k = 2$ allowed outputs. To scale this example up, let us consider a problem with $m = 100$ possible outputs and a data set of $n = 10$ samples, each having the same input vector $\boldsymbol{x}$ and $k = 10$ allowed outputs. In each of the 10 samples $\boldsymbol{y}^{(1)}, \dots \boldsymbol{y}^{(10)}$ allows output o_0 together with 9 different values from among $o_1 \dots o_{10}$.[5] In this data set, there are 10 outputs $o_1 \dots o_{10}$ that are "almost good" in the sense that they are acceptable for 9 out of 10 samples and there is a single output o_0 that is acceptable in all samples. Hence, the only consistent solution is to select o_0. This example highlights the challenge of identifying the correct label when some alternative label has a large "support", i.e., when it is acceptable by many samples, while not all of them.

5. E.g. $\boldsymbol{y}^{(1)} = \{o_0, o_2, o_3, \dots, o_{10}\}$, $\boldsymbol{y}^{(2)} = \{o_0, o_1, o_3, \dots, o_{10}\}$, $\dots \boldsymbol{y}^{(10)} = \{o_0, o_1, o_2, \dots, o_9\}$.

Network For each loss considered, we train an MLP with a single hidden layer of 50 neurons on this data set, with 1000 different random seeds. We employ Glorot (Glorot and Bengio, 2010) initialization.

Results We report average accuracy on the training set in Figure 7. The models trained with Libra-loss and Sag-loss robustly find the output that is consistent with all samples. However, for the other losses this is often not the case. Depending on random initialization, some of the suboptimal outputs can have higher initial probability, resulting in them getting greater gradients, even greater than o_0 which is promoted by all samples during optimization. This behavior arises when the strongly "supported", yet suboptimal, output has higher initial probability than the single optimal output.

The worst performers are NLL-loss and the two identification-based approaches: RC-loss and LWS-loss. We have seen earlier that the winner-take-all dynamics of NLL-loss makes it extremely sensitive to initialization. The same sensitivity holds for the identification-based approaches. These methods weigh the loss for each output with the model's own prediction: i.e., when allowed label A is predicted to be more likely than allowed label B, A will be promoted more, making the probability gap between A and B even greater. β-merit-loss reduces the winner-take-all effect and we get better results as we decrease β. Uniform-loss, which is completely insensitive to the current model configuration and merely tries to reach uniform distribution on the allowed outputs performs surprisingly well, although still consistently worse than Libra-loss. On BESS project'23, we drill down to provide visualizations of 10 randomly selected learning curves for each loss function.

6.2.2 Large Consistent Synthetic Data Set

While the previous example is useful to intuitively understand the harmful "winner-take-all" behavior of NLL-loss, it is very restrictive, since it assumes a setting with multiple competing samples for the same input vector. In a more realistic scenario there are few (or no) samples with the same input and hence the interaction among points is more subtle. More specifically, learned models are functions that display some degree of smoothness. As a result, samples with similar features, i.e., similar input vectors, will get similar predictions, affecting each other's prediction accuracy. In our next experiment, we aim to simulate this by building a large synthetic data set with partial labels. Our data set has $n = 100,000$ samples, $d = 100$ input dimension and $m = 100$ possible outputs. First, we produce a set of synthetic input vectors with their corresponding true labels, as follows: We uniformly sample m corners of a hypercube in $\mathbb{R}^{100}$, i.e., from $\{0,1\}^{100}$ which will function as our cluster centroids. Each cluster will correspond to one true label, ensuring that samples that have similar input will likely share their true output. Then, we utilize a mixture of m Gaussian distributions (having standard deviation 1) with our selected centroids, and sample $n = 100,000$ input vectors. Each input vector $\boldsymbol{x}^{(i)}$ is assigned a true output $\boldsymbol{y}^{(i)}_{\text{true}}$ corresponding to the Gaussian from which it was sampled.

With the input samples defined, we randomly select partial/distractor labels for each sample. Distractor selection is controlled by the following two parameters:

Definition 16 (r_{Dpool}) *In the context of a random PLL data set as above, the Distractor pool fraction (r_{Dpool}) is the fraction of the output labels that can appear as distractors for any given true label.*

For example, if $r_{Dpool} = 0.2$ and there are $m = 100$ outputs in total, then for each true label $c \in [m]$ we select (uniformly at random) $100 \cdot 0.2 - 1 = 19$, other labels, which—along with c—form the distractor pool $D(c)$. For each input $\boldsymbol{x}^{(i)}$ the partial labels are constrained to be from $D(\boldsymbol{y}^{(i)}_{\text{true}})$. The second parameter controls the *strength of distraction*:

Definition 17 (r_{Docc}) *The Distractor co-occurrence fraction (r_{Docc}) is the fraction of inputs that are affected by any particular distractor from the distractor pool. More precisely, for any label c and potential distractor $c' \in D(c)$ the fraction of inputs with true label c and distractor c' is r_{Docc}.*

For example, if $r_{Docc} = 0.1$ and there are 1000 inputs with true label c, then distractor $c' \in D(c)$ will be present $1000 \cdot 0.1 = 100$ times as a distractor in the label sets of inputs with true label c. A high r_{Docc} means that the distractors are strongly "supported", i.e., are almost indistinguishable from true labels.
In the preceding example (small consistent synthetic data set), $r_{Docc} = 0.9$ since each distractor occurs in 9 out of 10 samples and $r_{Dpool} = 0.11$, since 11 out of the 100 possible outputs appear in the label sets.
We note that r_{Dpool} and r_{Docc} are just two of the many possible ways of characterizing this noise model. r_{Dpool} was motivated by the observation that all losses are very sensitive to the number of distractors and the motivation for r_{Docc} comes from observing that in the real world rule learning data sets, high r_{Docc} made learning much harder (see Section 6.4). The employed noise model is *instance-independent*, meaning that partial label $\boldsymbol{y}$ is independent from input $\boldsymbol{x}$ given true label $\boldsymbol{y}_{\text{true}}$.

Network We alter r_{Dpool} and r_{Docc} and train models with various loss functions. As underlying network, We use the same MLP model from Wen et al. (2021), having 5 layers and 333,108 parameters. We run each experiment 9 times, using 3 seeds for data set generation and 3 seeds for training.

Results Figure 8 shows model accuracies for different loss functions, as well as r_{Dpool} and r_{Docc} values. On all plots, we see a clear downward trend in performance as we increase r_{Dpool}, with the exception of Libra-loss and Uniform-loss. We argue that this is due to the winner-take-all behavior: as we increase r_{Dpool}, there are more and more distractors, so the chance of one of them getting significantly greater initial probability than the true label increases, which makes it impossible to recover the true label. This trend is greatly exacerbated by increasing r_{Docc}: when r_{Docc} is high, distractors are "almost as good" as the true label, so it gets easy to confuse them. Libra-loss and Uniform-loss demonstrate extreme resistance against this kind of distraction. As in the previous experiment, Libra-loss performs consistently better than Uniform-loss.

6.3 Experiments with Real Data Sets and Synthetic Distractors

To better understand the practical value of learning methods for PLL and DS, we can start from a real fully-labelled data set instead of a synthetic one, and generate distractor labels

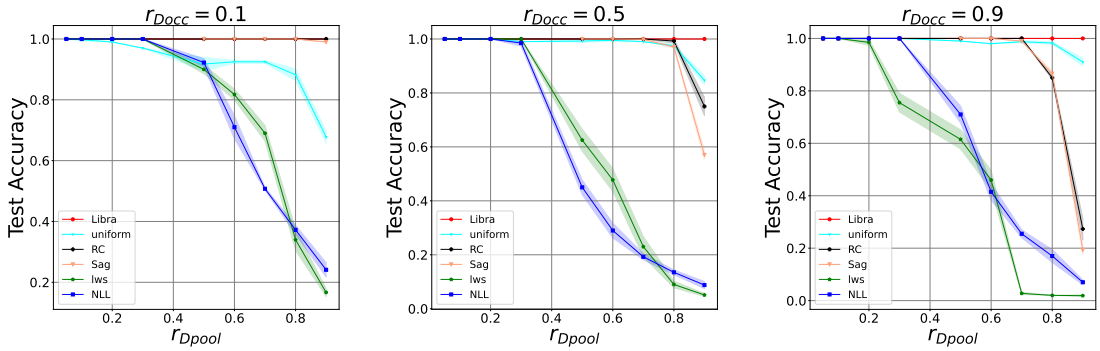


Figure 8: Test accuracy on the large consistent synthetic data set for different combinations of r_{Dpool} and r_{Docc} and different loss functions. We show mean values over 9 trials, using 3 seeds for data set generation and 3 seeds for training.

according to the noise model. This approach is often taken in the literature to evaluate PLL methods. We use the setup from Wen et al. (2021), starting from the CIFAR10 (Krizhevsky and Hinton, 2009) image classification benchmark and apply various true label dependent (instance-independent) noise models. Wen et al. (2021) defines three cases, to which we add two harder ones and refer to them as "Case 1" ... "Case 5". The noise models corresponding to these 5 cases are described in detail in Appendix D. CIFAR10 has 10 possible outputs and out of the 9 non-correct labels the expected number of distractors is 0.5, 0.6, 1.8, 4 and 7.1 for the 5 cases, respectively.

Network We train on this data set the CNN model from Wen et al. (2021), that has 9 convolutional layers and $4,434,570$ parameters. We run each experiment 9 times, using 3 seeds for data set generation and 3 seeds for training.

Results Figure 9 shows the performance of several loss functions trained on these data sets. Unsurprisingly, performance decreases as the distraction is stronger. However, the only loss that shows catastrophic collapse is LWS-loss. Also note that while Uniform-loss performs very well on purely synthetic inputs, it is clearly inferior to the other competitors in this setup. Some initial experiments with Libra-loss show easy overfitting, requiring careful early stopping to avoid a drop in final accuracy. We overcome this by introducing a weight $w_{\text{Lib}} = 1 - \sum_i y_i p_i$ that makes the loss vanish as the model gets close to fitting.[6] This weight is used in all subsequent experiments. Experiments with Sag-loss reveal that it is rather unstable. The explicit loss term that penalizes each disallowed label makes the average of the logits $\boldsymbol{z}$ tend to minus infinity and training quickly reaches a configuration that yields numerical instability. We managed to overcome this by adding an extra L2 regularization term to the loss that penalizes the magnitude of the logit vector:

$$\mathcal{L}_{\text{logit}} = \gamma_{\boldsymbol{z}} \sum_i z_i^2$$

6. We tried this weight for other losses as well, but it made no difference.

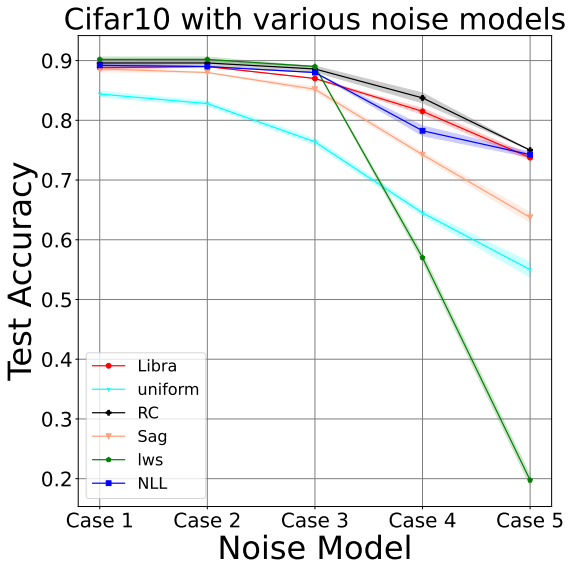


Figure 9: Performance of various loss functions on a PLL data set extracted from CIFAR10 and various noise models applied.

where $\gamma_{\boldsymbol{z}}$ is a hyperparameter determining the importance of this loss term and it is set to 0.01 in our experiments. This regularization successfully stabilized learning with the Sag-loss. Still, we find that it performs consistently worse than Libra-loss. All later experiments with Sag-loss makes use of this regularization term.
In our next experiment we evaluate the effect of changing r_{Dpool} and r_{Docc} on the much harder CIFAR100 data set, which has 100 labels.

