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 highly non-convex nature of the objective function. State-of-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-update 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, avoiding gradient chains and thus the vanishing gradient issue, allowing weight update parallelization, among other advantages. However, those techniques are mainly offline (batch), which limits their applicability to extremely large datasets or unlimited data streams in online, continual or reinforcement learning. The main contribution of our work is a novel online (stochastic/mini-batch) alternating minimization (AM) algorithm for training deep neural networks, together with the first theoretical convergence guarantees for AM in stochastic settings, and extensive empirical evaluation on various architectures and datasets, demonstrating advantages of the proposed approach as compared to both offline auxiliary variable methods and to the backpropagation-based stochastic gradient descent.

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-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 cannot handle *non-differentiable nonlinearities* and *does not allow parallel weight updates* across the layers. Finally, backpropagation is often criticized for being *biologically implausible*, since its error propagation mechanism does not influence neural activity, unlike known biological feedback mechanisms. For more details on limitations of backpropagation and SGD, both numerical and conceptual, in terms of their biological (im)plausibility, see (Le et al., 2011; Carreira-Perpinan & Wang, 2014; Taylor et al., 2016) and (Lee et al., 2015; Bartunov et al., 2018), respectively.

The issues discussed 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. Recently, several auxiliary-variable approaches were proposed, including the alternating direction method of multipliers (ADMM) (Taylor et al., 2016; Zhang et al., 2016) and alternating-minimization, or block coordinate descent (BCD) methods (Carreira-Perpinan & Wang, 2014; Zhang & Brand, 2017; Zhang & Kleijn, 2017; Askari et al., 2018; Zeng et al., 2018; Lau et al., 2018). A similar formulation, using Lagrange multipliers, was proposed earlier in (LeCun, 1986; Yann, 1987; LeCun et al., 1988), where a constrained formulation involving activations required the output of the previous layer to be equal to the

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

input of the next layer. This gave rise to the approach known as target propagation and its recent extensions (Lee et al., 2015; Bartunov et al., 2018); however, target propagation is somewhat different method than the above BCD and ADMM methods, training instead layer-wise inverses of the forward mappings.

While the BCD and ADMM methods discussed above assume an offline (batch) setting, requiring the full training dataset at each iteration, we will focus instead on the *online*, incremental learning approach, performing *online* alternating minimization (AM) over the network weights and auxiliary activation variables, yielding scalability to arbitrarily large dataset, and applicability on incremental/online learning settings.

Herein, we introduce the general online alternating minimization framework, provide theoretical convergence analysis, and propose two specific algorithms, AM-Adam which uses Adam *locally* (no gradient chains) for weight updates in optimization subproblems at each layer, and AM-mem, based on the *surrogate function* approach similar to the online dictionary learning of (Mairal et al., 2009), which relies on accumulation of second-order information (activation covariances and cross-covariances). Both methods demonstrate promising empirical results, clearly outperforming the corresponding off-line alternating minimization, as well as closely related approach of (Taylor et al., 2016); we obseve, however, that AM-Adam consistently outperforms AM-mem; often, it also converges faster than Adam and SGD initially, while typically matching the two as learning progresses.

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 (in case of AM-Adam) needs no more auxiliary variables than the layer-wise activations, i.e. as memory-efficient as standard SGD which stores activations for gradient computations. We assume arbitrary loss functions and nonlinearities (unlike, for example, formulation of (Zhang & Brand, 2017) based on ReLU assumption). Moreover, our empirical evaluation extends beyond fully-connected networks, commonly used to evaluate auxiliary-variable methods, and includes CNNs (LeNet-5), RNNs and discrete (binary) networks, on several datasets including multi-million-sample HIGGS dataset.

In summary, our contributions are:

- *algorithm:* a novel online (mini-batch) auxiliary-variable approach for training neural networks without gradient chain rule of backprop; as opposed to prior offline auxiliary-variable algorithms, our online method is scalabile to arbitrarily large datasets and applicable to continual learning.
- *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 faster initial learning than Adam and SGD, followed by similar asymptotic performance.
- our online method inherits common advantages of similar offline auxiliary-variable methods, including (1) *no vanishing gradients*; (2) handling *non-differentiable nonlinearities* more easily in local subproblems; (3) *possibility for parallelization weight updates across layers*, and (4) potentially higher level of *biological plausibility* due to explicit neuronal activation propagation, as compared to backprop.

