DistProp: A Scalable Approach to Lagrangian Training via Distributional Approximation

Anonymous authors
Paper under double-blind review

Abstract

We develop a multiple shooting method for learning in deep neural networks based on the Lagrangian perspective on automatic differentiation. Our method leverages ideas from saddle-point optimization to derive stable first-order updates to solve a specific constrained optimization problem. Most importantly, we propose a novel solution allowing us to run our algorithm over mini-batches with stochastic gradient fashion and to decouple the number of auxiliary variables with the size of the dataset. We show empirically that our method reliably achieves higher accuracy than other comparable local (biologically plausible) learning methods on MNIST, CIFAR10 and ImageNet.

1 Introduction

Neural network training is inherently sequential: the network outputs first, then the error signal backward layer-by-layer (Rumelhart et al., 1986). The problem with this approach is twofold: it can lead to unstable gradient computation (Hochreiter & Schmidhuber, 1997; Bengio et al., 1992) over long horizons while precluding layer-local parallelism (Crick, 1989). While a rich body of work tackling those issues can be found under the heading of biologically plausible backpropagation (Lee et al., 2015; Carreira-Perpiñán & Wang, 2014; Choromanska et al., 2019), those methods have been only been shown to work on small datasets.

In this paper, we approach this problem from a different angle: that of numerical methods for optimal control. As noted by LeCun et al. (1988), this is a natural connection to make since backprop has in fact been living a double life as the *adjoint equation* in the optimal control literature (Bryson, 1996). More specifically, we propose to decouple training within a neural network (or *differentiable program* more generally) through *multiple shooting* (Bock & Plitt, 1984): the idea of subdividing the optimization problem across several intervals whose solutions are then *glued back* as equality constraints to form a global solution. The resulting approach brings us into the rarely explored (Donti et al., 2021; Taylor et al., 2016) territory of constrained optimization for deep learning.

While the constrained perspective at the core of our multiple shooting approach has been leveraged in Carreira-Perpiñán & Wang (2014); Taylor et al. (2016); Gotmare et al. (2018) for the same purposes, the only types of solution methods explored so far have been unstable first-order Lagrangian-type methods (Bertsekas, 2016, Section4.4) or unconstrained approximations (Choromanska et al., 2019). A core contribution of this paper is to show that advances in GAN (Goodfellow et al., 2020) training via the body of work on game-theoretic optimization (Korpelevich, 1976; Gidel et al., 2019; Mertikopoulos et al., 2019) can be used to design stable first-order constrained solvers. Our experiments show that our new solver is stable across a wide range of hyper-parameters, robust under noise (stochastic optimization regime), and scalable to large datasets.

We also address a second important, but more subtle issue: the fact that a direct application of the Lagrangian perspective necessarily hides a direct dependency on the size of the dataset. This issue, rarely discussed in the literature, rules out the possibility of using mini-batch methods in a well-defined way other than in the full batch regime. We pinpoint the exact mathematical origins of this problem in section 3. We then propose to alleviate this problem by means of function approximation over the output variables. The resulting algorithm melds ideas from Target Propagation (TP) (Le Cun, 1986) and constrained optimization to control both the number and desired accuracy of the output variables.

2 BACKGROUND

A deep feed-forward neural network (Goodfellow et al., 2016) can be conceptualized as a composition of non-linear functions of the form:

$$f(\boldsymbol{x}; \boldsymbol{w}) \triangleq (f^{(L)} \circ \ldots \circ f^{(1)})(\boldsymbol{x}; \boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(L)}) \triangleq f^{(L)}(\ldots f^{(1)}(\boldsymbol{x}; \boldsymbol{w}^{(1)}) \ldots ; \boldsymbol{w}^{(L)}) \ ,$$

where x represents an "input", f a "layer", and $\{w_i\}_i^L$ are tunable "weights" and "biases". Furthermore, the learning problem is typically that of maximizing an objective of the form $J(\boldsymbol{w}) \triangleq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim P}\left[l(f(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y}))\right]$ where l is a "loss function" and $\boldsymbol{x}$ and $\boldsymbol{y}$ are random variables drawn from a joint distribution P. Under the empirical risk minimization (ERM) framework (Bottou & Vapnik, 1992), we can only optimize this objective via a sample average, leading us to the following unconstrained optimization problem:

$$\text{minimize} \ \ \hat{J}(\boldsymbol{w}) \triangleq \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|} \ell(f(\boldsymbol{x}^{(i)}; \boldsymbol{w}), \boldsymbol{y}^{(i)}), \ \ \text{given} \ \ (\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) \in \mathcal{D} \ \ ,$$

where $\mathcal{D}$ is a dataset of examples drawn i.i.d. from the joint. Rather than solving the above problem in the unconstrained form, our work instead consider the following equivalent constrained formulation:

$$\text{minimize} \ \ \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|} \ell(\boldsymbol{h}_L^{(i)}, \boldsymbol{y}^{(i)}) \ \ \text{where} \ \ \boldsymbol{h}_l^{(l)} = f_k(\boldsymbol{h}_{l-1}^{(i)}; \boldsymbol{w}) \ \ \text{given} \ \ \boldsymbol{h}_0^{(i)} = \boldsymbol{x}_i, (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D} \ . \quad (1)$$

In passing from the unconstrained version to the constrained one, we have introduced new optimization variables $\boldsymbol{h}_l^{(i)}, l \in (1, \ldots, L)$ for each layer. In essence, we have *lifted* our problem to one which now has parameters $\{\boldsymbol{w}\} \cup \{\boldsymbol{h}_l^{(i)}\}_{(i,l)\in\{1,\ldots,|\mathcal{D}|\}\times\{1,\ldots,L\}}$: that is $|\mathcal{D}| \times L$ more variables than the original problem. But in doing so, we have also gained two important properties: 1) the ability to compute network outputs *implicitly* and in parallel without a separate forward propagation phase 2) increased training stability over a long optimization horizon by "pinning down" the unstable modes (Ascher et al., 1995; Biegler, 1984).