Network We use the 18-block residual network from He et al. (2016), as implemented in Zai (2017). This model has $11,220,132$ parameters. We run each experiment 9 times, using 3 seeds for data set generation and 3 seeds for training.

Results Figure 10 shows the same trends as observed on Figure 8: performance degrades as r_{Dpool} (number of distractors) and r_{Docc} increase (strength of distraction) increase. However, Libra-loss shows remarkable robustness.

6.4 Rule Learning Experiments

In the following we experiment with the partially labelled rule learning data sets, introduced in Section 5. We remind the reader that these data sets contain negative samples, which are handled as described in Section 5. We also recall that these are sequence-to-sequence data sets, i.e., both the input and the output are represented as sequences of tokens. As described in Section 3, we use an autoregressive model $\boldsymbol{f_\theta}(\boldsymbol{x}, \boldsymbol{s}_{\text{prefix}})$ that outputs a distribution over single tokens in one step, conditioned on the preceding tokens. By sequentially evaluating all tokens in a sequence, we obtain the model predicted probability of the sequence. We recall that the output space of sequences is huge and we cannot compute the probability of all

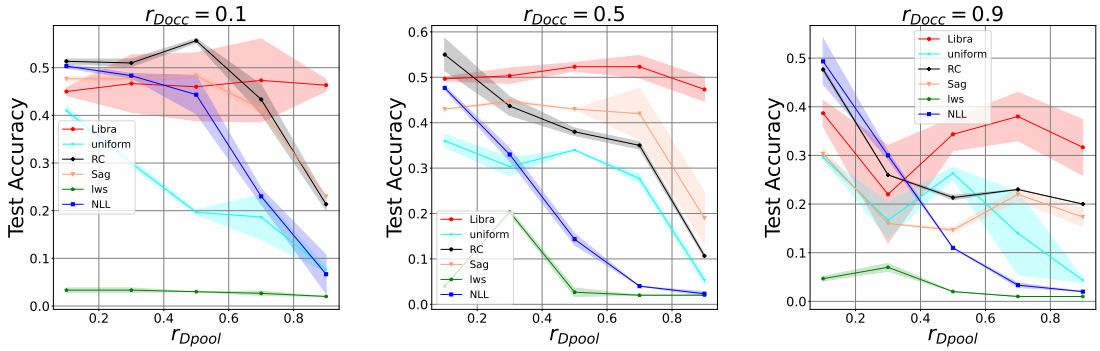


Figure 10: Test accuracy on CIFAR100 for different combinations of r_{Dpool} and r_{Docc} and different loss functions. We show mean values over 3 trials, using 3 seeds for data set generation.

sequences.[7] Even computing the probabilities of allowed sequences (via positive supervision) and explicitly forbidden sequences (via negative supervision) is computation heavy due to the sequential nature of evaluation. Consequently, we cannot use loss functions that depend on the probabilities of all possible outputs, such as Sag-loss or LWS-loss.

Given a sample $(\boldsymbol{x}, \boldsymbol{y})$, our primary evaluation metric is the probability of the model outputting an allowed output:

$$P_{pos} = \sum_i y_i p_i$$

Furthermore, we compute the probability of the model returning an output from any of the label sets of the negative samples (including training and test samples). Let $I_{neg} = \{i | (\boldsymbol{x}, \boldsymbol{y}) \text{ is a negative example and } y_i = 1\}$ denote the set of indices of all forbidden sequences. Then, the probability of selecting one of them is:

$$P_{neg} = \sum_{i \in I_{neg}} p_i$$

We are also interested in $H@k$ metrics, which is the ratio of inputs for which the k highest scoring outputs according to the model include either 1) an allowed (positive $H@k$) output or 2) a forbidden (negative $H@k$) output. Exactly determining the k highest scoring outputs is not tractable, as it would require evaluating all possible outputs. Thus we approximate this with beam-search, employing beamsize 10. All experiments employ a (70%, 15%, 15%) train-validation-test split.

6.4.1 CMT Challenges

We experiment with the CMT challenges, described earlier in Section 5.

7. This is true even if we employ some length limit that is large enough to derive rules in the supervision.

Network We train sequence-to-sequence models and use an encoder-decoder transformer architecture (Vaswani et al., 2017) with embedding dimension 128 and 4 encoder/decoder layers, having 2.5M parameters.
The largest rule in the training set contains 17 tokens and the model generated rules are restricted to 20 tokens. The γ hyperparameter that controls the tradeoff between positive and negative samples is empirically set to 3. A single experiment lasts for around 7hours on a single Nvidia A100 GPU.

Results Table 4 shows our experimental results. Given the different metrics, it is hard to come up with an unambiguous ordering of the loss functions. Nevertheless, Libra-loss clearly seems to perform best in terms of predicting allowed outputs for the test samples and β-merit-loss is second best. Uniform-loss is weaker, but performs consistently, while NLL-loss and RC-loss are overall quite weak and sometimes extremely weak. As for avoiding forbidden labels, Uniform-loss tends to perform best. But, this becomes somewhat vacuous given its mediocre performance on the allowed labels.

6.4.2 NPD Challenge

The NPD rule learning challenge is much harder than the CMT challenges, mostly due to the larger number of candidates (see Section 5 for details).

Network We train transformer models with embedding dimension 32 and 3 encoder/decoder layers, having 1M parameters.[8]
The largest rule in the training set contains 44 tokens and the model generated rules are restricted to 50 tokens. The γ hyperparameter that controls the tradeoff between positive and negative samples is empirically set to 0.001. A single experiment lasts for around 23 hours on a single Nvidia A100 GPU.

Results A particularity of this data set is that it contains inputs that share the same predicate, while having disjoint labels, forcing the model to attend both to the predicates and the constants in the input. Table 5 shows that Libra-loss performs best in terms of predicting allowed outputs in the evaluation set and is only marginally surpassed by RC-loss, while RC-loss completely fails to predict allowed labels. Uniform-loss is competitive for allowed outputs, but performs rather poorly in terms of avoiding forbidden labels. The results also show that the alignment produced by our solution is still far from perfect. However, we know of no other tools that can detect the rules in the NPD data set with or without supervision.

6.5 PLL Experiments with Real Data sets

To conclude our experiments, we adapt five real-world PLL data sets, each targeting a different task: Lost (Cour et al., 2011), Soccer Player (Zeng et al., 2013), and Yahoo!News (Guillaumin et al., 2010) for automatic face naming from video frames or images, MSRCv2 (Liu and Dietterich, 2012) for object classification and BirdSong (Briggs et al., 2012) for bird song classification.

8. We had to scale down the model size compared to that in the CMT experiments because candidate sets are larger and sequences are longer and we had to fit into the memory of a single Nvidia A100 GPU.

		Positive			Negative		
Loss	Distortion	P_{pos}	H@1	H@5	P_{neg}	H@1	H@5
Libra-loss	-	**0.97**	99%	100%	**0.03**	4%	7%
NLL-loss	-	0.83	83%	85%	0.17	17%	17%
RC-loss	-	0.71	0.71	71%	0.2	20%	20%
0.5-merit-loss	-	0.79	92%	100%	0.12	11%	25%
Uniform-loss	-	0.69	93%	99%	0.08	22%	29%
Libra-loss	renaming	**0.94**	93%	99%	0.11	9%	28%
NLL-loss	renaming	0.77	77%	79%	0.15	15%	15%
RC-loss	renaming	0.52	52%	52%	0.15	15%	15%
0.5-merit-loss	renaming	0.78	90%	100%	0.13	8%	29%
Uniform-loss	renaming	0.68	94%	100%	**0.08**	15%	31%
Libra-loss	restructuring	**0.93**	93%	100%	**0.03**	2%	26%
NLL-loss	restructuring	0.32	32%	32%	0.13	13%	13&
RC-loss	restructuring	0.18	18%	18%	0.7	7%	7%
0.5-merit-loss	restructuring	0.8	86%	99%	0.07	12%	21%
Uniform-loss	restructuring	0.76	96%	100%	0.06	6%	27%
Libra-loss	cleaning	0.87	89%	98%	0.14	10%	29%
NLL-loss	cleaning	0.71	71%	72%	0.10	10%	10%
RC-loss	cleaning	0.53	54%	54%	0.1	10%	10%
0.5-merit-loss	cleaning	**0.88**	91%	100%	0.17	18%	30%
Uniform-loss	cleaning	0.66	89%	98%	**0.07**	17%	29%
Libra-loss	denormalizing	**0.98**	100%	100%	0.14	16%	20%
NLL-loss	denormalizing	0.32	32%	32%	0.08	8%	8%
RC-loss	denormalizing	0.25	25%	25%	0.12	12%	12%
0.5-merit-loss	denormalizing	0.85	88%	100%	0.1	9%	17%
Uniform-loss	denormalizing	0.75	91%	100%	**0.04**	6%	20%

Table 4: P_{pos}, P_{neg}, H@1 and H@5 scores on the evaluation set of CMT data sets. For β-merit-loss, $\beta = 0.5$, which provided the best results based on a grid search with $\beta \in \{0.25, 0.5, 0.75\}$.

		Positive			Negative		
Loss	P_{pos}	H@1	H@5	P_{neg}	H@1	H@5
Libra-loss	**0.44**	**44%**	50%	0.02	2%	10%
NLL-loss	0.1	10%	11%	0.02	2%	2%
RC-loss	0.06	6%	9%	**0.01**	**1%**	**1%**
0.5-merit-loss	0.27	33%	45%	0.05	**1%**	19%
Uniform-loss	0.35	42%	**69%**	0.19	26%	26%

Table 5: P_{pos}, P_{neg}, H@1 and H@5 scores on the evaluation set of the NPD data set.

Network We perform experiments with two different models: the first Linear and the second a 3-layer MLP. We use learning rate 0.1 and weight decay with parameter 10^{-3}. We train for 300 epochs using Stochastic Gradient Descent with batches of size 256. All experiments are performed using Pytorch.

Results For this experiment we apply 10-fold cross validation to evaluate all losses, and we report the accuracy along with the standard deviation. We observe that Libra-loss achieves the top performance for almost all data sets, with the exception of Yahoo!News. In particular, Libra-loss is the winner by a large margin for three out of five data sets, namely Lost, MSRCv2 and SoccerPlayer, while for BirdSong, it closely follows the winner.

Loss, Model	**Lost**	**MSRCv2**	**BirdSong**	**Soccer**	**Yahoo**
NLL-loss, linear	61.6±3.2%	41.3±2.2%	70.9±1.5%	53.2±0.6%	**64.7±0.4%**
Libra-loss, linear	**69.8±3.1%**	42.8±1.7%	65.8±1.3%	**55.3±0.5%**	60.1±0.7%
Sag-loss, linear	65.1±2.3%	40.7±2.0%	62.1±1.3%	49.1±0.3%	47.6±0.6%
LWS-loss, linear	39.4±4.9%	28.0±4.2%	57.3±2.1%	49.0±0.0%	46.6±0.7%
RC-loss, linear	63.1±2.7%	40.9±2.1%	70.8±1.5%	54.0±0.5%	64.7±0.5%
0.5-merit-loss, linear	17.3±3.2%	15.7±3.8%	62.2±1.2%	49.0±0.0%	45.7±0.5%
Uniform-loss, linear	55.4±3.0%	36.9±1.7%	63.2±1.4%	49.3±0.6%	50.3±1.0%
NLL-loss, MLP	53.1±2.4%	48.9±1.8%	69.8±1.3%	52.3±0.5%	60.1±0.8%
Libra-loss, MLP	59.7±2.4%	**51.0±1.7%**	72.4±1.0%	52.7±0.4%	60.4±1.0%
Sag-loss, MLP	55.9±2.4%	48.7±1.7%	71.5±1.5%	53.7±0.5%	57.3±0.7%
LWS-loss, MLP	51.7±3.0%	47.3±1.5%	67.3±1.1%	50.3±0.4%	53.9±1.0%
RC-loss, MLP	53.5±2.6%	48.6±2.1%	**72.4±1.0%**	52.8±0.4%	60.4±0.7%
0.5-merit-loss, MLP	37.9±3.3%	39.8±1.7%	67.6±1.2%	49.0±0.0%	36.7±0.9%
Uniform-loss, MLP	54.3±2.8%	41.2±2.5%	66.0±1.7%	51.3±0.5%	49.2±1.1%

Table 6: Classification accuracy (mean±std) for five real-world data sets. **Soccer** and **Yahoo** stand for SoccerPlayer and Yahoo!News benchmarks. For each data set, the best method is indicated with **bold** and the second best with underline.

