BACKPROPAGATION THROUGH COMBINATORIAL ALGORITHMS: IDENTITY WITH PROJECTION WORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Embedding discrete solvers as differentiable layers has given modern deep learning architectures combinatorial expressivity and discrete reasoning capabilities. The derivative of these solvers is zero or undefined, therefore a meaningful replacement is crucial for effective gradient-based learning. Prior works rely on smoothing the solver with input perturbations, relaxing the solver to continuous problems, or interpolating the loss landscape with techniques that typically require additional solver calls, introduce extra hyper-parameters or compromise performance. We propose a principled approach to exploit the geometry of the discrete solution space to treat the solver as a negative identity on the backward pass and further provide a theoretical justification. Our experiments demonstrate that such a straightforward hyper-parameter-free approach is on-par with or outperforms previous more complex methods on numerous experiments such as Traveling Salesman Problem, Deep Graph Matching, and backpropagation through discrete samplers. Furthermore, we substitute the previously proposed problem-specific and label-dependent margin by a generic regularization procedure that prevents cost collapse and increases robustness.

1 INTRODUCTION

Deep neural networks have achieved astonishing results in solving problems on raw inputs. However, in key domains such as planning or reasoning, deep networks need to make discrete decisions, which can be naturally formulated via constrained combinatorial optimization problems. In many settings—including shortest path finding (Vlastelica et al., 2020; Berthet et al., 2020), optimizing rank-based objective functions (Rolínek et al., 2020a), keypoint MATCHING (Rolínek et al., 2020b; Paulus et al., 2021), Sudoku solving (Amos and Kolter, 2017; Wang et al., 2019), solving the knapsack problem from sentence descriptions (Paulus et al., 2021)—neural models that embed optimization modules as part of their layers achieve improved performance, data-efficiency, and generalization (Vlastelica et al., 2020; Amos and Kolter, 2017; Ferber et al., 2020; P. et al., 2021).

This paper explores the end-to-end training of deep neural network models with embedded discrete combinatorial algorithms (*solvers*, for short) and derives simple and efficient gradient estimators for these architectures. Deriving an informative gradient through the solver constitutes the main challenge, since the true gradient is, due to the discreteness, zero almost everywhere. Most notably, Blackbox Backpropagation (BB) by Vlastelica et al. (2020) introduces a simple method that yields an informative gradient by applying an informed perturbation to the solver input and calling the solver one additional time. This results in a gradient of an implicit piecewise-linear loss interpolation, whose locality is controlled by a hyperparameter λ.

We propose a fundamentally different strategy by dropping the constraints on the solver solutions and simply propagating the incoming gradient through the solver, effectively treating the discrete block as a negative identity on the backward pass. This approach can be seen as a generalization of the straight-through estimator (STE)—a popular technique for differentiating through discrete samples—to combinatorial optimization problems. While our gradient replacement is simple and cheap to compute, it comes with important considerations, as its naïve application can result in unstable learning behavior, as described in the following.

Our considerations are focused on invariances of typical combinatorial problems under specific transformations of the cost vector. These transformations usually manifest as projections or nor-

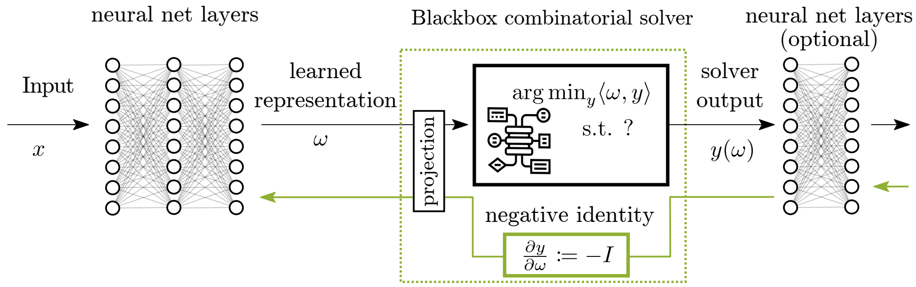


Figure 1: Hybrid architecture with blackbox combinatorial solver and Identity module (green dotted line) with the projection of a cost ω and negative identity on the backward pass.

malizations, e.g. as an immediate consequence of the linearity of the objective, the combinatorial solver is agnostic to normalization of the cost vector. Such invariances, if unattended, can hinder fast convergence when used in combination with adaptive optimizers, or can result in divergence and cost collapse (Rolínek et al., 2020a). We propose to exploit the knowledge of such invariances by including the respective transformations in the computation graph. On the forward pass this leaves the solution unchanged, but on the backward pass removes the malicious part of the update. In our experiments, we show that this technique is crucial to the success of our proposed method.

In addition, we improve the robustness of our method by adding noise to the cost vector, which induces a margin on the learned solutions and thereby subsumes previously proposed ground-truth-informed margins (Rolínek et al., 2020a). With these considerations taken into account, our simple method achieves strong empirical performance. Moreover, it avoids a costly call to the solver on the backward pass and requires fewer hyperparameters than previous methods.

Our contributions can be summarized as follows: **(i)** A hyperparameter-free method for linear-cost solver differentiation not requiring any additional calls to the solver on the backward pass. **(ii)** Exploiting invariances via cost projections tailored to the combinatorial problem. **(iii)** Increasing robustness by replacing previously proposed informed margin with a noise perturbation. **(iv)** Analysis of the robustness of differentiation methods to perturbations during training.

2 Related Work

Optimizers as Model Building Blocks. It has been shown in various application domains that optimization on prediction is beneficial for model performance and generalization. One such area is meta-learning, where methods backpropagate through multiple steps of gradient descent for few-shot adaptation in a multi-task setting (Finn et al., 2017; Raghu et al., 2020). Along these lines, algorithms that effectively embed more general optimizers into differentiable architectures have been proposed such as convex optimization (Agrawal et al., 2019a; Lee et al., 2019), quadratic programs (Amos and Kolter, 2017), conic optimization layers (Agrawal et al., 2019b), and more.

Combinatorial Solver Differentiation. Many important problems require discrete decisions and hence, using *combinatorial* solvers as layers have sparked research interest (Domke, 2012; Elmachtoub and Grigas, 2022). Methods, such as SPO (Elmachtoub and Grigas, 2022) and MIPaaL Ferber et al. (2020), assume access to true target costs, a scenario we are not considering. Berthet et al. (2020) differentiate through discrete solvers by sample-based smoothing. Blackbox Backpropagation (BB) by Vlastelica et al. (2020) returns the gradient of an implicitly constructed piecewise-linear interpolation. Modified instances of this approach have been applied in various settings such as ranking (Rolínek et al., 2020a), keypoint matching (Rolínek et al., 2020b), and imitation learning (Vlastelica et al., 2020). I-MLE by Niepert et al. (2021) adds noise to the solver to model a discrete probability distribution and uses the BB update to compute informative gradients. Previous works have also considered adding a regularization to the linear program, including differentiable top-k-selection (Amos et al., 2019) and differentiable ranking and sorting (Blondel et al., 2020). Another common approach is to differentiate a softened solver, for instance in (Wilder et al., 2019) or (Wang et al., 2019) for MaxSat. Finally, Paulus et al. (2021) extend approaches for learning the cost coefficients to learning also the constraints of integer linear programs.

Learning to Solve Combinatorial Problems. An orthogonal line of work to ours is differentiable learning of combinatorial algorithms or their improvement by data-driven methods. Examples of such algorithms include learning branching strategies for MIPs (Balcan et al., 2018; Khalil et al., 2016; Alvarez et al., 2017), learning to solve SMT formulas (Balunovic et al., 2018), learning to solve linear programs (Mandi and Guns, 2020; Tan et al., 2020). A natural way of dealing with the lack of gradient information in combinatorial problems is reinforcement learning which is prevalent among these methods (Khalil et al., 2016; Bello et al., 2017; Nazari et al., 2018; Zhang and Dietterich, 2000). Further progress has been made in applying graph neural networks for learning classical programming algorithms (Velickovic et al., 2018; 2020) and latent value iteration (Deac et al., 2020). Further work in this direction can be found in the review Li et al. (2021). Another interesting related area of work is program synthesis, or "learning to program", which has gained traction (Ellis et al., 2018; Inala et al., 2020).

3 Method

We consider architectures that contain differentiable blocks, such as neural network layers, and combinatorial blocks, as sketched in Fig. 1. In this work, a combinatorial block uses an algorithm (called *solver*) to solve an optimization problem of the form

$$y(\omega) = \arg\min_{y \in Y} \langle \omega, y \rangle, \tag{1}$$

where $\omega \in W \subseteq \mathbb{R}^n$ is the cost vector produced by a previous block, $Y \subset \mathbb{R}^n$ is *any finite* set of possible solutions and $y(\omega) \in Y$ is the solver's output. Without loss of generality, Y consists only of extremal points of its convex hull, as no other point can be a solution of optimization (1). This formulation covers linear programs as well as integer linear programs.

3.1 Differentiating through Combinatorial Solvers

We consider the case in which the solver is embedded inside the neural network, meaning that the costs ω are predicted by a backbone, the solver is called, and *the solution $y(\omega)$ is post-processed* before the final loss ℓ is computed. For instance, this is the case when a specific choice of the loss is crucial, or the solver is followed by additional learnable components.