2.1 SOLVING CONSTRAINED OPTIMIZATION PROBLEMS

The above constrained perspective calls for its dedicated optimization methods. Consider for a moment a generic equality-constrained problem of the form: minimize $f(\boldsymbol{x})$ subject to $g(\boldsymbol{x}) = 0$. By the first-order optimality condition (Bertsekas, 2016, 4.1.1) if $\boldsymbol{x}^\star$ is a regular local minimum of f such that $g(\boldsymbol{x}^\star) = \mathbf{0}$, then there must exist a unique Lagrange multiplier vector $\boldsymbol{\lambda}^\star$ such that $Df(\boldsymbol{x}^\star) + {\boldsymbol{\lambda}^\star}^\top Dg(\boldsymbol{x}^\star) = \mathbf{0}$. In this form, we can therefore view the problem of solving a constrained optimization program as a root-finding problem. In fact, the application of Newton's method to the root-finding problem $DL(\boldsymbol{x}, \boldsymbol{\lambda}) = \mathbf{0}$ where $L(\boldsymbol{x}, \boldsymbol{\lambda}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top g(\boldsymbol{x})$ can be seen Nocedal & Wright (1999) as an instance of the Sequential Quadratic Programming method. While SQP has been widely adopted in the industry Biegler (1984), its reliance on second-order information combined with the need to solve for inverse Hessian is problematic in the deep learning context where the number of variables is many orders of magnitudes larger than what state-of-the-art solvers are capable of handling. There are two main reasons for this: 1) the fact that many industrial solvers have taken a CPU-centric approach leveraging sparse operations (which are of little use in the dense compute model of GPUs) 2) that they often rely on finite-difference derivative estimation methods where Jacobians are represented explicitly (in comparison with the matrix-free mindset in deep learning).

These challenges bring about the need to consider a different class of approaches for constrained optimization. Rather than starting from the above Karush–Kuhn–Tucker (KKT) condition, we note that a necessary and sufficient condition for $\boldsymbol{x}^\star$ to be a minimum of $f(\boldsymbol{x})$ subject to $g(\boldsymbol{x}) = \mathbf{0}$ is that there exists a Lagrange multiplier vector $\boldsymbol{\lambda}^\star$ which is a saddle point of the Lagrangian (Zangwill, 1969), ie: $L(\boldsymbol{x}^\star, \boldsymbol{\lambda}) \leq L(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star) \leq L(\boldsymbol{x}, \boldsymbol{\lambda}^\star)$ for all $\boldsymbol{x}$ and $\boldsymbol{\lambda}$. This leads us to a game-theoretic formulation of our constrained problem as: $\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top g(\boldsymbol{x})$. As a first attempt, we could try to solve this problem by gradient-descent-ascent (GDA) via the Arrow-Hurwicz-Uzawa (Uzawa, 1958) algorithm:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \alpha D_x L(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}) \quad \text{and} \quad \boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \alpha D_\lambda L(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}) \ .$$

It can be shown that (Pschenichny & Danilin, 1975; Bertsekas, 2016) if $D_x^2 L(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)})$ is positive definite, then $[D_x L(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}), g(\boldsymbol{x}^{(k)}]$ must be a descent direction of the penalty function $p(\boldsymbol{x}, \boldsymbol{\lambda}) \triangleq \frac{1}{2}\|D_x L(\boldsymbol{x}, \boldsymbol{\lambda})\|^2 + \frac{1}{2}\|g(\boldsymbol{x})\|^2$. However, the positive definiteness of $D_x^2 L$ only holds when sufficiently close to $(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star)$ an may otherwise lead to non-descent directions and unstable behaviour. Drawing the analogy to GANs, we could also consider a regularized version of the Arrow-Hurwicz-Uzawa algorithm to stabilize training. This idea would then lead us to the class of Augmented Lagrangian methods which try to endow the problem with a "locally convex" structure (Bertsekas, 2016) by using an augmented Lagrangian $L_c(\boldsymbol{x}, \boldsymbol{\lambda}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top g(\boldsymbol{x}) + (c/2)\|g(\boldsymbol{x})\|^2$, where c should be chosen sufficiently small to ensure that $D_x^2 L_c$ is positive definite.

In order the avoid the shortcomings of GDA-type methods, our work uses recent findings by Gidel et al. (2019) showing that the Extragradient method of Korpelevich (1976) can be adapted to provide stable learning algorithms for GANs. In the context of constrained optimization and the saddle-point perspective, the ExtraGradient method is of the form:

$$\begin{aligned} \boldsymbol{x}^{(k+\frac{1}{2})} &= \boldsymbol{x}^{(k)} - \alpha D_x L(\boldsymbol{x}^{(k)}, \boldsymbol{\lambda}^{(k)}), \;\; \boldsymbol{\lambda}^{(k+\frac{1}{2})} = \boldsymbol{\lambda}^{(k)} + \alpha g(\boldsymbol{x}^{(k)}) \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} - \alpha D_x L(\boldsymbol{x}^{(k+\frac{1}{2})}, \boldsymbol{\lambda}^{(k+\frac{1}{2})}), \;\; \boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \alpha g(\boldsymbol{x}^{(k+\frac{1}{2})}) \;. \end{aligned} \tag{2}$$

As shown in our experiment, the combination of this algorithm the ExtraAdagrad step size adaptation strategy similar to Gidel et al. (2019) leads to increased stability and performance over the Arrow-Hurwicz-Uzawa/GDA and Augmented Lagrangian alternatives. We refer to this application of extragradient to the equality-constrained formulation or neural network training as ExtraProp.

