ALTERNATING DIFFERENTIATION FOR OPTIMIZATION LAYERS

Anonymous authors
Paper under double-blind review

ABSTRACT

The idea of embedding optimization problems into deep neural networks as optimization layers to encode constraints and inductive priors has taken hold in recent years. Most existing methods focus on implicitly differentiating Karush–Kuhn–Tucker (KKT) conditions in a way that requires expensive computations on the Jacobian matrix, which can be slow and memory-intensive. In this paper, we developed a new framework, named Alternating Differentiation (Alt-Diff), that differentiates optimization problems (here, specifically in the form of convex optimization problems with polyhedral constraints) in a fast and recursive way. Alt-Diff decouples the differentiation procedure into a primal update and a dual update in an alternating way. Accordingly, Alt-Diff substantially decreases the dimensions of the Jacobian matrix and thus significantly increases the computational speed of implicit differentiation. Further, we present the computational complexity of the forward and backward pass of Alt-Diff and show that Alt-Diff enjoys quadratic computational complexity in the backward pass. Another notable difference between Alt-Diff and state-of-the-arts is that Alt-Diff can be truncated for the optimization layer. We theoretically show that: 1) Alt-Diff can converge to consistent gradients obtained by differentiating KKT conditions; 2) the error between the gradient obtained by the truncated Alt-Diff and by differentiating KKT conditions is upper bounded by the same order of variables' truncation error. Therefore, Alt-Diff can be truncated to further increases computational speed without sacrificing much accuracy. A series of comprehensive experiments demonstrate that Alt-Diff yields results comparable to the state-of-the-arts in far less time.

1 INTRODUCTION

Recent years have seen a variety of applications in machine learning that consider optimization as a tool for inference learning Belanger & McCallum (2016); Belanger et al. (2017); Diamond et al. (2017); Amos et al. (2017); Amos & Kolter (2017); Agrawal et al. (2019a). Embedding optimization problems as optimization layers in deep neural networks can capture useful inductive bias, such as domain specific knowledge and priors. Unlike conventional neural networks, which are defined by an explicit formulation in each layer, optimization layers are defined implicitly by solving optimization problems. They can be treated as implicit functions where inputs are mapped to optimal solutions. However, training optimization layers together with explicit layers is not easy since explicit closed-form solutions typically do not exist for the optimization layers.

Generally, computing the gradients of the optimization layers can be classified into two main categories: differentiating the optimality conditions implicitly and applying unrolling methods. The ideas of differentiating optimality conditions have been used in bilevel optimization Kunisch & Pock (2013); Gould et al. (2016) and sensitivity analysis Bonnans & Shapiro (2013). Recently, OptNet Amos & Kolter (2017) and CvxpyLayer Agrawal et al. (2019a) have extended this method to optimization layers so as to enable end-to-end learning within the deep learning structure. However, these methods inevitably require expensive computation on the Jacobian matrix during the forward and backward pass, which are prone to instability and often intractable especially for large-scale optimization layers. Another direction to obtain the gradients of optimization layers is based on the unrolling methods, where an iterative first-order gradient method is applied. However, they are memory-intensive since all the intermediate results have to be recorded. Besides, unrolling methods

are not quite suitable to the constrained optimization problems as the expensive projection operators are needed.

In this paper, we aim to develop a new method that significantly increases the computational speed of the differentiation procedure for convex optimization problems with polyhedral constraints. Motivated by the scalability of the operator splitting method for optimization problem Glowinski & Le Tallec (1989); Stellato et al. (2020), we developed a new framework, namely Alternating Differentiation (Alt-Diff), that differentiates optimization layers in a fast and recursive way. Alt-Diff first decouples the constrained optimization problem into multiple subproblems based on the well-known alternating direction method of multipliers (ADMM) Boyd et al. (2011). Then, the differentiation operators for obtaining the derivatives of the primal and dual variables w.r.t the parameters are implemented on these subproblems in an alternating manner. Accordingly, Alt-Diff substantially decreases the dimensions of the Jacobian matrix, significantly increasing the computational speed of implicit differentiation. Unlike most existing methods that directly differentiate KKT conditions after obtaining the optimal solution for an optimization problem, Alt-Diff performs the forward and backward procedures simultaneously. Both the forward and backward procedures can be truncated without sacrificing much accuracy in terms of the gradients of optimization layers. Overall, our contributions are three-fold:

- We develop a new differentiation framework Alt-Diff that decouples the optimization layers in an alternating way. Alt-Diff significantly reduces the dimension of KKT matrix and thus increases the computational speed of implicit differentiation especially for large-scale problems. Notably, we show that Alt-Diff enjoys quadratic computational complexity in the backward pass.
- We theoretically show that: 1) Alt-Diff can converge to the consistent gradients obtained by differentiating KKT conditions; 2) the error between the gradient obtained by the truncated Alt-Diff and by differentiating KKT conditions is upper bounded by the same order of variables' truncation error. Therefore, Alt-Diff can be truncated to accelerate the computational speed without scarifying much accuracy.
- We conduct a series of experiments and show that Alt-Diff can achieve results comparable to state-of-the-art methods in much less time. The fast performance of Alt-Diff comes from the dimension reduction of KKT matrix and the truncated capability of Alt-Diff.

2 RELATED WORK

Differentiation for optimization layers Embedding optimization problems into deep learning architectures as optimization layers to integrate domain knowledge has received increasing attention in recent years Gould et al. (2016); Amos & Kolter (2017); Agrawal et al. (2019a); Gould et al. (2021). For example, Gould et al. Gould et al. (2016) collected some general techniques for differentiating argmin and argmax optimization problems based on the Implicit Function Theorem, while Amos et al. Amos & Kolter (2017) proposed a network architecture, namely OptNet, that integrates quadratic optimization into deep networks. This work was further extended for end-to-end learning with stochastic programming Donti et al. (2017). For more general convex optimization, Agrawal et al. Agrawal et al. (2019b) proposed CvxpyLayer that differentiates through disciplined convex programs and some sparse operations and LSQR Paige & Saunders (1982) to speed up the differentiation procedure. Although these have been worthwhile attempts to accelerate the differentiation procedure, they still largely focus on implicitly differentiating the KKT conditions in a way that requires expensive operations on a Jacobian matrix. A differentiation solver named JaxOpt has also been put forward that is based on an implicit automatic differentiation mechanism under the Jax framework Blondel et al. (2021).

Unrolling methods Another direction for differentiating optimization problems is the unrolling methods Monga et al. (2021); Domke (2012). These approximate the objective function with a first-order gradient method and then incorporate the gradient of the inner loop optimization into the training procedures Belanger & McCallum (2016); Belanger et al. (2017); Amos et al. (2017); Metz et al. (2019). This is an iterative procedure, which is usually truncated to a fixed number of iterations. Although unrolling methods are easy to implement, most of their applications are limited to unconstrained problems. If constraints are added, the unrolling solutions have to be projected into a feasible region, significantly increasing the computation costs. By contrast, Alt-Diff only requires a

very simple operation that projects the slack variable to the nonnegative orthant. This substantially improves the efficiency of subsequent updates and reduces the method's computational complexity.

Implicit models There has been growing interest in implicit models in recent years J. Z. Kolter & Johnson (2020); Zhang et al. (2020); Bolte et al. (2021). Implicit layers replace the traditional feed-forward layers of neural networks with fixed point iterations to compute inferences. They have been responsible for substantial advances in many applications, including Neural ODE Chen et al. (2018); Dupont et al. (2019), Deep Equilibrium Models Bai et al. (2019; 2020); Gurumurthy et al. (2021), nonconvex optimization problems Wang et al. (2019) and implicit 3D surface layers Michalkiewicz et al. (2019); Park et al. (2019), etc. Implicit layers have a similar computational complexity to optimization layers as they also requires solving a costly Jacobian-based equation based on the Implicit Function Theorem. Recently, Fung et al. (2021) proposed a Jacobian-free backpropagation method to accelerate the speed of training implicit layers. However, this method is not suitable for the optimization layers with complex constraints. In terms of training implicit models, Geng et al. (2021) also proposed a novel gradient estimate called phantom gradient which relies on fixed-point unrolling and a Neumann series to provide a new update direction; computation of precise gradient is forgone. Implicit models have also been extended to more complex learning frameworks, such as attention mechanisms Geng et al. (2020) and Graph Neural Networks Yang et al. (2021).

3 PRELIMINARY: DIFFERENTIABLE OPTIMIZATION LAYERS

We consider a parameterized convex optimization problems with polyhedral constraints:

$$\begin{aligned} \min_{x} \quad & f(x;\theta) \\ s.t. \quad & x \in \mathcal{C}(\theta) \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$ is the decision variable, the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is convex and the constraint is a polyhedron parameterized by $\theta \in \mathbb{R}^m$: $\mathcal{C}(\theta) = \{x | A(\theta)x = b(\theta), \quad G(\theta)x \leq h(\theta)\}$. For any given θ, a solution of optimization problem (1) is $x^\star \in \mathbb{R}^n$ that minimizes $f(x;\theta)$ while satisfying the constraints, i.e. $x^\star \in \mathcal{C}(\theta)$. Then the optimization problem (1) can be viewed as a mapping that maps the parameters θ to the solution $x^\star$. Here, we focus on convex optimization with affine constraints due to its wide applications in control systems Guo & Wang (2010), signal processing Mattingley & Boyd (2010), communication networks Luong et al. (2017); Bui et al. (2017), etc.

Since optimization problems can capture well-defined specific domain knowledge in a model-driven way, embedding such optimization problems into neural networks within an end-to-end learning framework can simultaneously leverage the strengths of both the model-driven and the data-driven methods. Unlike conventional neural networks, which are defined through explicit expressions of each layer, an optimization layer embedded in optimization problem (1) is defined as follows:

Definition 3.1. (*Optimization Layer*) A layer in a neural network is defined as an optimization layer if its input is the optimization parameters $\theta \in \mathbb{R}^m$ and its output $x^\star \in \mathbb{R}^n$ is the solution of the optimization problem (1).