6.6 Discussion of the Experimental Results

We draw together some of the main takeaways from our analysis.

- NLL-loss *tends to perform poorly in the presence of a softmax layer.* This is due to the winner-take-all bias. This loss works well in easier situations, when there are many distractors (high r_{Dpool}) or some distractors are present in many samples (high r_{Docc}), performance drops steeply in more complex settings.

- *Identification-based methods are also susceptible to winner-take-all bias.* This is because initially incorrect predictions can make erroneous labels being promoted more than the correct one. Analogously to NLL-loss, this effect is exacerbated as r_{Dpool} and r_{Docc} values increase.

- β-merit-loss *improves over* NLL-loss. Decreasing β reduces the effect of winner-take-all. Often, it is best to push it to the extreme, which is Uniform-loss.

- *The most extreme antidote to winner-take-all is* Uniform-loss*, which can perform surprisingly well.* Uniform-loss is the opposite of identification-based methods, and completely avoids winner-take-all. Indeed, it always explores multiple options equally, even after developing some experience and signal. We find that it performs well on

synthetic data sets (Figures 7 and 8), where being cautious is useful. But it does not perform well on real data sets where it is important to "exploit" as well as explore (Figures 9 and 10, Tables 4 and 5).

- Libra-loss *tends to perform best, especially on harder challenges.* Libra-loss overcomes the winner-take-all bias by design, allowing more balanced exploration of alternatives. On the other hand, it is more flexible than Uniform-loss, as it can adapt to experience accumulated during training. This ability is less important in synthetic data sets, but yields large performance difference in real data sets.

- Sag-loss *performs decently, but it can easily become unstable.* This is because the magnitude of the logit vector increases quickly during learning, leading to numerical instability. This problem can be overcome with L2 regularization on the logits. However, not even the regularized variant ever performs better than Libra-loss.

7. Conclusion

In this paper we identify a bias phenomenon that emerges in partial label learning based on neural architectures with a softmax layer. We provide a loss function which is tailored towards addressing the situation, and argue that it is, up to a differentiable transformation, canonical. We also give an experimental evaluation of its performance. We discuss some of the issues left over from this work.

7.1 Winner-take-all and characterization theorems

We have proven our main theoretical results in a restricted setting, both in terms of the normalization function (softmax) and the update mechanism - (gradient descent). It remains to investigate winner-take-all results and the PRP property in more general settings.

7.2 Loss functions

We have looked at loss functions that focus on combating a certain bias phenomenon; but there are obviously many other desiderata within learning. It remains to investigate how properties like PRP can be incorporated in the setting where there are additional objectives in play.

7.3 Rule learning

Partial supervision is a special case of symbolic supervision in the form of logical constraints. In a learning framework where supervision is intermediated by the presence of logical constraints, more general forms of symbolic supervision can emerge, not merely disjunctions of literals and their negations as in our application. We will investigate the broader question of symbolic supervision in the future; we think our work shows promise in tailoring loss functions for supervision intermediated by more general formulas.

Acknowledgments

 This work has been supported by Hungarian National Excellence Grant 2018-1.2.1-NKP-00008, the Hungarian Artificial Intelligence National Laboratory (RRF-2.3.1-21-2022-00004), ELTE TKP 2021-NKTA-62 funding scheme. It has also been supported by the UK's Engineering and Physical Science Research Center under Oxford's EPSRC Impact Acceleration Account Award EP/R511742/1 and EPSRC EP/T022124/1, as well as the National Science Foundation (NSF) under award numbers IIS-1762268 and IIS-1956096. We thank Varun Kanade for his guidance and feedback on preliminary versions of this work. We also thank Beatrix Benkö for her help with visualizations. We sincerely thank JMLR editor Luc de Raedt for all his work in managing this submission.

Appendix A. Winner-Take-All Theorem

In this section we prove the winner-take-all property of the NLL-loss as stated in Theorem 4. Throughout the section, we assume a softmax regression model $\boldsymbol{p} = \boldsymbol{f_\theta}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{\theta} \cdot \boldsymbol{x})$. Furthermore, since the theorem deals with convergence on a single sample, we assume without loss of generality that input $\boldsymbol{x}$ is given in one-hot representation, i.e, the logit vector $\boldsymbol{z}$ is $\boldsymbol{\theta}_j$ for some j, i.e., directly updateable. We use $\hat{p} = \sum_i y_i p_i$ to denote the sum of probabilities of acceptable outputs.
We begin by calculating the gradients of the NLL-loss with respect to the logits.

Lemma 18 *The gradient vector* $grad = \frac{\partial \mathcal{L}_{\text{NLL}}(\boldsymbol{p},\boldsymbol{y})}{\partial \boldsymbol{z}}$ *of the* NLL-loss *with respect to the logit vector* $\boldsymbol{z} = \boldsymbol{\theta} \cdot \boldsymbol{x}$ *is given by*

$$grad_j = \frac{p_j}{\hat{p}} \left(-y_j + \hat{p}\right)$$

Proof

$$\begin{aligned} grad_j &= \sum_{\{i|y_i=1\}} \frac{\partial \mathcal{L}_{\text{NLL}}(\boldsymbol{p},\boldsymbol{y})}{\partial p_i} \frac{\partial p_i}{\partial z_j} = \sum_{\{i|y_i=1\}} -\frac{1}{\sum_{\{k|y_k=1\}} p_k} p_i(\delta_{ij} - p_j) \\ &= \frac{1}{\sum_{\{k|y_k=1\}} p_k} \left(-y_j p_j + \sum_{\{y_i=1\}} p_i p_j\right) = \frac{p_j}{\sum_{\{k|y_k=1\}} p_k} \left(-y_j + \sum_{\{y_i=1\}} p_i\right) \\ &= \frac{p_j}{\hat{p}} \left(-y_j + \hat{p}\right) \end{aligned}$$

■

Next, we compare the ratio of probabilities of two allowed outputs and show that the ratio changes monotonically during training.

Lemma 19 *Let* m, n *be two acceptable outputs, i.e.* $y_m = y_n = 1$. *Let* $\boldsymbol{p}'$ *denote the updated probability vector after a Gradient-update operation with some positive learning rate* λ. *Then it holds that* $\frac{\boldsymbol{p}'_m}{\boldsymbol{p}'_n} > \frac{p_m}{p_n}$ *exactly when* $p_m > p_n$.

Proof Since we know that $y_m = y_n = 1$ the gradients computed in Lemma 18 reduce to

$$\begin{aligned} grad_m &= \frac{p_m}{\hat{p}}(-1 + \hat{p}) \\ grad_n &= \frac{p_n}{\hat{p}}(-1 + \hat{p}) \end{aligned}$$

After the update step, the ratio of model predicted probabilities are:

$$\begin{aligned} \frac{\boldsymbol{p}'_m}{\boldsymbol{p}'_n} &= \frac{\text{softmax}(\boldsymbol{z}')_m}{\text{softmax}(\boldsymbol{z}')_n} = \frac{e^{\boldsymbol{z}'_m}}{e^{\boldsymbol{z}'_n}} = \frac{e^{z_m - \lambda \frac{p_m}{\hat{p}}(-1+\hat{p})}}{e^{z_n - \lambda \frac{p_n}{\hat{p}}(-1+\hat{p})}} \\ &= \frac{e^{z_m}}{e^{z_n}} \frac{e^{-\lambda \frac{p_m}{\hat{p}}(-1+\hat{p})}}{e^{-\lambda \frac{p_n}{\hat{p}}(-1+\hat{p})}} = \frac{p_m}{p_n} e^{\lambda(p_m - p_n)\frac{1-\hat{p}}{\hat{p}}} \end{aligned}$$

Since $0 < \hat{p} < 1$, the exponent has the same sign as $p_m - p_n$. From this it follows that the ratio increases exactly when $p_m > p_n$ and remains the same when $p_m = p_n$. This concludes our proof. ■

We can now prove Theorem 4, which we restate here:

Theorem 4 (Winner-take-all) *Consider the softmax regression model $\boldsymbol{f_\theta}(\boldsymbol{x})$. Fix a datapoint $(\boldsymbol{x}, \boldsymbol{y})$, and let J be the set of acceptable outputs such that for every $j \in J$, $p_j = \boldsymbol{f_\theta}(x)_j$ is maximal among the allowed output probabilities. Then the Gradient-update operation with $\mathcal{L} = \mathsf{NLL}$-loss from Equation (2) yields a limit distribution*

$$p_j = \begin{cases} \frac{1}{|J|} & \text{if } j \in J \\ 0 & \text{otherwise} \end{cases}$$

Proof The model probability of all outputs $j \in J$ is the same initially, and it follows from Lemma 19 that their ratios remain 1 during training. Let I denote the set of acceptable outputs and let $I^c = \mathcal{Y} \setminus I$ denote its complement. Recall that we update the k^{th} logit z_k as

$$z_k = z_k - \lambda \left(p_k - \frac{p_k y_k}{\hat{p}} \right)$$

In any state where none of the z_i are $\pm\infty$, the gradient is nonzero and hence that state cannot be a convergence point. Consequently, $\boldsymbol{z}$ can only converge to a state where at least one logit is $\pm\infty$. Note that the sum of logits is constant because the sum of the gradients at each step is zero:

$$\sum_i \left(-p_i + \frac{p_i y_i}{\hat{p}} \right) = -\sum p_i + \frac{1}{\hat{p}} \sum_i y_i p_i = 0$$

Given that the sum of z_i is constant and that there is some logit that converges to $\pm\infty$, there must be a logit which converges to ∞. The disallowed logits are decreasing, so an allowed logit must converge to ∞.

