Beyond Backprop: Online Alternating Minimization with Auxiliary Variables

Anna Choromanska* [1] **Benjamin Cowen*** [1] **Sadhana Kumaravel*** [2] **Ronny Luss*** [2] **Mattia Rigotti*** [2]
Irina Rish* [2] **Brian Kingsbury** [2] **Paolo DiAchille** [2] **Viatcheslav Gurev** [2] **Ravi Tejwani** [3] **Djallel Bouneffouf** [2]

Abstract

Despite significant recent advances in deep neural networks, training them remains a challenge due to the highly non-convex nature of the objective function. State-of-the-art methods rely on error backpropagation, which suffers from several well-known issues, such as vanishing and exploding gradients, inability to handle non-differentiable nonlinearities and to parallelize weight-updates across layers, and biological implausibility. These limitations continue to motivate exploration of alternative training algorithms, including several recently proposed auxiliary-variable methods which break the complex nested objective function into local subproblems. However, those techniques are mainly offline (batch), which limits their applicability to extremely large datasets, as well as to online, continual or reinforcement learning. The main contribution of our work is a novel online (stochastic/mini-batch) alternating minimization (AM) approach for training deep neural networks, together with the first theoretical convergence guarantees for AM in stochastic settings and promising empirical results on a variety of architectures and datasets.

1. Introduction

Backpropagation (backprop) (Rumelhart et al., 1986) has been the workhorse of neural net learning for several decades, and its practical effectiveness is demonstrated by recent successes of deep learning in a wide range of applications. Backprop (chain rule differentiation) is used to compute gradients in state-of-the-art learning algorithms such as stochastic gradient descent (SGD) (Robbins & Monro, 1985) and its variations (Duchi et al., 2011; Tieleman & Hinton, 2012; Zeiler, 2012; Kingma & Ba, 2014).

However, backprop has several drawbacks as well, including the commonly known *vanishing gradient* issue, resulting from recursive application of the chain rule through multiple layers of deep and/or recurrent networks (Bengio et al., 1994; Riedmiller & Braun, 1993; Hochreiter & Schmidhuber, 1997; Pascanu et al., 2013; Goodfellow et al., 2016). Although several approaches were proposed to address this issue, including Long Short-Term Memory (Hochreiter & Schmidhuber, 1997), RPROP (Riedmiller & Braun, 1993), and rectified linear units (ReLU) (Nair & Hinton, 2010), the fundamental problem with computing gradients of a deeply nested objective function remains. Moreover, backpropagation does not apply directly to *non-differentiable nonlinearities* and *does not allow parallel weight updates* across the layers (Le et al., 2011; Carreira-Perpiñán & Wang, 2014; Taylor et al., 2016).

Also, besides its computational issues, backprop is often criticized from a neuroscience perspective as a biologically implausible learning mechanism (Lee et al., 2015; Bartunov et al., 2018; Krotov & Hopfield, 2019; Sacramento et al., 2018; Guerguiev et al., 2017), due to multiple factors including the need for "a distinct form of information propagation (error feedback) that *does not influence neural activity*, and hence does not conform to known biological feedback mechanisms underlying neural communication" (Bartunov et al., 2018)[1].

The issues mentioned above continue to motivate research on alternative algorithms for neural net learning. Several approaches were proposed recently, introducing *auxiliary variables* associated with hidden unit activations in order to decompose the highly coupled problem of optimizing a nested loss function into multiple, loosely coupled, simpler subproblems. These include alternating direction method of multipliers (ADMM) (Taylor et al., 2016; Zhang et al., 2016) and alternating-minimization or block coordinate descent (BCD) methods (Carreira-Perpiñán & Wang, 2014; Zhang & Brand, 2017; Zhang & Kleijn, 2017; Askari et al., 2018; Zeng et al., 2018; Lau et al., 2018; Gotmare et al., 2018).

A similar formulation, using Lagrange multipliers, was pro-

*Equal contribution [1] ECE NYU Tandon [2]IBM T.J. Watson Research Center [3]MIT. Correspondence to: Irina Rish <IBM>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

[1]Gradient chain computation yields *non-local* synaptic weight updates which depend on the activity and computations of all downstream neurons, rather than only local signals from adjacent neurons (Whittington & Bogacz, 2019; Krotov & Hopfield, 2019).

posed earlier in (LeCun, 1986; 1987; LeCun et al., 1988), where a constrained formulation involving activations required the output of the previous layer to be equal to the input of the next layer, leading to the *target propagation* algorithm and recent extensions (Lee et al., 2015; Bartunov et al., 2018) (unlike BCD and ADMM, target prop uses layer-wise inverses of the forward mappings). These methods are viewed as somewhat more bio-plausible alternatives to backprop due to explicit propagation of (noisy/nondeterministic) neuronal activity and (layer-)local synaptic updates (see (Bartunov et al., 2018) for details). Note that the above bio-plausibility arguments are equally applicable to auxiliary-variable methods based on explicit optimization of (noisy) neural activations, and breaking the weight update problem into local, layer-wise optimization subproblems.

In this paper, we propose a *novel activation-propagation approach*, which, similarly to prior BCD and ADMM approaches, performs alternating minimization of network weights and auxiliary activation variables. However, unlike those methods, which all assume an offline (batch) setting and require the full training dataset at each iteration, our method is an *online*, incremental learning approach, that performs *stochastic (minibatch) alternating minimization (AM)*. Two variants of AM are proposed, *AM-Adam* and *AM-mem*, which use different approaches for optimizing local subproblems.

Note that, unlike ADMM-based methods (Taylor et al., 2016; Zhang et al., 2016) and some previously proposed BCD methods (Zeng et al., 2018), our approach does not require Lagrange multipliers and only uses one set of auxiliary variables per layer: it is as memory-efficient as standard SGD, which stores activation values for gradient computations. The same distinction, along with multiple others (discussed in Supplementary Material), exists between our method and another recently proposed alternating-minimization scheme, ProxProp (Thomas Frerix, 2018). Also, we assume arbitrary loss functions and nonlinearities (unlike, for example, (Zhang & Brand, 2017) which assumes ReLU nonlinearities), and perform extensive empirical evaluation beyond the fully-connected networks, commonly used to evaluate auxiliary-variable methods.

In summary, our contributions include:

- *algorithm(s):* a novel *online (mini-batch) auxiliary-variable approach* for training neural networks without the gradient chain rule of backprop; unlike prior offline (batch) auxiliary-variable algorithms, our method can scale to arbitrarily large datasets and is applicable in continual and reinforcement learning settings;
- *theory:* to the best of our knowledge, we propose the first general theoretical convergence guarantees of alternating minimization in the stochastic setting. We show that the error of AM decays at the sub-linear rate $O((1/t)^{3/2} + 1/t)$ as a function of the iteration t;
- *extensive empirical evaluation* on a variety of network architectures and datasets, demonstrating significant advantages of our method vs. offline counterparts, as well as somewhat faster initial convergence as compared to SGD and Adam, followed by similar asymptotic performance;
- our online method inherits common advantages of similar offline auxiliary-variable methods, including (1) *no vanishing gradients*, (2) handling of *non-differentiable nonlinearities* more easily in local subproblems, and (3) the *possibility for parallelizing weight updates across layers*;
- similarly to target propagation approaches (LeCun, 1986; 1987; Lee et al., 2015; Bartunov et al., 2018), our method is based on an *explicit propagation of neural activity* and *local synaptic updates*, which is one step closer to a more *biologically plausible* credit assignment mechanism than backprop; see (Bartunov et al., 2018) for a detailed discussion on this topic.