The optimization layer can be treated as an implicit function $\mathcal{F} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ with $\mathcal{F}(x^\star, \theta) = \mathbf{0}$. In the deep learning architecture, optimization layers are implemented together with other explicit layers in an end-to-end framework. During the training procedure, the chain rule is used to back propagate the gradient through the optimization layer. Given a loss function $\mathcal{R}$, the derivative of the loss w.r.t the parameter θ of the optimization layer is

$$\frac{\partial \mathcal{R}}{\partial \theta} = \frac{\partial \mathcal{R}}{\partial x^\star} \frac{\partial x^\star}{\partial \theta}. \tag{2}$$

Obviously, the derivative $\frac{\partial \mathcal{R}}{\partial x^\star}$ can be easily obtained by automatic differentiation techniques on explicit layers, such as fully connected layers and convolutional layers. However, since explicit closed-form solutions typically do not exist for optimization layers, this brings additional computational difficulties during implementation. Recent work, such as OptNet Amos & Kolter (2017) and CvxpyLayer Agrawal et al. (2019a) have shown that the Jacobian $\frac{\partial x^\star}{\partial \theta}$ can be derived by differentiating the KKT conditions of the optimization problem based on the Implicit Function Theorem Krantz & Parks (2002). This is briefly recalled in Lemma 3.2.

Lemma 3.2. *(Implicit Function Theorem) Let $\mathcal{F}(x,\theta)$ denote a continuously differentiable function with $\mathcal{F}(x^\star,\theta)=\mathbf{0}$. Suppose the Jacobian of $\mathcal{F}(x,\theta)$ is invertible at $(x^\star;\theta)$, then the derivative of the solution $x^\star$ with respect to θ is*

$$\frac{\partial x^\star}{\partial \theta} = -\left[\mathcal{J}_{\mathcal{F};x}\right]^{-1}\mathcal{J}_{\mathcal{F};\theta}, \tag{3}$$

where $\mathcal{J}_{\mathcal{F};x} := \nabla_x \mathcal{F}(x^\star,\theta)$ and $\mathcal{J}_{\mathcal{F};\theta} := \nabla_\theta \mathcal{F}(x^\star,\theta)$ are respectively the Jacobian of $\mathcal{F}(x,\theta)$ w.r.t. x and θ.

It is worth noting that the differentiation procedure needs to calculate the optimal value $x^\star$ and the inverse of Jacobian matrix $\mathcal{J}_{\mathcal{F};x}$, which is very computationally expensive, especially for the large-scale problems. Although the matrix factorization techniques in OptNet and sparse optimization techniques in CvxpyLayer are proposed to alleviate this issue, the computational difficulty is still $O\left(T(n+n_c)^3\right)$, where T is the number of iterations required by the inner solvers, n and n_c are respectively the number of variables and constraints. This means that differentiating KKT conditions is not quite scalable to large optimization problems. Therefore, our main target is to develop a new method that can significantly increase the computational speed of the differentiation procedure.

4 Alternating differentiation for optimization layers

This section provides the details of the Alt-Diff algorithm for optimization layers. Alt-Diff decomposes the differentiation of a large-scale KKT system into a smaller problems that are solved in a primal-dual alternating way (see Section 4.1). The convergence analysis of Alt-Diff is also given. Alt-Diff reduces the computational complexity of the model and significantly improves the computational efficiency of the optimization layers. In Section 4.2, we present the computational complexity of forward and backward pass of Alt-Diff. Specifically, Alt-Diff enjoys quadratic complexity in the backward pass. Finally, we theoretically analyze the truncated capability of Alt-Diff in Section 4.3. Notably, Alt-Diff can be truncated for inexact solutions to further increase computational speeds without sacrificing much accuracy.

4.1 Alternating differentiation

Motivated by the scalability of the operator splitting method Glowinski & Le Tallec (1989); Stellato et al. (2020), Alt-Diff first decouples a constrained optimization problem into multiple subproblems based on ADMM. Each splitted operator is then differentiated to establish the derivatives of the primal and dual variables w.r.t the parameters in an alternating fashion. The augmented Lagrange function of problem (1) with a quadratic penalty term is:

$$\begin{aligned}&\max_{\lambda,\nu}\min_{x,s\geq 0}\mathcal{L}(x,s,\lambda,\nu;\theta)\\ &\quad= f(x;\theta)+\langle\lambda,Ax-b\rangle+\langle\nu,Gx+s-h\rangle+\frac{\rho}{2}(\|Ax-b\|^2+\|Gx+s-h\|^2),\end{aligned} \tag{4}$$

where $s\geq 0$ is a non-negative slack variable, and $\rho>0$ is a hyperparameter associated with the penalty term. Here and later in this paper, we omit the parameter θ in the polyhedral constraints to simplify the formulation. Accordingly, the following ADMM procedures are used to alternatively update the primary, slack and dual variables:

$$\begin{cases} x_{k+1} = \arg\min_{x}\mathcal{L}(x,s_k,\lambda_k,\nu_k;\theta), & \text{(5a)}\\ s_{k+1} = \arg\min_{s\geq 0}\mathcal{L}(x_{k+1},s,\lambda_k,\nu_k;\theta), & \text{(5b)}\\ \lambda_{k+1} = \lambda_k+\rho(Ax_{k+1}-b), & \text{(5c)}\\ \nu_{k+1} = \nu_k+\rho(Gx_{k+1}+s_{k+1}-h). & \text{(5d)}\end{cases}$$

Note that the primal variable x_{k+1} is updated by solving an unconstrained optimization problem. The update of the slack variable s_{k+1} is easily obtained by a closed-form solution via a Rectified Linear Unit (ReLU) as

$$s_{k+1} = \mathbf{ReLU}_{s\geq 0}\left(-\frac{1}{\rho}\nu_k-(Gx_{k+1}-h)\right). \tag{6}$$

The differentiations for the slack and dual variables are trivial and can be done using an automatic differentiation technique. Therefore, the computational difficulty is now concentrated around differentiating the primal variables, which can be done using the Implicit Function Theorem. Applying a differentiation technique to alternating procedure (5) leads to:

$$\begin{cases} \dfrac{\partial x_{k+1}}{\partial \theta} = -\left(\nabla_x^2 \mathcal{L}(x_{k+1})\right)^{-1} \nabla_{x,\theta} \mathcal{L}(x_{k+1}), & \text{(7a)} \\ \dfrac{\partial s_{k+1}}{\partial \theta} = -\dfrac{1}{\rho} \mathbf{sgn}(s_{k+1}) \cdot \mathbf{1}^T \odot \left(\dfrac{\partial \nu_k}{\partial \theta} + \rho \dfrac{\partial (G x_{k+1} - h)}{\partial \theta} \right), & \text{(7b)} \\ \dfrac{\partial \lambda_{k+1}}{\partial \theta} = \dfrac{\partial \lambda_k}{\partial \theta} + \rho \dfrac{\partial (A x_{k+1} - b)}{\partial \theta}, & \text{(7c)} \\ \dfrac{\partial \nu_{k+1}}{\partial \theta} = \dfrac{\partial \nu_k}{\partial \theta} + \rho \dfrac{\partial (G x_{k+1} + s_{k+1} - h)}{\partial \theta}, & \text{(7d)} \end{cases}$$

where $\nabla_x^2 \mathcal{L}(x_{k+1}) = \nabla_x^2 f(x_{k+1})^T + \rho A^T A + \rho G^T G$, $\odot$ represents Hadamard production, $\mathbf{sgn}(s_{k+1})$ denotes a function such that $\mathbf{sgn}(s_{k+1}^i) = 1$ if $s_{k+1}^i \geq 0$ and $\mathbf{sgn}(s_{k+1}^i) = 0$ vice versa.

We summarize the procedure of Alt-Diff in Algorithm 1 and provide a framework of Alt-Diff in Appendix A. The convergence behavior of Alt-Diff can be inspected by checking $\|x_{k+1} - x_k\| / \|x_k\| < \epsilon$, where ϵ is a predefined threshold.

Algorithm 1 Alt-Diff

Parameter: θ from the previous layer
Initialize slack variable and dual variables
while $\|x_{k+1} - x_k\| / \|x_k\| \geq \epsilon$ **do**
  Forward update by (5)
  Primal update $\frac{\partial x_{k+1}}{\partial \theta}$ by (7a)
  Slack update $\frac{\partial s_{k+1}}{\partial \theta}$ by (7b)
  Dual update $\frac{\partial \lambda_{k+1}}{\partial \theta}$ and $\frac{\partial \nu_{k+1}}{\partial \theta}$ by (7c) and (7d)
end while
return $x^\star$ and its gradient $\frac{\partial x^\star}{\partial \theta}$

Notably, Alt-Diff is somehow similar to unrolling methods as in Domke (2012); Foo et al. (2007). However, these unrolling methods were designed for unconstrained optimization. If constraints are added, the unrolling solutions have to be projected into a feasible region. Generally this projector operators are very computationally expensive. By contrast, Alt-Diff can decouple constraints from the optimization and only involves a very simple operation that projects the slack variable s to the nonnegative orthant $s \geq 0$. This significantly improves the efficiency of subsequent updates and reduces the overall computational complexity of Alt-Diff. Besides, conventional unrolling methods usually need more memory consumption as all the intermediate computational results have to be recorded. However, when updating (7) continuously, Alt-Diff does not need to save the results of the previous round. Instead, it only saves the results of the last round, that is, the previous $\frac{\partial x_k}{\partial \theta}$ is replaced by the $\frac{\partial x_{k+1}}{\partial \theta}$ iteratively.

Convergence analysis

Lemma 4.1. *(Fixed point iteration Kirk (1965)) Suppose $\mathcal{G}(\tilde{x}, \theta)$ is an L-Lipschitz continuous function, for any $\tilde{x}_0 \in \mathcal{X}$, by the iteration $\tilde{x}_{k+1} = \mathcal{G}(\tilde{x}_k, \theta)$ converges to a fixed point $\tilde{x}^\star(\theta)$.*

The linear system (7) can be viewed as an operator $\mathcal{G}(\tilde{x}, \theta)$, with $\tilde{x}$ collecting the variables $[x, s, \lambda, \nu]$. By Lemma 4.1, we can derive the following theorem.