3 Breaking the Dependence on the Dataset Size

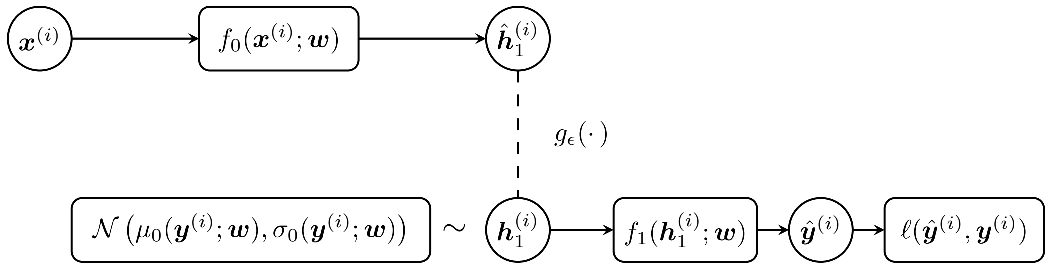

$\boldsymbol{x}^{(i)}$
$f_0(\boldsymbol{x}^{(i)}; \boldsymbol{w})$
$\hat{\boldsymbol{h}}_1^{(i)}$
$g_\epsilon(\cdot)$
$\mathcal{N}\left(\mu_0(\boldsymbol{y}^{(i)}; \boldsymbol{w}), \sigma_0(\boldsymbol{y}^{(i)}; \boldsymbol{w})\right)$
$\sim$
$\boldsymbol{h}_1^{(i)}$
$f_1(\boldsymbol{h}_1^{(i)}; \boldsymbol{w})$
$\hat{\boldsymbol{y}}^{(i)}$
$\ell(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)})$

Figure 1: Flow of computation in a two-layer network. The constraint function g_ϵ ensures that $\hat{\boldsymbol{h}}_1^{(i)}$, is within a designed number of standard deviations from the mean of a learned Gaussian distribution assigned to the class $\boldsymbol{y}^{(i)}$. Subsequent layers propagate samples from this distribution rather than the transformed examples directly

The fact that the constrained formulation in equation 1 has a direct dependency on the number of examples in the dataset is easy to overlook. This stems from the fact that the formulation in equation 1 is inherently deterministic: the *stochastic counterpart* or Sample Average Approximation (SAA) Rubinstein (1981) arising from the empirical risk minimization principle. Hence, the dataset $\mathcal{D}$ is *bound* to the equality constrained program itself via the auxiliary variables $\{h_1, \dots, h_L\}$. More precisely, given a dataset $\mathcal{D}$ we need $|\mathcal{D}| \cdot L \cdot H$ constraints, for a network of L layers and hidden activations of uniform size H. This problem is further exacerbated by the need to also maintain Lagrange multipliers when solving programs using primal-dual methods, which doubles this number.

The solution that is put forward in this paper consists in decoupling the dataset itself with the auxiliary variables via function approximation. That is, instead of maintaining a unique auxiliary variable and Lagrange multiplier pair per example, our algorithm tries to learn a mapping between a data point and a corresponding learned *prototype*. However, in order to maintain diversity and avoid potential *collapses* in the data representation, we add Gaussian noise around that prototype (the mean) with a certain learned standard deviation. Hence, instead of maintaining $|\mathcal{D}| \times L \times H$ variables, we are capable of reducing this number down to $N \times L \times H$ where N is the desired number of synthetic samples drawn from the distributions centred at each of the M prototypes where M is the number of classes. In the context of ImageNet (Deng et al., 2009) for example where $|\mathcal{D}| = 14M$, we now only have to learn $M = 1000$ prototypes, and found that $N < 8$ and often $N = 1$ is sufficient to achieve good performance.

More precisely, the training procedure goes as follows. For each input $\boldsymbol{x}^{(i)}$ from the original dataset, we compute the corresponding output through the first layer (or any other chosen split) f_0. Then, rather than requiring that $\boldsymbol{h}_0^{(i)} - f_0(\boldsymbol{x}^{(i)}; \boldsymbol{w}) = \mathbf{0}$ as in the usual multiple-shooting formulation, we require instead that the z-score $\left(f_0(\boldsymbol{x}^{(i)}; \boldsymbol{w}) - \mu(\boldsymbol{y}^{(i)}; \boldsymbol{w}_\mu^{(0)})\right) / \sigma(\boldsymbol{y}^{(i)}; \boldsymbol{w}_\sigma^{(0)})$ of that output with respect to the corresponding learned Gaussian must be below a certain threshold. Here, μ and σ are simple shallow models of the form $\mu(\boldsymbol{y}; \boldsymbol{w}_\mu) \triangleq \boldsymbol{w}_\mu^\top \boldsymbol{y}$ and $\sigma(\boldsymbol{y}; \boldsymbol{w}_\sigma) \triangleq \log(1 + \exp(\boldsymbol{w}_\sigma^\top \boldsymbol{y}))$ assigning a given class to a corresponding mean and variance for the hidden states. Note that despite enforcing the z-score criterion via $|\mathcal{D}| \times H$ equality constraints, we are not introducing as many intermediary variables as the size of the dataset. Given those functions μ and σ, we are then free to generate as many samples as allowed by our computational budget: this is the key property allowing a decoupling of the subsequent layers.