2. Alternating Minimization: Breaking Gradient Chains with Auxiliary Variables

We denote as $(\boldsymbol{X}, \boldsymbol{Y}) = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ a dataset of n labeled samples, where $\boldsymbol{x}_t$ and $\boldsymbol{y}_t$ are the sample and its (vector) label at time t, respectively (e.g., one-hot m-dimensional vector $\boldsymbol{y}$ encoding discrete labels with m possible values). We assume $\boldsymbol{x} \in \mathbb{R}^N$, and $\boldsymbol{y} \in \{0, 1\}^m$. Given a fully-connected neural network with L hidden layers, $\boldsymbol{W^j}$ denotes the $m_j \times m_{j-1}$ link weight matrix associated with the links from layer $j-1$ to layer j, where m_j is the number of nodes at layer j. $\boldsymbol{W}^{L+1}$ denotes the $m_L \times m$ weight matrix connecting the last hidden layer L with the output. We denote the set of all weights $\boldsymbol{W} = \{\boldsymbol{W}^1, ..., \boldsymbol{W}^{L+1}\}$.

Optimization problem. Training a fully-connected neural network with L hidden layers consists of minimizing, with respect to weights $\boldsymbol{W}$, the loss $\mathcal{L}(y, f(\boldsymbol{W}, \boldsymbol{x}_L))$ involving a nested function $f(\boldsymbol{W}, \boldsymbol{x}_L) = f_{L+1}(\boldsymbol{W}_{L+1}, f_L(\boldsymbol{W}_L, f_{L-1}(\boldsymbol{W}_{L-1}, ...f_1(\boldsymbol{W}_1, \boldsymbol{x})...)$; this can be re-written as constrained optimization:

$$\min_{\boldsymbol{W}} \quad \sum_{t=1}^{n} \mathcal{L}(\boldsymbol{y}_t, \boldsymbol{a}_t^L, \boldsymbol{W}^{L+1}),\ where\ \boldsymbol{a}_t^l = \sigma_l(\boldsymbol{c}_t^l),$$
$$s.t.\ \boldsymbol{c}_t^l = \boldsymbol{W}^l \boldsymbol{a}_t^{l-1},\ l = 1, ..., L,\ and\ \boldsymbol{a_t}^0 = \boldsymbol{x}_t. \quad (1)$$

In the above formulation, we use $\boldsymbol{a}_t^l$ as shorthand (not a new variable) denoting the *activation* vector of hidden units in layer l, where σ is a nonlinear activation function (e.g, ReLU, $tanh$, etc) applied to *code* $\boldsymbol{c}^l$, a new *auxiliary variable* that must be equal to a linear transformation of the previous-layer activations.

For classification problems, we use the multinomial loss as our objective function: $\mathcal{L}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{W}) = -\log P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{W})$

$$= \quad -\sum_{i=1}^{m} \boldsymbol{y}_i(\boldsymbol{w}_i^T \boldsymbol{x}) + \log(\sum_{l=1}^{m} \exp(\boldsymbol{w}_l^T \boldsymbol{x})), \qquad (2)$$

where w_i is the i^{th} column of $\boldsymbol{W}$, y_i is the i^{th} entry of the one-hot vector encoding $\boldsymbol{y}$, and the class likelihood is modeled as $P(y_i = 1|\boldsymbol{x}, \boldsymbol{W}) = \exp(\boldsymbol{w}_i^T \boldsymbol{x}) / \sum_{l=1}^{m} \exp(\boldsymbol{w}_l^T \boldsymbol{x})$.

Offline Alternating Minimization. We start with an offline optimization problem formulation, for a given dataset of n samples, which is similar to (Carreira-Perpiñán & Wang, 2014) but uses multinomial instead of quadratic loss, and a different set of *auxiliary variables*. Namely, we use the following relaxation of the constrained formulation in eq. 1:

$$f(\boldsymbol{W}, \boldsymbol{C}) = \sum_{t=1}^{n} \mathcal{L}(y_t, \sigma_L(\boldsymbol{c}_t^L), \boldsymbol{W}^{L+1}) + \mu \sum_{t=1}^{n} \sum_{l=1}^{L} ||\boldsymbol{c}_t^l - \boldsymbol{W}^l \sigma_{l-1}(\boldsymbol{c}_t^{l-1})||_2^2. \qquad (3)$$

This problem can be solved by alternating minimization (AM), or block-coordinate descent (BCD), over weights $\boldsymbol{W} = \{\boldsymbol{W}^1, ..., \boldsymbol{W}^{L+1}\}$ and codes $\boldsymbol{C} = \{\boldsymbol{c}_1^1, ..., \boldsymbol{c}_1^L, ...\boldsymbol{c}_n^1, ..., \boldsymbol{c}_n^L, \}$. Each iteration involves optimizing $\boldsymbol{W}$ for fixed $\boldsymbol{C}$, followed by fixing $\boldsymbol{W}$ and optimizing $\boldsymbol{C}$. The parameter $\mu > 0$ acts as a regularization weight. As in (Carreira-Perpiñán & Wang, 2014), we use an adaptive scheme for gradually increasing μ over iterations[2]

Online Alternating Minimization. The offline alternating minimization outlined above is not scalable to extremely large datasets (even data-parallel methods, such as (Taylor et al., 2016), are inherently limited by the number of cores available), and not suitable for incremental, continual/lifelong (Ring, 1994; Thrun, 1995; 1998) or reinforcement learning scenarios with potentially infinite data streams. To overcome those limitations, we propose a general *online AM* algorithmic scheme and present *two specific algorithms* which differ in optimization approaches used for updating $\boldsymbol{W}$; both algorithms are later evaluated and compared empirically.

Our approach is outlined in Algorithms 2.1, 2.2, and 2.3, omitting implementation details such as the adaptive μ schedule, hyperparameters controlling the number of iterations in optimization subroutines, and several others; we will make our code available online. As an input, the method takes an initial $\boldsymbol{W}$ (e.g., random), initial penalty weight μ, learning rate for the predictive layer, η, and a Boolean variable Mem, indicating which optimization method to use for $\boldsymbol{W}$ updates; if $Mem = 1$, a memory-based approach (discussed below) is selected, and initial memory matrices $\boldsymbol{A_0}$, $\boldsymbol{B_0}$ (described below) will be provided (typically, both are initialized to all zeros unless we want to retain the memory of some prior learning experience, e.g. in a continual learning scenario). The algorithm processes samples one at a time (but can easily be generalized to mini-batches); the current sample is encoded in its representations at each layer (**encodeInput** procedure, Algorithm 2.2), and an output prediction is made based on such encodings. The prediction error is computed, and the backward code updates follow as shown in the **updateCodes** procedure, where the code vector at layer l is optimized with respect to the only two parts of the global objective that the code variables participate in. Once the codes are updated, the *weights can be optimized in parallel across the layers* (in **updateWeights** procedure, Algorithm 2.3) since fixing codes breaks the weight optimization problem into layer-wise independent subproblems. We next discuss each step in detail.