If $J = I$, i.e., all acceptable outputs have the same initial probability, then we are done, since z_k for $k \in I$ are increasing and z_k for $k \in I^c$ are decreasing and this only stops if $\hat{p} = 1$, so the limit is uniform distribution over J. So we can assume that $J \neq I$. After T update steps, the value of the k^{th} logit with $k \in I$ will be

$$z_k(T) = z_k(0) + \lambda \sum_{t=0}^{T-1} \left(\frac{1}{\hat{p}(t)} - 1 \right) p_k(t)$$

Let $j \in J$, $\iota \in I \setminus J$ and $c = \frac{p_j(0)}{p_\iota(0)}$. Due to our assumption that J contains all allowed logits with maximal probabilities at time $t = 0$, we have that $c > 1$. Furthermore, we know from Lemma 19 that p_j grows faster than p_ι in every update step, hence $\frac{p_j(t)}{p_\iota(t)} \geq c$ for every $t \geq 0$. This gives us a lower bound on $z_j(T)$:

$$z_j(T) \geq z_j(0) + \lambda c \sum_{t=0}^{T-1} \left(\frac{1}{\hat{p}(t)} - 1 \right) p_\iota(t) = z_j(0) - c z_\iota(0) + c z_\iota(T)$$

If $z_\iota \to \infty$, then the above calculation shows that $z_j \to \infty$ and that

$$\frac{p_\iota^\infty}{p_j^\infty} = \lim_{t\to\infty} e^{z_\iota(t)-z_j(t)} \leq \lim_{t\to\infty} e^{C+(1-c)z_\iota(t)} = 0$$

where p^∞ denotes the limit distribution and $C = cz_\iota(0) - z_j(0)$ is some constant. The limit goes to 0 because $1-c<0$, so $(1-c)z_\iota(t) \to -\infty$.
We now consider the possibility that z_ι does not go to infinity as the number of updates increases. Thus there exists δ such that $z_\iota(t) < \delta$ for all t and

$$\frac{p_\iota^\infty}{p_j^\infty} = \lim_{t\to\infty} e^{z_\iota(t)-z_j(t)} \leq \lim_{t\to\infty} e^{\delta-z_j(t)} = 0$$

This is because we showed previously that some allowed logit must converge to infinity and z_j is greater than any other allowed logit, hence $z_j \to \infty$.
We conclude that $\frac{p_\iota^\infty}{p_j^\infty} = 0$ in the limit state. Therefore, we showed that all probabilities in J^c converge to 0.
We know that $z_{j_1}(t) = z_{j_2}(t)$ for all $j_1, j_2 \in J$ throughout the training because we apply the same gradient at each step. From this it follows that p converges to a uniform distribution over J. ■

Appendix B. Theorems Related to the Libra-loss

In this section we prove the characterization theorems for loss functions satisfying the $\mathrm{PRP_s}$ property. We recall the two theorems:

Theorem 8 *The* Libra-loss *function has the* $\mathrm{PRP_s}$ *property.*

Theorem 10 *Let $\mathcal{L}$ be an acceptable-dependent function on m outputs that has the* $\mathrm{PRP_s}$ *property. Then there exists a function $h : \mathbb{R} \times [m] \to \mathbb{R}$ that is continuously differentiable in its first argument such that $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y}) = h(\mathcal{L}_{\mathrm{Lib}}(\boldsymbol{p}, \boldsymbol{y}), k)$ where $k = \sum_i y_i$.*

We also recall the definition of the Libra-loss:

$$\mathcal{L}_{\mathrm{Lib}}(\boldsymbol{p}, \boldsymbol{y}) = \log\left(1 - \sum_i y_i p_i\right) - \frac{1}{k}\sum_i y_i \log(p_i)$$

Before beginning the proofs, we give a property of loss function with the $\mathrm{PRP_s}$ property that will be easier to work with.

Theorem 20 *Let $\mathcal{L} : \mathbb{R}^n \times \{0,1\}^n \to \mathbb{R}$ be a differentiable loss function. Then the* $\mathrm{PRP_s}$ *property holds for $\mathcal{L}$ if and only if $\mathcal{L}$ satisfies the following system of equations for all index pair (m,n) such that $y_m = y_n = 1$:*

$$\sum_i \frac{\partial \mathcal{L}(p,y)}{\partial p_i} p_i(\delta_{im} - p_m) = \sum_i \frac{\partial \mathcal{L}(p,y)}{\partial p_i} p_i(\delta_{in} - p_n)$$

where δ_{ij} is the Kronecker function.

Proof [Proof of Theorem 20]
Let us compute the updated probabilities $\boldsymbol{p}'$:

$$\begin{aligned} grad &:= \frac{\partial \mathcal{L}(\boldsymbol{p}, \boldsymbol{x})}{\partial \boldsymbol{z}} \\ p' &= \text{softmax}(\boldsymbol{z}') = \text{softmax}(\boldsymbol{z} - \lambda grad) \\ p'_i &= \frac{e^{\boldsymbol{z}'_i}}{\sum e^{\boldsymbol{z}'_j}} = \frac{e^{z_i - \lambda grad_i}}{\sum e^{\boldsymbol{z}'_j - \lambda grad_j}} \\ \frac{p_m}{p_n} &= \frac{e^{z_m}}{e^{z_n}} \\ \frac{\boldsymbol{p}'_m}{\boldsymbol{p}'_n} &= \frac{e^{z_m - \lambda grad_m}}{e^{z_n - \lambda grad_n}} = \frac{p_1}{p_2} \frac{e^{\lambda grad_n}}{e^{\lambda grad_m}} \end{aligned}$$

Here we recall that λ is the learning rate. The last equation above shows that the ratios remain the same if $\frac{e^{\lambda grad_n}}{e^{\lambda grad_m}} = 1$, i.e., $grad_m = grad_n$. Let us decompose the gradient using the chain rule:

$$grad_j = \frac{\partial \mathcal{L}}{\partial z_j} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{p}} \frac{\partial \boldsymbol{p}}{\partial z_j} = \sum_i \frac{\partial \mathcal{L}}{\partial p_i} \frac{\partial p_i}{\partial z_j} = \sum_i \frac{\partial \mathcal{L}}{\partial p_i} p_i(\delta_{ij} - p_j)$$

For each pair of allowed outputs (m, n), the loss function has to satisfy the differential equation $grad_m = grad_n$, i.e.:

$$\sum_i \frac{\partial \mathcal{L}}{\partial p_i} p_i(\delta_{im} - p_m) = \sum_i \frac{\partial \mathcal{L}}{\partial p_i} p_i(\delta_{in} - p_n)$$

This concludes our proof. ■

Proof [Proof of Theorem 8.]
Recall that the PRP_s property assumes a softmax regression model and that the logit vector $\boldsymbol{z}$ is a parameter vector. In order to show that $\mathcal{L}_\text{Lib}$ has the PRP_s property, we need to show that partial derivatives in z_j are equal whenever $y_j = 1$. First, we compute the partial derivatives in the probabilities p_j.

$$\begin{aligned} \frac{\partial \mathcal{L}_\text{Lib}}{\partial p_j} &= \frac{\partial \big[\log(1 - \sum_i y_i p_i) - \frac{1}{k} \sum_i y_i \log p_i\big]}{\partial p_j} \\ &= \begin{cases} -\frac{1}{1 - \sum_i y_i p_i} - \frac{1}{k} \frac{1}{p_i} & \text{if } j \in I \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Now, we compute the partial derivatives in z_j. Recall that $\frac{\partial p_i}{\partial z_j}$ is the partial derivative of the softmax function which is $p_i(\delta_{ij} - p_j)$.

$$\frac{\partial \mathcal{L}_{\text{Lib}}}{\partial z_j} = \sum_i \frac{\partial \mathcal{L}}{\partial p_i}\frac{\partial p_i}{\partial z_j} = \sum_i -y_i \left(\frac{1}{1-\sum_{i'} y_{i'}p_{i'}} + \frac{1}{kp_i}\right) p_i(\delta_{ij} - p_j)$$
$$= -\frac{1}{1-\sum_i y_i p_i}\sum_i y_i p_i(\delta_{ij} - p_j) - \frac{1}{k}\sum_i y_i(\delta_{ij} - p_j) = -\frac{y_j p_j - p_j\sum_i y_i p_i}{1-\sum_i y_i p_i} - \frac{1}{k}y_j + p_j$$
$$= \frac{(1-y_j)p_j}{1-\sum_i y_i p_i} - \frac{y_j}{k} = \begin{cases} -\frac{1}{k} & \text{if } y_j = 1 \\ \frac{p_j}{1-\sum_i y_i p_i} & \text{if } y_j = 0 \end{cases}$$

As we can see, the gradients of the logits with $y_i = 1$ are equal, hence $\mathcal{L}_{\text{Lib}}$ has the PRP_s property. ■

Proof [Proof of Theorem 10.]
For given h_k continuously differentiable functions, let $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y}) = h_{|\boldsymbol{y}|}(\mathcal{L}_{\text{Lib}}(\boldsymbol{p}, \boldsymbol{y}))$. We showed earlier in Theorem 20 that $\mathcal{L}$ has the PRP_s property if it satisfies a linear differential equation and we have shown that $\mathcal{L}_{\text{Lib}}$ satisfies it. We know that $\frac{\partial \mathcal{L}}{\partial p_i} = h'_{|\boldsymbol{y}|}(\mathcal{L}_{\text{Lib}}(\boldsymbol{p}, \boldsymbol{y}))\frac{\partial \mathcal{L}_{\text{Lib}}}{\partial p_i}$. Multiplying the equations in Theorem 20 for $\mathcal{L}_{\text{Lib}}$ with $h'_{|\boldsymbol{y}|}(\mathcal{L}_{\text{Lib}}(\boldsymbol{p}, \boldsymbol{y}))$ yields the equations for $\mathcal{L}$. Therefore $\mathcal{L}$ also has the PRP_s property.
The proof of the converse statement consists of several steps, which we will label for better transparency.
1. Consider a loss function $\mathcal{L}$ that has the PRP_s property and satisfies the technical assumptions in the statement of the theorem. According to Theorem 20, the PRP_s property is equivalent to a differential equation which is an invariant of $\mathcal{L}$ at any given set of labels $\boldsymbol{y}$. Therefore, we only consider the case when $\boldsymbol{y}$ is fixed such that the first k outputs are the allowed ones, i.e., $y_i = 1 \leftrightarrow i \leq k$. Since $\boldsymbol{y}$ is fixed, we can treat $\mathcal{L}$ and $\mathcal{L}_{\text{Lib}}$ as functions over $\boldsymbol{p}$, omitting $\boldsymbol{y}$ from its domain. According to Theorem 20, the partial derivatives of $\mathcal{L}$ satisfy the system of equations:

$$\sum_i \frac{\partial \mathcal{L}}{\partial p_i} p_i(\delta_{im} - p_m) = \sum_i \frac{\partial \mathcal{L}}{\partial p_i} p_i(\delta_{in} - p_n)$$

for all $m, n \leq k$. To understand these equations better, we define the parameterized matrix $A \in \mathbb{R}^k \to \mathbb{R}^{k\times k}$ such that $A_{i,j} = p_j(\delta_{ij} - p_i)$ where $i, j \leq k$, i.e., we only consider rows and columns corresponding to allowed outputs. We will use the apostrophe notion for denoting the Jacobian matrix of a smooth function, i.e., $\mathcal{L}'$ is the vector of partial derivatives of $\mathcal{L}$ with respect to the logits of allowed outputs. Note that, at any input value in $\mathbb{R}^k$, the m^{th} entry of $A\mathcal{L}'$ is the left-hand side of the above equation:

$$(A\mathcal{L}')_m = \sum_i p_i(\delta_{im} - p_m)\mathcal{L}'_i$$

Therefore the above system of equations is equivalent to the value of $A\mathcal{L}'$ at any input being a constant vector. That is, $A\mathcal{L}'$ is of the form $\langle \kappa \dots \kappa \rangle$ for some function $\kappa : \mathbb{R}^k \to \mathbb{R}$. At any input value, the corresponding matrix A is invertible if and only if $\det A \neq 0$. Below, we show, by direct calculation, that $\det A = (1 - \sum p_i)\prod p_i$.

Lemma 21 *Let $A \in \mathbb{R}^{k\times k}$ denote the matrix such that $A_{i,j} = p_j(\delta_{ij} - p_i)$ and let $v = A^{-1}\underline{1}$. Then* $\det A = (1 - \sum p_i)\prod p_i^{-1}$.