We aim to train the entire architecture end-to-end, which requires computing gradients in a layer-wise manner during backpropagation. However, the true derivative of the solver $y(\omega)$ is *either zero or undefined*, as the relation between the optimal solution $y(\omega)$ and the cost vector ω is piecewise constant. Thus, it is crucial to contrive a *meaningful replacement* for the true zero gradient of the combinatorial block. See Fig. 1 for an illustration.

3.2 Identity Update: Intuition in Simple Case

On the backward pass, the negated incoming gradient $-\mathrm{d}\ell/\mathrm{d}y$ gives us the local information of where we expect solver solutions with a lower loss. We first consider the simple scenario in which $-\mathrm{d}\ell/\mathrm{d}y$ points directly towards another solution y^*, referred to as the "target". This means that there is some $\eta > 0$ such that $-\eta\frac{\mathrm{d}\ell}{\mathrm{d}y} = y^* - y(\omega)$. This happens, for instance, if the final layer coincides with the ℓ_2 loss (then $\eta = 1/2$), or for ℓ_1 loss with Y being a subset of $\{0,1\}^n$ (then $\eta = 1$).

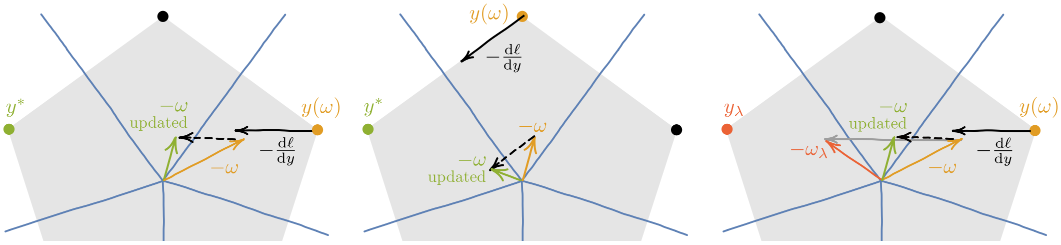
Our aim is to update ω in a manner which decreases the objective value associated with y^*, i.e. $\langle \omega, y^* \rangle$, and increases the objective value associated with the current solution $y(\omega)$, i.e. $\langle \omega, y(\omega) \rangle$. Therefore, y^* will be favoured over $y(\omega)$ as the solution in the updated optimization problem. This motivates us to set the replacement for the true zero gradient $\mathrm{d}\ell/\mathrm{d}\omega$ to

$$\tfrac{\mathrm{d}}{\mathrm{d}\omega}\langle \omega, y^* - y(\omega) \rangle = y^* - y(\omega) = -\eta\tfrac{\mathrm{d}\ell}{\mathrm{d}y}. \tag{2}$$

The scaling factor η is subsumed into the learning rate, therefore we propose the update

$$\Delta^{\mathrm{I}}\omega = -\tfrac{\mathrm{d}\ell}{\mathrm{d}y}. \tag{3}$$

This corresponds to simply treating the solver as a negated identity on the backward pass, hence we call our method "Identity". An illustration of how the repeated application of this update leads to the correct solution is provided in Fig. 2a and Fig. 2b.



(a) Id update to neighboring solution (b) Id update to final solution (c) BB update (identical for some λ)

Figure 2: Intuitive illustration of the Identity (Id) gradient (without normalization) and its equivalence to Blackbox Backpropagation (BB) when $-\mathrm{d}\ell/\mathrm{d}y$ points directly to a target y^*. The cost and solution spaces are overlayed; the cost space partition into sets resulting in the same solution are drawn in blue.

Comparison to Blackbox Backpropagation. To strengthen the intuition of why update (3) results in a sensible update, we offer a comparison to BB by Vlastelica et al. (2020) that proposes an update

$$\Delta^{\mathrm{BB}}\omega = \frac{1}{\lambda}\big(y_\lambda(\omega) - y(\omega)\big), \tag{4}$$

where $y(\omega)$ is the solution from the forward pass and, on the backward pass, the solver is called again on a perturbed cost to get $y_\lambda(\omega) = y(\omega + \lambda \mathrm{d}\ell/\mathrm{d}y)$. Here, $\lambda > 0$ is a hyperparameter that controls the locality of the implicit linear interpolation of the loss landscape.

Observe that $y_\lambda(\omega)$ serves as a target for our current $y(\omega)$, similar to the role of y^* in computation (2). If λ in BB coincides with a steps size that Identity needs to reach the target y^* with update $\Delta^{\mathrm{I}}\omega$, then

$$\Delta^{\mathrm{BB}}\omega = \frac{1}{\lambda}\Big(y\big(\omega + \lambda\tfrac{\mathrm{d}\ell}{\mathrm{d}y}\big) - y(\omega)\Big) = \frac{1}{\lambda}\big(y^* - y(\omega)\big) = -\frac{\eta}{\lambda}\frac{\mathrm{d}\ell}{\mathrm{d}y} = \frac{\eta}{\lambda}\Delta^{\mathrm{I}}\omega. \tag{5}$$

Therefore, if $-\mathrm{d}\ell/\mathrm{d}y$ points directly towards a neighboring solution, the Identity update is equivalent to the BB update with the smallest λ that results in a non-zero gradient in (4). This situation is illustrated in Fig. 2c. However, Identity does not require an additional call to the solver on the backward pass, nor does it have an additional hyperparameter that needs to be tuned.

3.3 Identity Update: General Case

We will now consider the general case, in which we do not expect the solver to find a better solution in the direction $\Delta^{\mathrm{I}}\omega = -\mathrm{d}\ell/\mathrm{d}y$ due to the constraints on Y. We show that if we ignore the constraints on Y and simply apply the Identity method, we still achieve a sensible update.

Let $\omega = \omega_0$ be a fixed initial cost. For a fixed step size $\alpha > 0$, we iteratively update the cost using Identity, i.e. we set $\omega_{k+1} = \omega_k - \alpha\Delta^{\mathrm{I}}\omega_k$ for $k \in \mathbb{N}$. We aim to show that performing these updates leads to a solution with a lower loss. As gradient-based methods cannot distinguish between a nonlinear loss ℓ and its linearization $f(y) = \ell(y(\omega)) + \langle y - y(\omega), \mathrm{d}\ell/\mathrm{d}y\rangle$ at the point $y(\omega)$, we can safely work with f in our considerations. We desire to find solutions with lower linearized loss than our current solution $y(\omega)$, i.e. points in the set

$$Y^*\big(y(\omega)\big) = \big\{y \in Y : f(y) < f\big(y(\omega)\big)\big\}. \tag{6}$$

Our result guarantees that a solution with a lower linearized loss is always found if one exists. The proof is in Suppl. 8.

Theorem 1. *For sufficiently small $\alpha > 0$, either $Y^*\big(y(\omega)\big)$ is empty and $y(\omega_k) = y(\omega)$ for every $k \in \mathbb{N}$, or there is $n \in \mathbb{N}$ such that $y(\omega_n) \in Y^*\big(y(\omega)\big)$ and $y(\omega_k) = y(\omega)$ for all $k < n$.*

3.4 Exploiting Solver Invariants

In practice, Identity can lead to problematic cost updates when the optimization problem is invariant to a certain transformation of the cost vector ω. For instance, adding the same constant to every component of ω will not affect its rank or top-k indices. Formally, this means that there exists a mapping $P\colon \mathbb{R}^n \to \mathbb{R}^n$ of the cost vector ω that does not change the optimal solver solution, i.e.

$$\arg\min_{y\in Y}\langle\omega, y\rangle = \arg\min_{y\in Y}\langle P(\omega), y\rangle \quad \text{for every } \omega \in W. \tag{7}$$

Linear Transforms. Let us demonstrate why invariants can be problematic in a simplified case assuming that P is linear. Consider an incoming gradient $\mathrm{d}\ell/\mathrm{d}y$, for which the Identity method suggests the cost update $\Delta^{\mathrm{I}}\omega = -\mathrm{d}\ell/\mathrm{d}y$. We can uniquely decompose $\Delta^{\mathrm{I}}\omega$ into $\Delta^{\mathrm{I}}\omega = \Delta^{\mathrm{I}}\omega_1 + \Delta^{\mathrm{I}}\omega_0$ where $\Delta^{\mathrm{I}}\omega_1 = P(\Delta^{\mathrm{I}}\omega) \in \operatorname{Im} P$ and $\Delta^{\mathrm{I}}\omega_0 = (I - P)\Delta^{\mathrm{I}}\omega \in \ker P$. Now observe that only the parallel update $\Delta^{\mathrm{I}}\omega_1$ affects the updated optimization problem, as

$$\begin{aligned}\arg\min_{y\in Y}\langle \omega - \alpha\Delta^{\mathrm{I}}\omega, y\rangle &= \arg\min_{y\in Y}\langle P(\omega - \alpha\Delta^{\mathrm{I}}\omega), y\rangle \\ &= \arg\min_{y\in Y}\langle P(\omega - \alpha\Delta^{\mathrm{I}}\omega_1), y\rangle = \arg\min_{y\in Y}\langle \omega - \alpha\Delta^{\mathrm{I}}\omega_1, y\rangle\end{aligned} \tag{8}$$

for every $\omega \in W$ and for any step size $\alpha > 0$. In the second equality we used

$$P(\omega - \alpha\Delta^{\mathrm{I}}\omega) = P(\omega - \alpha\Delta^{\mathrm{I}}\omega_1) - \alpha P(\Delta^{\mathrm{I}}\omega_0) = P(\omega - \alpha\Delta^{\mathrm{I}}\omega_1), \tag{9}$$

exploiting linearity and idempotency of P.