Algorithm 2.1 Online Alternating Minimization (AM)

Require: $(\boldsymbol{x}, \boldsymbol{y}) \sim p(\boldsymbol{x}, y)$ (data stream sampled from distribution $p(\boldsymbol{x}, \boldsymbol{y})$; initial weights $\boldsymbol{W}_0$; $\mu \in \mathbb{R}^+$ (quadratic penalty weight); $\eta \in \mathbb{R}^+$ (top-layer weight update step size); Mem (indicates the type of optimization method for **updateWeights**; if "yes", input initial memory matrices $\boldsymbol{A_0}$ and $\boldsymbol{B_0}$).

while more samples **do**
  Input $(\boldsymbol{x}_t, y_t)$
  $\boldsymbol{C} \leftarrow$ **encodeInput**($\boldsymbol{x}_t$, $\boldsymbol{W}_{t-1}$) % forward: compute linear activations at layers $1, ..., L$
  $\boldsymbol{C} \leftarrow$ **updateCodes**($\boldsymbol{C}$, y_t, $\boldsymbol{W}_{t-1}$, μ) % backward: error propagation by activation (code) changes
  $\boldsymbol{W}_t \leftarrow$ **updateWeights**($\boldsymbol{W}_{t-1}$, $\boldsymbol{x}_t$, y_t, $\boldsymbol{C}$, μ, η, Mem)
end while
return $\boldsymbol{W}_t$

Algorithm 2.2 Activation Propagation (Code Update) Steps

encodeInput($\boldsymbol{x}$, $\boldsymbol{W}$)

$\boldsymbol{c}^0 = \boldsymbol{x}$
for $l = 1$ to L **do**
  $\boldsymbol{c}^l = \boldsymbol{W}^l \sigma_{l-1}(\boldsymbol{c}^{l-1})$
  % $\sigma_0(\boldsymbol{x}) = \boldsymbol{x}, \sigma_l(\boldsymbol{x}) = ReLU(\boldsymbol{x})$ for $l = 1, ..., L$
end for
return $\boldsymbol{C}$

updateCodes($\boldsymbol{C}$, $\boldsymbol{y}$, $\boldsymbol{W}$, λ_C, μ)

$\boldsymbol{c}^L \leftarrow$ Solve Problem (4), $\boldsymbol{c}^0 = \boldsymbol{x}$
for $l = L - 1$ to 1 **do**
  $\boldsymbol{c}^l \leftarrow$ Solve Problem (5)
end for
return $\boldsymbol{C}$

Activation propagation: forward and backward passes. In an online setting, we only have access to the current sample $\boldsymbol{x}_t$ at time t, and thus can only compute the corresponding codes $\boldsymbol{c}_t^l$ using the weights computed so far. Namely, given input $\boldsymbol{x}_t$, we compute the last-layer activations $\boldsymbol{a}_t^L = \sigma_L(\boldsymbol{c}_t^L)$ in a forward pass, propagating activa-

[2] Note that sparsity (l_1 regularization) on both $\boldsymbol{c}$ and $\boldsymbol{W}$ could be easily added to the objective in eq. 3 and would not change the computational complexity of the algorithms detailed below (we can use proximal instead of gradient methods).

Algorithm 2.3 Weight and Memory Update Steps

updateWeights($\boldsymbol{W}$, $\boldsymbol{x}$, y, $\boldsymbol{C}$, μ, η, Mem)

$\boldsymbol{W}^{L+1} = \boldsymbol{W}^{\boldsymbol{L+1}} - \eta \nabla_{\boldsymbol{W}} \mathcal{L}(y, \sigma_L(\boldsymbol{c}^L), \boldsymbol{W}^{\boldsymbol{L+1}})$
for $l = 1$ to L **do**
  if Mem **then**
    $(\boldsymbol{A}^{\boldsymbol{l}}{}_t, \boldsymbol{B}^{\boldsymbol{l}}{}_t) \leftarrow$ **updateMemory**($\boldsymbol{A}^{\boldsymbol{l}}{}_{t-1}$, $\boldsymbol{B}^{\boldsymbol{l}}{}_{t-1}, \boldsymbol{C}^{\boldsymbol{l}}$)
    % $\hat{f}^l(\boldsymbol{W}^{\boldsymbol{l}}) \equiv Tr(\boldsymbol{W}^T \boldsymbol{W} \boldsymbol{A}^l) - 2Tr(\boldsymbol{W}^T \boldsymbol{B}^l)$
    $\boldsymbol{W}^l = \arg\min_W \hat{f}^l(\boldsymbol{W})$
  else
    %(parallel) local update of each layer weights,
    %(*independently of other layers (unlike backprop)*
    $\boldsymbol{W}^l \leftarrow$ **SGD**($\boldsymbol{W}^{\boldsymbol{l}}$, $\boldsymbol{x}$, y, $\boldsymbol{C}^{\boldsymbol{l}}$, μ, η)
  end if
end for
return $\boldsymbol{W}$

updateMemory($\boldsymbol{A}$, $\boldsymbol{B}, \boldsymbol{C}$)
for $l = 1$ to L **do**
  $\boldsymbol{a} = \sigma_{l-1}(\boldsymbol{c}^{l-1})$, $\boldsymbol{A}^l \leftarrow \boldsymbol{A}^l + \boldsymbol{a}\boldsymbol{a}^T$, $\boldsymbol{B}^l \leftarrow \boldsymbol{B}^l + \boldsymbol{c}^l \boldsymbol{a}^T$
end for
return $\boldsymbol{A}, \boldsymbol{B}$

tions from input to the last layer, and make a prediction about y_t, incurring the loss $\mathcal{L}(y_t, \boldsymbol{a}_t^L, \boldsymbol{W}^{L+1})$. We now propagate this error back to all activations. This is achieved by solving a sequence of optimization problems:

$$\begin{aligned} \boldsymbol{c}^L = \quad & \arg\min_{\boldsymbol{c}} \mathcal{L}(y, \sigma_L(\boldsymbol{c}), \boldsymbol{W}^{L+1}) \\ & + \quad \mu ||\boldsymbol{c} - \boldsymbol{W}^L \sigma_{L-1}(\boldsymbol{c}^{L-1})||_2^2 \end{aligned} \tag{4}$$

$$\begin{aligned} \boldsymbol{c}^l = \quad & \arg\min_{\boldsymbol{c}} \mu ||\boldsymbol{c}^{l+1} - \boldsymbol{W}^{l+1} \sigma_l(\boldsymbol{c})||_2^2 \\ & + \quad \mu ||\boldsymbol{c} - \boldsymbol{W}^l \sigma_{l-1}(\boldsymbol{c}^{l-1})||_2^2, \end{aligned} \tag{5}$$

for $l = L-1, ..., 1$.