In a classical multiple-shooting formulation, we would then require that the the intermediary variable at the second layer $\boldsymbol{h}_1^{(i)}$ satisfies $\boldsymbol{h}_1^{(i)} - f_1(\boldsymbol{h}_0^{(i)}; \boldsymbol{w}) = \mathbf{0}$. However, since the very goal of our approach is to avoid maintaining this variable $\boldsymbol{h}_1^{(i)}$ explicitly for each original datapoint, we use again our z-score criterion, but this time evaluated over a smaller set of synthetic hidden states generated by drawing N samples from $\tilde{\boldsymbol{h}}_0^{(j)} \sim \mathcal{N}(\mu(j; \boldsymbol{w}_\mu^{(0)}), \sigma(j; \boldsymbol{w}_\sigma^{(0)}))$ for each class $j = 1, \ldots, M$. These $N \times M$ additional constraints are of then of the form $(f(\tilde{\boldsymbol{h}}_0^{(j)}; \boldsymbol{w}_0) - \mu(i; \boldsymbol{w}_\mu^{(1)}))/\sigma(i; \boldsymbol{w}_\sigma^{(1)})$. The same procedure (summarized in algorithm 1) is then repeated for the subsequent layers.

3.1 Algorithmic Considerations

For the procedure outlined above to be a well-defined mathematical program as well as for it to yield well-behaved solutions, additional algorithmic considerations need to be taken into account.

Derandomization via Reparameterization The procedure outlined above is incompatible with the *static* and deterministic structure of a usual equality constraint problem by the very fact that it involves sampling synthetic activations at every layer. While a chance-constrained (Hof et al., 1996) formulation may account for our computational model, we choose instead to generate a fixed snapshot of the layer-specific datasets and account for the stochastic nature of the process via reparametrization (Rezende et al., 2014). That is, we pre-generate a vector of Normal variates $\boldsymbol{\epsilon} \sim \mathcal{N}(0, 1)$ at initialization time and later, deterministically generate the hidden states as $\boldsymbol{h}_l^{(i)} = \mu(\boldsymbol{y}^{(i)}; \boldsymbol{w}_\mu^{(l)}) + \boldsymbol{\epsilon} \odot \sigma(\boldsymbol{y}^{(i)}; \boldsymbol{w}_\sigma^{(l)})$. In that sense, given the original dataset $\mathcal{D}$ and generated normal variates, the corresponding constrained program is fully determined and remains unchanged through optimization.

Adaptive Step Sizes In order to further account for the stochasticity coming from sampling mini-batches, we use the adaptive step size method, AdaGrad, adapted from Gidel et al. (2019) which itself is an adaptation of Adam to the ExtraGradient case. This modification is crucial to ensure robustness of ExtraGradient when passing to the stochastic and high dimensional setting, ExtraAdagrad applies the AdaGrad (Duchi et al., 2011) algorithm to both extrapolation and optimization steps as if both were usual classical optimization iterations.

To have a more compact notation let us define a general constraint $g_l^{(i)}(\boldsymbol{h}_l^{(i)}, \boldsymbol{w}) = \begin{cases} \boldsymbol{h}_l^{(i)} - f_l(\boldsymbol{h}_l^{(i)}; \boldsymbol{w}), & \text{otherwise} \\ \boldsymbol{h}_l^{(i)} - f_l(\boldsymbol{x}^{(i)}; \boldsymbol{w}), & \text{l=0} \end{cases}$, where the only difference is the lack of hidden state for the first constraint.

ϵ-constraints In practice, with very deep neural networks, noisy samples and imperfect representation, enforcing constraints to be exactly zero induces instability in the learning procedure, one solution is to change the constraints to be insensitive in a range $(-\epsilon, \epsilon)$, this can be done by replac-

Algorithm 1 Constrained learning with dataset decoupling over two "layers" or blocks

```
initialize $\boldsymbol{w}_0$
initialize $\boldsymbol{\epsilon}_k \sim \mathcal{N}(0,1)\,\forall k \in (0,\dots,K)$
for $k \in (0,...,K)$ do
    sample $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) \in D$ uniformly at random.
    $g_0^{(i)} \leftarrow \frac{\mu(\boldsymbol{y}^{(i)};\boldsymbol{w}_k) - f_0(\boldsymbol{x}^{(i)};\boldsymbol{w})}{\sigma(\boldsymbol{y}^{(i)};\boldsymbol{w}_k)}$
    $h_0^{(i)} \leftarrow \mu(\boldsymbol{y}^{(i)};\boldsymbol{w}_k) + \boldsymbol{\epsilon}_k \sigma(\boldsymbol{y}^{(i)};\boldsymbol{w}_k)$    ▷ dataset Projection; $h_0^{(i)}$ is a synthetic hidden state
    $\hat{\boldsymbol{y}}^{(i)} \leftarrow f_1(h_0^{(i)}, \boldsymbol{w}_k)$
    $L_i(\boldsymbol{w},\boldsymbol{\lambda}) \leftarrow \ell_i(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) + \boldsymbol{\lambda}_{\boldsymbol{y}^{(i)}} g^{(i)}$
    $\begin{bmatrix}\boldsymbol{w}_{k+\frac{1}{2}}\\ \boldsymbol{\lambda}_{k+\frac{1}{2}}\end{bmatrix} \leftarrow \begin{bmatrix}\boldsymbol{w}_k\\ \boldsymbol{\lambda}_k\end{bmatrix} - \alpha_k \begin{bmatrix}\nabla_{\boldsymbol{w}} L_i(\boldsymbol{w}_k,\boldsymbol{\lambda}_k)\\ -\nabla_{\boldsymbol{\lambda}} L_i(\boldsymbol{w}_k,\boldsymbol{\lambda}_k)\end{bmatrix}$
    $\begin{bmatrix}\boldsymbol{w}_{k+1}\\ \boldsymbol{\lambda}_{k+1}\end{bmatrix} \leftarrow \begin{bmatrix}\boldsymbol{w}_k\\ \boldsymbol{\lambda}_k\end{bmatrix} - \alpha_{k+\frac{1}{2}} \begin{bmatrix}\nabla_{\boldsymbol{w}} L_i(\boldsymbol{w}_{k+\frac{1}{2}},\boldsymbol{\lambda}_{k+\frac{1}{2}})\\ -\nabla_{\boldsymbol{\lambda}} L_i(\boldsymbol{w}_{k+\frac{1}{2}},\boldsymbol{\lambda}_{k+\frac{1}{2}})\end{bmatrix}$
end for
```