In the case when a user has no control about the incoming gradient $\Delta^{\mathrm{I}}\omega = -\mathrm{d}\ell/\mathrm{d}y$, the update $\Delta^{\mathrm{I}}\omega_0$ in $\ker P$ can be much larger in magnitude than $\Delta^{\mathrm{I}}\omega_1$ in $\operatorname{Im} P$. In theory, this is not very problematic, as updates in $\ker P$ do not affect the optimization problem. However, in practice these irrelevant updates can lead to explosion of the cost vector as well as problems with adaptive optimizers, which will take into account the irrelevant updates to adjust the learning rate and hence slowing down the convergence.

Therefore, it is desirable to discard the irrelevant part of the update. Consequently, for a given incoming gradient $\mathrm{d}\ell/\mathrm{d}y$, we remove the irrelevant part ($\ker P$) and return only its projected part

$$\Delta^{\mathrm{I}}\omega_1 = -P\tfrac{\mathrm{d}\ell}{\mathrm{d}y}. \tag{10}$$

Nonlinear Transforms. For a nonlinear P, we replace $P(\omega - \alpha\Delta^{\mathrm{I}}\omega)$ in the above-mentioned considerations by its affine approximation $P(\omega) - \alpha P'(\omega)\Delta^{\mathrm{I}}\omega$. Now, the term $P'(\omega)$ plays the role of the linear projection. Equalities in (8) then hold as well for all α sufficiently small instead of globally. Consistently with the linear case, we return a projected gradient

$$-P'(\omega)\tfrac{\mathrm{d}\ell}{\mathrm{d}y} \tag{11}$$

for a given ω and an incoming gradient $\mathrm{d}\ell/\mathrm{d}y$. Note that for linear P, we have $P'(\omega) = P$ and hence gradient (11) generalizes update (10).

Another view on invariant transforms is the following. Consider our combinatorial layer as a composition of two sublayers. First, given ω, we simply perform the map $P(\omega)$ and then pass it to the argmin solver. Clearly, on the forward pass the transform P does not affect the solution $y(\omega)$. However, the derivative of the combinatorial layer is the composition of the derivatives of its sublayers, i.e. $P'(\omega)$ for the transform and negative identity for the argmin. Consequently, we get the very same gradient (11) on the backward pass. In conclusion, enforcing guarantees on the forward pass by a mapping P is in this sense dual to projecting gradients onto $\operatorname{Im} P'$.

Examples. In our experiments, we encounter two types of invariant mappings. The first one is the standard projection onto a hyperplane. It is always applicable when all the solutions in Y are contained in a hyperplane, i.e. there exists a unit vector $a \in \mathbb{R}^n$ and a scalar $b \in \mathbb{R}$ such that $\langle a, y\rangle = b$ for all $y \in Y$. Consider the projection of ω onto the subspace orthogonal to a given by $P_{\text{plane}}(\omega|a) = \omega - \langle a, \omega\rangle a$. This results in

$$\arg\min_{y\in Y}\langle P_{\text{plane}}(\omega|a), y\rangle = \arg\min_{y\in Y}\langle \omega, y\rangle - \langle a, \omega\rangle b = \arg\min_{y\in Y}\langle \omega, y\rangle \quad \text{for every } \omega \in W, \tag{12}$$

thereby fulfilling assumption (7). This projection is relevant for the ranking and top-k experiment, in which the solutions live on a hyperplane with the normal vector $a = \mathbf{1}/\sqrt{n}$. Therefore, the projection

$$P_{\text{plane}}(\omega|\mathbf{1}) = \omega - \frac{\langle \mathbf{1}, \omega\rangle}{n}\mathbf{1} \tag{13}$$

is applied on the forward pass, which simply amounts to subtracting the mean from the cost vector.

The other invariant mapping arises from the stability of the argmin solution to the magnitude of the cost vector. Due to this invariance, the projection onto the unit sphere

$$P_{\text{sphere}}(\omega) = \omega / \|\omega\| \tag{14}$$

also fulfills assumption (7). As the invariance to the cost magnitude is independent of the solutions Y, normalization P_{sphere} is always applicable and we therefore use it in every experiment. Observe that

$$P'_{\text{sphere}}(\omega) = \left(\frac{I}{\|\omega\|} - \frac{\omega \otimes \omega}{\|\omega\|^3} \right) \tag{15}$$

and the first order approximation of P_{sphere} corresponds to the projection onto the tangent hyperplane given by $a = \omega$ and $b = 1$.

3.5 Preventing Cost Collapse

Any argmin solver (1) is a piecewise constant mapping inducing a partitioning into convex cones $W_y = \{\omega \in W : y(\omega) = y\}$, $y \in Y$, on which the solution does not change. The aim of the backbone network is to suggest a cost $\omega \in W_y$ that leads to a correct solution y. However, if the predicted cost $\omega \in W_y$ lies close to the boundary of W_y, it is potentially sensitive to small perturbations. The more partition sets $\{W_{y_1}, \ldots, W_{y_k}\}$ meet at a given boundary point ω, the more brittle such a prediction ω is, since all the solutions $\{y_1, \ldots, y_k\}$ are attainable in any neighbourhood of ω. For example, the zero cost $\omega = 0$ is one of the most prominent points, as it belongs to the boundary of *all* partition sets. However, the origin is not the only problematic point, for example in the ranking problem, every point $\omega = \lambda\mathbf{1}$, $\lambda > 0$, is *equally bad* as the origin.

The goal is to achieve predictions that are *far* from the boundaries of these partition sets. For $Y = \{0,1\}^n$, Rolínek et al. (2020a) propose modifying the cost to $\omega' = \omega + \frac{\alpha}{2}y^* - \frac{\alpha}{2}(1 - y^*)$ before the solver to induce an *informed margin* α. However, this requires access to the ground-truth solution y^*.

In general, when the ground-truth is not available, we add a symmetric noise $\xi \sim p(\xi)$ to the predicted cost before feeding it to the solver. Since all the partition sets' boundaries have zero measure (as they are of a lower dimension), this almost surely induces a margin from the boundary of the size $\mathbb{E}[|\xi|]$. Indeed, if the cost is closer to the boundary, the expected outcome will be influenced by the injected noise and incorrect solutions will increase the expected loss giving an incentive to push the cost further away from the boundary.

In principle, careful design of the projection map P from Sec. 3.4 can also prevent instabilities. This would require that $\operatorname{Im} P$ avoids the boundaries of the partition sets, which is difficul to fully achieve in practice. However, even *reducing* the size of the problematic set by a projection is beneficial. For example, the normalization P_{sphere} avoids brittleness around the origin, and P_{plane} avoids instabilities around every $\omega = \lambda\mathbf{1}$ for ranking. For the other—less significant, but still problematic—boundaries, the noise injection still works in general without any knowledge about the structure of the solution set.

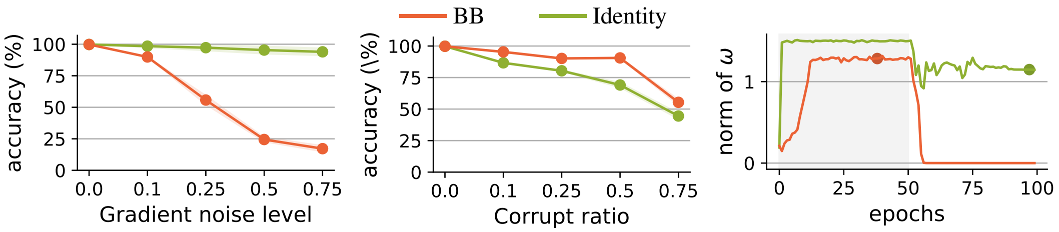
When combining noise and projections, we apply the noise *before* projecting by P since the cost collapse occurs already before applying P. The noise is going to be projected as well and any unnecessary components of the noise are removed. As solvers are agnostic to cost magnitude, the network may suppress the effect of the fixed-sized noise by increasing the suggested cost magnitude. Therefore, the network learns costs that are properly scaled with respect to the fixed noise. In contrast, applying the noise *after* normalization P_{sphere} impede the ability of the model to scale the costs which therefore requires a careful tuning of the noise magnitude for the model to find a robust solution.

In experiments, the noise is sampled uniformly from $\{-\frac{\alpha}{2}, \frac{\alpha}{2}\}^n$, where $\alpha > 0$ is a hyperparameter.

4 Experiments

We demonstrate Identity yields performance that is on par with or outperforms BB, the main competing method. In addition, we show the importance of applying the correct projections according to Sec. 3.4.