Proof First, note that $A = B \cdot \mathrm{diag}(p_1, \dots p_k)$, where $B = (\delta_{ij} - p_i)_{i,j}$. The inverse of $\mathrm{diag}(p_1, \dots p_k)$ is $\mathrm{diag}(p_1^{-1}, \dots p_k^{-1})$. The determinant of a diagonal matrix is just the product of the diagonal entries. So we only need to show that the determinant of B is $1 - \Sigma p_i$. Subtracting the last column from any other will not change the determinant, but will simplify the calculation

$$B = \begin{pmatrix} 1-p_1 & -p_1 & -p_1 & \dots & -p_1 \\ -p_2 & 1-p_2 & -p_2 & \dots & -p_2 \\ \vdots & & & \ddots & \\ -p_k & -p_k & -p_k & \dots & 1-p_k \end{pmatrix} \rightarrow \quad B_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & -p_1 \\ 0 & 1 & 0 & \dots & -p_2 \\ \vdots & & & \ddots & \\ -1 & -1 & -1 & \dots & 1-p_k \end{pmatrix}$$

Now using the definition of determinant, $\det B = \sum_\pi \prod_i B_{i,\pi(i)}$, where π goes over every permutation, we see that the only non-zero products are $1 - p_k$ and $-p_1, -p_2, \cdots - p_{k-1}$. Hence $\det B = \det B_1 = 1 - p_1 - \cdots - p_k$. This completes the derivation. ■

In particular, the lemma above tells us that A is invertible over any non-degenerate probability distribution. If A is invertible, then we can simply calculate $\mathcal{L}' = \kappa A^{-1}\underline{1}$. Let v denote $A^{-1}\underline{1}$. Thus $\mathcal{L}' = \kappa v$. Let κ_{Lib} be the value of κ for $\mathcal{L}_{\mathrm{Lib}}$. We can show:

Claim 22 *κ_{Lib} is never 0, assuming $\sum p_i$ is neither 0 nor 1.*

Proof At a point where κ_{Lib} is 0, we have $\mathcal{L}'_{\mathrm{Lib}}$ is 0, since $\mathcal{L}'_{\mathrm{Lib}} = \kappa_{\mathrm{Lib}} v_{\mathrm{Lib}}$. But above we have calculated that $\frac{\partial \mathcal{L}_{\mathrm{Lib}}}{\partial z_j}$ is $-\frac{1}{k}$ if $\quad y_j = 1$ and $\frac{p_j}{1-\sum_i y_i p_i}$ if $y_j = 0$. Clearly this is not 0 when $\mathbf{p}$ is nontrivial. ■

From the claim it follows that at every point, $\mathcal{L}'$ and $\mathcal{L}'_{\mathrm{Lib}}$ only differ by a constant multiple. Of course, we are not interested in the derivatives of the loss functions, but in the functions themselves.
Before we move on with the remainder of the proof, here is an outline of the steps.

1. We argue that $\mathcal{L}' = \kappa v$ for some $\kappa : \mathbb{R}^k \to \mathbb{R}$. and that $\mathcal{L}' = d \cdot L'_{\mathrm{Lib}}$ for some constant function d.

2. The sets $H_z = \mathcal{L}_{\mathrm{Lib}}^{-1}(\{z\})$ are path-connected.

3. We argue that $\mathcal{L}$ is constant on H_z for any z. Restated, this means that the function $h : \mathbb{R} \to \mathbb{R}$ as required by theorem (but not necessarily smooth) exists. This will make use of the first items above.

4. The h function is continuously differentiable.

We have already shown the first item above, modulo the gap of showing the determinant of A is nonzero, and also that κ_{Lib} is never 0.

2. Note that in this item, we are only reasoning about $\mathcal{L}_{\text{Lib}}$, and not the generic loss function $\mathcal{L}$. Let $H_z = \mathcal{L}_{\text{Lib}}^{-1}(\{z\}) = \{\boldsymbol{p} | \mathcal{L}_{\text{Lib}}(\boldsymbol{p}, \boldsymbol{x}) = z\}$ be the preimage of z. Let $\mathcal{P} = \{(p_1, \dots, p_k) | p_i \in (0,1), \sum_j p_j < 1\}$ denote the space of the projection onto the first k coordinates of the non-degenerate probability distribution over m categories.
Note that $\mathcal{P}$ is an open path-connected subset of $\mathbb{R}^k$ and hence a differentiable manifold. At the same time, the range of $\mathcal{L}_{\text{Lib}}$ is $\mathbb{R}$, which is also a differentiable manifold. Thus we can view $\mathcal{L}_{\text{Lib}}$ as a smooth map between manifolds $\mathcal{P}$ and $\mathbb{R}$. Our next goal will be:

Claim 23 *Each H_z is a differentiable manifold.*

Proof Let X, Y be two differentiable manifolds and $f : X \to Y$ a smooth map between them. We say that $y \in Y$ is regular if for every $x \in f^{-1}(y)$ the map $df_x : T_x X \to T_y Y$ is surjective, where $T_x X$ is the tangent space of X in x.
We will use the following elementary result about differentiable manifolds.

Fact B.1 *If $y \in Y$ is a regular value of f, then $f^{-1}(y)$ is a differentiable submanifold of X.*

We want to show that $d\mathcal{L}_{\text{Lib}}$ is surjective everywhere, in order to argue, using the fact above, that the pre-image of a single point is a differentiable manifold.
Since $\mathbb{R}$ is one-dimensional, $d\mathcal{L}_{\text{Lib}}$ is not surjective precisely when $d\mathcal{L}_{\text{Lib}}$ is the zero map. Equivalently the gradient is zero; it follows from Claim 22 that this can only occur on the boundary of $\mathcal{P}$. Therefore, any z is a regular value of H, and consequently H_z is a differentiable submanifold of $\mathcal{P}$. ■

For any probability distribution $\boldsymbol{p}$ over m categories with $\sum_{j=1}^{k} p_j = 1$ we assign a line that goes through $\boldsymbol{p}$ and 0, let $l_{\boldsymbol{p}} = \{\omega \boldsymbol{p} | \omega \in (0,1)\}$ denote this line. Note that $l_{\boldsymbol{p}}$ does not contain either $(p_1, \dots p_k)$ or 0 and it lies in $\mathcal{P}$, i.e., $l_{\boldsymbol{p}} \subset \mathcal{P}$. Informally, $l_{\boldsymbol{p}}$ represents the possible ways of "scaling down" some target distribution that assigns all the mass to acceptable elements. We make the following claim, where again $\mathbf{y}$ is fixed to sum to k.

Claim 24 $\mathcal{L}_{\text{Lib}}$ *takes every value precisely once on $l_{\boldsymbol{p}}$.*

Proof To prove the claim, observe that the loss is

$$\log(1 - \omega \sum_i y_i p_i) - \frac{1}{k} \sum_i y_i \log(\omega p_i) = \log(1-\omega) - \log(\omega) - \frac{1}{k} \sum_i y_i \log(p_i)$$

where we used that $\sum_i y_i p_i = 1$ and $\sum_i y_i = k$. It is clear that when $\omega \to 0$ it converges to ∞ and when $\omega \to 1$ it converges to $-\infty$. Now, we show that the above mapping is monotonically strictly decreasing in ω, and consequently $\mathcal{L}_{\text{Lib}}$ takes every value of $\mathbb{R}$ precisely once on $l_{\boldsymbol{p}}$. It is sufficient if the derivative with respect to ω is less than zero. The derivative is $-\frac{1}{1-\omega} - \frac{1}{\omega}$ which is clearly less than zero. Hence the claim is proven. ■

Recall that we are interested in showing path connectedness of the set H_z, the pre-image of singletons under $\mathcal{L}_{\text{Lib}}$. By the claim above, we know that as we vary the lines $l_{\boldsymbol{p}}$, $\mathcal{L}_{\text{Lib}}$ always hits H_z exactly once on the line, but the point at which it hits H_z varies with $\boldsymbol{p}$.

Let $\pi : \mathcal{P} \to S^{k-1}$ the projection given by $\pi(x) = \frac{x}{\|x\|}$. Here $\|\cdot\|$ is the 2-norm. Note that the preimage of a point under π is precisely an $l_{\boldsymbol{p}}$ line for some $\boldsymbol{p}$. Since we have shown above that $\mathcal{L}_{\text{Lib}}$ takes every value once over a fixed $l_{\boldsymbol{p}}$ line, we conclude that π is a bijection between $\pi(\mathcal{P})$ and H_z.

We will now use the fact that H_z is a manifold by Claim 23. It is known that for manifolds, connected and path-connected are equivalent properties. If H_z were not connected, then there would be U, V disjoint non-empty open sets such that $H_z \subset U \cup V$. Observe that π is an open map, $\pi(\mathcal{P})$ is connected, and $\pi\mathcal{P} = \pi(H_z) \subset \pi(U \cup V)$. We cannot have two disjoint open sets covering the connected set $\pi(\mathcal{P})$. Thus the sets $\pi(U)$ and $\pi(V)$ must overlap: there are $u \in U, v \in V$ points such that $\pi(u) = \pi(v)$. Thus there are two distinct points in the pre-image of π with the same value. Since the pre-image is an $l_{\boldsymbol{p}}$ line, this contradicts Claim 24.

3. We show that $\mathcal{L}$ is constant on H_z for any z, and that a $h : \mathbb{R} \to \mathbb{R}$ function exists such that $\mathcal{L} = h(\mathcal{L}_{\text{Lib}})$. The idea will be that for any $a \neq b \in H_z$ we show $\mathcal{L}(b) - \mathcal{L}(a) = 0$. We do this by computing $\mathcal{L}(b) - \mathcal{L}(a)$ as an integral of a quantity, over a path in H_z between a and b, using the fact that H_z is path-connected. The quantity will involve a dot product with the derivative of $\mathcal{L}$, and we will use part (1) to argue that this dot product is always 0. We will make use of the following result from multi-variable calculus

Proposition 25 *For any $F : \mathbb{R}^m \to \mathbb{R}$ continuously differentiable function and $\gamma : [0, 1] \to \mathbb{R}^m$ differentiable path from $\gamma(0) = a$ to $\gamma(1) = b$, we have $F(b) - F(a) = \int_\gamma \langle F', d\gamma \rangle$.*

Applying this to $\mathcal{L}$, we get

$$\mathcal{L}(b) - \mathcal{L}(a) = \int_\gamma \langle \mathcal{L}', d\gamma \rangle$$

Applying what we showed about $\mathcal{L}'$ in part (1), we have that this integral simplifies as follows:

$$= \int_\gamma \langle \kappa v, d\gamma \rangle = \int_\gamma \frac{\kappa}{\kappa_{\text{Lib}}} \langle \mathcal{L}'_{\text{Lib}}, d\gamma \rangle$$

In the last line, we used the assumption that $\kappa_{prp} > 0$, so we can divide by it. We now use another fact from calculus:

Proposition 26 *For any smooth H, the gradient H' is orthogonal to the tangent plane of a constant surface H_z.*

Now note that γ lies in H_z, so $d\gamma$ is in the tangent plane of H_z. So the inner product $\langle \mathcal{L}', d\gamma \rangle = 0$, for every point of γ. And since $\mathcal{L}'_{\text{Lib}}$ is a constant multiple of $\mathcal{L}'$ by part (1), we have $\langle \mathcal{L}'_{\text{Lib}}, d\gamma \rangle = 0$ for every point of γ. This implies that $\mathcal{L}(a) = \mathcal{L}(b)$ and that there exists some $h : \mathbb{R} \to \mathbb{R}$ function such that $\mathcal{L} = h(\mathcal{L}_{\text{Lib}})$, though it is not necessarily differentiable or even continuous.