Weights Update Step. Different online (stochastic) optimization methods can be applied to update the weights at each layer, using a *surrogate* objective function defined more generally than in (Mairal et al., 2009) as follows: $\hat{f}_{[t':t]}(\boldsymbol{W}) = f(\boldsymbol{W}, \boldsymbol{C}_{[t':t]})$, where f is defined in eq. 3 and $\boldsymbol{C}_{[t':t]}$ denotes codes for all samples from time t' to time t, computed at previous iterations. When $t' = 1$, we simplify the notation to $\hat{f}_t(\boldsymbol{W})$, and when $t' = t$, the surrogate is the same as the true objective on the current-time codes $f(\boldsymbol{W}, \boldsymbol{C}_t)$. The surrogate objective decomposes into $L+1$ independent terms, $\hat{f}_t(\boldsymbol{W}) = \sum_{l=1}^{L+1} \hat{f}_t^l(\boldsymbol{W}^l)$, which allows for *parallel weight optimization across all layers*:

$$\boldsymbol{W}^{L+1} = \arg\min_{\boldsymbol{W}} \left\{ \hat{f}_t^{L+1}(\boldsymbol{W}) \equiv \sum_{i=1}^{t} \mathcal{L}(y_i, \sigma_L(\boldsymbol{c}_i^L), \boldsymbol{W}) \right\}.$$

For layers $l = 1, ..., L$, we have

$$\boldsymbol{W}^l = \arg\min_{\boldsymbol{W}} \left\{ \hat{f}_t^l(\boldsymbol{W}) \equiv \mu \sum_{i=1}^{t} \|\boldsymbol{c}_i^l - \boldsymbol{W} \sigma_{l-1}(\boldsymbol{c}_i^{l-1})\|_2^2 \right\}. \tag{6}$$

In general, computing a surrogate function with $t' < t$ would require storing all samples and codes in that time interval. Thus, for the $\boldsymbol{W}^{L+1}$ update, we always use $t' = t$ (current sample), and optimize $f^{L+1}(\boldsymbol{W})$ via stochastic gradient descent (SGD) (step 1 in **updateWeights**, Algorithm 2.3). However, in case of quadratic loss (intermediate layers), we have more options. One is to use SGD again, or its adaptive-rate version such as Adam. This option is selected when $Mem = False$ is passed to **updateWeights** function in Algorithm 2.3. We call that method *AM-Adam*.

Alternatively, we can use the memory-efficient surrogate-function computation as in (Mairal et al., 2009), where $t' = 1$, i.e. the surrogate function accumulates the memory of all previous samples and codes, as described below; we hypothesize that such an approach, here called *AM-mem*, can be useful in continual learning as a potential mechanism to alleviate the catastrophic forgetting issue.

Co-Activation Memory. We now summarize the memory-based approach. Denoting activation in layer l as $\boldsymbol{a}^l = \sigma_l(\boldsymbol{c}^l)$, and following (Mairal et al., 2009), we can rewrite the above objective in eq. 6 using the following:

$$\sum_{i=1}^{t} ||\boldsymbol{c}_i^l - \boldsymbol{W}\boldsymbol{a}_i^l||_2^2 = Tr(\boldsymbol{W}^T \boldsymbol{W} \boldsymbol{A}_t^l) - 2Tr(\boldsymbol{W}^T \boldsymbol{B}_t^l), \tag{7}$$

where $\boldsymbol{A}_t^l = \sum_{i=1}^t \boldsymbol{a}_i^{l-1} (\boldsymbol{a}_i^{l-1})^T$ and $\boldsymbol{B}_t^l = \sum_{i=1}^t \boldsymbol{c}_i^l (\boldsymbol{a}_i^{l-1})^T$ are the "memory" matrices (i.e. *co-activation memories*), compactly representing the accumulated strength of co-activations in each layer (matrices $\boldsymbol{A}_{\boldsymbol{t}}^{\boldsymbol{l}}$, i.e. covariances) and across consecutive layers (matrices $\boldsymbol{B}_{\boldsymbol{t}}^{\boldsymbol{l}}$, or cross-covariances). At each iteration t, once the new input sample $\boldsymbol{x}_t$ is encoded, the matrices are updated (**updateMemory** function, Algorithm 2.3) as

$$\boldsymbol{A_t} \leftarrow \boldsymbol{A_t} + \boldsymbol{a}_t^{l-1} (\boldsymbol{a}_t^{l-1})^T \text{ and } \boldsymbol{B} \leftarrow \boldsymbol{B_t} + \boldsymbol{c}_t^l (\boldsymbol{a}_t^{l-1})^T.$$

It is important to note that, using memory matrices, we are effectively optimizing the weights at iteration t with respect to all previous samples and their previous linear activations at all layers, without the need for an explicit storage of these examples. Clearly, AM-SGD is even more memory-efficient since it does not require any memory matrices. Finally, to optimize the quadratic surrogate in eq. 7, we follow (Mairal et al., 2009) and use *block-coordinate descent*, iterating over the columns of the corresponding weight matrices; however, rather than always iterating until convergence, we make the number of such iterations an additional hyperparameter.

3. Theoretical analysis

We will next provide theoretical convergence analysis for a general stochastic alternating minimization (AM) scheme. Under certain assumptions that we will discuss, the algorithms proposed in the previous section fall into the category of approaches that comply with these guarantees, although

our theory is applicable to a wider family of AM algorithms. *To the best of our knowledge, we provide the first theoretical convergence guarantees of AM in the stochastic setting*.

Setting. Let in general $f(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$ denote the function to be optimized using AM, where in the i^{th} step of the algorithm, we optimize f with respect to $\boldsymbol{\theta}_i$ and keep other arguments fixed. Let K denote total number of arguments. For the theoretical analysis, we consider a smooth approximation to f as done in the literature (Schmidt et al., 2007; Lange et al., 2014).

Let $\{\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \ldots, \boldsymbol{\theta}_K^*\}$ denote the global optimum of f computed on the entire data population. For the sake of the theoretical analysis we assume that the algorithm knows the lower-bound on the radii of convergence $r_1, r_2, \ldots, r_K$ for $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K$.[3] Let $\nabla_i f^1$ denote the gradient of f computed for a single data sample $(\boldsymbol{x}, \boldsymbol{y})$ and taken with respect to the i^{th} argument of the function f (weights or codes from Algorithm 2.1). In the next section, we refer to $\nabla_i f(\cdot)$ as the gradient of f with respect to $\boldsymbol{\theta}_i$ computed for the entire data population, i.e. an infinite number of samples ("oracle gradient"). We assume in the i^{th} step ($i = 1, 2, \ldots, K$), the AM algorithm performs the update:

$$\boldsymbol{\theta}_i^{t+1} = \Pi_i(\boldsymbol{\theta}_i^t - \eta^\tau \nabla_i f^1(\boldsymbol{\theta}_1^{t+1}, \ldots, \boldsymbol{\theta}_{i-1}^{t+1}, \boldsymbol{\theta}_i^t, \boldsymbol{\theta}_{i+1}^t, \ldots, \boldsymbol{\theta}_K^t)), \tag{8}$$

where t denotes time, Π_i denotes the projection onto the Euclidean ball $B_2(\frac{r_i}{2}, \boldsymbol{\theta}_i^0)$ of some given radius $\frac{r_i}{2}$ centered at the initial iterate $\boldsymbol{\theta}_i^0$. Thus, given any initial vector $\boldsymbol{\theta}_i^0$ in the ball of radius $\frac{r_i}{2}$ centered at $\boldsymbol{\theta}_i^*$, we are guaranteed that all iterates remain within an r_i-ball of $\boldsymbol{\theta}_i^*$. This is true for all $i = 1, 2, \ldots, K$. The re-projection step of eq. 8 implies that starting close enough to the optimum and taking small steps leads to convergence rate of Theorem 3.1. The radiuses dictate how convergence is affected if the iterates stray further from the optimum through the variable σ^2 defined before that theorem.