4.1 Globe Traveling Salesperson Problem



(a) Gradient noise (σ) (b) Label corruption ρ (c) Cost norm ($\|\omega\|$)

Figure 4: Susceptibility to perturbations and cost collapse in TSP(20). (a) Adding noise to the gradient $\mathrm{d}\ell/\mathrm{d}y$ with std sigma effects Identity much less than BB. (b) Corrupting labels with probability ρ/k in y^*. (c) Average cost norm $\|w\|$ over training epochs. The markers indicate the early stopping time (best validation performance), also used in the other plots.

We consider the Traveling Salesperson Problem (TSP) experiment from Vlastelica et al. (2020). Given a set of k countries flags, the goal for TSP(k) is to predict the optimal tour on globe through the corresponding capitals. The training dataset for TSP(k) consists of 10 000 examples, where each has a k element subset sampled from 100 country flags as input and the output is the adjacency matrix of the optimal tour. We consider $k = 5, 10, 20$. The architecture details can be found in Suppl. 6.1.

Table 1: Accuracy for TSP(k).

k	BB ↑	Identity ↑
5	99.74	99.74
10	99.75	99.77
20	99.86	99.83

Comparison to BB. Results in Tab. 1 confirm that both Identity and BB achieve nearly identical test accuracy. In contrast to the Identity, BB has an additional hyperparameter λ that needs to be tuned. In Fig. 3 we present the performance depending on λ and find that BB is robust over 2 orders of magnitude.

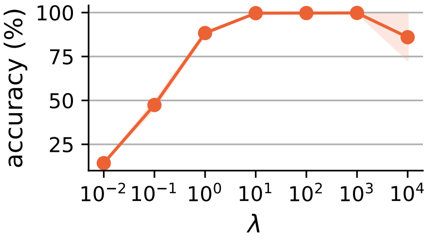


Figure 3: BB test accuracy over parameter λ for TSP(20).

Corrupting Gradients. We studied the robustness of these methods to potential perturbations during training. We consider two scenarios: noisy gradients and corrupted labels.
Noisy Gradients. We inflict the gradient with noise to simulate changes in layers after the solver. So we add $\xi \sim \mathcal{N}(0, \sigma^2)$ to $\mathrm{d}\ell/\mathrm{d}y$.
Corrupt Labels. In real-world settings, incorrect labels are inevitably occurring. We study random but fixed corruptions of labels in the training data, by flipping entries in y^* with probability ρ/k.

In Fig. 4 we empirically show that Identity performs well and even outperforms BB under perturbations. Figure 4c shows the average norm of the cost vector w in the course of training under gradient noise with $\sigma = 0.25$.

After epoch 50 cost collapse occurs quickly for BB, but not for Identity.

4.2 Deep Graph Matching

Given a source and a target image showing an object of the same class (eg. airplane), annotated with a set of keypoints (e.g. right wing), the goal is to match the sets of key points from visual information without any access to keypoint annotation. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be the graphs representing the keypoints in the images. We denote by $\mathbf{v} \in \{0,1\}^{|V_1||V_2|}$, $\mathbf{e} \in \{0,1\}^{|E_1||E_2|}$ the indicator vector of matched vertices and edges respectively. Given two cost vectors $c^v \in \mathbb{R}^{|V_1||V_2|}$ and $c^e \in \mathbb{R}^{|E_1||E_2|}$, we formulate the graph matching problem as

$$\mathrm{GM}(c^v, c^e) = \arg\min_{\mathbf{v},\mathbf{e}} \langle c^v, \mathbf{v}\rangle + \langle c^e, \mathbf{e}\rangle. \tag{16}$$

We follow the same experimental design from (Rolínek et al., 2020b) - including the architecture of the neural network and the hyper-parameter configuration to train the network. We run experiments on SPair-71k Min et al. (2019) and Pascal VOC (with Berkeley annotations) Everingham et al. (2010). Table 2 presents the average matching accuracy across all the classes. For this experiment Identity uses P_{sphere} projection. Again Identity is on par with BB. Interestingly, the margin-inducing noise has small, but opposite effects in the two datasets.

Table 2: Matching accuracy for Deep Graph Matching. Statistics is over 5 restarts. Identity uses P_{sphere} projection.

		Matching accuracy ↑	
Dataset	α	BB	Identity
SPAIR	0	78.89 ± 0.14	78.96 ± 0.29
	0.05	78.72 ± 0.26	79.03 ± 0.30
	0.1	79.47 ± 0.27	79.38 ± 0.38
VOC Basic	0	80.07 ± 0.51	79.90 ± 0.19
	0.05	77.21 ± 0.83	78.59 ± 0.47
	0.1	78.58 ± 0.48	78.96 ± 0.26

4.3 Backpropagating Through Discrete Samplers

The sampling process of distributions of discrete random variables can often be reparameterized approximately as the solution to a noisy $\arg\max$ optimization problem (Paulus et al., 2020). We consider models following our general hybrid architecture Fig. 1 where the combinatorial solver is part of the sampling procedure for the discrete probability distribution $y \sim p(y;\omega)$, which can be written as

$$y = \arg\max_{y \in Y} \langle \omega + \epsilon, y \rangle \tag{17}$$

with appropriate noise distribution ϵ. We call this the *sampling solver*. Typically a Gumbel distribution is used for ϵ, but Niepert et al. (2021) show that for modelling top-k distributions a sum of gamma distribution is more suitable.

Discrete Variational Auto-Encoder. In a discrete variational autoencoder (DVAE), the network layers before the *sampling solver* represent the encoder and the layers after the *sampling solver* the decoder. We consider the task of training such a DVAE on the Mnist dataset. The discrete distribution is the distribution of k-hot binary vectors (or top-k assignments of length 20). We use the same setup as Jang et al. (2017) and Niepert et al. (2021), including the hyper-parameters. Details can be found in Suppl. 6.4. The loss is the Negative Evidence Lower Bound (N-ELBO), which is computed as the sum of the reconstruction losses (binary cross-entropy loss on output pixels) and the KL divergence between the marginals of the discrete latent variables and the uniform distribution. In Fig. 5 we compare our method to I-MLE, which uses the BB update to compute informative gradients for the distribution parameters ω, and thereby can be seen as an application of BB to discrete distributions. We observe that Identity achieves a significantly lower N-ELBO. For this experiment Identity uses $P_{\text{sphere}} \circ P_{\text{plane}}$ projection.

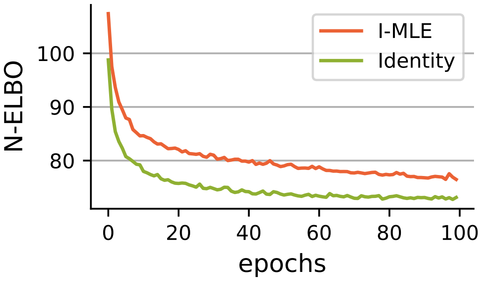


Figure 5: Negative ELBO on the MNIST test-set ($k = 10$), comparing I-MLE (Niepert et al., 2021) with Identity.

Table 3: Results for Aroma. Numbers of the baselines[†] are taken from Niepert et al. (2021). Identity uses $P_{\text{sphere}} \circ P_{\text{plane}}$ projection.

	Test MSE ×100 ↓		
Method	$k = 5$	$k = 10$	$k = 15$
Identity	2.58 ± 0.13	2.62 ± 0.21	2.68 ± 0.23
I-MLE[†]	2.62 ± 0.05	2.71 ± 0.10	2.91 ± 0.18
L2X[†]	5.75 ± 0.30	6.68 ± 1.08	7.71 ± 0.64
Softsub[†]	2.57 ± 0.12	2.67 ± 0.14	2.52 ± 0.07

Learning to Explain. Another application of discrete distributions is learning to explain. We consider the BeerAdvocate dataset that consists of reviews of different aspects of beer: appearance, aroma, taste and palate and a rating score on a scale of 0 and 1. The goal is to identify a subset k of the words in the text that best explain a given aspect rating Chen et al. (2018); Sahoo et al. (2021). We follow the experimental setup of Niepert et al. (2021). Details can be found in Suppl. 6.5. Again, the problem is modeled as a k-hot binary latent representation (mask) with $k \in \{5, 10, 15\}$. For this experiment Identity uses $P_{\text{sphere}} \circ P_{\text{plane}}$ projection. We compare Identity against I-MLE, L2X Chen et al. (2018), and

Softsub Xie and Ermon (2019) in Table 3. Softsub is a relaxation based method designed specifically for subset sampling, in contrast to Identity and I-MLE which are generic. We observe that Identity outperforms I-MLE and L2X, and is on par with Softsub for all $k \in \{5, 10, 15\}$.

4.4 Optimizing Rank Based Metrics – Image Retrieval

In this section, we apply the methods to differentiation of ranking that appears in many evaluation metrics in computer vision, for instance. We consider the image retrieval benchmark *CUB-200-2011* Welinder et al. (2010). For the experiment, we follow the setup of Rolínek et al. (2020a) which uses the most standard processing steps. Further details are provided in Suppl. 6.6. To apply our method, ranking needs to be cast as a combinatorial $\arg\min$ optimization problem with linear cost by simple application of the permutation inequality, as shown in Rolínek et al. (2020a). It holds that $\mathbf{rk}(\omega) = \arg\min_{y \in \Pi_n} \langle \omega, y \rangle$, where Π_n is the set of all rank permutations and $\omega \in \mathbb{R}^n$ the vector of individual scores of the n elements.