ing the original constraint by an epsilon-insensitive version:

$$g_\epsilon(\boldsymbol{h}_l^{(i)}, \boldsymbol{\theta}) = \begin{cases} g(\boldsymbol{h}_l^{(i)}, \boldsymbol{\theta}) - \epsilon, & \text{if } g(\boldsymbol{h}_l^{(i)}, \boldsymbol{\theta}) > \epsilon \\ g(\boldsymbol{h}_l^{(i)}, \boldsymbol{\theta}) + \epsilon, & \text{if } g(\boldsymbol{h}_l^{(i)}, \boldsymbol{\theta}) < -\epsilon \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

We transform our constraint in this manner using the hard shrinkage function.

Activation Functions
When using activation functions such as ReLU, we restrict the output domain of a layer, to improve the optimization dynamics one should also include such transformation to the relative state variable to lie in the same domain as the previous function of f_t, for example, if the network output is bound in $\mathbb{R}^n_+$ by non-linearities such as ReLu, the state should also be bound to $\mathbb{R}^n_+$ by re-parametrizing the h_l as $h_l = \text{ReLU}(\hat{h}_l)$ where $\hat{h}_l$ behaves as a free parameter in the same way h_l would.

Initialize to a valid solution
Initializing the intermediate states h_l to, or close to a valid forward pass greatly increases stability by avoiding unnecessary primal and dual updates, in the case of approximation by a distribution, initializing the means to random points on the unit sphere and the standard deviation to one allowing the initial class distributions to be distinguishable.

Sparse gradients
in the exact setting where we have as many constraints as the dataset, using sparse gradients avoids calculating gradients with zero entries for the non-batch values, an important implementation detail in word embedding methods for Natural Language processing tasks for example.

4 Related Works

An important consequence of this decoupling from the first property is also that we can now compute layer outputs independently in parallel given the variables $\boldsymbol{h}_l^{(i)}$. The fact that we can now jointly *forward propagate* (implicitly) and optimize at the same time is an idea known in the optimal control literature under the the so-called *multiple shooting* (Bock & Plitt, 1984) methods, which are said to be *simultaneous* methods (Biegler, 1984). In the jargon of numerical methods for the control of differential equations (Ascher et al., 1995), this is as if we were performing *simulation* (solving an initial value problem) implicitly rather than explicitly via time-stepping; simultaneously rather than sequentially. A second benefit of the multiple shooting formulation is that of stability. We can show precisely (Ascher et al., 1995, Theorem 4.45) that despite the introduction of additional variables, numerical solutions to boundary value problems can be more stable if obtained by multiple shooting with the right number and location of the breakpoints. Intuitively, the fact that we introduce additional variables into our problem allows us to "pindown" the unstable modes of the system at those breakpoints. Translated into deep learning terms, those breakpoints would help us deal with exploding or vanishing gradients (Hochreiter, 1998; Bengio et al., 1994).

Difference Target Propagation (DTP) In Difference Target Propagation (Lee et al., 2015) defining the proper *inverse function* adds additional complexity and requires careful fine-tuning, similarly to DistProp, DTP injects noise in the training procedure, in our case, the generalization bonus is a welcomed side effect while in DTP the magnitude of the injected Gaussian noise is not a learnable parameter but rather it has to be defined by the designer.

Online Alternating Minimization (OAM) Choromanska et al. (2019) and similar methods like Taylor et al. (2016), taking inspiration from the method Alternating Direction Method of Multipliers (ADMM), (Boyd et al., 2011), also splits the optimization problem into smaller and simpler sub-problems similarly to our method, but unlike OAM we solve both the learning and the gluing problems simultaneously rather than sequentially, and can work on mini-batches rather than full-batches.

Direct Feedback Alignment (DFA) Nøkland (2016) does away with the classical gradient signal and replaces it with predetermined random matrices to transport errors backward directly to each layer of the network, although DFA performs well in simple tasks, and it is easily scaled to bigger datasets such as ImageNet, the performance degrades quickly as the tasks complexity increases.