Remark 3.1. *The difference between the AM scheme we analyze and the Algorithm 2.1 can be summarized as follows: i) only a single SGD step is taken with respect to weights and then codes (while Algorithm 2.1 can optimize codes till convergence at each iteration); ii) gradient direction is approximated with respect to a single data sample (in practice, Algorithm 2.1 uses mini-batches), and iii) re-projection step is included, unlike in Algorithm 2.1.*

We argue that the general AM scheme analyzed here leads to the worst-case theoretical guarantees with respect to the original setting from Algorithm 2.1, i.e. we expect the convergence rate for the original setting to be no worse than the one dictated by the obtained guarantees. This is because we allow only a single stochastic update (i.e. computed on a single data point) with respect to an appropriate argument (when keeping other arguments fixed) in each step of AM, whereas in Algorithm 2.1 and related schemes in the literature, one may increase the size of the data mini-batch in each AM step (semi-stochastic setting). The convergence rate in the latter case is typically better (Nesterov, 2014). Finally, note that the analysis does not consider running the optimizer more than once before changing the argument of an update, e.g., when obtaining sparse code $\boldsymbol{c}$ for a given data point $(\boldsymbol{x}, \boldsymbol{y})$ and fixed weights. We expect this to have a minor influence on the convergence rate as our analysis specifically considers a local convergence regime, where we expect that running the optimizer once produces good enough parameter approximations. Moreover, note that by preventing each AM step to be performed multiple times, we analyze a more stochastic (noisier) version of parameter updates.

[3] This assumption is potentially easy to eliminate with a more careful choice of the step size in the first iterations.

Statistical guarantees for AM algorithms. The theoretical analysis we provide here is an extension to the AM setting of recent work on statistical guarantees for the EM algorithm (Balakrishnan et al., 2017).

We first discuss necessary assumptions that we make. Let $L(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K) = -f(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$ and denote $L_d^*(\boldsymbol{\theta}_d) = L(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \ldots, \boldsymbol{\theta}_{d-1}^*, \boldsymbol{\theta}_d, \boldsymbol{\theta}_{d+1}^*, \ldots, \boldsymbol{\theta}_{K-1}^*, \boldsymbol{\theta}_K^*)$. Let $\Omega_1, \Omega_2, \ldots, \Omega_K$ denote non-empty compact convex sets such that for any $i = \{1, 2, \ldots, K\}, \boldsymbol{\theta}_i \in \Omega_i$. The following three assumptions are made on $L_d^*(\boldsymbol{\theta}_d)$ ($d = 1, 2, \ldots, K$) and the objective function $L(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$.

Assumption 3.1 (*Strong concavity*)**.** *The function $L_d^*(\boldsymbol{\theta}_d)$ is strongly concave for all pairs $(\boldsymbol{\theta}_{d,1}, \boldsymbol{\theta}_{d,2})$ in the neighborhood of $\boldsymbol{\theta}_d^*$. That is*

$$L_d^*(\boldsymbol{\theta}_{d,1}) - L_d^*(\boldsymbol{\theta}_{d,2}) - \langle \nabla_d L_d^*(\boldsymbol{\theta}_{d,2}), \boldsymbol{\theta}_{d,1} - \boldsymbol{\theta}_{d,2} \rangle \leq -\frac{\lambda_d}{2} \|\boldsymbol{\theta}_{d,1} - \boldsymbol{\theta}_{d,2}\|_2^2,$$

where $\lambda_d > 0$ is the strong concavity modulus.

Assumption 3.2 (*Smoothness*)**.** *The function $L_d^*(\boldsymbol{\theta}_d)$ is μ_d-smooth for all pairs $(\boldsymbol{\theta}_{d,1}, \boldsymbol{\theta}_{d,2})$. That is*

$$L_d^*(\boldsymbol{\theta}_{d,1}) - L_d^*(\boldsymbol{\theta}_{d,2}) - \langle \nabla_d L_d^*(\boldsymbol{\theta}_{d,2}), \boldsymbol{\theta}_{d,1} - \boldsymbol{\theta}_{d,2} \rangle \geq -\frac{\mu_d}{2} \|\boldsymbol{\theta}_{d,1} - \boldsymbol{\theta}_{d,2}\|_2^2,$$

where $\mu_d > 0$ is the smoothness constant.

Next, we introduce the gradient stability (GS) condition that holds for any d from 1 to k.

Assumption 3.3 (*Gradient stability (GS)*)**.** *We assume $L(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \ldots, \boldsymbol{\theta}_K)$ satisfies GS (γ_d) condition, where $\gamma_d \geq 0$, over Euclidean balls $\boldsymbol{\theta}_1 \in B_2(r_1, \boldsymbol{\theta}_1^*), \ldots, \boldsymbol{\theta}_{d-1} \in B_2(r_{d-1}, \boldsymbol{\theta}_{d-1}^*), \boldsymbol{\theta}_{d+1} \in B_2(r_{d+1}, \boldsymbol{\theta}_{d+1}^*), \ldots, \boldsymbol{\theta}_K \in B_2(r_K, \theta_K^*)$ of the form*

$$\|\nabla_d L_d^*(\boldsymbol{\theta}_d) - \nabla_d L(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)\|_2 \leq \gamma_d \sum_{\substack{i=1 \\ i \neq d}}^{K} \|\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^*\|_2.$$

We also define the following bound σ on the expected value of the norm of the gradients of our objective function (commonly done in the stochastic gradient descent convergence theorems as well). Define $\sigma = \sqrt{\sum_{d=1}^{K} \sigma_d^2}$ where

$$\sigma_d^2 = \sup\{\mathbb{E}[\|\nabla_d L_1(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)\|_2^2] : \boldsymbol{\theta}_1 \in B_2(r_1, \boldsymbol{\theta}_1^*) \ldots \boldsymbol{\theta}_K \in B_2(r_k, \boldsymbol{\theta}_k^*)\}$$

The following theorem then gives a recursion on the expected error obtained at each iteration of Algorithm 1.