Sparse Gradient & Difference between Identity and BB. When training the network using the recall loss we find that Identity performs worse than BB as shown in the first row of Tab. 4. The reason seems to lie in the sparseness of the gradient information of the ranking-based loss: it only yields a gradient for the relevant images (positive labels). In BB (4), a new ranking y_λ is produced for a shifted input. The difference between the two rankings is then the gradient, which contains information for positive and negative examples. This process is illustrated in Suppl. 6.6.

Table 4: Recall r@1 for *CUB-200-2011*.

		Recall r@1 ↑	
P	α	BB	Identity
–	0	64.70 ± 0.27	47.49 ± 1.51
–	0.01	65.17 ± 0.14	47.76 ± 0.48
$P_{\text{sphere}} \circ P_{\text{plane}}$	0	64.72 ± 0.01	61.36 ± 0.35
$P_{\text{sphere}} \circ P_{\text{plane}}$	0.001	64.72 ± 0.46	61.21 ± 0.01

In Tab. 4, we compare the setting without any margin-inducing method (labeled *none*) with the introduced noisy solver inputs for different noise sizes ξ and with the projection method (13), (14). This showcases that the projection step is crucial in some cases. Still Identity does not perform as well as BB in this task.

5 Conclusion

In this work we present a simple approach for computing gradients through combinatorial solvers with a linear cost function, by treating the solver as a *negative identity* mapping during the backward pass in conjunction with exploiting invariances of the solver via projections. This approach, which we call *Identity*, is hyperparameter-free and does not require any additional call to the solver on the backward pass, thus enjoying a considerable speedup during training when compared to more complex approaches. We analyze the robustness to perturbations during training that can come from subsequent layers or incorrect labels. We find that Identity is much more robust to noisy gradients than the prominent BB baseline method and behaves similarly robust under corrupted labels. The introduced projection scheme, that is typically just a normalization step, helps to prevent cost collapse and renders additional robustification steps unnecessary. We demonstrate in numerous experiments from various application areas, that Identity achieves performance that is on par with BB. Our experiments also cover the challenging setup of placing the solver in the middle of the architecture, and the results give confidence that our simple solution is a viable alternative to more complex methods. Only in the ranking experiments we see a discrepancy which we want to address in future work. Another promising area of future work would be to investigate more complex post-processing steps and objective functions after the solver output, or to investigate in more detail the effect of specific cost projections.

REFERENCES

Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BkevoJSYPB.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis R. Bach. Learning with differentiable perturbed optimizers. *CoRR*, abs/2002.08676, 2020. URL https://arxiv.org/abs/2002.08676.

Michal Rolínek, Vít Musil, Anselm Paulus, Marin Vlastelica P., Claudio Michaelis, and Georg Martius. Optimizing rank-based metrics with blackbox differentiation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 7617–7627. Computer Vision Foundation / IEEE, 2020a. doi: 10.1109/CVPR42600.2020.00764. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Rolinek_Optimizing_Rank-Based_Metrics_With_Blackbox_Differentiation_CVPR_2020_paper.html.

Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVIII*, volume 12373 of *Lecture Notes in Computer Science*, pages 407–424. Springer, 2020b. doi: 10.1007/978-3-030-58604-1_25. URL https://doi.org/10.1007/978-3-030-58604-1_25.

Anselm Paulus, Michal Rolínek, Vít Musil, Brandon Amos, and Georg Martius. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8443–8453. PMLR, 2021. URL http://proceedings.mlr.press/v139/paulus21a.html.

Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 2017. URL http://proceedings.mlr.press/v70/amos17a.html.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 2019. URL http://proceedings.mlr.press/v97/wang19e.html.

Aaron M. Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1504–1511. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/5509.

Marin Vlastelica P., Michal Rolínek, and Georg Martius. Neuro-algorithmic policies enable fast combinatorial generalization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 10575–10585. PMLR, 2021. URL http://proceedings.mlr.press/v139/vlastelica21a.html.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017. URL http://proceedings.mlr.press/v70/finn17a.html.

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of MAML. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rkgMkCEtPB.

Akshay Agrawal, Brandon Amos, Shane T. Barratt, Stephen P. Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9558–9570, 2019a. URL https://proceedings.neurips.cc/paper/2019/hash/9ce3c52fc54362e22053399d3181c638-Abstract.html.

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 10657–10665. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.01091. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Lee_Meta-Learning_With_Differentiable_Convex_Optimization_CVPR_2019_paper.html.

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. Differentiating through a cone program. *J. Appl. Numer. Optim*, 1(2):107–115, 2019b. URL https://doi.org/10.23952/jano.1.2019.2.02.

Justin Domke. Generic methods for optimization-based modeling. In Neil D. Lawrence and Mark A. Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, Spain, April 21-23, 2012*, volume 22 of *JMLR Proceedings*, pages 318–326. JMLR.org, 2012. URL http://proceedings.mlr.press/v22/domke12.html.

Adam N. Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Manag. Sci.*, 68(1):9–26, 2022. doi: 10.1287/mnsc.2020.3922. URL https://doi.org/10.1287/mnsc.2020.3922.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit MLE: backpropagating through discrete exponential family distributions. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 14567–14579, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/7a430339c10c642c4b2251756fd1b484-Abstract.html.

Brandon Amos, Vladlen Koltun, and J. Zico Kolter. The limited multi-label projection layer. *CoRR*, abs/1906.08707, 2019. URL http://arxiv.org/abs/1906.08707.

Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 950–959. PMLR, 2020. URL http://proceedings.mlr.press/v119/blondel20a.html.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1658–1665. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33011658. URL https://doi.org/10.1609/aaai.v33i01.33011658.

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 353–362. PMLR, 2018. URL http://proceedings.mlr.press/v80/balcan18a.html.

Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L. Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 724–731. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12514.

Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS J. Comput.*, 29(1):185–195, 2017. URL https://doi.org/10.1287/ijoc.2016.0723.

Mislav Balunovic, Pavol Bielik, and Martin T. Vechev. Learning to solve SMT formulas. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10338–10349, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/68331ff0427b551b68e911eebe35233b-Abstract.html.

Jayanta Mandi and Tias Guns. Interior point solving for lp-based prediction+optimisation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7272–7282. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/51311013e51adebc3c34d2cc591fefee-Paper.pdf.

Yingcong Tan, Daria Terekhov, and Andrew Delong. Learning linear programs from optimal decisions. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 19738–19749, 2020. URL https://proceedings.neurips.cc/paper/2020/file/e44e875c12109e4fa3716c05008048b2-Paper.pdf.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Bk9mxlSFx.

MohammadReza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9861–9871, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/9fb4651c05b2ed70fba5afe0b039a550-Abstract.html.

Wei Zhang and Thomas G Dietterich. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Research*, 1(1):1–38, 2000.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Petar Velickovic, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SkgKO0EtvS.

Andreea Deac, Petar Velickovic, Ognjen Milinkovic, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolic. XLVIN: executed latent value iteration nets. *CoRR*, abs/2010.13146, 2020. URL https://arxiv.org/abs/2010.13146.

Bingjie Li, Guohua Wu, Yongming He, Mingfeng Fan, and Witold Pedrycz. An overview and experimental study of learning-based optimization algorithms for vehicle routing problem. *CoRR*, abs/2107.07076, 2021. URL https://arxiv.org/abs/2107.07076.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6062–6071, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/6788076842014c83cedadbe6b0ba0314-Abstract.html.

Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1l8oANFDH.

Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. Spair-71k: A large-scale benchmark for semantic correspondence. *CoRR*, abs/1908.10543, 2019. URL http://arxiv.org/abs/1908.10543.

Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *Int. J. Comput. Vis.*, 88(2): 303–338, 2010. doi: 10.1007/s11263-009-0275-4. URL https://doi.org/10.1007/s11263-009-0275-4.

Max B. Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J. Maddison. Gradient estimation with stochastic softmax tricks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/3df80af53dce8435cf9ad6c3e7a403fd-Abstract.html.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 882–891. PMLR, 2018. URL http://proceedings.mlr.press/v80/chen18j.html.

Subham Sekhar Sahoo, Subhashini Venugopalan, Li Li, Rishabh Singh, and Patrick Riley. Scaling symbolic methods using gradients for neural model explanation. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=V5j-jdoDDP.

Sang Michael Xie and Stefano Ermon. Reparameterizable subset sampling via continuous relaxations. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 3919–3925. ijcai.org, 2019. doi: 10.24963/ijcai.2019/544. URL https://doi.org/10.24963/ijcai.2019/544.

Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Fatih Çakir, Kun He, Xide Xia, Brian Kulis, and Stan Sclaroff. Deep metric learning to rank. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 1861–1870. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00196. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Cakir_Deep_Metric_Learning_to_Rank_CVPR_2019_paper.html.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

6 Appendix

6.1 Globe TSP

Architecture. A set of k flags is presented to a convolutional neural network that output k 3-dimensional coordinates. These points are then projected onto a 3D unit sphere and then used to construct the $k \times k$ distance matrix which is fed to a TSP solver. Then the solver outputs the adjacency matrix indicating the edges present in the TSP tour. The loss function is an L1 loss between the predicted tour and the true tour. The neural network is expected to learn the correct coordinates of the countries' capitals on Earth, up to rotations of the sphere. We use the `gurobi` solver for the MIP formulation of TSP.

For the Globe TSP experiment we use a convolutional neural network with 2 conv. layers ((channels, kernel_size, stride) = [[20, 4, 2], [50, 4, 2]]) and 1 fully connected layer of size 500 that predicts vector of dimension $3k$ containing the k 3-dimensional representations of the respective countries' capital cities. These representations are projected onto the unit sphere and the matrix of pairwise distances is fed to the TSP solver. The network was trained using Adam optimizer with a learning rate 10^{-4} for 100 epochs and a batch size of 50. For BB, the hyper-parameter λ was set to 20.

As described in Sec. 3.5 adding noise to the cost vector can avoid cost collapse and act as a margin-inducing procedure. We apply noise, $\xi \in \{0.1, 0.2, 0.5\}$, for the first 50 epochs (of 100 in total) to prevent cost collapse. For this dataset, it was observed that finetuning the weights after applying noise helped improve the accuracy. Margin is only important initially as it allows you to not get stuck in a local optima around zero-cost. Once avoided, margin doesn't play a useful role anymore because there are no large distribution shifts between the train and test set. Hence, noise wasn't applied for the entirety of the training phase. We have verified the benefits of adding noise experimentally in Table 5.

	$\xi = 0$	$\xi = 0.1$	$\xi = 0.2$	$\xi = 0.5$
TSP($k = 5$)	91.09 ± 0.07	99.67 ± 0.10	99.67 ± 0.10	99.68 ± 0.09
TSP($k = 10$)	85.31 ± 0.15	99.72 ± 0.04	99.76 ± 0.07	99.76 ± 0.04
TSP($k = 20$)	88.45 ± 0.88	99.78 ± 0.04	94.50 ± 10.53	99.80 ± 0.06

Table 5: Adding noise ξ during the initial 50 epochs of training prevents cost collapse and therefore improves the test accuracy.

A difference between BB and Identity can be observed when applying gradient noise, simulating larger architectures where the combinatorial layer is followed by further learnable layers. In Figure 6, we present the training loss curves in this case. As we see, BB starts to diverge right after the margin-inducing noise is not added anymore i.e. after epoch 50. However, Identity converges as the training progresses.

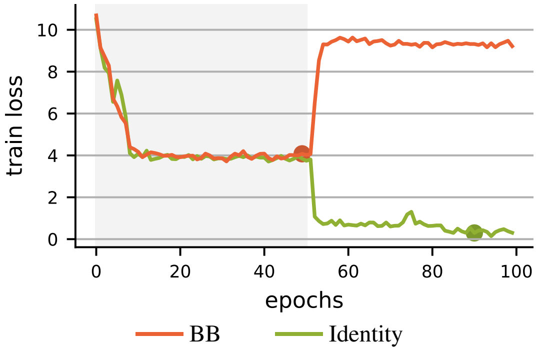


Figure 6: Training curves for TSP ($k = 5$) for gradient noise level 0.50. To check for the robustness of the methods, we apply gradient noise during the entire training time (100 epochs). The shaded grey region highlights the epochs for which margin-inducing noise (ξ) was applied to the inputs to prevent cost collapse. The dots on the training curve represent early stopping epochs.

6.2 Warcraft Shortest Path

Taken from Vlastelica et al. (2020), the aim of the Warcraft Shortest Path experiment is to predict the shortest path between the top-left and bottom-right vertex in a $k \times k$ Warcraft terrain map. The path is represented by an indicator matrix of vertices that appear along the path.

The non-negative vertex costs of the $k \times k$ grid are computed by a modified Resnet18 (He et al., 2016) architecture using softplus on the output. The network receives supervision in form of L1 loss between predicted and target paths. We consider only the hardest case from the dataset with map sizes 32×32.

Due to the non-uniqueness of solutions we use the ratio between true and predicted path costs as an evaluation metric: $\langle \omega^*, y(\omega) \rangle / \langle \omega^*, y^* \rangle$, with ω^* being the ground truth cost vector. Note that lower scores are better and 1.0 is the perfect score. Table 6 shows that Identity performs comparatively to BB.

Table 6: Cost ratio (suggested vs. true path costs) for Warcraft Shortest Path 32×32. BB and Identity work similarly well. Cost transform P or noise does not affect the performance significantly.

		Cost ratio ×100 ↓	
P	α	BB	Identity
–	0	100.9 ± 0.1	101.0 ± 0.1
P_{sphere}	0	100.9 ± 0.1	101.2 ± 0.1
–	0.2	101.1 ± 0.1	101.0 ± 0.1

We folow the same experimental design as Vlastelica et al. (2020) and do the same modification to the ResNet18 architecture, except that we use *softplus* to make the network output positive. The model is trained with Adam for 50 epochs with learning rate 5×10^{-3}. The learning rate is divided by 10 after 30 epochs. For the BB method we use $\lambda = 20$. The noise, when used, is applied for the whole duration of training.

6.3 Deep Graph Matching

We follow the same experimental design from Rolínek et al. (2020b), including the architecture of the nerual network and the hyperparameter configuration to train the network. We run experiments on SPair-71k Min et al. (2019) and Pascal VOC (with Berkeley annotations) Everingham et al. (2010). The optimizer used is Adam with an initial learning rate of 2×10^{-3} which is halved four times in regular intervals. Learning rate for finetuning the VGG weights is multiplied with 10^{-2}. Image pairs in batches of 8 are processed and we set the BB hyperparameter $\lambda = 80$.

6.4 DVAE on MNIST

Consider the models described by the equations $\theta = f_e(x)$, $y \sim p(y;\theta)$, $z = f_d(y)$ where $x \in \mathcal{X}$ is the input, $o \in \mathcal{O}$ is the output, and $f_e \colon \mathcal{X} \to \theta$ is the encoder neural network that maps the input x to the logits θ and $f_d \colon \mathcal{Y} \to \mathcal{Z}$ is the decoder neural network, and where $\mathcal{Y}$ is the set of all k-hot vectors. Following Niepert et al. (2021), we set ϵ in sampling (17) as the Sum-of-Gamma distribution given by

$$\mathrm{SoG}(k,\tau,s) = \frac{\tau}{k}\left(\sum_{i=1}^{s} \mathrm{Gamma}\left(\frac{1}{k}, \frac{k}{i}\right) - \log s\right), \tag{18}$$

where s is a positive integer and $\mathrm{Gamma}(\alpha,\beta)$ is the Gamma distribution with (α,β) as the shape and scale parameters.

We follow the exact same training procedure from Niepert et al. (2021). The encoder and the decoder were feedforward neural networks with the architectures: 512-256-20 × 20 and 256-512-784 respectively. We used MNIST dataset for the problem which consisted of $50,000$ training examples and $10,000$ validation and test examples each. We train the model for 100 epochs and record the loss on the test data. In this experiment we use $k = 10$ i.e. sample 10-hot binary vectors from the latent space. We sample the noise from SoG with $s = 10$ and $\tau = 10$.

6.5 Learning to Explain

The entire dataset consists of 80k reviews for the aspect Appearance and 70k reviews for all the remaining aspects. Following the experimental setup of Niepert et al. (2021), the dataset was divided into 10 different evenly sized validation / test splits of the 10k held out set and compute mean and standard deviation over 10 models, each trained on one split. For this experiment the neural network from Chen et al. (2018) was used which had 4 convolutional and 1 dense layer. The neural network outputs the parameters θ of the pdf $\mathrm{p}(y; \theta)$ over k-hot binary latent masks with $K = 5, 10, 15$. The same hyperparameter configuration was the same as that of Niepert et al. (2021).

6.6 Rank-based Metrics – Image Retrieval Experiments

Ranking as a Solver. Ranking can be cast as a combinatorial $\arg\max$ optimization problem with linear cost by simple application of the permutation inequality as shown in Rolínek et al. (2020a). It holds that

$$\mathbf{rk}(\omega) = \underset{y \in \Pi_n}{\arg\max} -\langle \omega, y \rangle, \tag{19}$$

where Π_n is the set of all rank permutations and $\omega \in \mathbb{R}^n$ the vector of individual scores.

A popular metric is recall at K, denoted by $r@K$, which can be formulated using ranking as

$$r@K(\omega, y^*) = \begin{cases} 1 & \text{if there is } i \in \mathrm{rel}(y^*) \text{ with } \mathbf{rk}(\omega)_i \leq K \\ 0 & \text{otherwise,} \end{cases} \tag{20}$$

where $K \in \mathbb{N}$, $\omega \in \mathbb{R}^n$ are the scores, $y^* \in \{0, 1\}^n$ are the ground-truth labels and $\mathrm{rel}(y^*) = \{i : y_i^* = 1\}$ is the set of relevant label indices (positive labels). Consequently, $r@K$ is going to be 1 if a positively labeled element is among the predicted top K ranked elements. Due to computational restrictions, at training time, instead of computing $r@K$ over the whole dataset a sample-based version is used.