4. We claim that h should be differentiable. Let d be a vector. By $\partial_d \mathcal{L}_{\text{Lib}}(p) \neq 0$, then the directional derivative of $\mathcal{L}$ is

$$\begin{aligned}\partial_d \mathcal{L}(\boldsymbol{p}) &= \lim_{\epsilon \to 0} \frac{\mathcal{L}(\boldsymbol{p} + \epsilon d) - \mathcal{L}(\boldsymbol{p})}{h} \\ &= \lim_{\epsilon \to 0} \frac{h(\mathcal{L}_{\text{Lib}}(\boldsymbol{p} + \epsilon d)) - h(\mathcal{L}_{\text{Lib}}(\boldsymbol{p}))}{\mathcal{L}_{\text{Lib}}(\boldsymbol{p} + \epsilon d) - \mathcal{L}_{\text{Lib}}(\boldsymbol{p})} \cdot \frac{\mathcal{L}_{\text{Lib}}(\boldsymbol{p} + \epsilon d) - \mathcal{L}_{\text{Lib}}(\boldsymbol{p})}{\epsilon} \\ &= \partial_d \mathcal{L}_{\text{Lib}}(\boldsymbol{p}) \lim_{\epsilon \to 0} \frac{h(\mathcal{L}_{\text{Lib}}(\boldsymbol{p} + \epsilon d)) - h(\mathcal{L}_{\text{Lib}}(\boldsymbol{p}))}{\mathcal{L}_{\text{Lib}}(\boldsymbol{p} + \epsilon d) - \mathcal{L}_{\text{Lib}}(\boldsymbol{p})}\end{aligned}$$

By assumption $\mathcal{L}'_{\text{Lib}}$ and $\mathcal{L}'$ exist and they are continuous, therefore the above limit also exists which is just the derivative of h at $\mathcal{L}_{\text{Lib}}(\boldsymbol{p})$. That means that h is indeed continuously differentiable on the domain of $\mathcal{L}_{\text{Lib}}$, which is $\mathbb{R}$. Note that we fixed y at the very beginning. There are only finitely many such y over a set of m outputs, so we have a h function for every y, and putting these together gets the h that we want. ■

Appendix C. Theorems related to the Sag-loss

In this section we prove the characterization theorems for loss functions satisfying the bi-PRP$_s$ property. We recall the two theorems:

Theorem 14 *The* Sag-loss *function has the* bi-PRP$_s$ *property and for any continuously differentiable family of $h_i : \mathbb{R} \to \mathbb{R}$ functions $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y}) = h_k(\mathcal{L}_{\text{Sag}}(\boldsymbol{p}, \boldsymbol{y}))$ also satisfies the* bi-PRP$_s$ *property, where $k = \sum_i y_i$.*

Theorem 15 *Let $\mathcal{L}$ be a function that has the* bi-PRP$_s$ *property, invariant under the permutation of the input (i.e., $\forall \pi \in S_n, \mathcal{L}(\pi \circ \boldsymbol{p}, \pi \circ \boldsymbol{y}) = \mathcal{L}(\boldsymbol{p}, \boldsymbol{y})$). Then there exist $h_i : \mathbb{R} \to \mathbb{R}$ continuously differentiable functions such that $\mathcal{L}(\boldsymbol{p}, \boldsymbol{y}) = h_k(\mathcal{L}_{\text{Sag}}(\boldsymbol{p}, \boldsymbol{y}))$.*

We also recall the definition of the Sag-loss:

$$\mathcal{L}_{\text{Sag}}(\boldsymbol{p}, \boldsymbol{y}) = \underbrace{-\frac{1}{k} \sum_i y_i \log(p_i)}_{\text{Allowed term}} + \underbrace{\frac{1}{m-k} \sum_i (1 - y_i) \log(p_i)}_{\text{Disallowed term}}$$

As before, let $k = \sum_i y_i$, and $m - k = \sum_i (1 - y_i)$ denote the number of acceptable and unacceptable labels, respectively.

Proof [Proof of Theorem 14.]
Recall that the bi-PRP$_s$ property assumes a softmax regression model and that the logit vector $\boldsymbol{z}$ is a parameter vector. In order to show that $\mathcal{L}_{\text{Sag}}$ has the bi-PRP$_s$ property, we need to show that partial derivatives in z_j are equal whenever $y_j = 1$ and they are also equal whenever $y_j = 0$. First, we compute the partial derivatives in the probabilities p_j.

$$\frac{\partial}{\partial p_i} \mathcal{L}_{\text{Sag}}(\boldsymbol{p}, \boldsymbol{y}) = -\frac{1}{k} \frac{y_i}{p_i} + \frac{1}{m-k} \frac{1 - y_i}{p_i} = \begin{cases} -\frac{1}{k} \frac{1}{p_i} & \text{if } y_i = 1 \\ \frac{1}{m-k} \frac{1}{p_i} & \text{otherwise} \end{cases}$$

Now, we compute the partial derivatives in logit z_j. Recall that $\frac{\partial p_i}{\partial z_j}$ is the partial derivative of the softmax function which is $p_i(\delta_{ij} - p_j)$.

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\text{Sag}}}{\partial z_j} &= \sum_i \frac{\partial \mathcal{L}_{\text{Sag}}}{\partial p_i}\frac{\partial p_i}{\partial z_j} = \sum_i \left(-\frac{1}{k}\frac{y_i}{p_i} + \frac{1}{m-k}\frac{1-y_i}{p_i}\right) p_i(\delta_{ij} - p_j) \\
&= \sum_i \left(-\frac{1}{k}y_i + \frac{1}{m-k}(1-y_i)\right)(\delta_{ij} - p_j) \\
&= \left(-\frac{1}{k}y_j + \frac{1}{m-k}(1-y_j)\right) - p_j \sum_i \left(-\frac{1}{k}y_i + \frac{1}{m-k}(1-y_i)\right) \\
&= -\frac{1}{k}y_j + \frac{1}{m-k}(1-y_j) = \begin{cases} -\frac{1}{k} & \text{if } y_i = 1 \\ \frac{1}{m-k} & \text{otherwise} \end{cases}
\end{aligned}$$

We can observe that the gradients of the logits with $y_i = 1$ and those with $y_i = 0$ are equal, indicating that the loss function satisfies the bi-PRP$_{\text{s}}$ property.

■

Proof [Proof of Theorem 10.]
The proof follows along the same lines as in the the PRP$_{\text{s}}$ case. The revised outline is just as before:

1. We argue that $\mathcal{L}' = \kappa v$ for some $\kappa : \mathbb{R}^k \to \mathbb{R}$. and that $\mathcal{L}' = d \cdot \mathcal{L}'_{\text{Sag}}$ for some constant function d.

2. We show that the sets $H_z = \mathcal{L}_{\text{Sag}}^{-1}(\{z\})$ are path-connected.

3. We argue that $\mathcal{L}$ is constant on H_z for any z. Restated, this means that the function $h : \mathbb{R} \to \mathbb{R}$ as required by theorem (but not necessarily smooth) exists.

4. The h function is continuously differentiable.

1. Consider a loss function $\mathcal{L}$ that has the bi-PRP$_{\text{s}}$ property and satisfies the technical assumptions in the statement of the theorem. Let $\mathcal{L}'_{accept}$ denote the gradient restricted to acceptable inputs, and $\mathcal{L}'_{unaccept}$ the restriction to unacceptable outputs. We let $\mathcal{L}'_{\text{Sag},accept}$ and $\mathcal{L}'_{\text{Sag},unaccept}$ denote the special case where the loss is the Sag-loss. First, we show that $\mathcal{L}'$ and $\mathcal{L}'_{\text{Sag}}$ are scalar multiples of one another at any $\mathbf{p}$. Based on our assumption that the ratios of gradients for acceptable and unacceptable inputs are equivalent, we can infer that $\mathcal{L}'_{accept}$ and $\mathcal{L}'_{\text{Sag},accept}$ are scalar multiples of each other, as are $\mathcal{L}'_{unaccept}$ and $\mathcal{L}'_{\text{Sag},unaccept}$. However, we still need to prove that the constants for both pairs are identical.
Let v_{accept} and $v_{unaccept}$ denote the gradients of $\mathcal{L}$ with respect to the acceptable and unacceptable logits. Similarly, for $\mathcal{L}_{\text{Sag}}$, we use $v_{\text{Sag},accept}, v_{\text{Sag},unaccept}$. Furthermore, we will use v for the gradients of a general $\mathcal{L}$ (with respect to *logits*), without restricting to particular outputs. We similarly use v_{Sag} for the full gradient vector of $\mathcal{L}_{\text{Sag}}$, with respect to logits. Since $\mathcal{L}$ satisfies the bi-PRP$_{\text{s}}$ property, we have $v_{accept} = \kappa_{accept}\underline{1}$ and $v_{unaccept} = \kappa_{unaccept}\underline{1}$

for some $\kappa_{accept}, \kappa_{unaccept}$ scalars. For any $\mathcal{L}$, the gradients on the logits add to 0, since:

$$\sum_j \frac{\partial \mathcal{L}}{\partial z_j} = \sum_j \sum_i \frac{\partial \mathcal{L}}{\partial p_i}\frac{\partial p_i}{\partial z_j} = \sum_i \frac{\partial \mathcal{L}}{\partial p_i} \sum_j \frac{\partial p_i}{\partial z_j} = \sum_i \frac{\partial \mathcal{L}}{\partial p_i} \sum_j p_i(\delta_{ij} - p_j)$$
$$= \sum_i \frac{\partial \mathcal{L}}{\partial p_i} p_i \sum_j (\delta_{ij} - p_j) = 0$$

The last equality follows because the p_j form a probability distribution, hence for any fixed i, $\sum_j(\delta_{ij} - p_j) = 0$.
Since the gradients on the logits add to zero, we have

$$0 = v_{accept} + v_{unaccept} = k\kappa_{accept} + (m-k)\kappa_{unaccept}$$
$$0 = v_{\text{Sag},accept} + v_{\text{Sag},unaccept} = k\kappa_{\text{Sag},accept} + (m-k)\kappa_{\text{Sag},unaccept}$$

For this, it is easy to see that the ratios $\kappa_{accept} : \kappa_{\text{Sag},accept}$ and $\kappa_{unaccept} : \kappa_{\text{Sag},unaccept}$ have to be equal. Thus we have derived the following result:

Proposition 27 *v and v_{Sag} are scalar multiples of one another.*

Recall that the goal of part (1) of the proof is to show that $\mathcal{L}' = d \cdot \mathcal{L}'_{\text{Sag}}$, i.e., the gradients with respect to the probabilities of $\mathcal{L}$ and $\mathcal{L}_{\text{Sag}}$ are scalar multiple of one another. Proposition 27 shows the analog for the gradients with respect to the logits. But because of the chain rule, the gradients with respect to the probabilities and the logits are connected by a linear transformation. We define the vector to vector function A by

$$A_{i,j} = p_j(\delta_{ij} - p_i)$$

This is quite similar to the function A in the earlier proof of Theorem 10, but this time i, j range over all inputs, not just acceptable ones. The equality

$$\frac{\partial \mathcal{L}}{\partial z_j} = \sum_i \frac{\partial \mathcal{L}}{\partial p_i} p_i \sum_j (\delta_{ij} - p_j)$$

can be expressed in matrix multiplication terms as

$$v = A\mathcal{L}'$$

If A were invertible, then $\mathcal{L}' = A^{-1}v$, and it would follow that $\mathcal{L}' = d \cdot \mathcal{L}'_{\text{Sag}}$. Unfortunately, this is not true. From the fact that the function uses all inputs, which sum to 1, we can infer that $\det A = 0$, and so we cannot take the inverse of A over the entire input space.
Let V be the orthogonal complement of $\underline{1}$. This is all real vectors whose dot product with $\underline{1}$ is 0; that is, vectors whose sum is 0. We claim that V is invertible when we restrict to these vectors:

Claim 28 *A is invertible over V.*

We mentioned above that the gradients sum to 0, and the gradient with respect to the logits—that is, a v above—must be in V. Thus, from Claim 28 we are able to take an inverse of A over the relevant vectors, and derive that the partials with respect to the probabilities are scalar multiples, as before. We now turn to the proof of Claim 28.