Theorem 3.1. *Given the stochastic AM gradient iterates of the version of Algorithm 2.1 given in eq. 8 with decaying step size $\{\eta^t\}_{t=0}^{\infty}$ and $\gamma < \frac{2\xi}{3(K-1)}$, the error at iteration $t+1$ satisfies recursion*

$$\mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \leq (1-q^t)\mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{t}\|_2^2\right] + \frac{(\eta^t)^2}{1-(K-1)\eta^t\gamma}\sigma^2, \quad (9)$$

where $\boldsymbol{\Delta}_d^{t+1} := \boldsymbol{\theta}_d^{t+1} - \boldsymbol{\theta}_d^*$ for $d = 1, 2, \ldots, K$, $\gamma := \max_{i=1,2,\ldots,K} \gamma_i$, $q^t = 1 - \frac{1-2\eta^t\xi+2\eta^t\gamma(K-1)}{1-(K-1)\eta^t\gamma}$, and $\xi := \min_{i=1,2,\ldots,K} \frac{2\mu_i\lambda_i}{\mu_i+\lambda_i}$.

The recursion in Theorem 3.1 is expanded in the Supplementary Material to prove the final convergence theorem stated as follows:

Theorem 3.2. *Given the stochastic AM gradient iterates of the version of Algorithm 2.1 given in eq. 8 with decaying step size $\eta^t = \frac{3/2}{[2\xi-3\gamma(K-1)](t+2)+\frac{3}{2}(K-1)\gamma}$ and assuming that $\gamma < \frac{2\xi}{3(K-1)}$, the error at iteration $t+1$ satisfies*

$$\mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \leq \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right]\left(\frac{2}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \frac{9}{[2\xi-3\gamma(K-1)]^2(t+3)}, \quad (10)$$

where $\boldsymbol{\Delta}_d^{t+1} := \boldsymbol{\theta}_d^{t+1} - \boldsymbol{\theta}_d^*$ for $d = 1, 2, \ldots, K$, $\gamma := \max_{i=1,2,\ldots,K} \gamma_i$, and $\xi := \min_{i=1,2,\ldots,K} \frac{2\mu_i\lambda_i}{\mu_i+\lambda_i}$.

4. Experiments

We compare on several datasets (MNIST, CIFAR10, HIGGS) our online alternating minimization algorithms, *AM-mem* and *AM-Adam* (using mini-batches instead of single samples at each time point), against backrop-based online methods, SGD and Adam (Kingma & Ba, 2014), as well as against the offline auxiliary-variable ADMM method of (Taylor et al., 2016), using code provided by the authors[4], and against the two offline versions of our methods, *AM-Adam-off* and *AM-mem-off*, which simply treat the training dataset as a single minibatch, i.e. one AM iteration is equivalent to one epoch over the training dataset. All our algorithms were implemented in PyTorch (Paszke et al., 2017); we also used PyTorch implementation of *SGD* and *Adam*. Hyperparameters used for each method were optimized by grid search on a validation subset of training data. Most results were averaged over at least 5 different weight initializations.

Note that most of the prior auxiliary-variable methods were evaluated only on fully-connected networks (Carreira-Perpiñán & Wang, 2014; Taylor et al., 2016; Zhang et al., 2016; Zhang & Brand, 2017; Zeng et al., 2018; Askari et al., 2018), while *we also experiment with RNNs and CNNs, as well as with discrete (nondifferentiable) networks*.

Fully-connected nets: MNIST, CIFAR10, HIGGS. We experiment with fully-connected networks on the standard MNIST (LeCun, 1998) dataset, consisting of 28×28 gray-scale images of hand-drawn digits, with 50K samples, and a test set of 10K samples. We evaluate two different 2-hidden-layer architectures, with equal hidden layer sizes of 100 and 500, and ReLU activations. Figure 1 zooms in on the performance of the online methods, *AM-Adam*, *AM-mem*, *SGD* and *Adam*, over 50 minibatches of size 200 each. We observe that, on both architectures, *AM-Adam is comparable to (in early stages, even slightly better than) SGD and Adam*, while *AM-mem* is comparable with them on the larger architecture, and falls between *SGD* and *Adam* on the smaller one. Next, Figure 2 continues to 50 epochs, now including the offline methods (which require at least 1 epoch over the full dataset, by definition). Our *AM-Adam* method matches *SGD* and *Adam*, reaching 0.98 accuracy. Our second method, *AM-mem* only yields 0.91 and 0.96 on the 100-node and 500-node networks, respectively. *All offline methods are significantly outperformed by the online ones; e.g., Taylor's ADMM learns very slowly until about 10 epochs, being greatly outperformed even by our offline versions, but later catches up with offline* AM-mem *on the 100-node network; it is still inferior to all other methods on the 500-node architecture.*

Figures 3 and 4 show similar results for the same experiment setting, on the CIFAR10 dataset (5000 training and 10000 test samples). Again, our *AM-Adam* performs slightly better than SGD and Adam on the first 50 minibatches (same size 200 as before), and even on 50 epochs for the 1-100 archi-

[4]We choose Taylor's ADMM among several auxiliary methods proposed recently, since it was the only one capable of handling very large datasets due to massive data parallelization; also, some other methods were not designed for classification task, e.g. (Carreira-Perpiñán & Wang, 2014) trained autoencoders, (Zhang et al., 2016) learned hashing.

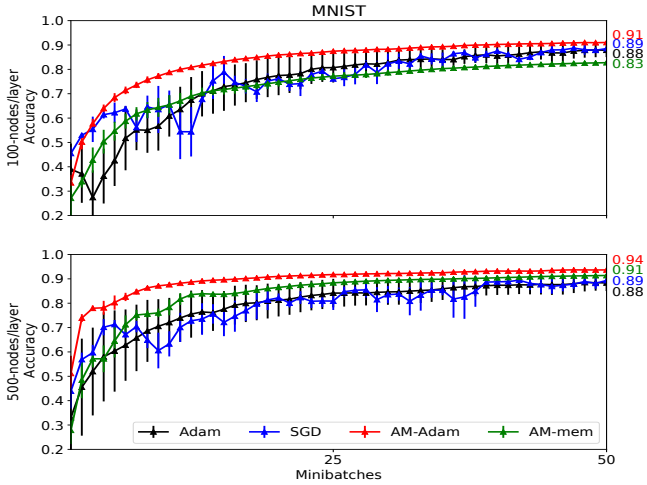

Figure 1. MNIST (fully-connected nets, 2 layers): online methods, first epoch; 50 mini-batches, 200 samples each.

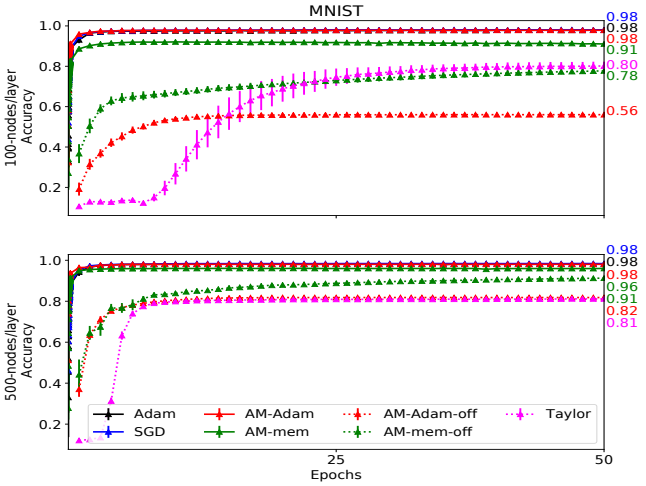

Figure 2. MNIST (fully-connected nets, 2 layers): online vs. offline methods vs. Taylor's ADMM, 50 epochs.