Theorem 4.2. *Suppose the gradient of the solution $x^\star$ with respect to the parameters θ obtained by CvxpyLayer Agrawal et al. (2019a) through differentiating KKT conditions implicitly is $\frac{\partial x^\star}{\partial \theta}$. Let $\frac{\partial x_k}{\partial \theta}$ denote the gradient obtained by the k-th primal differentiation (7a). Then*

$$\lim_{k \to \infty} \frac{\partial x_k}{\partial \theta} = \frac{\partial x^\star}{\partial \theta}. \tag{8}$$

This theorem shows that the gradient obtained by Alt-Diff is consistent with the one obtained by the gradient of the optimality conditions implicitly. The detailed proof as well as the procedure of differentiating KKT conditions is presented in the Appendix C.

4.2 COMPLEXITY ANALYSIS OF ALT-DIFF

In the forward pass, the unconstrained optimization problem (5a) can be easily solved by Newton's methods where the inverse of Hessian, i.e. $(\nabla_x^2 \mathcal{L}(x_{k+1}))^{-1}$ is needed. It can be directly used in the backward pass (7a) to accelerate the computation after the forward pass. The computation is even more efficient in the case of quadratic programmings, where $(\nabla_x^2 \mathcal{L}(x_{k+1}))^{-1}$ is constant and only needs to be computed once. More details are referred to Appendix B.1. As Alt-Diff decouples the objective function and constraints, it is obviously that the dimension of Hessian matrix $\nabla_x^2 \mathcal{L}(x_{k+1})$ is reduced to the number of variables n. This differs from the OptNet Amos & Kolter (2017) and CvxpyLayer Agrawal et al. (2019a) that directly differentiate through the KKT conditions, leading to a $(n + n_c)$ - dimensional Hessian matrix.

The forward pass of solving the optimal solutions (5a) requires $\mathcal{O}(Mn^3)$ in general convex target functions, where M is the iteration number of inner solvers; the complexity of the backward pass of Alt-Diff is reduced to $\mathcal{O}(kn^2)$ benefiting from the inherited $(\nabla_x^2 \mathcal{L}(x_{k+1}))^{-1}$ obtained in the forward pass. We present the comparison of complexity between CvxpyLayer and Alt-Diff in Table 1. Since OptNet only works for Quadratic programming and shares similar complexity with CvxpyLayer, so we do not list it here. As shown in OptNet and CvxpyLayer, there are some ways to speed up the cubic complexity in the forward pass by solving linear systems via matrix decomposition, conjugate gradient methods, etc, these techniques can also be applied in Alt-Diff.

Table 1: Time complexity comparision between CvxpyLayer and Alt-Diff

	General cases		Quadratic Programmings	
	Forward	Backward	Forward	Backward
CvxpyLayer	$\mathcal{O}\big(T(n+n_c)^3\big)$	$\mathcal{O}\big((n+n_c)^3\big)$	$\mathcal{O}\big(T(n+n_c)^3\big)$	$\mathcal{O}\big((n+n_c)^3\big)$
Alt-Diff	$\mathcal{O}\big(Mn^3\big)$	$\mathcal{O}\big(kn^2\big)$	$\mathcal{O}\big(n^3+(k-1)n^2\big)$	$\mathcal{O}\big(kn^2\big)$

For more clarity, we take several optimization layers for examples to show the complexity of Alt-Diff, including Quadratic Layer Amos & Kolter (2017), constrained Sparsemax Layer Malaviya et al. (2018) and constrained Softmax Layer Martins & Astudillo (2016). These optimization layers were proposed to integrate specific domain knowledge in the learning procedure. Detailed derivations for the primal differentiation procedure (7a) are provided in the Appendix B.2.

4.3 TRUNCATED CAPABILITY OF ALT-DIFF

As shown in recent work Fung et al. (2021); Geng et al. (2021), exact gradients are not necessary during the learning process of neural networks. Therefore, we can truncate the iterative procedure of Alt-Diff given some threshold values ϵ. Existing methods need to complete the forward pass and then calculate the gradient of the optimization layers Amos & Kolter (2017); Barratt (2018); Agrawal et al. (2019a). In Alt-Diff, the forward and backward passes are implemented iteratively, therefore it can be easily truncated. Next, we give theoretical analysis to show the influence of truncated Alt-Diff.

Assumption A *(L-smooth)* The first order derivatives of loss function $\mathcal{R}$ and the second derivatives of the augmented Lagrange function $\mathcal{L}$ are L-Lipschitz. For $\forall x_1, x_2 \in \mathbb{R}^n$,

$$\|\nabla_x \mathcal{R}(\theta; x_1) - \nabla_x \mathcal{R}(\theta; x_2)\| \leq L_1 \|x_1 - x_2\|, \tag{9a}$$

$$\|\nabla_x^2 f(x_1) - \nabla_x^2 f(x_2)\| \leq L_2 \|x_1 - x_2\|, \tag{9b}$$

$$\|\nabla \mathcal{L}_{x,\theta}(x_1) - \nabla \mathcal{L}_{x,\theta}(x_2)\| \leq L_3 \|x_1 - x_2\|. \tag{9c}$$

Assumption B *(Bound of gradients)* The first and second order derivative of function $\mathcal{L}$ and $\mathcal{R}$ are bounded as follows:

$$\nabla_x \mathcal{R}(\theta; x) \preceq \mu_1 I, \qquad \nabla_x^2 \mathcal{L}(x) \succeq \mu_2 I, \qquad \nabla_{x,\theta} \mathcal{L}(x) \preceq \mu_3 I, \tag{10}$$

where μ_1, μ_2 and μ_3 are positive constants. The above inequalities hold for $\forall x \in \mathbb{R}^n$.

Assumption C (*Nonsingular Hessian)* Function $f(x)$ is twice differentiable. The Hessian matrix of augmented Lagrange function $\mathcal{L}$ is invertiable at any point $x \in \mathbb{R}^n$.

Theorem 4.3. *(Error of gradient obtained by truncated Alt-Diff) Suppose x_k is the truncated solution at the k-th iteration, the error between the gradient $\frac{\partial x_k}{\partial \theta}$ derived by the truncated Alt-Diff and the real one $\frac{\partial x^\star}{\partial \theta}$ has an upper bound, i.e.,*

$$\left\| \frac{\partial x_k}{\partial \theta} - \frac{\partial x^\star}{\partial \theta} \right\| \leq C_1 \|x_k - x^\star\|, \tag{11}$$

where $C_1 = \dfrac{L_3}{\mu_2} + \dfrac{\mu_3 L_2}{\mu_2^2}$ is a constant.

Please refer to Appendix D for our proof. The theorem has shown that the primal differentiation (7a) obtained by Alt-Diff has the same order of error brought by the truncated iterations of (5a). Moreover, we can derive the following corollary with respect to the general loss function $\mathcal{R}$ accordingly.

Corollary 4.4. *(Error of the inexact gradient in terms of loss function) Followed by Theorem 4.3, the error of the gradient w.r.t. θ in loss function $\mathcal{R}$ is bounded as follows:*

$$\|\nabla \mathcal{R}(\theta; x_k) - \nabla \mathcal{R}(\theta; x^\star)\| \leq C_2 \|x_k - x^\star\|, \tag{12}$$

where $C_2 = L_1 + \dfrac{\mu_3 L_1 + \mu_1 L_3}{\mu_2} + \dfrac{\mu_1 \mu_3 L_2}{\mu_2^2}$ is a constant.

Please refer to Appendix E for our proof. Truncating the iterative process in advance will result in fewer steps and higher computational efficiency without sacrificing much accuracy. In the next section, we will show the truncated capability of Alt-Diff by several numerical experiments.

5 Experimental results

In this section, we evaluate Alt-Diff over a series of experiments to demonstrate its performance in terms of computational speed as well as accuracy. First, we implemented Alt-Diff over several optimization layers, including the sparse and dense constrained quadratic layers and constrained softmax layers. In addition, we applied the Alt-Diff to the real-world task of energy generation scheduling under a predict-then-optimize framework. Moreover, we also tested Alt-Diff with a quadratic layer in an image classification task on the MNIST dataset. To verify the truncated capability of Alt-Diff, we also implemented Alt-Diff with different values of tolerance. All the experiments were implemented on a Core Intel(R) i7-10700 CPU @ 2.90GHz with 16 GB of memory.

5.1 Numerical experiments on several optimization layers

In this section, we compare Alt-Diff with OptNet and CvxpyLayer on several optimization layers, including the constrained Sparsemax layer, the dense Quadratic layer and the constrained Softmax layer which are referred to typical cases of sparse, dense quadratic problems and problems with general convex objective function, respectively.

In the dense quadratic layer, the parameters P, q, A, b, G, h were randomly generated from the same random seed with $P \succeq 0$. The tolerance ϵ is set as 10^{-3} for OptNet, CvxpyLayer and Alt-Diff. As the parameters in optimization problems are generated randomly, we compared Alt-Diff with the "dense" mode in CvxpyLayer. All the numerical experiments were executed 5 times with the average times reported as the results in Table 2. It can be seen that OptNet runs much faster than CvxpyLayer for the dense quadratic layer, while our Alt-Diff outperforms both the two counterparts. Additionally, the superiority of Alt-Diff becomes more evident with the increase of problem size.

Fig. 1 shows the trend of Jacobian $\frac{\partial x_k}{\partial b}$ with iterations in primal update (7a). As shown in Theorem 4.2, the gradient obtained by Alt-Diff can gradually converge to the results obtained by KKT derivative as the number of iterations increases. The results in Fig. 1 also validate this theorem. The detailed results of other optimization layers are provided in Appendix F.