Proof

Recall that A is the Jacobian of the softmax function, which is a surjective function from $\mathbb{R}^n$, the space of logits, to the space of probability distributions over m categories. The latter is an $m-1$ dimensional subspace of $\mathbb{R}^n$. We already showed that $\underline{1}$ is in the kernel of A. Let f denote the softmax function, then $A := df$. Since f is a smooth and surjective function, the rank of df is equal to the dimension of the codomain, i.e. the space of probability distributions, which has dimension $m-1$. It follows that $\dim \ker A = m - \text{rank } df = 1$, consequently $\ker A$ is generated by $\underline{1}$ and so A is invertible over V, as required. ■

2. Analogous to what we did in the PRP case, we argue for path-connectedness of $\mathcal{L}_{\text{Sag}}$. Let $H_z = \mathcal{L}_{\text{Sag}}^{-1}(\{z\}) = \{\boldsymbol{p} | \mathcal{L}_{\text{Sag}}(\boldsymbol{p}, \boldsymbol{y}) = z\}$ be the preimage of z. Let $\mathcal{P}$ be the set of distributions with each probability non-zero and neither the acceptable nor the unacceptable outputs sum to 1.

Note that $\mathcal{P}$ is an open path-connected subset of $\mathbb{R}^n$ and hence a differentiable manifold. At the same time, the range of $\mathcal{L}_{\text{Sag}}$ is $\mathbb{R}$, which is also a differentiable manifold. Thus we can view $\mathcal{L}_{\text{Sag}}$ as a smooth map between manifolds $\mathcal{P}$ and $\mathbb{R}$. We will show the analogous claim as for PRP:

Claim 29 *Each H_z is a differentiable manifold.*

Proof Let X, Y be two differentiable manifolds and $f : X \to Y$ a smooth map between them. We say that $y \in Y$ is regular if for every $x \in f^{-1}(y)$ the map $df_x : T_x X \to T_y Y$ is surjective, where $T_x X$ is the tangent space of X in x.

We again use that fact that if $y \in Y$ is a regular value of f, then $f^{-1}(y)$ is a differentiable submanifold of X. We show that $d\mathcal{L}_{\text{Sag}}$ is surjective everywhere, in order to argue, using the fact above, that the pre-image of a single point is a differentiable manifold.

Since $\mathbb{R}$ is one-dimensional, $d\mathcal{L}_{\text{Sag}}$ is not surjective precisely when $d\mathcal{L}_{\text{Sag}}$ is the zero map. Equivalently the gradient is zero, which can only occur on the boundary of $\mathcal{P}$. Therefore, any z is a regular value of H, and consequently H_z is a differentiable submanifold of $\mathcal{P}$. ■

For any probability distribution $\boldsymbol{p} = (p_1 \dots p_n)$ over m categories, we let $D_{\boldsymbol{p}}$ denote all distributions that agree with $\boldsymbol{p}$ on both the ratios of acceptable values, as well as on the ratio of unacceptable values, with both of these nonzero. That is, $D_{\boldsymbol{p}}$ is the subset of $\mathcal{P}$ that we get by fixing the ratios for both acceptable and unacceptable values.

We again proceed analogously to the PRP case:

Claim 30 *For each fixed $\boldsymbol{y}$ having 1 on entries for $\mathbf{a}$ and 0 on entries for $\mathbf{u}$, $\mathcal{L}_{\text{Sag}}$ takes every value precisely once on $D_{\boldsymbol{p}}$.*

Proof Let us fix non-trivial distributions $\mathbf{a}$ on acceptable outputs and $\mathbf{u}$ on unacceptable outputs with the sum of the entries of both coming to 1. $D_{\boldsymbol{p}}$ consists of the distributions

$\omega\mathbf{a}$, $(1-\omega)\mathbf{u}$ for all $0 < \omega < 1$. To prove the claim, observe that the loss is

$$-\frac{1}{k}\left(\sum_{i\in A}\log(\omega a_i)\right) + \frac{1}{m-k}\sum_{i\in U}\log((1-\omega)u_i)$$

Here A are the indices of acceptable values and U the indices of unacceptable values. Note that this simplifies to an expression of the form

$$-\log(\omega) - \frac{1}{k}\sum_{i\in A}\log(a_i) + \log(1-\omega) + \frac{1}{m-k}\sum_{i\in U}u_i$$

If we ignore terms without ω, this is $-\log(\omega)+\log(1-\omega)$. Thus we see, as in the PRP case, when $\omega \to 0$ it converges to ∞ and when $\omega \to 1$ it converges to $-\infty$. And differentiating with respect to ω, we see that the above mapping is monotonically strictly decreasing in ω, and consequently $\mathcal{L}_{\text{Sag}}$ takes every value of $\mathbb{R}$ precisely once on the set. ■

Recall that we are interested in showing path connectedness of the set H_z, the pre-image of singletons under $\mathcal{L}_{\text{Sag}}$. By the claim above, we know that as we vary $\boldsymbol{p}$, $\mathcal{L}_{\text{Sag}}$ will always hit H_z exactly once on the set $D_{\boldsymbol{p}}$, but the point at which it hits H_z will vary with $\boldsymbol{p}$.
Let π be the quotient map equating two elements if they are in the same $D_{\boldsymbol{p}}$. Thus by definition the preimage of a point under π is precisely a set $D_{\boldsymbol{p}}$ for some $\boldsymbol{p}$. Since we have observed above that $\mathcal{L}_{\text{Sag}}$ takes every value once over a fixed $D_{\boldsymbol{p}}$, we conclude that π is a bijection between $\pi(\mathcal{P})$ and H_z. We will show in the next paragraph that π is an open map, but first introduce a useful lemma.

Lemma 31 *If f is a quotient map, then f is open if and only if*

$$U \subset X \text{ is open } \Rightarrow f^{-1}(f(U)) \text{ is open}$$

Proof If f is open, then $f(U)$ is open and so $f^{-1}(f(U))$. For the converse, the fact that $f^{-1}(f(U))$ is open implies that $f(U)$ is open, because f is a quotient map. Since this holds for every U open, it follows that f is open. ■

We use this fact to show that π is open. More precisely, we have that for every open $U \subset \mathcal{P}$, $\pi^{-1}(\pi(U)) = \bigcup_{\boldsymbol{p}\in U} D_{\boldsymbol{p}}$. Unfortunately, the sets $D_{\boldsymbol{p}}$ lines are not open subsets; so we cannot deduce directly that $\bigcup_{\boldsymbol{p}\in U} D_{\boldsymbol{p}}$ is open. Let $\mathcal{S}$ denote the set of linear functions that send probabilities to probabilities, such that the ratio is preserved for the acceptable and also for the unacceptable outputs. Since the functions in $\mathcal{S}$ are linear, they are also open maps. Moreover, we have $D_{\boldsymbol{p}} = \cup_{S\in\mathcal{S}} S(\boldsymbol{p})$. Therefore, we can write $\bigcup_{\boldsymbol{p}\in U} D_{\boldsymbol{p}} = \bigcup_{S\in\mathcal{S}} S(U)$. Since U is open, each $S(U)$ is also open, and thus so is the union over all S. We conclude that $\pi^{-1}(\pi(U))$ is open.
We will now use the fact that H_z is a manifold by Claim 29. It is known that for manifolds, connectedness and path-connectedness are equivalent. If H_z were not connected, then there would be U, V disjoint non-empty open sets such that $H_z \subset U \cup V$. Note that the image of π is connected: we start with a connected space, namely the whole probability space, and take quotient by a continuous function. Thus $\pi(\mathcal{P})$ is connected, and $\pi\mathcal{P} = \pi(H_z) \subset$

$\pi(U \cup V) = \pi(U) \cup \pi(V)$. Note that $\pi(U), \pi(V)$ are open because π is an open map. We cannot have two disjoint open sets covering the connected set $\pi(\mathcal{P})$. Thus the sets $\pi(U)$ and $\pi(V)$ must overlap: there are $u \in U, v \in V$ points such that $\pi(u) = \pi(v)$. Thus there are two distinct points in the pre-image of π with the same value. Since the pre-image is a $D_{\boldsymbol{p}}$ line, this contradicts Claim 30.

3. We show that $\mathcal{L}$ is constant on H_z for any z, and that a $h : \mathbb{R} \to \mathbb{R}$ function exists such that $\mathcal{L} = h(\mathcal{L}_{\text{Sag}})$.

The idea will be that for any $a \neq b \in H_z$ we show $\mathcal{L}(b) - \mathcal{L}(a) = 0$. We do this by computing $\mathcal{L}(b) - \mathcal{L}(a)$ as an integral of a quantity, over a path in H_z between a and b, using the fact that H_z is path-connected. The quantity will involve a dot product with the derivative of $\mathcal{L}$, and we will use part (1) to argue that this dot product is always 0.

We will again make use of Proposition 25, which states that $F(b) - F(a) = \int_\gamma \langle F', d\gamma \rangle$. We can again apply this to $\mathcal{L}$ to get

$$\mathcal{L}(b) - \mathcal{L}(a) = \int_\gamma \langle \mathcal{L}', d\gamma \rangle$$

By Proposition 26, the inner product with $\mathcal{L}'_{\text{Sag}}$ in place of $\mathcal{L}'$ is 0 within a constant surface H_z. And again since $\mathcal{L}'$ is always a scalar multiple of $\mathcal{L}'_{\text{Sag}}$, we conclude In the last line, we used the assumption that the gradients of $\mathcal{L}_{\text{Sag}}$ and $\mathcal{L}$ have a constant ratio.

The argument that h is differentiable is almost identical to the argument for PRP. ■

Appendix D. Label Dependent Noise Models for Synthetic PLL Data Sets

In Subsection 6.3 we described a model for adding distractors synthetically to a real data set. Here we provide more detail.

Wen et al. (2021) introduces three PLL noise models for classification with $m = 10$ labels. The models are instance- independent, i.e., the noise only depends on the true label. Figure 9 presents results based on 5 such noise matrices. Of these the first three are taken directly from Wen et al. (2021) and the last two are harder variants created by us.

The noise models are represented as $[m \times m]$ matrices M where M_{ij} represents the probability of label j becoming a distractor given true label i. In the following we describe these 5 noise matrices.