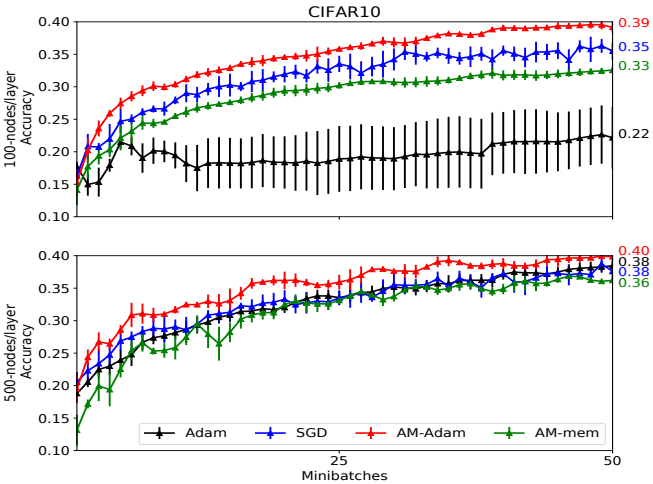

Figure 3. CIFAR10 (fully-connected nets): online methods, 1st epoch. 2 hidden layers with 100 (top) and 500 (bottom) units each; 250 mini-batches, 200 samples each.

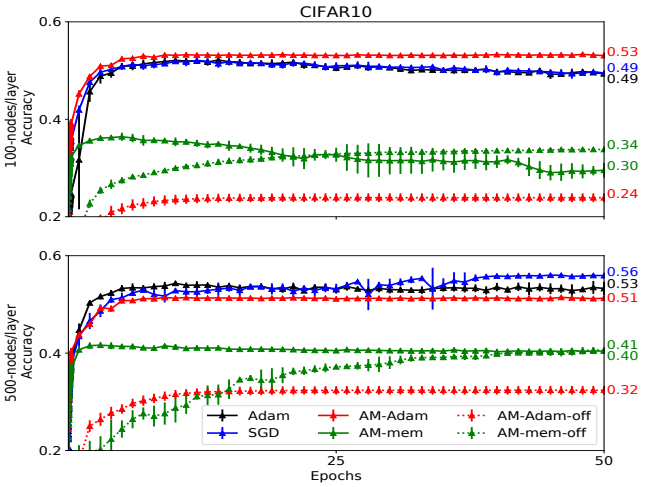

Figure 4. CIFAR10 (fully-connected networks): online vs. offline, 50 epochs. Similar experiments to Figure 2.

tecture, reaching 0.53 vs 0.49 accuracy of SGD and Adam, but falls a bit behind on the larger 1-500 architecture with 0.51 vs 0.53 and 0.56, respectively. Our second algorithm, *AM-mem*, is clearly dominated by all the three methods above. Also, we ran the two offline AM versions, which were again greatly outperformed by the online methods. *In the remaining experiments, we focus on our best-performing method, online AM-Adam.*

HIGGS, fully-connected, 1-300 ReLU network. In Figure 5, we compare our *online AM-Adam* approach against *SGD*, *Adam* and the offline ADMM method of Taylor, on a very large HIGGS dataset, containing 10,500,000 training samples (28 features each) and 500,000 test samples. Each datapoint is labeled as either a signal process producing a Higgs boson or a background process which does not. We use the same architecture (a single-hidden layer network with ReLU activations and 300 hidden nodes) as in (Taylor et al., 2016), and the same training/test data sets. For all online methods, we use minibatches of size 200, so one epoch over the 10.5M samples equals 52,500 iterations.

While Taylor's method was reported to achieve 0.64 accuracy on the whole dataset (using data parallelization on 7200 cores to handle the whole dataset as a batch) (Taylor et al., 2016), the online methods achieve the same accuracy much faster (less than 1000 iterations/200K samples for our *AM-Adam*, and less than 2000 iterations for *SGD* and *Adam*; within only 20,000 iterations (less than a half of training samples), *AM-Adam*, *SGD* and *Adam* 0.70, 0.69 and 0.71, respectively, and continue to improve slowly, reaching after one epoch, 0.71, 0,71 and 0.72, respectively. (Our *AM-mem*

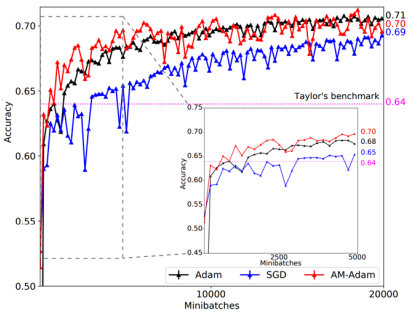

Figure 5. HIGGS dataset.

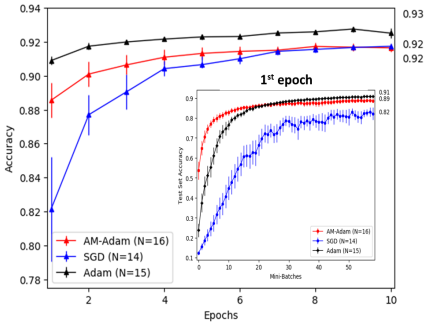

Figure 6. RNN-15, Sequential MNIST.

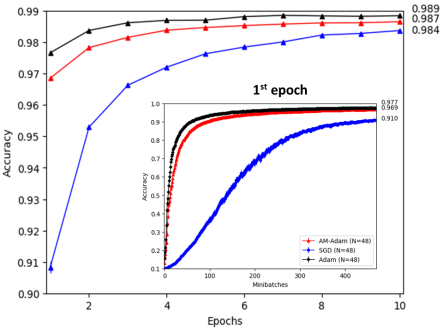

Figure 7. CNN: LeNet5, MNIST.

version quickly reached 0.6 together with *AM-Adam*, but then slowed down, reaching only 0.61 on the 1st epoch).

In summary, *on HIGGS dataset,* AM-Adam*, SGD and Adam clearly outperform Taylor's offline ADMM, while using less than a half of the 1st epoch, and quickly reaching Taylor's 0.64 accuracy benchmark after observing only a tiny fraction (less than 0.01%) of the 10.5M dataset. Both Adam and AM-Adam perform very closely, both outperforming SGD.*

RNN on MNIST. Next, we evaluate our method on Sequential MNIST (Le et al., 2015), where each image is vectorized and fed to the RNN as a sequence of $T = 784$ pixels. We use the standard Elman RNN architecture with $tanh$ activations among hidden states and ReLU applied to the output sequence before making a prediction (we use larger minibatches of 1024 samples to reduce training time). *AM-Adam* was adapted to work on such RNN architecture (see Appendix for details). Figure 6 shows the results using $d = 15$ hidden units (see Appendix for $d = 50$), averaged over N weight initializations, for 10 epochs, with a zoom-in on the first epoch inset. *AM-Adam performs similarly to Adam in the 1st epoch, and outperforms SGD up to epoch 6, matching SGD's performance afterwards.*

CNN (LeNet-5), MNIST. Next, we experiment with CNNs, using LeNet-5 (LeCun et al., 1998) on MNIST (Figure 7). Similarly to RNN result, AM-Adam clearly outperforms SGD, while being somewhat outperformed by Adam.