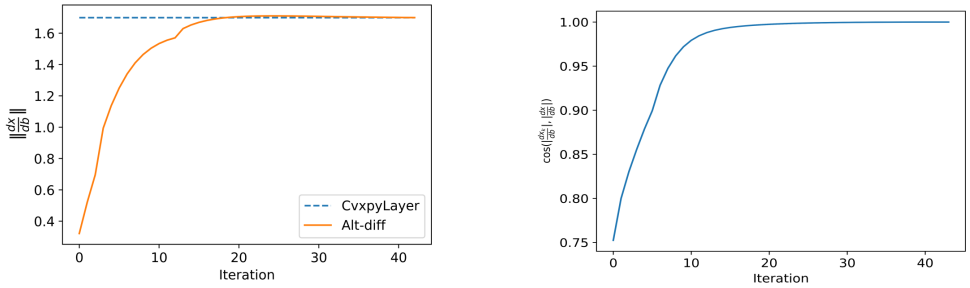
5.2 Energy generation scheduling

To test the accuracy of sparse layers, we applied the Alt-Diff to the real-world task of energy generation scheduling Donti et al. (2017) under a predict-then-optimize framework Elmachtoub & Grigas

Table 2: Comparison of running time (s) and cosine distances of gradients in dense Quadratic layers with tolerance $\epsilon = 10^{-3}$.

	tiny	small	medium	large
Num of variable n	1500	3000	5000	10000
Num of ineq. m	500	1000	2000	5000
Num of eq. p	200	500	1000	2000
Num of elements	4.84×10^6	2.03×10^7	6.4×10^7	2.89×10^8
OptNet	0.81	9.33	40.71	306.09
CvxpyLayer (total)	25.34	212.86	1056.18	-
Initialization	1.86	10.51	38.73	-
Canonicalization	0.37	1.47	4.40	-
Forward	14.00	117.06	572.24	-
Backward	9.11	83.82	440.80	-
Alt-Diff (total)	**0.71**	**5.75**	**20.20**	**154.52**
Inversion	0.11	0.91	3.03	21.12
Forward and backward	0.60	4.84	17.17	133.40
Cosine Dist.	0.999	0.999	0.999	0.998

"-" represents that the solver cannot generate the gradients.



(a) The norm variation trends. (Blue dotted line is the gradient $\frac{\partial x^\star}{\partial b}$ obtained by CvxpyLayer).

(b) The cosine distance of the obtained gradient between Alt-Diff and CvxpyLayer.

Figure 1: The variation of primal variable gradient of quadratic layers. (The threshold of iteration is set as 10^{-3})

(2021); Mandi et al. (2020); Mandi & Guns (2020). Predict-then-optimize is an end-to-end learning model, in which some unknown variables are first predicted by machine learning methods and are then successively optimized. The main idea of predict-then-optimize is to use the optimization loss (13) to guide the prediction, rather than using the prediction loss as in the normal learning style.

$$L(\hat{\theta}, \theta) = \frac{1}{2} \sum_{i=1}^{m} (x_i^\star(\hat{\theta}) - x_i^\star(\theta))^2 \tag{13}$$

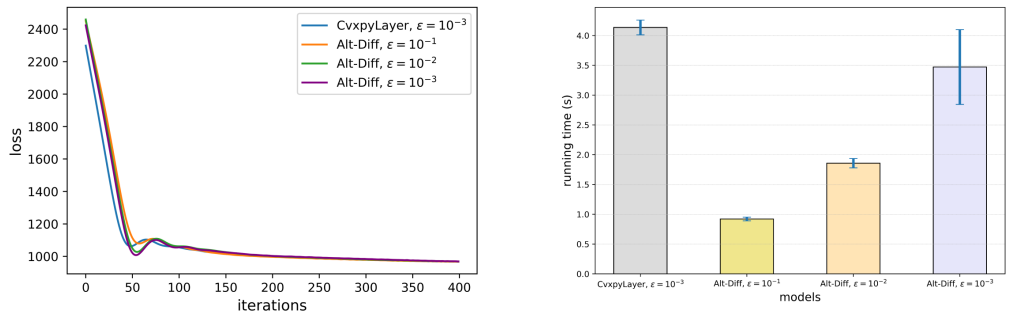
We consider the energy generation scheduling task based on the actual power demand for electricity in a certain region web. In this setting, a power system operator must decide the electricity generation to schedule for the next 24 hours based on some historical electricity demand information. In this paper, we used the hourly electricity demand data over the past 72 hours to predict the real power demand in the next 24 hours. The predicted electricity demand was then input into the following optimization problem to schedule the power generation:

$$\begin{aligned} \min_{x_k} \quad & \sum_{k=1}^{24} \|x_k - P_{d_k}\|^2 \\ s.t. \quad & |x_{k+1} - x_k| \leq r, \quad k = 1, 2, \cdots, 23 \end{aligned} \tag{14}$$

where P_{d_k} and x_k denote the power demand and power generation at time slot k respectively. Due to physical limitations, the variation of power generation during a single time slot is not allowed to

exceed a certain threshold value r. During training procedures, we use a neural network with two hidden layers to predict the electricity demand of the next 24 hours based on the data of the previous 72 hours. All the P_{d_k} have been normalized into the $[0, 100]$ interval.

The optimization problem in this task can be treated like the optimization layer previously considered. As shown in Table 4, since the constraints are sparse, OptNet runs much slower than the "lsqr" mode in CvxpyLayer. Therefore, we respectively implemented Alt-Diff and CvxpyLayer to obtain the gradient of power generation with respect to the power demand. During the training procedure, we used the Adam optimizer Kingma & Ba (2014) to update the parameter. Once complete, we compared the results obtained from CvxpyLayer (with tolerance 10^{-3}) and those from Alt-Diff under different levels of truncated thresholds, varying from 10^{-1}, 10^{-2} to 10^{-3}. The results are shown in Figure 2. We can see that the losses for CvxpyLayer and Alt-Diff with different truncated threshold values are almost the same, but the running time of Alt-Diff is much smaller than CvxpyLayer especially in the truncated cases.



(a) The convergence behaviors.

(b) The average running time.

Figure 2: The experimental results of Alt-diff and CvxpyLayer for energy generation scheduling.

5.3 Image Classification

We follow similar experimental settings in OptNet Amos & Kolter (2017) to embed the optimization layer into the neural networks for image classification task on MNIST dataset. We replace a layer by a dense quadratic optimization layer in the neural networks and then compare the computational speed and accuracy between Alt-Diff and OptNet. As we have shown that OptNet runs much faster than CvxpyLayer in dense quadratic layers, therefore we only compare Alt-Diff with OptNet here. More experimental details are provided in the Appendix F.

6 Conclusion

In this paper, we have proposed Alt-Diff for computationally efficient differentiation on optimization layers associated with convex objective functions and polyhedral constraints. Alt-Diff differentiates the optimization layers in a fast and recursive way. Unlike differentiation on the KKT conditions of optimization problem, Alt-Diff decouples the differentiation procedure into a primal update and a dual update by an alternating way. Accordingly, Alt-Diff substantially decreases the dimensions of the Jacobian matrix and significantly increase computational speed. Further, we have presented the computational complexity of forward and backward pass of Alt-Diff and showed that Alt-Diff enjoys quadratic computational complexity in the backward pass. We have also theoretically show the convergence of Alt-Diff and the truncated error of Alt-Diff given some general assumptions. Comprehensive experiments have demonstrated the efficiency of Alt-Diff compared to the state-of-the-arts. Currently, Alt-Diff is designed for specific optimization layers in the form of convex objective functions with polyhedral constraints. Extending Alt-Diff to more general convex optimization layers and nonconvex optimization layers are under consideration in our future work.

REFERENCES

Eletrical load. https://dataminer2.pjm.com/. Accessed: 2021-12-30.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable convex optimization layers. *arXiv preprint arXiv:1910.12430*, 2019a.

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. Differentiating through a cone program. *arXiv preprint arXiv:1904.09043*, 2019b.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pp. 146–155. PMLR, 2017.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Multiscale deep equilibrium models. *Advances in Neural Information Processing Systems*, 33:5238–5250, 2020.

Shane Barratt. On the differentiability of the solution to convex optimization problems. *arXiv preprint arXiv:1804.05098*, 2018.

David Belanger and Andrew McCallum. Structured prediction energy networks. In *International Conference on Machine Learning*, pp. 983–992. PMLR, 2016.

David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structured prediction energy networks. In *International Conference on Machine Learning*, pp. 429–439. PMLR, 2017.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.

Jérôme Bolte, Tam Le, Edouard Pauwels, and Tony Silveti-Falls. Nonsmooth implicit differentiation for machine-learning and optimization. *Advances in neural information processing systems*, 34: 13537–13549, 2021.

J Frédéric Bonnans and Alexander Shapiro. *Perturbation analysis of optimization problems*. Springer Science & Business Media, 2013.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Nicola Bui, Matteo Cesana, S Amir Hosseini, Qi Liao, Ilaria Malanchini, and Joerg Widmer. A survey of anticipatory mobile networking: Context-based classification, prediction methodologies, and optimization techniques. *IEEE Communications Surveys & Tutorials*, 19(3):1790–1821, 2017.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Steven Diamond, Vincent Sitzmann, Felix Heide, and Gordon Wetzstein. Unrolled optimization with deep priors. *arXiv preprint arXiv:1705.08041*, 2017.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326. PMLR, 2012.

Priya L Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. *arXiv preprint arXiv:1703.04529*, 2017.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *Advances in Neural Information Processing Systems*, 32, 2019.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 2021.

Chuan-sheng Foo, Andrew Ng, et al. Efficient multiple hyperparameter learning for log-linear models. *Advances in neural information processing systems*, 20, 2007.

Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Jfb: Jacobian-free backpropagation for implicit networks. *arXiv preprint arXiv:2103.12803*, 2021.

Zhengyang Geng, Meng-Hao Guo, Hongxu Chen, Xia Li, Ke Wei, and Zhouchen Lin. Is attention better than matrix decomposition? In *International Conference on Learning Representations*, 2020.

Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. *Advances in Neural Information Processing Systems*, 34:24247–24260, 2021.

Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

Roland Glowinski and Patrick Le Tallec. *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*. SIAM, 1989.

Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016.

Stephen Gould, Richard Hartley, and Dylan Campbell. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):3988–4004, 2021.

Lei Guo and Hong Wang. *Stochastic distribution control system design: a convex optimization approach*. Springer, 2010.

Swaminathan Gurumurthy, Shaojie Bai, Zachary Manchester, and J Zico Kolter. Joint inference and input optimization in equilibrium networks. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

D. Duvenaud J. Z. Kolter and M. Johnson. Deep implicit layers - neural odes, deep equilibirum models, and beyond. `http://implicit-layers-tutorial.org/`, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

William Art Kirk. A fixed point theorem for mappings which do not increase distances. *The American mathematical monthly*, 72(9):1004–1006, 1965.

Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

Karl Kunisch and Thomas Pock. A bilevel optimization approach for parameter learning in variational models. *SIAM Journal on Imaging Sciences*, 6(2):938–983, 2013.

Nguyen Cong Luong, Ping Wang, Dusit Niyato, Yonggang Wen, and Zhu Han. Resource management in cloud networking using economic analysis and pricing models: A survey. *IEEE Communications Surveys & Tutorials*, 19(2):954–1001, 2017.

Chaitanya Malaviya, Pedro Ferreira, and André FT Martins. Sparse and constrained attention for neural machine translation. *arXiv preprint arXiv:1805.08241*, 2018.

Jayanta Mandi and Tias Guns. Interior point solving for LP-based prediction+ optimisation. *arXiv preprint arXiv:2010.13943*, 2020.

Jayanta Mandi, Peter J Stuckey, Tias Guns, et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 1603–1610, 2020.

Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pp. 1614–1623. PMLR, 2016.

John Mattingley and Stephen Boyd. Real-time convex optimization in signal processing. *IEEE Signal processing magazine*, 27(3):50–61, 2010.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pp. 4556–4565. PMLR, 2019.

Mateusz Michalkiewicz, Jhony K Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4743–4752, 2019.

Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

Christopher C Paige and Michael A Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71, 1982.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 165–174, 2019.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12 (4):637–672, 2020.

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 *ICML*, 2019.

Yongyi Yang, Tang Liu, Yangkun Wang, Jinjing Zhou, Quan Gan, Zhewei Wei, Zheng Zhang, Zengfeng Huang, and David Wipf. Graph neural networks inspired by classical iterative algorithms. In *International Conference on Machine Learning*, pp. 11773–11783. PMLR, 2021.

Qianggong Zhang, Yanyang Gu, Michalkiewicz Mateusz, Mahsa Baktashmotlagh, and Anders Eriksson. Implicitly defined layers in neural networks. *arXiv preprint arXiv:2003.01822*, 2020.

Appendix

We present some details for the proposed Alt-Diff algorithm, including the framework of Alt-Diff, the derivation of some specific layers, the proof of theorems, detailed experimental results and implementation notes.

A Framework of Alt-Diff

The Alt-Diff framework for optimization layers within an end-to-end learning architecture is illustrated in Figure 3.

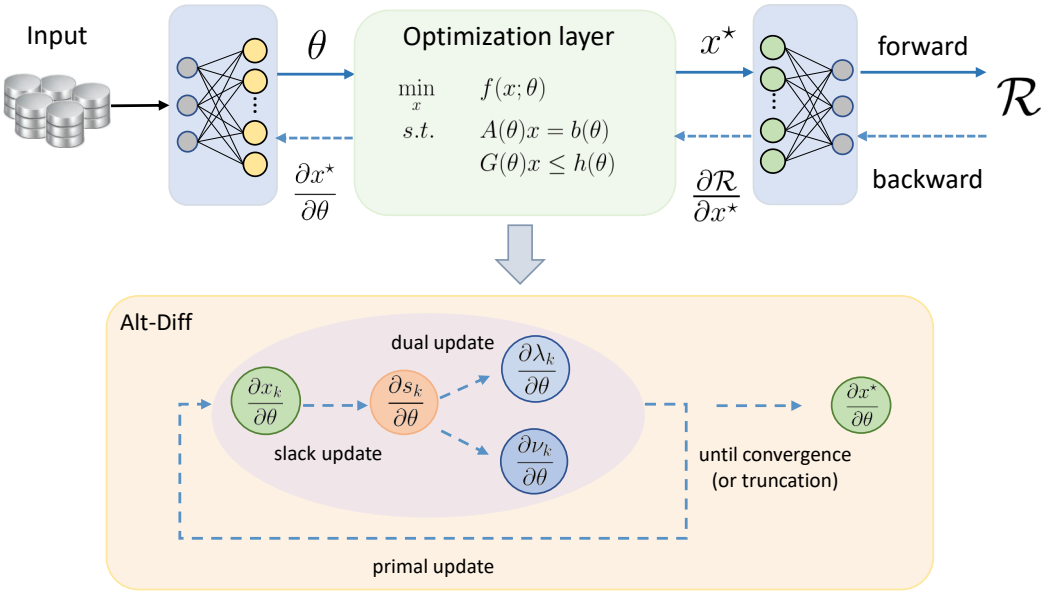


Figure 3: The model architecture of Alt-Diff for optimization layers.

B Primal Differentiation

B.1 Inheritance of Hessian matrix

According to the first-order optimality condition in forward pass (5a), we have

$$\nabla \mathcal{L}(x_{k+1}) = \nabla f(x_{k+1}) + A^T\lambda_k + G^T\nu_k + \rho A^T(Ax_{k+1} - b) + \rho G^T(Gx_{k+1} + s_k - h) = 0. \tag{15}$$

Generally, the equation (15) can be efficiently solved by iterative methods such as Newton's methods as follows:

$$\begin{aligned} x_{k+1}^i &= x_{k+1}^{i-1} - \alpha\Big(\nabla_x^2 \mathcal{L}\big(x_{k+1}^{i-1}\big)\Big)^{-1} \nabla \mathcal{L}(x_{k+1}^{i-1}) \\ &= x_{k+1}^{i-1} - \alpha\Big(\nabla_x^2 f\big(x_{k+1}^{i-1}\big) + \rho A^T A + \rho G^T G\Big)^{-1} \nabla \mathcal{L}(x_{k+1}^{i-1}), \end{aligned} \tag{16}$$

where x_{k+1}^i denotes the value of x_{k+1} at the i-th Newton's iteration and α is the step size. Therefore, when x_{k+1}^i converges at its optimal value x_{k+1}, i.e., after the forward pass, the inverse of Hessian $\big(\nabla_x^2 \mathcal{L}(x_{k+1})\big)^{-1}$ can be directly used to accelerate the computation of the backward pass (7a). Especially, when $f(x)$ is quadratic, i.e. $f(x) = \frac{1}{2}x^T P x + q^T x$, the inverse of Hessian becomes constant as:

$$\left(\nabla_x^2 \mathcal{L}(x_{k+1})\right)^{-1} = \left(P^T + \rho A^T A + \rho G^T G\right)^{-1}. \tag{17}$$

B.2 SPECIAL LAYERS

We take Quadratic Layer Amos & Kolter (2017), constrained Sparsemax Layer Malaviya et al. (2018) and constrained Softmax Layer Martins & Astudillo (2016) as special cases in order to show the computation of backward pass (7). The details of these layers are provided in the first two columns of Table 3. The differentiation procedure (7b) - (7d) for these layers are exactly the same. The only difference relies on the primal differentiation procedure (7a), which is listed in the last column of Table 3.

Table 3: The primal differentiation procedure of Alt-Diff for several optimization layers.

Layers	Optimization problems	Primal differentiation (7a)
Constrained Sparsemax Layer	$\min_x \ \|x-y\|_2^2 \ \ s.t.\ 1^T x = 1,\ 0 \le x \le u$	$\frac{\partial x_{k+1}}{\partial \theta} = -\left((2+2\rho)I + \rho \mathbf{1}\cdot\mathbf{1}^T\right)^{-1} \mathcal{L}_{x\theta}(x_{k+1})$
Quadratic Layer	$\min_x \ \frac{1}{2}x^T P x + q^T x \ \ s.t.\ Ax = b,\ Gx \le h$	$\frac{\partial x_{k+1}}{\partial \theta} = -\left(P^T + \rho A^T A + \rho G^T G\right)^{-1} \mathcal{L}_{x\theta}(x_{k+1})$
Constrained Softmax Layer	$\min_x \ -y^T x + H(x) \ \ s.t.\ 1^T x = 1,\ 0 \le x \le u$	$\frac{\partial x_{k+1}}{\partial \theta} = -\left(\mathbf{diag}^{-1}(x) + 2\rho I + \rho \mathbf{1}\cdot\mathbf{1}^T\right)^{-1} \mathcal{L}_{x\theta}(x_{k+1})$

where $\mathcal{L}_{x\theta}(x_{k+1}) := \frac{\partial}{\partial\theta}\nabla_x \mathcal{L}(x_{k+1}, s_k, \lambda_k, \nu_k; \theta)$, $P \succeq 0$ in the Quadratic layer and $H(x) = \sum_{i=1}^{n} x_i \log(x_i)$ is the negative entropy in the constrained Softmax layer. We next give the detailed derivation for the last column of Table 3.

Proof. For the Quadratic layer and the contrained Sparsemax layer, we substitute $\nabla f(x_{k+1}) = P^T x_{k+1}$ into (15) and calculate the derivative with respect to θ, i.e.

$$P^T \frac{\partial x_{k+1}}{\partial \theta} + \frac{\partial A^T \lambda_k}{\partial \theta} + \frac{\partial G^T \nu_k}{\partial \theta} + \rho \frac{\partial A^T (A x_{k+1} - b)}{\partial \theta} + \rho \frac{\partial G^T (G x_{k+1} + s_k - h)}{\partial \theta} = 0. \tag{18}$$

Therefore we can derive the formulation of primal gradient update:

$$\begin{aligned} \frac{\partial x_{k+1}}{\partial \theta} &= -\left(P^T + \rho A^T A + \rho G^T G\right)^{-1} \left(\frac{\partial A^T \lambda_k}{\partial \theta} + \frac{\partial G^T \nu_k}{\partial \theta} - \rho \frac{\partial (A^T b - G^T (s_k - h))}{\partial \theta} \right) \\ &= -\left(P^T + \rho A^T A + \rho G^T G\right)^{-1} \mathcal{L}_{x\theta}(x_{k+1}). \end{aligned} \tag{19}$$

Compared with (17), we can find that the forward pass and backward pass in quadratic layers shared the same Hessian matrix, which significantly reduces the computational complexity.