Case	Noise Matrix	Description
1	$\begin{bmatrix} 1 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.5 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$	There is a single potential distractor for each true label, which is present with probability 0.5. The expected number of distractors is 0.5.
2	$\begin{bmatrix} 1 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 \\ 0.3 & 1 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 1 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 1 & 0.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 1 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & 1 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 1 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 1 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 1 & 0.3 \\ 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 1 \end{bmatrix}$	There are two potential distractors for each true label, each of which is present with probability 0.3. The expected number of distractors is 0.6.
3	$\begin{bmatrix} 1 & 0.5 & 0.3 & 0.1 & 0 & 0 & 0 & 0.1 & 0.3 & 0.5 \\ 0.5 & 1 & 0.5 & 0.3 & 0.1 & 0 & 0 & 0 & 0.1 & 0.3 \\ 0.3 & 0.5 & 1 & 0.5 & 0.3 & 0.1 & 0 & 0 & 0 & 0.1 \\ 0.1 & 0.3 & 0.5 & 1 & 0.5 & 0.3 & 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0.3 & 0.5 & 1 & 0.5 & 0.3 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0.3 & 0.5 & 1 & 0.5 & 0.3 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 & 0.3 & 0.5 & 1 & 0.5 & 0.3 & 0.1 \\ 0.1 & 0 & 0 & 0 & 0.1 & 0.3 & 0.5 & 1 & 0.5 & 0.3 \\ 0.3 & 0.1 & 0 & 0 & 0 & 0.1 & 0.3 & 0.5 & 1 & 0.5 \\ 0.5 & 0.3 & 0.1 & 0 & 0 & 0 & 0.1 & 0.3 & 0.5 & 1 \end{bmatrix}$	For each true label, there are 2 potential distractors with probability 0.5, 2 with probability 0.3 and 2 with probability 0.1. The expected number of distractors is 1.8.
4	$\begin{bmatrix} 1 & 0.2 & 0.8 & 0.8 & 0.8 & 0.4 & 0.4 & 0.2 & 0.2 & 0.2 \\ 0.2 & 1 & 0.2 & 0.8 & 0.8 & 0.8 & 0.4 & 0.4 & 0.2 & 0.2 \\ 0.2 & 0.2 & 1 & 0.2 & 0.8 & 0.8 & 0.8 & 0.4 & 0.4 & 0.2 \\ 0.2 & 0.2 & 0.2 & 1 & 0.2 & 0.8 & 0.8 & 0.8 & 0.4 & 0.4 \\ 0.4 & 0.2 & 0.2 & 0.2 & 1 & 0.2 & 0.8 & 0.8 & 0.8 & 0.4 \\ 0.4 & 0.4 & 0.2 & 0.2 & 0.2 & 1 & 0.2 & 0.8 & 0.8 & 0.8 \\ 0.8 & 0.4 & 0.4 & 0.2 & 0.2 & 0.2 & 1 & 0.2 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.4 & 0.4 & 0.2 & 0.2 & 0.2 & 1 & 0.2 & 0.8 \\ 0.8 & 0.8 & 0.8 & 0.4 & 0.4 & 0.2 & 0.2 & 0.2 & 1 & 0.2 \\ 0.2 & 0.8 & 0.8 & 0.8 & 0.4 & 0.4 & 0.2 & 0.2 & 0.2 & 1 \end{bmatrix}$	For each true label, there are 4 potential distractors with probability 0.2, 3 with probability 0.8 and 2 with probability 0.4. The expected number of distractors is 4.
5	$\begin{bmatrix} 1 & 0.9 & 0.8 & 0.8 & 0.8 & 0.7 & 0.7 & 0.6 & 0.9 & 0.9 \\ 0.9 & 1 & 0.9 & 0.8 & 0.8 & 0.8 & 0.7 & 0.7 & 0.6 & 0.9 \\ 0.9 & 0.9 & 1 & 0.9 & 0.8 & 0.8 & 0.8 & 0.7 & 0.7 & 0.6 \\ 0.6 & 0.9 & 0.9 & 1 & 0.9 & 0.8 & 0.8 & 0.8 & 0.7 & 0.7 \\ 0.7 & 0.6 & 0.9 & 0.9 & 1 & 0.9 & 0.8 & 0.8 & 0.8 & 0.7 \\ 0.7 & 0.7 & 0.6 & 0.9 & 0.9 & 1 & 0.9 & 0.8 & 0.8 & 0.8 \\ 0.8 & 0.7 & 0.7 & 0.6 & 0.9 & 0.9 & 1 & 0.9 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.7 & 0.7 & 0.6 & 0.9 & 0.9 & 1 & 0.9 & 0.8 \\ 0.8 & 0.8 & 0.8 & 0.7 & 0.7 & 0.6 & 0.9 & 0.9 & 1 & 0.9 \\ 0.9 & 0.8 & 0.8 & 0.8 & 0.7 & 0.7 & 0.6 & 0.9 & 0.9 & 1 \end{bmatrix}$	For each true label, there are 3 potential distractors with probability 0.9, 3 with probability 0.8, 2 with probability 0.7 and 1 with probability 0.6. The expected number of distractors is 7.1.

References

Kareem Ahmed, Eric Wang, Kai-Wei Chang, and Guy Van den Broeck. Neuro-symbolic entropy regularization. In *UAI*, 2022. URL https://proceedings.mlr.press/v180/ahmed22a.html.

Leonard E. Baum and Ted Petrie. Statistical inference for probabilistic functions of finite-state Markov chains. *Annals of Mathematical Statistics*, 37(6):1554–1563, 1966. URL https://doi.org/10.1214/aoms/1177699147.

BESS project'23. Better exploration for symbolic supervision project webpage, 2023. URL https://sites.google.com/view/symbolicsupervision.

Forrest Briggs, Xiaoli Z Fern, and Raviv Raich. Rank-loss support instance machines for miml instance annotation. In *SIGKDD*, 2012. URL https://doi.org/10.1145/2339530.2339616.

Etienne Côme, Latifa Oukhellou, Thierry Denoeux, and Patrice Aknin. Learning from partially supervised data using mixture models and belief functions. *Pattern Recogn.*, 42 (3):334–348, mar 2009. ISSN 0031-3203. URL https://doi.org/10.1016/j.patcog.2008.07.014.

Timothee Cour, Ben Sapp, and Ben Taskar. Learning from partial labels. *Journal of Machine Learning Research*, 12(5):1501–1536, 2011. URL https://www.jmlr.org/papers/volume12/cour11a/cour11a.pdf.

James Curran and Steven Clark. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–553, 2017. URL https://aclanthology.org/J07-4004.pdf.

Luc De Raedt and Sašo Džeroski. First-order jk-clausal theories are PAC-learnable. *Artif. Intell.*, 70(1-2):375–392, 1994. URL https://doi.org/10.1016/0004-3702(94)90112-0.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018. URL https://www.jair.org/index.php/jair/article/download/11172/26376/.

Lei Feng and Bo An. Partial label learning with self-guided retraining. In *AAAI*, 2019. URL https://dl.acm.org/doi/10.1609/aaai.v33i01.33013542.

Lei Feng, Jiaqi Lv, Bo Han, Miao Xu, Gang Niu, Xin Geng, Bo An, and Masashi Sugiyama. Provably consistent partial-label learning. In *NeurIPS*, 2020. URL https://dl.acm.org/doi/abs/10.5555/3495724.3496643.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010. URL https://proceedings.mlr.press/v9/glorot10a.html.

Georg Gottlob, Nicola Leone, and Francesco Scarcello. On the complexity of some inductive logic programming problems. *New Generation Computing*, 17(1):53–75, 1999. URL https://doi.org/10.1007/BF03037582.

Yves Grandvalet and Yoshua Bengio. Learning from Partial Labels with Minimum Entropy. CIRANO Working Papers 2004s-28, CIRANO, May 2004. URL https://ideas.repec.org/p/cir/cirwor/2004s-28.html.

Matthieu Guillaumin, Jakob Verbeek, and Cordelia Schmid. Multiple instance metric learning from automatically labeled bags of faces. In *ECCV*, 2010. URL https://doi.org/10.1007/978-3-642-15549-9_46.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *ACL*, 2017. URL https://aclanthology.org/P17-1097.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. URL https://doi.org/10.1109/CVPR.2016.90.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *ACL*, 2016. URL https://aclanthology.org/P16-1228.

Gauri Jagatap, Ameya Joshi, Animesh Basak Chowdhury, Siddharth Garg, and Chinmay Hegde. Adversarially robust learning via entropic regularization. *Frontiers in Artificial Intelligence*, 4, 2022. URL https://doi.org/10.3389/frai.2021.780843.

Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G. Skjæveland, Evgenij Thorstensen, and Jose Mora. BootOX: Practical Mapping of RDBs to OWL 2. In *ISWC*, 2015. URL https://doi.org/10.1007/978-3-319-25010-6_7.

Rong Jin and Zoubin Ghahramani. Learning with multiple labels. In *NeurIPS*, 2002. URL https://dl.acm.org/doi/10.5555/2968618.2968733.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Liping Liu and Thomas Dietterich. A conditional multinomial mixture model for superset label learning. In *NeurIPS*, 2012. URL https://dl.acm.org/doi/10.5555/2999134.2999196.

Emanuele Marconato, Samuele Bortolotti, Emile van Krieken, Antonio Vergari, Andrea Passerini, and Stefano Teso. BEARS make neuro-symbolic models aware of their reasoning shortcuts. In *Conference on Uncertainty in Artificial Intelligence*, 2024. URL https://openreview.net/forum?id=pDcM1k7mgZ.

Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *ICML*, 2020. URL https://dl.acm.org/doi/10.5555/3524938.3525582.

Nam Nguyen and Rich Caruana. Classification with partial labels. In *KDD*, 2008. URL https://doi.org/10.1145/1401890.1401958.

Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton. Regularizing neural networks by penalizing confident output distributions. In *ICLR*, 2017. URL https://openreview.net/forum?id=HyhbYrGYe.

Christoph Pinkel, Carsten Binnig, Evgeny Kharlamov, and Peter Haase. IncMap: pay as you go matching of relational schemata to OWL ontologies. In *ISWC-PD*, 2013. URL https://dl.acm.org/doi/10.5555/2874399.2874456.

Christoph Pinkel, Carsten Binnig, Ernesto Jiménez-Ruiz, Wolfgang May, Dominique Ritze, Martin G. Skjæveland, Alessandro Solimando, and Evgeny Kharlamov. RODI: A benchmark for automatic mapping generation in relational-to-ontology data integration. In *ESWC*, 2015. URL https://doi.org/10.1007/978-3-319-18818-8_2.

Christoph Pinkel, Carsten Binnig, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Wolfgang May, Andriy Nikolov, Ana Sasa Bastinos, Martin G. Skjæveland, Alessandro Solimando, Mohsen Taheriyan, Christian Heupel, and Ian Horrocks. RODI: Benchmarking relational-to-ontology mapping generation quality. *Semantic Web*, 9(1):25–52, 2018. URL https://doi.org/10.3233/SW-170268.

Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008. URL http://www.w3.org/TR/rdf-sparql-query/.

Meng Qu, Junkun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs. In *ICLR*, 2021. URL https://openreview.net/forum?id=tGZu6DlbreV.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NeurIPS*, 2017. URL https://dl.acm.org/doi/10.5555/3294996.3295136.

Martin G. Skjæveland, Espen H. Lian, and Ian Horrocks. Publishing the Norwegian Petroleum Directorate's FactPages as Semantic Web Data. In *ISWC*, 2013. URL https://doi.org/10.1007/978-3-642-41338-4_11.

Yingjie Tian, Xiaotong Yu, and Saiji Fu. Partial label learning: Taxonomy, analysis and outlook. *Neural Networks*, 161(C):708–734, 2023. URL https://doi.org/10.1016/j.neunet.2023.02.019.

Grigorios Tsoumakas and Ioannis Katakis. Multi label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007. URL http://mlkd.csd.auth.gr/publication_details.asp?publicationID=219.

Emile van Krieken, Pasquale Minervini, Edoardo Ponti, and Antonio Vergari. On the independence assumption in neurosymbolic learning. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=S1gSrruVd4.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. URL https://dl.acm.org/doi/10.5555/3295222.3295349.

W3C. Web ontology language, 2012. URL www.w3.org/OWL/.

Hongwei Wen, Jingyi Cui, Hanyuan Hang, Jiabin Liu, Yisen Wang, and Zhouchen Lin. Leveraged weighted loss for partial label learning. In *ICML*, 2021. URL http://proceedings.mlr.press/v139/wen21a.html.

Yaqi Xie, Ziwei Xu, Mohan S Kankanhalli, Kuldeep S Meel, and Harold Soh. Embedding symbolic knowledge into deep networks. In *NeurIPS*, 2019. URL https://dl.acm.org/doi/abs/10.5555/3454287.3454668.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, 2018. URL https://proceedings.mlr.press/v80/xu18h.html.

Yao Yao, Jiehui Deng, Xiuhua Chen, Chen Gong, Jianxin Wu, and Jian Yang. Deep discriminative CNN with temporal ensembling for ambiguously-labeled image classification. In *AAAI*, 2020. URL https://doi.org/10.1609/aaai.v34i07.6959.

Wei Ai Cun Zai. Practice on CIFAR100 using PyTorch. https://github.com/weiaicunzai/pytorch-cifar100, 2017.

Zinan Zeng, Shijie Xiao, Kui Jia, Tsung-Han Chan, Shenghua Gao, Dong Xu, and Yi Ma. Learning by associating ambiguously labeled images. In *CVPR*, 2013. URL https://doi.org/10.1109/CVPR.2013.97.