5 Experiments

In order to empirically investigate the advantage of the Extragradient applied to the Lagrangian perspective on neural network training as well as our distributional approximation, we use the network architecture of (LeCun et al., 1999) composed of a variable number of convolutional layers each followed by a pooling layer as *feature extractors* and a variable number of fully-connected layers as *classifier*. We evaluate ExtraProp in MNIST with the AlexNet architecture, whic we split after the convolutional layers; for DistProp, we remove the ReLU non-linearity. For the CIFAR-10, we use a similar network with one extra convolution and one extra linear layer while the network structure for ImageNet is again, AlexNet as described in Krizhevsky (2014). We benchmark the proposed methods against state-of-the-art methods from the literature on biologically plausible backprop (which share the same layer-local ideals) as well as plain stochastic gradient descent (SGD).

For both OAM and DTP, we use a LeNet baseline structure, as reported in their respective papers and official implementations, In the ImageNet experiments, due to the memory requirement, we replaced OAM with Direct Feedback Alignment (DFA) Nøkland (2016), while DFA is not as close to our method as OAM, they are both alternatives to classical backpropagation and DFA can be used on the scale of ImageNet, more details on the differences between our algorithm and the benchmark methods refer to Section 4.

The MNIST model was trained for 6500 iterations, the LeNet-style CIFAR-10 model was trained for 20000 iterations, on ImageNet, the models were trained for 50000 iterations, all models were trained with AdaGrad (Duchi et al., 2011) for both ExtraProp and DistProp and their own optimizer for the baseline algorithms. For the ImageNet task, even just storing the multipliers ($\boldsymbol{\lambda}$) for the constraint ($l = 1$) block is unfeasible, to reduce the memory footprint we then reformulate the constraints making sure the feasible set of solutions is unchanged: $g_\epsilon(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{w}) = \mathrm{ReLu}(||\frac{f_0(\boldsymbol{x}^{(i)};\boldsymbol{w}) - \mu(\boldsymbol{y}^{(i)};\boldsymbol{w})}{\sigma(\boldsymbol{y}^{(i)};\boldsymbol{w})}||_1 - \epsilon)$ so that the new multipliers for the first block are $\boldsymbol{\lambda} \in \mathbb{R}^{|D|}$ rather than $\in \mathbb{R}^{|D| \times H}$.

Impact of Extragradient We also investigate the effectiveness of ExtraGradient as optimizer, in figure 3 GDA (Gradient Descent Ascent) refers to an alternative of ExtraProp where the extrapolation step is avoided, and Regularization refers to Gradient Descent, where the constraint is considered a regularization term based on the defect, we omit the scheduling of the penalty as convergence is never reached. Extrapolation greatly helps to stabilize the optimization dynamics, an analysis and plots of the constraint defect over the optimization process can be found in Fig. 2

Augmented Lagrangian It is often useful to augment the Lagrangian objective to smooth the optimization landscape around the feasible zone, in our experiments this augmentation did not noticeably help, we hypothesize that the dual problem is already smooth enough, this is also reflected in the low sensitivity to constraint related hyperparameters, as shown in section 5.1.

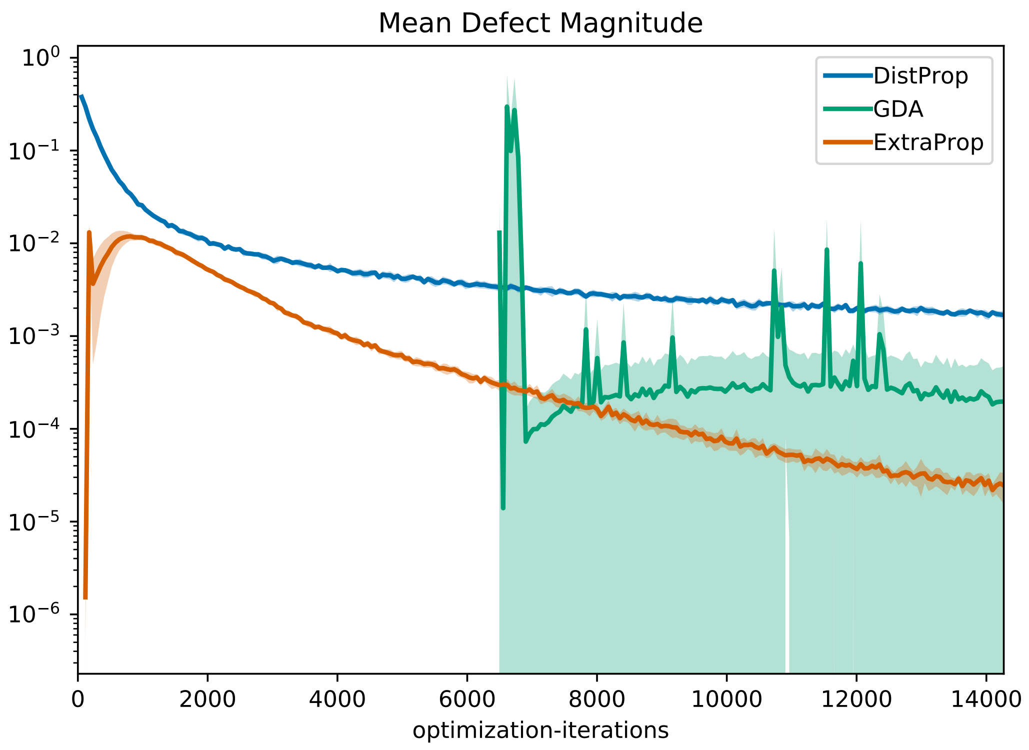


Figure 2: Log-scale mean absolute defect and it's standard deviation across 20 runs vs gradient steps on the MNIST dataset, GDA starts with zero defect but the optimization dynamics are often eventually destabilized stopping any progress in terms of objective function.