The constrained Sparsemax Layer is a special quadratic layer that with sparse constraints, where $P = 2I$, $A = \mathbf{1}^T$ and $G = [-I, I]^T$. Therefore the primal differentiation (7a) becomes

$$\frac{\partial x_{k+1}}{\partial \theta} = -\left((2+2\rho)I + \rho \mathbf{1}\cdot\mathbf{1}^T\right)^{-1} \mathcal{L}_{x\theta}(x_{k+1}). \tag{20}$$

For constrained Softmax layer, the augmented Lagrange function is:

$$\begin{aligned} \mathcal{L} = -y^T x + \sum_{i=1}^{n} x_i \log(x_i) + \lambda^T (\mathbf{1}^T x - 1) + \nu (G x + s - h) \\ + \frac{\rho}{2}\left(\|\mathbf{1}^T x - 1\|^2 + \|G x + s - h\|^2 \right), \end{aligned} \tag{21}$$

where $G = [-I, I]^T$ and $h = [\mathbf{0}, \mathbf{u}]^T$. Taking the gradient of the first-order optimality condition (15), (18) can be replaced by

$$\mathbf{diag}^{-1}(x) \frac{\partial x_{k+1}}{\partial \theta} + \frac{\partial \mathbf{1}^T \lambda_k}{\partial \theta} + \frac{\partial G^T \nu_k}{\partial \theta} + \rho \frac{\partial \mathbf{1}(\mathbf{1}^T x_{k+1} - 1)}{\partial \theta} + \rho \frac{\partial G^T (G x_{k+1} + s_k - h)}{\partial \theta} = 0. \tag{22}$$

Similarly, the formulation of primal gradient update of the constrained Softmax layer is:

$$\begin{aligned}\frac{\partial x_{k+1}}{\partial \theta} &= -\left(\mathbf{diag}^{-1}(x) + \rho G^T G + \rho \mathbf{1}\cdot\mathbf{1}^T\right)^{-1}\left(\frac{\partial \mathbf{1}^T\lambda_k}{\partial\theta} + \frac{\partial G^T\nu_k}{\partial\theta} + \rho\frac{\partial G^T(s_k-h)}{\partial\theta}\right)\\ &= -\left(\mathbf{diag}^{-1}(x) + 2\rho I + \rho\mathbf{1}\cdot\mathbf{1}^T\right)^{-1}\mathcal{L}_{x\theta}(x_{k+1}).\end{aligned} \tag{23}$$

□

From Table 3, we can see the implementation of primal differentiation for the constrained Sparsemax layers Malaviya et al. (2018) and the quadratic optimization layers Amos & Kolter (2017) are quite simple. The Jacobian matrix involved in the primal update keeps the same during the iteration, thus the computational complexity of Alt-Diff for these layers reduces to $\mathcal{O}(kn^2)$.

C ALT-DIFF AND DIFFERENTIATING KKT

We first consider the gradient obtained by differentiating KKT conditions implicitly, and then show that it is the same as the result obtained by Alt-Diff.

C.1 DIFFERENTIATION OF KKT CONDITIONS

In convex optimization problem (1), the optimal solution is implicitly defined by the following KKT conditions:

$$\begin{cases}\nabla f(x) + A^T(\theta)\lambda + G^T(\theta)\nu = 0\\ A(\theta)x - b(\theta) = 0\\ \mathbf{diag}(\nu)(G(\theta)x - h(\theta)) = 0\end{cases} \tag{24}$$

where λ and ν are the dual variables, $\mathbf{diag}(\cdot)$ creates a diagonal matrix from a vector. Suppose $\tilde{x}$ collects the optimal solutions $[x^\star, \lambda^\star, \nu^\star]$, in order to obtain the derivative of the solution $\tilde{x}$ w.r.t. the parameters $A(\theta), b(\theta), G(\theta), h(\theta)$, we need to calculate $\mathcal{J}_{\mathcal{F};\tilde{x}}$ and $\mathcal{J}_{\mathcal{F};\theta}$ by differentiating KKT conditions (24) as follows Amos & Kolter (2017); Agrawal et al. (2019a); Zhang et al. (2020):

$$\left[\ \mathcal{J}_{\mathcal{F};x^\star}\ \middle|\ \mathcal{J}_{\mathcal{F};\lambda}\ \middle|\ \mathcal{J}_{\mathcal{F};\nu}\ \right] = \left[\begin{array}{c|c|c} f_{xx}^T(x^\star) & A^T(\theta) & B^T(\theta)\\ A(\theta) & 0 & 0\\ \mathbf{diag}(\nu)G(\theta) & \mathbf{diag}(G(\theta)x - h(\theta)) & 0\end{array}\right] \tag{25a}$$

$$\left[\ \mathcal{J}_{\mathcal{F};A}\ \middle|\ \mathcal{J}_{\mathcal{F};b}\ \middle|\ \mathcal{J}_{\mathcal{F};G}\ \middle|\ \mathcal{J}_{\mathcal{F};h}\ \right] = \left[\begin{array}{c|c|c|c} \nu^T\otimes I & 0 & \lambda^T\otimes I & 0\\ I\otimes(x^\star)^T & -I & 0 & 0\\ 0 & 0 & \lambda I\otimes(x^\star)^T & -I\end{array}\right] \tag{25b}$$

where $f_{xx}(x) := \nabla_x^2 f(x)$, $\otimes$ denotes the Kronecker product.

C.2 PROOF OF THEOREM 4.2

Theorem 4.2. *Suppose the gradient of the solution $x^\star$ with respect to the parameters θ obtained by CvxpyLayer Agrawal et al. (2019a) through differentiating KKT conditions implicitly is $\frac{\partial x^\star}{\partial\theta}$. Let $\frac{\partial x_k}{\partial\theta}$ denote the gradient obtained by the k-th primal differentiation (7a). Then*

$$\lim_{k\to\infty}\frac{\partial x_k}{\partial\theta} = \frac{\partial x^\star}{\partial\theta}. \tag{26}$$

Proof. Note that the alternating procedure of (5) is based on ADMM, which is convergent for convex optimization problems, therefore the convergence of differentiation procedure (7) is obviously. So we only need to prove that after (7) converges, we can get (3).

Suppose the optimal solutions by the alternating procedure of (5) are $[x_{\text{opt}}, s_{\text{opt}}, \lambda_{\text{opt}}, \nu_{\text{opt}}]$, then

$$\lim_{k\to\infty}\frac{\partial x_k}{\partial\theta} = \frac{\partial x_{\text{opt}}}{\partial\theta}. \tag{27}$$

After convergence, the differentiation procedure (7) becomes

$$\begin{cases} \frac{\partial x_{\text{opt}}}{\partial \theta} = -\left(f_{xx}(x_{\text{opt}})^T + \rho A^T A + \rho G^T G\right)^{-1} L_{x\theta}(x_{\text{opt}}), & \text{(28a)} \\ \frac{\partial s_{\text{opt}}}{\partial \theta} = -\frac{1}{\rho}\mathbf{sgn}(s_{\text{opt}}) \cdot \mathbf{1}^T \odot \left(\frac{\partial \nu_{\text{opt}}}{\partial \theta} + \rho \frac{\partial (G x_{\text{opt}} - h)}{\partial \theta}\right), & \text{(28b)} \\ \frac{\partial \lambda_{\text{opt}}}{\partial \theta} = \frac{\partial \lambda_{\text{opt}}}{\partial \theta} + \rho \frac{\partial (A x_{\text{opt}} - b)}{\partial \theta}, & \text{(28c)} \\ \frac{\partial \nu_{\text{opt}}}{\partial \theta} = \frac{\partial \nu_{\text{opt}}}{\partial \theta} + \rho \frac{\partial (G x_{\text{opt}} + s_{\text{opt}} - h)}{\partial \theta}. & \text{(28d)} \end{cases}$$

In order to show (26), we just need to show

$$\frac{\partial x_{\text{opt}}}{\partial \theta} = \frac{\partial x^\star}{\partial \theta}. \tag{29}$$

Recall that the differentiation through KKT conditions on the optimal solution $[x^\star, \lambda^\star, \nu^\star]$ are as follows,

$$\begin{cases} f_{xx}(x^\star)\frac{\partial x^\star}{\partial \theta} + \frac{\partial A^T \lambda^\star}{\partial \theta} + \frac{\partial G^T \nu^\star}{\partial \theta} = 0, & \text{(30a)} \\ \frac{\partial (A x^\star - b)}{\partial \theta} = 0, & \text{(30b)} \\ \frac{\partial \nu^\star}{\partial \theta}(G x^\star - h) + \nu^\star \frac{\partial (G x^\star - h)}{\partial \theta} = 0. & \text{(30c)} \end{cases}$$

By the convergence of ADMM, it is known that $[x_{\text{opt}}, \lambda_{\text{opt}}, \nu_{\text{opt}}]$ is equal to $[x^\star, \lambda^\star, \nu^\star]$. Since in convex optimization problems, the KKT condition (24) and its derivative (30) have unique solutions, therefore we only need to show that (30) can be directly derived by (28).

(28a)$\Longrightarrow$(30a):

$$\begin{aligned} & \left(f_{xx}(x_{\text{opt}})^T + \rho A^T A + \rho G^T G\right) \frac{\partial x_{\text{opt}}}{\partial \theta} \\ = \; & -\frac{\partial A^T \lambda_{\text{opt}}}{\partial \theta} - \frac{\partial G^T \nu_{\text{opt}}}{\partial \theta} + \rho \frac{\partial (A^T b - G^T (s_{\text{opt}} - h))}{\partial \theta}. \end{aligned} \tag{31}$$

Therefore,

$$\begin{aligned} & f_{xx}(x)^T \frac{\partial x_{\text{opt}}}{\partial \theta} + \frac{\partial A^T \lambda_{\text{opt}}}{\partial \theta} + \frac{\partial G^T \nu_{\text{opt}}}{\partial \theta} \\ = \; & -(\rho A^T A + \rho G^T G)\frac{\partial x_{\text{opt}}}{\partial \theta} + \rho \frac{\partial (A^T b - G^T (s_{\text{opt}} - h))}{\partial \theta} \\ = \; & -\rho \frac{\partial (A x_{\text{opt}} - b)}{\partial \theta} - \rho \frac{\partial (G x_{\text{opt}} + s_{\text{opt}} - h)}{\partial \theta} \\ = \; & 0. \end{aligned} \tag{32}$$

Clearly, (32) is exactly the same as (30a).