Binary nets (nondifferentiable activations), MNIST. Finally, to investigate the ability of our method to handle non-differentiable networks, we consider an architecture originally investigated in (Lee et al., 2015) to evaluate another type of auxiliary-variable approach, called Difference Target Propagation (DTP). The model is a 2-hidden layer fully-connected network (784-500-500-10), whose first hidden layer uses the non-differentiable $sign$ transfer function (while the second hidden layer uses $\tanh$). Target propagation approaches were motivated by the goal of finding more biologically plausible mechanisms for credit assignment in the brain's neural networks as compared to standard backprop, which, among multiple other biologically-implausible aspects, does not model the neuronal activation propagation explicitly, and does not handle non-differentiable binary activations (spikes) (Lee et al., 2015; Bartunov et al., 2018). In (Lee et al., 2015), DTP was applied to the above discrete network, and compared to a backprop-based *straight-through estimator (STE)*, which simply ignores the derivative of the step function (which is 0 or infinite) in the back-propagation phase. *While DTP took about 200 epochs to reach 0.2 error, matching the STE performance (Figure 3 in (Lee et al., 2015)), our AM-Adam with binary activations reaches the same error in less than 20 epochs (Figure 8).*

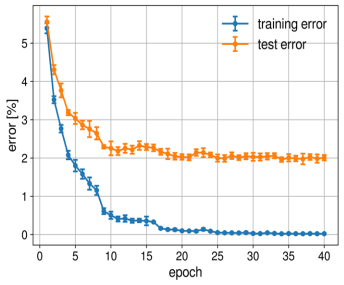

Figure 8. Binary net, MNIST.

5. Conclusions

We proposed a novel online alternating-minimization approach for neural network training; it builds upon previously proposed offline methods that break the nested objective into easier-to-solve local subproblems via inserting auxiliary variables corresponding to activations in each layer. Such methods avoid gradient chain computation and potential issues associated with it, including vanishing gradients, lack of cross-layer parallelization, and difficulties handling non-differentiable nonlinearities. However, unlike prior art, our approach is online (mini-batch), and thus can handle arbitrarily large datasets and continual learning settings. We proposed two variants, AM-mem and AM-Adam, and found that AM-Adam works better. Also, AM-Adam greatly outperforms offline methods on several datasets and architectures; when compared to state-of-the-art backprop methods such as (standard) SGD and Adam, AM-Adam typically matches their performance over multiple epochs, and may even learn somewhat faster initially, in small-data regimes. AM-Adam also converged faster than another related method, difference target propagation, on a discrete (non-differentiable) network. Finally, to the best of our knowledge, we are the first to provide theoretical guarantees for a wide class of online alternating minimization approaches including ours.

References

Askari, A., Negiar, G., Sambharya, R., and El Ghaoui, L. Lifted neural networks. arXiv:1805.01532 [cs.LG], 2018.

Balakrishnan, S., Wainwright, M. J., and Yu, B. Statistical guarantees for the em algorithm: From population to sample-based analysis. *Ann. Statist.*, 45(1):77–120, 02 2017. doi: 10.1214/16-AOS1435. URL https://doi.org/10.1214/16-AOS1435.

Bartunov, S., Santoro, A., Richards, B., Marris, L., Hinton, G. E., and Lillicrap, T. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pp. 9390–9400, 2018.

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Carreira-Perpiñán, M. and Wang, W. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pp. 10–19, 2014.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Gotmare, A., Thomas, V., Brea, J., and Jaggi, M. Decoupling back-propagation using constrained optimization methods. *Proc. of ICML 2018 Workshop on Credit Assignment in Deep Learning and Deep Reinforcement Learning*, 2018.

Guerguiev, J., Lillicrap, T. P., and Richards, B. A. Towards deep learning with segregated dendrites. *ELife*, 6:e22901, 2017.

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

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

Krotov, D. and Hopfield, J. J. Unsupervised learning by competing hidden units. *Proceedings of the National Academy of Sciences*, pp. 201820458, 2019.

Lange, M., Zühlke, D., Holz, O., and Villmann, T. Applications of lp-norms and their smooth approximations for gradient based learning vector quantization. In *ESANN*, 2014.

Lau, T. T.-K., Zeng, J., Wu, B., and Yao, Y. A proximal block coordinate descent algorithm for deep neural network training. *arXiv preprint arXiv:1803.09082*, 2018.

Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. Y. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pp. 265–272. Omnipress, 2011.

Le, Q. V., Jaitly, N., and Hinton, G. E. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

LeCun, Y. Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pp. 233–240. Springer, 1986.

LeCun, Y. *Modèles connexionnistes de l'apprentissage*. PhD thesis, PhD thesis, These de Doctorat, Universite Paris 6, 1987.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

LeCun, Y., Touresky, D., Hinton, G., and Sejnowski, T. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, pp. 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. Difference target propagation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 498–515. Springer, 2015.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, 2009.

Nair, V. and Hinton, G. E. Rectified linear units improve Restricted Boltzmann Machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014. ISBN 1461346916, 9781461346913.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Riedmiller, M. and Braun, H. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pp. 586–591. IEEE, 1993.

Ring, M. B. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.

Robbins, H. and Monro, S. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pp. 102–109. Springer, 1985.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature*, 323(6088): 533, 1986.

Sacramento, J., Costa, R. P., Bengio, Y., and Senn, W. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems*, pp. 8721–8732, 2018.

Schmidt, M., Fung, G., and Rosales, R. Fast optimization methods for l1 regularization: A comparative study and two new approaches. In Kok, J. N., Koronacki, J., Mantaras, R. L. d., Matwin, S., Mladenič, D., and Skowron, A. (eds.), *ECML*, 2007.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pp. 2722–2731, 2016.

Thomas Frerix, Thomas Möllenhoff, M. M. D. C. Proximal backpropagation. *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1706.04638.

Thrun, S. A lifelong learning perspective for mobile robot control. In *Intelligent Robots and Systems*, pp. 201–214. Elsevier, 1995.

Thrun, S. Lifelong learning algorithms. In *Learning to learn*, pp. 181–209. Springer, 1998.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Whittington, J. C. and Bogacz, R. Theories of error backpropagation in the brain. *Trends in cognitive sciences*, 2019.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zeng, J., Lau, T. T.-K., Lin, S., and Yao, Y. Global convergence in deep learning with variable splitting via the kurdyka-lojasiewicz property. *arXiv preprint arXiv:1803.00225*, 2018.

Zhang, G. and Kleijn, W. B. Training deep neural networks via optimization over graphs. arXiv:1702.03380 [cs.LG], 2017.

Zhang, Z. and Brand, M. Convergent block coordinate descent for training Tikhonov regularized deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 1719–1728, 2017.

Zhang, Z., Chen, Y., and Saligrama, V. Efficient training of very deep neural networks for supervised hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1487–1495, 2016.