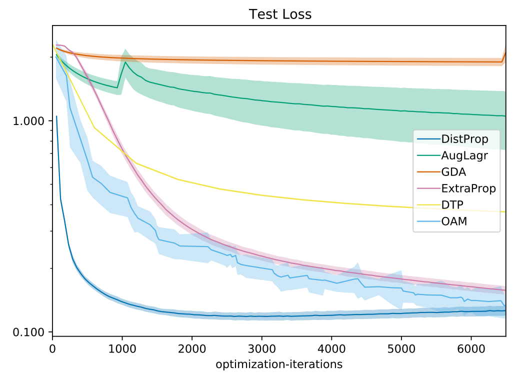
5.1 Robustness to hyper-parameters

We investigate the effect of hyperparameter selection on the optimality of solutions, ideally, we would like to have a wide range of stable hyperparameters configurations. We ran 170 experiments with different parameters, 103 (shown) achieve top 10% performance, the experiments were run using a Bayesian hyperparameter search method uses based on a Gaussian Process [2] with a configuration space as reported in Fig. 4. The results show that the hyperparameters related to the lifting variables in DistProp are robust to a wide range of values.

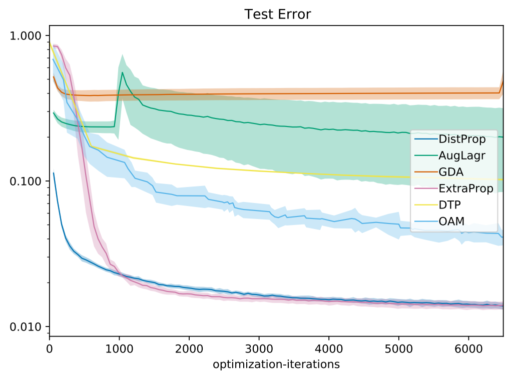
6 Conclusions

We proposed a novel online method to train neural networks, this work expands the target propagation class of algorithms allowing to divide the problem into more manageable sub-problems linked by explicit constraints, we show that the cost of adding auxiliary variables can be removed by reformulating the problem to an approximated form that breaks free of dependence on the dataset, so doing we remove the need to propagate the dataset in its entirety across the network. We show empirical accuracy comparable and superior to standard stochastic gradient methods in simple settings, and improve on state of the art algorithms in target propagation in terms of performance, stability and scalability. Furthermore, we show that given a few small modifications, first-order constrained optimization methods such as the Lagrangian method can be well behaved in modern tasks like ImageNet.

[2]`https://docs.wandb.ai/guides/sweeps/configuration#method`



(a) Log-scale Test Loss



(b) Log-scale Test Error

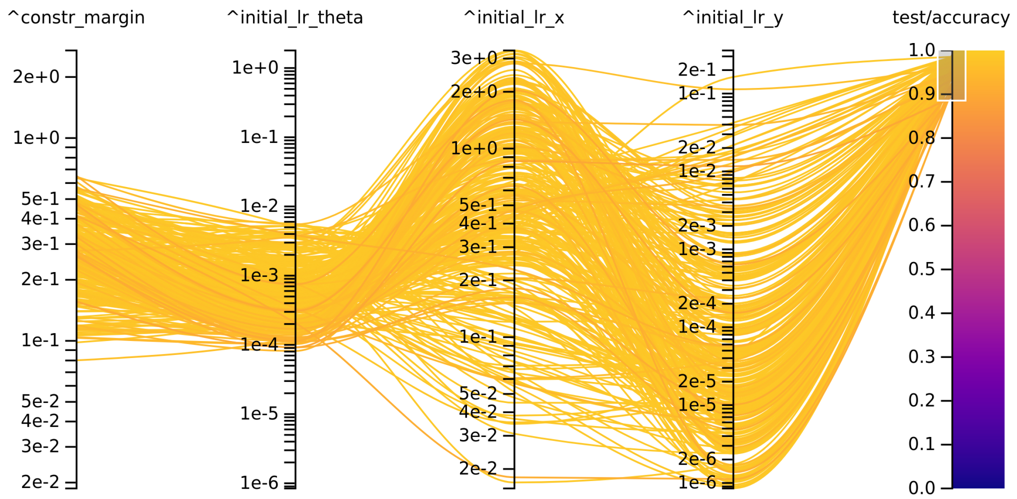
Figure 3: Here we show mean and standard deviation of the of generalization scores across 10 random seeds, OAM and DTP are shown for reference. On MNIST both ExtraGradient based methods (ExtraProp and DistProp) have stable optimization dynamics and good performance, GDA lacks both performance and stability, while a regularization approach suffers from high variance.the DistProp formulation achieves a lower generalization loss, possibly due optimizing a wider support rather than just the training distribution.

Table 1: Experimental results

Dataset	Algorithm	Test Accuracy	Final Defect
Mnist	SGD	0.9881	0
	Regularized	0.9627	0.1
	GDA	0.9743	0.00072
	ExtraProp	0.9827	0.0000045
	DistProp	**0.9903**	0.0026
CIFAR10	SGD	0.6309	0.
	Regularized	0.1726	0.
	GDA	0.1611	0.0008598
	ExtraProp	0.5833	0.5833
	DistProp	**0.6540**	0.07447 [1]
ImageNet TOP-1	DTP	0.0166	-
	DFA	0.0692	-
	DistProp	**0.1110**	0.
ImageNet TOP-5	DTP	0.0544	-
	DFA	0.1746	-
	DistProp	**0.3059**	0.