(28c)$\Longrightarrow$(30b): By (28c), we can obtain

$$\frac{\partial (A x_{\text{opt}} - b)}{\partial \theta} = 0, \tag{33}$$

which is exactly the same as (30b).

(28b) and (28d)$\Longrightarrow$(30c): By (28b) and (28d), we can obtain:

$$\begin{aligned} \rho \frac{\partial (G x_{\text{opt}} - h)}{\partial \theta} &= -\rho \frac{\partial s_{\text{opt}}}{\partial \theta} \\ &= \mathbf{sgn}(s_{\text{opt}}) \cdot \mathbf{1}^T \odot \left(\frac{\partial \nu_{\text{opt}}}{\partial \theta} + \rho \frac{\partial (G x_{\text{opt}} - h)}{\partial \theta}\right) \end{aligned} \tag{34}$$

We discuss this issue in two situations. Since the equality $Gx_{\text{opt}} + s_{\text{opt}} - h = 0$ holds by optimality, we have $\langle \nu_{\text{opt}}, s_{\text{opt}} \rangle = 0$ using the complementary slackness condition.

- If $s_{\text{opt},i} > 0$, then $\nu_{\text{opt},i} = 0$, we obtain

$$\begin{aligned} \rho \frac{\partial (G_i x_{\text{opt}} - h_i)}{\partial \theta} &= \mathbf{sgn}(s_{\text{opt},i}) \cdot \mathbf{1}^T \odot \left(\frac{\partial \nu_{\text{opt},i}}{\partial \theta} + \rho \frac{\partial (G_i x_{\text{opt}} - h_i)}{\partial \theta} \right) \\ &= \rho \mathbf{1} \cdot \mathbf{1}^T \odot \left(\frac{\partial \nu_{\text{opt},i}}{\partial \theta} + \rho \frac{\partial (G_i x_{\text{opt}} - h_i)}{\partial \theta} \right). \end{aligned} \tag{35}$$

  Therefore, $\dfrac{\partial \nu_{\text{opt},i}}{\partial \theta} = 0$ and

$$\frac{\partial \nu_{\text{opt},i}}{\partial \theta} (G_i x_{\text{opt}} - h_i) + \nu_{\text{opt},i} \frac{\partial (G_i x_{\text{opt}} - h_i)}{\partial \theta} = 0. \tag{36}$$

- If $s_{\text{opt},i} = 0$, then $G_i x_{\text{opt}} - h_i = 0$ and

$$\begin{aligned} \rho \frac{\partial (G_i x_{\text{opt}} - h_{\text{opt},i})}{\partial \theta} &= \mathbf{sgn}(s_i) \cdot \mathbf{1}^T \odot \left(\frac{\partial \nu_i}{\partial \theta} + \rho \frac{\partial (G_i x_{\text{opt}} - h_i)}{\partial \theta} \right) \\ &= \mathbf{0} \cdot \mathbf{1}^T \odot \left(\frac{\partial \nu_i}{\partial \theta} + \rho \frac{\partial (G_i x_{\text{opt}} - h_{\text{opt},i})}{\partial \theta} \right) \\ &= 0. \end{aligned} \tag{37}$$

  Therefore,

$$\frac{\partial \nu_{\text{opt},i}}{\partial \theta} (G_i x_{\text{opt}} - h_i) + \nu_{\text{opt},i} \frac{\partial (G_i x_{\text{opt}} - h_i)}{\partial \theta} = 0. \tag{38}$$

Therefore, (36) and (38) show (30c) holds. □

D ERROR OF THE TRUNCATED ALT-DIFF

Theorem 4.3. *(Error of gradient obtained by truncated Alt-Diff) Suppose x_k is the truncated solution at the k-th iteration, the error between the gradient $\frac{\partial x_k}{\partial \theta}$ derived by the truncated Alt-Diff and the real one $\frac{\partial x^\star}{\partial \theta}$ has an upper bound, i.e.,*

$$\left\| \frac{\partial x_k}{\partial \theta} - \frac{\partial x^\star}{\partial \theta} \right\| \leq C_1 \|x_k - x^\star\|, \tag{39}$$

where $C_1 = \dfrac{L_3}{\mu_2} + \dfrac{\mu_3 L_2}{\mu_2^2}$ is a constant.

Proof. In the differentiation procedure of Alt-Diff, we have our primal update procedure (7a), so that

$$\begin{aligned} \frac{\partial x_k}{\partial \theta} - \frac{\partial x^\star}{\partial \theta} &= -\nabla_x^2 \mathcal{L}^{-1}(x_k) \nabla_{x,\theta} \mathcal{L}(x_k) + \nabla_x^2 \mathcal{L}^{-1}(x^\star) \nabla_{x,\theta} \mathcal{L}(x^\star) \\ &= -\nabla_x^2 \mathcal{L}^{-1}(x_k) \nabla_{x,\theta} \mathcal{L}(x_k) + \nabla_x^2 \mathcal{L}^{-1}(x_k) \nabla_{x,\theta} \mathcal{L}(x^\star) \\ &\quad - \nabla_x^2 \mathcal{L}^{-1}(x_k) \nabla_{x,\theta} \mathcal{L}(x^\star) + \nabla_x^2 \mathcal{L}^{-1}(x^\star) \nabla_{x,\theta} \mathcal{L}(x^\star) \\ &= \nabla_x^2 \mathcal{L}^{-1}(x_k) \underbrace{\left(\nabla_{x,\theta} \mathcal{L}(x^\star) - \nabla_{x,\theta} \mathcal{L}(x_k) \right)}_{\Delta_1} \\ &\quad + \underbrace{\left(\nabla_x^2 \mathcal{L}^{-1}(x^\star) - \nabla_x^2 \mathcal{L}^{-1}(x_k) \right)}_{\Delta_2} \nabla_{x,\theta} \mathcal{L}(x^\star). \end{aligned} \tag{40}$$

By Assumption (9c), we can obtain that

$$\|\Delta_1\| = \|\nabla \mathcal{L}_{x,\theta}(x^\star) - \nabla \mathcal{L}_{x,\theta}(x_k)\| \leq L_3 \|x^\star - x_k\|. \tag{41}$$

Since we have computed the closed form of the Hessian of augmented Lagrange function $\nabla_x^2\mathcal{L}(x_k) = \nabla_x^2 f(x_k)^T + \rho A^T A + \rho G^T G$, by assumption (10) and Cauchy-Schwartz inequalities, we have

$$\begin{aligned}\|\Delta_2\| &= \|\nabla_x^2\mathcal{L}^{-1}(x^\star) - \nabla_x^2\mathcal{L}^{-1}(x_k)\| \\ &= \|\nabla_x^2\mathcal{L}^{-1}(x_k)\big(\nabla_x^2\mathcal{L}(x_k) - \nabla_x^2\mathcal{L}(x^\star)\big)\nabla_x^2\mathcal{L}^{-1}(x^\star)\| \\ &\leq \|\nabla_x^2\mathcal{L}^{-1}(x_k)\| \cdot \|\nabla_x^2\mathcal{L}(x_k) - \nabla_x^2\mathcal{L}(x^\star)\| \cdot \|\nabla_x^2\mathcal{L}^{-1}(x^\star)\| \\ &\leq \frac{1}{\mu_2^2}\|\nabla_x^2 f(x^\star) - \nabla_x^2 f(x_k)\|.\end{aligned} \tag{42}$$

Therefore the difference of Jacobians is

$$\begin{aligned}\Big\|\frac{\partial x_k}{\partial\theta} - \frac{\partial x^\star}{\partial\theta}\Big\| &= \|\nabla_x^2\mathcal{L}^{-1}(x_k)\cdot\Delta_1 + \Delta_2\cdot\nabla_{x,\theta}\mathcal{L}(x^\star)\| \\ &\leq \|\nabla_x^2\mathcal{L}^{-1}(x_k)\|\cdot\|\Delta_1\| + \frac{1}{\mu_2^2}\|\nabla_{x,\theta}\mathcal{L}(x)\|\cdot\|\nabla_x^2 f(x^\star) - \nabla_x^2 f(x_k)\| \\ &\leq \Big(\frac{L_3}{\mu_2} + \frac{\mu_3 L_2}{\mu_2^2}\Big)\|x^\star - x_k\|.\end{aligned} \tag{43}$$

□

E ERROR OF THE GRADIENT OF LOSS FUNCTION

Corollary 4.4. *(Error of the inexact gradient in terms of loss function) Followed by Theorem 4.3, the error of the gradient w.r.t. θ in loss function $\mathcal{R}$ is bounded as follows:*

$$\|\nabla\mathcal{R}(\theta; x_k) - \nabla\mathcal{R}(\theta; x^\star)\| \leq C_2\|x_k - x^\star\|, \tag{44}$$

where $C_2 = L_1 + \dfrac{\mu_3 L_1 + \mu_1 L_3}{\mu_2} + \dfrac{\mu_1\mu_3 L_2}{\mu_2^2}$ is a constant.