RaMBO Loss. Since $r@K$ depends only on the top-ranked element from y^*, the supervision for $r@K$ is very sparse. To circumvent this, Rolínek et al. (2020a) propose a loss

$$\ell_{\text{recall}}(\omega, y^*) = \frac{1}{|\,\mathrm{rel}(y^*)|} \sum_{i \in \mathrm{rel}(y^*)} \log\bigl(1 + \mathbf{rk}(\omega)_i - \mathbf{rk}(\omega^+)_i\bigr), \tag{21}$$

where $\mathbf{rk}(\omega^+)_i$ denotes the rank of the i-th element only within the relevant ones.

Experimental Configuration. We follow the exact training procedure from Rolínek et al. (2020a). The neural network was trained for 80 epochs with early stopping with a weight decay of 4×10^{-4}. Learning rate was set to 5×10^{-6} and dropped by 70% after 35 epochs with a higher learning rate for the embedding layer as mentioned in Çakir et al. (2019). BB parameter λ was set to 0.5 for Online Products, 0.4 for In-shop clothes and to 0.05 for CUB200.

Sparse Gradient & Difference between Identity and BB. When training the network using RaMBO loss (21) we find that Identity performs worse than BB as shown in Table 4. The reason seems to lie in the sparseness of the gradient information of RaMBO loss (21): it only yields a gradient for the relevant images (positive labels). In BB (4), a new ranking y_λ is produced for a shifted input. The difference between the two rankings is then the gradient, which contains information for positive and negative examples. This process is illustrated in Figure 7.

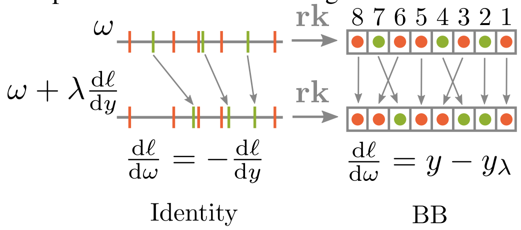
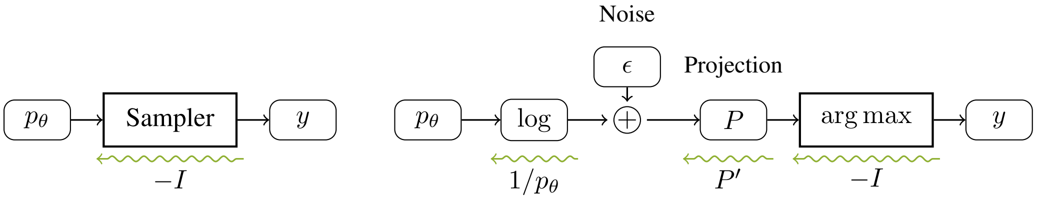

ω
$\omega + \lambda \frac{\mathrm{d}\ell}{\mathrm{d}y}$
rk
8 7 6 5 4 3 2 1
rk
$\frac{\mathrm{d}\ell}{\mathrm{d}\omega} = -\frac{\mathrm{d}\ell}{\mathrm{d}y}$
$\frac{\mathrm{d}\ell}{\mathrm{d}\omega} = y - y_\lambda$
Identity
BB

Figure 7: Visualization of the different ranking gradients. Left: the scores ω, with $\mathrm{rel}(y^*)$ in green and their shifted counterparts below. The grey arrows correspond to the Identity-gradient. Right: BB performs another ranking y_λ with the shifted ω which yields a denser gradient.

7 Straigth Through vs. Identity

Straight-through estimator (STE) was introduced by Bengio et al. (2013) to differentiate through samples drawn from a stochastic Bernoulli process. For $y \sim p(y;\theta)$, it ignores the sampling process and sets the derivative $\mathrm{d}y/\mathrm{d}\theta = I$. Superficially, it looks like Identity is exactly the same as the Straight-through estimator. However, in order to use Identity, the sampling process needs to be formulated as an argmin/argmax problem. Identity sets the gradient for argmax operator to I, but before the argmax, we project $\log p(y;\theta) + \epsilon$ where $\log p(y;\theta)$ represents the logits of the density $p(y;\theta)$. Projection P consists of P_{plane} and P_{sphere} given by (13) and (14), respectively. Figure 8 illustrates the computation graphs for STE and Identity that results in different backward passes.



(a) Straigth-through estimator

(b) Identity

Figure 8: Computation graph for Straight-through estimator and Identity. The green arrow denotes the derivative in the backward pass. Observe that the resulting derivatives of y w.r.t. p_θ are different.

8 Method

8.1 Proof of Theorem 1

For an initial cost ω and a step size $\alpha > 0$, we set

$$\omega_0 = \omega \quad \text{and} \quad \omega_{k+1} = \omega_k - \alpha\Delta^{\mathrm{I}}\omega_k \quad \text{for } k \in \mathbb{N}, \tag{22}$$

in which $\Delta^{\mathrm{I}}\omega_k$ denotes the Identity update at the solution point $y(\omega_k)$, i.e.

$$\Delta^{\mathrm{I}}\omega_k = -\frac{\mathrm{d}\ell}{\mathrm{d}y}\big(y(\omega_k)\big). \tag{23}$$

We shall simply write $\mathrm{d}\ell/\mathrm{d}y$ when no confusion is likely to happen. Recall that the set of better solutions is defined as

$$Y^*(y) = \big\{y' \in Y : f(y') < f\big(y\big)\big\}, \tag{24}$$

where f is the linearization of the loss ℓ at the point y defined by

$$f(y') = \ell(y) + \Big\langle y' - y, \frac{\mathrm{d}\ell}{\mathrm{d}y}\Big\rangle. \tag{25}$$

In principle, ties in the solver may occur and hence the mapping $\omega \mapsto y(\omega)$ is not well-defined for all cost ω unless we specify, how the ties are resolved. Typically, this is not an issue in most of the considerations. However, in our exposition, we need to avoid certain rare discrepancies. Therefore, we assume that the solver will always favour the solution from the previous iteration if possible, i.e. from $\langle y(\omega_k), \omega_k\rangle = \big\langle y(\omega_{k-1}), \omega_k\big\rangle$ it follows that $y(\omega_k) = y(\omega_{k-1})$.

Theorem 2. *Assume that $(\omega_k)_{k=0}^{\infty}$ is the sequence as in (22) for some initial cost ω and step size $\alpha > 0$. Then the following holds:*

(i) *Either $Y^*\big(y(\omega)\big)$ is non-empty and there is some $\alpha_{\max} > 0$ such that for every $\alpha < \alpha_{\max}$ there is $n \in \mathbb{N}$ such that $y(\omega_n) \in Y^*\big(y(\omega)\big)$ and $y(\omega_k) = y(\omega)$ for all $k < n$,*

(ii) *or $Y^*\big(y(\omega)\big)$ is empty and for every α it is $y(\omega_k) = y(\omega)$ for all $k \in \mathbb{N}$.*

We prove this statement in multiple parts.

Proposition 1. *Let $\alpha > 0$ and $(\omega_k)_{k=0}^{\infty}$ be as in (22). If $Y^*\big(y(\omega)\big)$ is non-empty, then there exists $n \in \mathbb{N}$ such that $y(\omega_n) \neq y(\omega)$.*

Proof. We proceed by contradiction. Assume that $Y^*\big(y(\omega)\big)$ is non-empty and $y(\omega_k) = y(\omega)$ for all $k \in \mathbb{N}$.

Take any $y^* \in Y^*\big(y(\omega)\big)$. By definition of $Y^*\big(y(\omega)\big)$, we have that

$$\xi = \left\langle y^* - y(\omega), \frac{\mathrm{d}\ell}{\mathrm{d}y} \right\rangle < 0. \tag{26}$$

As $y(\omega_k) = y(\omega)$ for all $k \in \mathbb{N}$, it is

$$\omega_k = \omega - k\alpha\Delta^{\mathrm{I}}\omega = \omega + k\alpha\frac{\mathrm{d}\ell}{\mathrm{d}y} \tag{27}$$

and therefore

$$\langle y^* - y(\omega), \omega_k\rangle = \langle y^* - y(\omega), \omega\rangle + k\alpha\left\langle y^* - y(\omega), \frac{\mathrm{d}\ell}{\mathrm{d}y}\right\rangle = \langle y^* - y(\omega), \omega\rangle + k\alpha\xi. \tag{28}$$

Since $\xi < 0$, the latter term tends to minus infinity. Consequently, there exists some $n \in \mathbb{N}$ for which

$$\langle y^*, \omega_n\rangle < \langle y(\omega), \omega_n\rangle \tag{29}$$

contradicting the fact that $y(\omega)$ is the minimizer for ω_n. □

Let us now make a simple auxiliary observation about argmin solvers. The mapping

$$\omega \mapsto y(\omega) = \arg\min_{y\in Y}\langle \omega, y\rangle \tag{30}$$

is a piecewise constant function and hence induces a partitioning of its domain W into non-overlapping sets on which the solver is constant. Let us denote the pieces by

$$W_y = \{\omega \in W : y(\omega) = y\} \quad \text{for } y \in Y. \tag{31}$$

We claim that W_y is a convex cone. Indeed, if $\omega \in W_y$, clearly $\lambda\omega \in W_y$ for any $\lambda > 0$. Next, if $\omega_1, \omega_2 \in W_y$ and $\lambda \in (0, 1)$, then $y(\lambda\omega_1) = y$ and $y\big((1-\lambda)\omega_2\big) = y$ and hence y is also a minimizer for $\lambda\omega_1 + (1-\lambda)\omega_2$.