7 Future Work

Without the dependency on the dataset, and having alleviated the problem of vanishing gradients it would be interesting to understand the limits in terms of network depth and advantage with respect of residual networks (He et al., 2016), initial experiments showed promising results for up to 100 layers but in-dept analysis of such settings is beyond the scope of this work.



Hyperparameter (Column Name)	**Lower Bound**	**Upper Bound**
Constraint Margin, ϵ (constr_margin)	0.1	0.64
Weights Learning rate (initial_lr_theta)	0.00009	0.0056
W_μ, W_σ Learning rate (initial_lr_x)	0.2273	3.282
λ Learning rate (initial_lr_y)	0.0000001	0.0158

Figure 4: Robustness to hyper-parameters, for DistProp on MNIST. Each line represents an experiment with values defined by the point of intersection with the parameter axis, shown is the range of hyper-parameters for which performance does not drop more than 10%, i.e. where test accuracy is above 89%. The table reports a possible interpretation of the ranges for each hyperparameters for which the performance does not degrade more than 10% from the best performer, due to the complex interaction between hyperparameters, these values should be taken as an heuristic at best.

7.1 Reproducibility

An implementation of this paper is available on GitHub[3], it contains all the code necessary to reproduce the methods in this paper, to ensure exact reproducibility over the years of experiments we also tracked the exact code version and hyperparameters[4]

References

Uri M. Ascher, Robert M. M. Mattheij, and Robert D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Society for Industrial and Applied Mathematics, 1995. doi: 10.1137/1.9781611971231. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611971231`.

Y. Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5:157–66, 02 1994. doi: 10.1109/72.279181.

Yoshua Bengio, Renato De Mori, Giovanni Flammia, and Ralf Kompe. Global optimization of a neural network-hidden markov model hybrid. *IEEE transactions on Neural Networks*, 3(2): 252–259, 1992.

Dimitri Bertsekas. *Nonlinear programming*. Athena Scientific, Belmont, Massachusetts, 2016. ISBN 978-1886529052.

[3]`https://anonymous.4open.science/r/constrained_nn-8013/`
[4]`https://github.com/ministry-of-silly-code/experiment_buddy/`

Lorenz T. Biegler. Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation. *Computers & Chemical Engineering*, 8:243–247, 1984.

Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

Léon Bottou and Vladimir Naumovich Vapnik. Local learning algorithms. *Neural Computation*, 4: 888–900, 1992.

Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

A.E. Bryson. Optimal control-1950 to 1985. *IEEE Control Systems Magazine*, 16(3):26–33, 1996. doi: 10.1109/37.506395.

Miguel Carreira-Perpiñán and Weiran Wang. Distributed optimization of deeply nested systems. In *Journal of Machine Learning Research*, volume 33, pp. 10–19, 2014. URL `http://eecs.ucmerced.edu`.

Anna Choromanska, Benjamin Cowen, Sadhana Kumaravel, Ronny Luss, Mattia Rigotti, Irina Rish, Paolo Diachille, Viatcheslav Gurev, Brian Kingsbury, Ravi Tejwani, et al. Beyond backprop: Online alternating minimization with auxiliary variables. In *International Conference on Machine Learning*, pp. 1193–1202. PMLR, 2019.

Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.

John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *J. Mach. Learn. Res.*, 2011.

Gauthier Gidel, Hugo Berard, Pascal Vincent, and Simon Lacoste-Julien. A variational inequality perspective on generative adversarial nets. In *ICLR*, 2019. (to appear).

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11): 139–144, October 2020. ISSN 0001-0782. doi: 10.1145/3422622. URL `https://doi.org/10.1145/3422622`.

Akhilesh Gotmare, Valentin Thomas, Johanni Brea, and Martin Jaggi. Decoupling backpropagation using constrained optimization methods. In *Iclr*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.*, 6:107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

John Hof, Michael Bevers, and James Pickens. Chance-constrained optimization with spatially autocorrelated forest yields. *Forest Science*, 42(1):118–123, 1996.

GM Korpelevich. The extragradient method for finding saddle points and other problems. *Ekonomika i matematicheskie metody*, 12:2010–2010, 1976.

Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *ArXiv*, abs/1404.5997, 2014.

Yann Le Cun. Learning process in an asymmetric threshold network. In E. Bienenstock, F. Fogelman Soulié, and G. Weisbuch (eds.), *Disordered Systems and Biological Organization*, pp. 233–240, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg. ISBN 978-3-642-82657-3.

Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pp. 21–28, 1988.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pp. 319–345. Springer, 1999.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pp. 498–515. Springer, 2015.

Panayotis Mertikopoulos, Bruno Lecouat, Houssam Zenati, Chuan-Sheng Foo, Vijay Chandrasekhar, and Georgios Piliouras. Optimistic mirror descent in saddle-point problems: Going the extra (gradient) mile. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=Bkg8jjC9KQ`.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 1999.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *NIPS*, 2016.

B. N. Pschenichny and Y.M. Danilin. *Numerical Methods in Extremal Problems*. MIR, 1975.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

Reuven Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1981. ISBN 0471089176.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pp. 2722–2731. PMLR, 2016.

H. Uzawa. *Studies in linear and nonlinear programming*. Stanford University Press, 1958.

W.I. Zangwill. *Nonlinear Programming: A Unified Approach*. Prentice-Hall international series in management. Prentice-Hall, 1969. ISBN 9780136235798.