Proof. Firstly, optimization layer can be reformulated as the following bilevel optimization problems Ghadimi & Wang (2018):

$$\begin{aligned}\min_\theta \quad & \mathcal{R}(\theta; x^\star(\theta)) \\ s.t. \quad & x^\star(\theta) = \arg\min_{x\in\mathcal{C}(\theta)} f(x;\theta),\end{aligned} \tag{45}$$

where $\mathcal{R}$ denotes the loss function, therefore we can derive the following conclusions by Implicit Function Theorem. In *outer optimization problems*, we have

$$\nabla\mathcal{R}(\theta;\tilde{x}) = \nabla_\theta\mathcal{R}(\theta;\tilde{x}) + \nabla_x\mathcal{R}(\theta;\tilde{x})\frac{\partial\tilde{x}}{\partial\theta}. \tag{46}$$

Therefore,

$$\begin{aligned}\Delta &= \nabla\mathcal{R}(\theta; x_k) - \nabla\mathcal{R}(\theta; x^\star) \\ &= \underbrace{\Big(\nabla_\theta\mathcal{R}(\theta; x_k) - \nabla_\theta\mathcal{R}(\theta; x^\star)\Big)}_{\Delta_3} + \underbrace{\Big(\nabla_x\mathcal{R}(\theta; x_k)\frac{\partial x_k}{\partial\theta} - \nabla_x\mathcal{R}(\theta; x^\star)\frac{\partial x^\star}{\partial\theta}\Big)}_{\Delta_4}.\end{aligned} \tag{47}$$

The second term is:

$$\begin{aligned}\Delta_4 &= \nabla_x\mathcal{R}(\theta; x_k)\frac{\partial x_k}{\partial\theta} - \nabla_x\mathcal{R}(\theta; x^\star)\frac{\partial x^\star}{\partial\theta} \\ &= \nabla_x\mathcal{R}(\theta; x_k)\frac{\partial x_k}{\partial\theta} - \nabla_x\mathcal{R}(\theta; x_k)\frac{\partial x^\star}{\partial\theta} \\ &\quad + \nabla_x\mathcal{R}(\theta; x_k)\frac{\partial x^\star}{\partial\theta} - \nabla_x\mathcal{R}(\theta; x^\star)\frac{\partial x^\star}{\partial\theta} \\ &= \underbrace{\Big(\nabla_x\mathcal{R}(\theta; x_k) - \nabla_x\mathcal{R}(\theta; x^\star)\Big)}_{\Delta_5}\frac{\partial x^\star}{\partial\theta} + \nabla_x\mathcal{R}(\theta; x_k)\Big(\frac{\partial x_k}{\partial\theta} - \frac{\partial x^\star}{\partial\theta}\Big).\end{aligned} \tag{48}$$

Combining the results in (48), assumption (9a) and Theorem 4.3, we can obtain that the differences of truncated gradient with optimal gradients is

$$\begin{aligned}\|\Delta\| &= \|\nabla\mathcal{R}(\theta;x_k) - \nabla\mathcal{R}(\theta;x^\star)\| \\ &\leq \|\Delta_3\| + \|\Delta_5\| \cdot \left\|\frac{\partial x^\star}{\partial\theta}\right\| + \|\nabla_x\mathcal{R}(\theta;x_k)\| \cdot \left\|\frac{\partial x_k}{\partial\theta} - \frac{\partial x^\star}{\partial\theta}\right\| \\ &\leq L_1\Big(1+\frac{\mu_3}{\mu_2}\Big)\|x_k - x^\star\| + \mu_1\Big(\frac{L_3}{\mu_2} + \frac{\mu_3 L_2}{\mu_2^2}\Big)\|x_k - x^\star\| \\ &= \Big(L_1 + \frac{\mu_3 L_1 + \mu_1 L_3}{\mu_2} + \frac{\mu_1\mu_3 L_2}{\mu_2^2}\Big)\|x_k - x^\star\|.\end{aligned} \tag{49}$$

Therefore, the gradient of loss function $\mathcal{R}(\theta;x_k)$ shares the same order of error as the truncated solution x_k. □

F Experimental Details

In this section, we give more simulation results as well as some implementation details. Our source code for these experiments is available at https://anonymous.4open.science/r/Alt-Diff-CECB/.

F.1 Additional Numerical Experiments

In constrained Sparsemax layer, "lsqr" mode of CvxpyLayer is applied for fair comparison. The total running time of CvxpyLayer includes the canonicalization, forward and backward pass, retrieval and initialization. The comparison results are provided in Table 4. Alt-Diff can obtain the competitive results as OptNet and CvxpyLayer with lower executing time. The tolerances of the gradient are all set as 10^{-3} for OptNet, CvxpyLayer and Alt-Diff. All the numerical experiments were executed 5 times with the average times reported as the results in Table 4.

Table 4: Comparison of running time (s) and cosine distances of gradients in constrained Sparsemax layers with tolerance $\epsilon = 10^{-3}$.

	small	medium	large	large
Num of variable n	3000	5000	10000	20000
Num of constraints n_c	6000	10000	20000	40000
OptNet	18.79	73.11	839.09	-
CvxpyLayer (total)	3.61	18.14	123.79	1024.69
Initialization	3.50	17.79	122.61	1020.41
Canonicalization	0.00	0.00	0.01	0.02
Forward	0.07	0.27	0.44	1.35
Backward	0.03	0.07	0.73	2.91
Alt-Diff (total)	**3.16**	**9.96**	**58.51**	**539.19**
Inversion	1.21	4.71	34.35	287.83
Forward and backward	1.95	5.25	24.15	251.36
Cos Dist.	0.999	0.998	0.997	0.998

"-" represents that the solver cannot generate the gradients.

From Table 4, we can find that OptNet is much slower than CvxpyLayer in solving sparse Quadratic programmings. CvxpyLayer requires a lot of time in the initialization procedure. But once the initialization procedure completed, calling CvxpyLayer is very fast. However, Alt-Diff only needs to compute the inverse matrix once, significantly reducing the computational speed.

As a special case for general convex objective functions, we adopted $f(x) = -y^T x + \sum_{i=1}^n x_i \log(x_i)$, which is not a quadratic programming problem. Thus we can only compare Alt-Diff with CvxpyLayer. The constraints $Ax = b$ and $Gx \leq h$ are randomly generated with dense coefficients. In Alt-Diff, each primal update x_{k+1} in (5a) is carried out by Newton's methods with the tolerance set as 10^{-4}. The comparison of cosine distance and running time is in Table 5. We can find that Alt-Diff significantly outperforms in all these cases especially for optimization problems with large size.

Table 5: Comparison of running time (s) and cosine distances of gradients in constrained Softmax layers with tolerance $\epsilon = 10^{-3}$. Since OptNet only works for quadratic optimization problems, thus we cannot compare Alt-Diff with it for the constrained Softmax layers.

	tiny	small	medium	large
Num of ineq. n	1000	3000	5000	10000
Num of ineq. m	300	1000	2000	3000
Num of eq. p	100	500	1000	1000
Num of elements	1.96×10^6	2.03×10^7	6.4×10^7	1.96×10^8
CvxpyLayer (total)	5.90	121.63	1328.99	-
Initialization	0.26	3.05	9.29	-
Canonicalization	0.04	0.50	1.82	-
Forward	1.89	18.37	164.30	-
Backward	3.70	99.71	1153.58	-
Alt-Diff (total)	**4.15**	**14.60**	**92.20**	**438.31**
Cosine Dist.	0.994	0.996	0.999	-

"-" represents that the solver cannot generate the gradients.

F.2 IMAGE CLASSIFICATION

All neural networks are implemented using PyTorch with Adam optimizer Kingma & Ba (2014). We replaced a layer as optimization layer in the neural networks, and compared the running time and test accuracy in MNIST dataset using OptNet and Alt-Diff. The batch size is set as 64 and the learning rate is set as 10^{-3}. We ran 30 epoches and provided the running time and test accuracy in Tabel 6. As we have shown that OptNet runs much faster than CvxpyLayer in dense quadratic layers, therefore we only compare Alt-Diff with OptNet.

We used two convolutional layers followed by ReLU activation functions and max pooling for feature extraction. After that, we added two fully connected layers with 200 and 10 neurons, both using the ReLU activation function. Among them, an optimization layer with an input of 200 dimensions and an output of 200 dimensions is added. We set the dimension of the inequality and equality constraints both as 50. Following similar settings in OptNet, we used the quadratic objective function $f(x) = \frac{1}{2}x^T Px + q^T x$ here, while taking q as the input of the optimization layer and the optimal $x^\star$ as the output. Finally, the 10 neurons obtained by the fully connected layer were input into Softmax layer, and the nonnegative log likelihood was used as the loss function here.

Table 6: Comparison of OptNet and Alt-Diff on MNIST with tolerance $\epsilon = 10^{-3}$.

models	Test accuracy (%)	Time per epoch (s)
OptNet	98.95 ± 0.25	307.16 ± 18.21
Alt-Diff	98.97 ± 0.17	167.66 ± 4.40

From the above table, we can see Alt-Diff runs much faster and yields similar accuracy compared to OptNet. We also show the training and testing performance of Alt-Diff and OptNet on MNIST in Figure 4. Obviously, Alt-Diff have similar performance under different tolerance values, further verifying the truncated capability of Alt-Diff as claimed in Corollary 4.4.

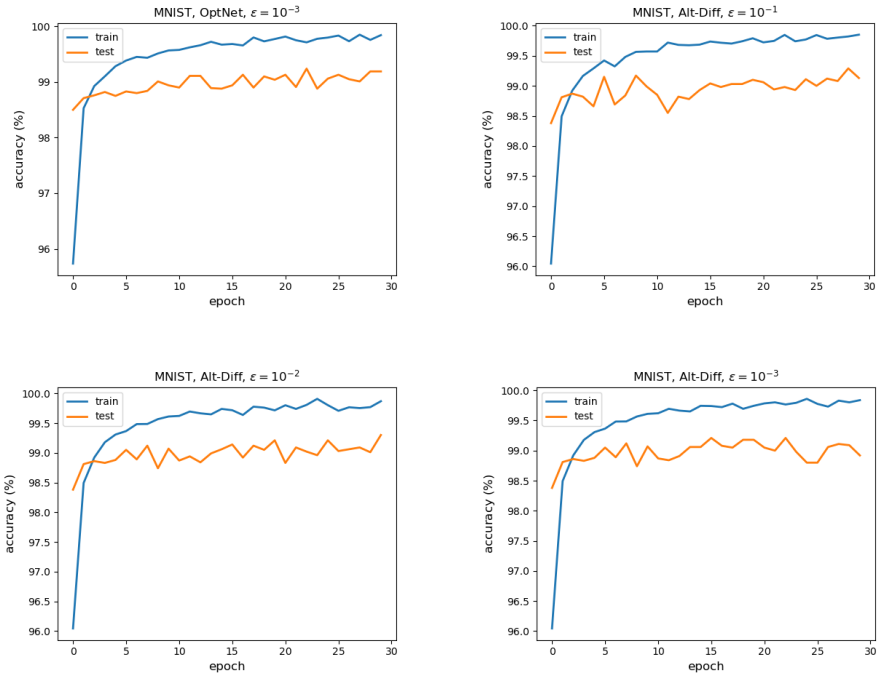


Figure 4: The training and testing performance of Alt-Diff and OptNet on MNIST.