Proposition 2. *Assume that $(\omega_k)_{k=0}^\infty$ is the sequence as in (22) for some initial cost ω and step size α. Then either for every α it is $y(\omega_k) = y(\omega)$ for all $k \in \mathbb{N}$, or there is some $\alpha_{\max} > 0$ such that for every $\alpha < \alpha_{\max}$ there is $n \in \mathbb{N}$ such that $y(\omega_k) = y(\omega)$ for all $k < n$ and $y(\omega_n) \neq y(\omega)$.*

Proof. Let us define $w\colon [0, \infty) \to W$ and $\gamma\colon [0, \infty) \to Y$ as

$$w(\alpha) = \omega - \alpha\Delta^{\mathrm{I}}\omega \quad \text{and} \quad \gamma(\alpha) = y\big(w(\alpha)\big) \quad \text{for } \omega \in W, \tag{32}$$

respectively. As γ is a composition of an affine function w and a piecewise constant solver, it is itself piecewise constant function. Therefore, γ induces a partitioning of its domain $[0, \infty)$ into disjoint sets on which γ is constant. In fact, these sets are intervals, say $I_1, \dots, I_m$, as intersections of the line segment $\{w(\alpha) : \alpha > 0\}$ and convex cones W_y. Consequently, $m \le |Y|$.

Now, If $m = 1$, then $I_1 = [0, \infty)$ and $y(\omega_k)$ is constant $y(\omega)$ for all $k \in \mathbb{N}$ whatever $\alpha > 0$ is. In the rest of the proof, we assume that $m \ge 2$. Assume that the intervals $I_1, \dots, I_m$ are labeled along increasing α, i.e. if $\alpha_1 \in I_i$ and $\alpha_2 \in I_j$ then $\alpha_1 < \alpha_2$ if and only if $i < j$.

We define the upper bound on the step size to $\alpha_{\max} = |I_2|$. Assume that $\alpha < \alpha_{\max}$ and $(\omega_k)_{k=1}^\infty$ is given. Let $n = \min\{k \in \mathbb{N} : w(\alpha k) \in I_2\}$, i.e. the first index when the sequence $y(\omega_k)$ switches. Clearly $y(\omega_k) = \gamma(\alpha k) = y(\omega)$ for $k = 0, \dots, n-1$ and $y(\omega_n) = \gamma(\alpha n) \neq y(\omega)$.

□

Proposition 3. *Let $\alpha > 0$ and $(\omega_k)_{k=0}^\infty$ be as in (22). Assume that $y(\omega_k) = y(\omega)$ for all $k < n$ and $y(\omega_n) \neq y(\omega)$. Also assume that from $\langle y(w_k), \omega_k\rangle = \big\langle y(\omega_{k-1}), \omega_k\big\rangle$ it follows that $y(w_k) = y(\omega_{k-1})$. Then*

$$f\big(y(\omega_n)\big) < f\big(y(\omega)\big), \tag{33}$$

where f is the linerarized loss at $y(\omega)$.

Proof. As $y(\omega_k) = y(\omega)$ for all $k < n$, it is

$$\omega_k = \omega - n\alpha\Delta^{\mathrm{I}}\omega = \omega + n\alpha\frac{\mathrm{d}\ell}{\mathrm{d}y}. \tag{34}$$

Therefore, as $y(\omega_n)$ is the minimizer for the cost ω_k, we have

$$0 \le \langle y(\omega) - y(\omega_n), \omega_n\rangle = \langle y(\omega) - y(\omega_n), \omega\rangle + n\alpha\Big\langle y(\omega) - y(\omega_n), \frac{\mathrm{d}\ell}{\mathrm{d}y}\Big\rangle. \tag{35}$$

Now, since $y(\omega)$ is the minimizer for ω, it is $\langle y(\omega) - y(\omega_n), \omega\rangle \le 0$ and therefore

$$0 \le \Big\langle y(\omega) - y(\omega_n), \frac{\mathrm{d}\ell}{\mathrm{d}y}\Big\rangle. \tag{36}$$

Consequently,

$$f\big(y(\omega_n)\big) = \ell\big(y(\omega)\big) + \Big\langle y(\omega_n) - y(\omega), \frac{\mathrm{d}\ell}{\mathrm{d}y}\Big\rangle \le \ell\big(y(\omega)\big) = f\big(y(\omega)\big). \tag{37}$$

Equality is attained only if

$$\langle y(\omega_n) - y(\omega_{n-1}), \omega_n\rangle = 0. \tag{38}$$

This together with $y(\omega_n) \neq y(\omega) = y(\omega_{n-1})$ violates the assumption that the solver ties are broken in favour of the previously attained solution, therefore the inequality in (37) is strict. □

Proof of Theorem 2. Assume $Y^*\big(y(\omega)\big)$ is non-empty. It follows from Proposition 1 that there exists $m \in \mathbb{N}$ such that $y(\omega_m) \neq y(\omega)$. It follows from Proposition 2 that there is some $\alpha_{\max} > 0$ such that for every $\alpha < \alpha_{\max}$ there is $n \in \mathbb{N}$ such that $y(\omega_k) = y(\omega)$ for all $k < n$ and $y(\omega_n) \neq y(\omega)$. From Proposition 3 it also follows that $f\big(y(\omega_n)\big) < f\big(y(\omega)\big)$. Combining these results we have $y(\omega_n) \in Y^*\big(y(\omega)\big)$ by definition. This proves the first part of the theorem.

We prove the second part of the theorem by contradiction. Assume that $Y^*\big(y(\omega)\big)$ is empty and there exists some α such that $y(\omega_k) = y(\omega)$ for all $k \in \mathbb{N}$ and $y(\omega_n) \neq y(\omega)$. From Proposition 3 we know that $f\big(y(\omega_n)\big) < f\big(y(\omega)\big)$ and therefore $y(\omega_n) \in Y^*\big(y(\omega)\big)$. This is in contradiction to $Y^*\big(y(\omega)\big)$ being empty. □

8.2 Cost Collapse

Any argmin solver (30) is a piecewise constant mapping inducing a partitioning into convex cones W_y, $y \in Y$, defined in (31) on which the solution does not change. The aim of the backbone network is to suggest a cost that leads to a correct solution. The solution of this task is clearly far from being unique, since any $\omega \in W_y$ leads to a solution y. However, not all the suggestions are equally good. For instance, if the predicted cost $\omega \in W_y$ lies close to the boundary of W_y, it it potentially sensitive to a shift in statistics across domains and to small perturbations. The more partition sets $\{W_{y_1}, \dots, W_{y_k}\}$ meet at a given boundary point ω, the more brittle such a prediction ω is, since all the solutions $\{y_1, \dots, y_k\}$ are attainable in any neighbourhood of ω. For example, the zero cost $\omega = 0$ is one of the most prominent points, as it belongs to the boundary of *all* partition sets. However, the origin is not the only problematic point, for example in the ranking problem, every point $\omega = \lambda\mathbf{1}$, $\lambda > 0$, is *equally bad* as the origin.

Therefore, our goal is to achieve predictions that are *far* from the boundaries of these partition sets. To do so, we add a symmetric noise $\xi \sim p(\xi)$ to the predicted cost before feeding it to the solver. Since all the partition sets' boundaries have zero measure (as they are of a lower dimension), this almost surely induces a margin from the boundary of the size $\mathbb{E}[|\xi|]$. Indeed, if the cost is closer to the boundary, the expected outcome will be influenced by the injected noise and incorrect solutions will increase the expected loss giving an incentive to push the cost further away from the boundary.

In principle, careful design of the projection map P from Sec. 3.4 can also prevent instabilities. This would require that $\operatorname{Im} P$ avoids the boundaries of the partition sets, which is difficult to fully achieve in practice. However, even reducing the size of the problematic set by a projection is beneficial. For example, the normalization P_{sphere} avoids brittleness around the origin, and P_{plane} avoids instabilities around every $\omega = \lambda\mathbf{1}$ for ranking. For the other—less significant, but still problematic—boundaries, the noise injection still works in general without any knowledge about the structure of the solution set.

When combining noise and projections, we apply the noise *before* projecting by P since the cost collapse occurs already before applying P. The noise is going to be projected as well and any unnecessary components of the noise are removed. As solvers are agnostic to cost magnitude, the network may suppress the effect of the fixed-sized noise by increasing the suggested cost magnitude. Therefore, the network learns costs that are properly scaled with respect to the fixed noise. In contrast, applying the noise *after* normalization P_{sphere} impede the ability of the model to scale the costs which therefore requires a careful tuning of the noise magnitude for the model to find a robust solution.