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 numbers of nodes in the 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:

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

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: $\mathcal{L}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{W}) = -\log P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{W})$

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

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-Perpinan & 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:

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

This problem can be solved by alternating minimization (AM), or block-coordinate descent (BCD), over the weights $\boldsymbol{W} = \{\boldsymbol{W}^1, ..., \boldsymbol{W}^{L+1}\}$ and the codes $\boldsymbol{C} = \{\boldsymbol{c}_1^1, ..., \boldsymbol{c}_1^L, ...\boldsymbol{c}_n^1, ..., \boldsymbol{c}_n^L, \}$. Each AM 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-Perpinan & Wang, 2014), we use an adaptive scheme for gradually increasing μ with the number of iterations[1].

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). Furthermore, offline AM is not suitable for *online learning*, where the input samples arrive incrementally, one at a time or in small mini-batches, from a potentially infinite data stream. This scenario is also typical in practical settings such as *continual* (or *lifelong*) learning (Ring, 1994; Thrun, 1995; 1998), which can be viewed as online learning from nonstationary inputs, e.g. a (potentially infinite) sequence of different tasks/datasets.

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 several implementation details such as 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); current sample is encoded in its representations at each layer (**encodeInput** procedure, Algorithm 2.2), and 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 $\boldsymbol{B_0}$).

```
1: while more samples do
2:    Input $(\boldsymbol{x}_t, y_t)$
3:    $\boldsymbol{C} \leftarrow$ encodeInput( $\boldsymbol{x}_t$, $\boldsymbol{W}_{t-1}$) % forward: compute linear activations at layers $1, ..., L$
4:    $\boldsymbol{C} \leftarrow$ updateCodes( $\boldsymbol{C}$, $y_t$, $\boldsymbol{W}_{t-1}$, $\mu$) % backward: error propagation by activation (code) changes
5:    $\boldsymbol{W}_t \leftarrow$ updateWeights( $\boldsymbol{W}_{t-1}$, $\boldsymbol{x}_t$, $y_t$, $\boldsymbol{C}$, $\mu$, $\eta$, $Mem$)
6: end while
7: return $\boldsymbol{W}_t$
```

Algorithm 2.2 Activation Propagation (Code Update) Steps

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

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

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

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

[1] 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); however, we tried it in our experiment and have not yet see much benefit; for clarity, we removed it from all formulations.

Algorithm 2.3 Weight and Memory Update Steps

```
updateWeights( $\boldsymbol{W}$, $\boldsymbol{x}$, $y$, $\boldsymbol{C}$, $\mu$, $\eta$, $Mem$)

1:  $\boldsymbol{W}^{L+1} = \boldsymbol{W}^{L+1} - \eta\nabla_{\boldsymbol{W}}\mathcal{L}(y, \sigma_L(\boldsymbol{c}^L), \boldsymbol{W}^{L+1})$
2:  for $l = 1$ to $L$ do
3:    if $Mem$ then
4:      $(\boldsymbol{A^l}_t, \boldsymbol{B^l}_t) \leftarrow$ updateMemory( $\boldsymbol{A^l}_{t-1}$, $\boldsymbol{B^l}_{t-1}$, $\boldsymbol{C^l}$)
5:      % $\hat{f}^l(\boldsymbol{W^l}) \equiv Tr(\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{A}^l) - 2Tr(\boldsymbol{W}^T\boldsymbol{B}^l)$
6:      $\boldsymbol{W}^l = \arg\min_W \hat{f}^l(\boldsymbol{W})$
7:    else
8:      %(parallel) local update of each layer weights,
9:      %(independently of other layers (unlike backprop)
10:     $\boldsymbol{W}^l \leftarrow$ SGD( $\boldsymbol{W}^l$, $\boldsymbol{x}$, $y$, $\boldsymbol{C}^l$, $\mu$, $\eta$)
11:   end if
12: end for
13: return $\boldsymbol{W}$

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

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 $\boldsymbol{c}^l_t$ using the weights computed so far. Namely, given input $\boldsymbol{x}_t$, we compute the last-layer activations $\boldsymbol{a}^L_t = \sigma_L(\boldsymbol{c}^L_t)$ in a forward pass, propagating activations from input to the last layer, and make a prediction about y_t, incurring the loss $\mathcal{L}(y_t, \boldsymbol{a}^L_t, \boldsymbol{W}^{L+1})$. 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, $\hat{f}_t(\boldsymbol{W}) = \sum_{l=1}^{L+1}\hat{f}^l_t(\boldsymbol{W}^l)$, which allows for *parallel weight optimization across all layers,* as follows:

$$\boldsymbol{W}^{L+1} = \arg\min_{\boldsymbol{W}} \left\{ \hat{f}^{L+1}_t(\boldsymbol{W}) \equiv \sum_{i=1}^{t} \mathcal{L}(y_i, \sigma_L(\boldsymbol{c}^L_i), \boldsymbol{W}) \right\}. \tag{6}$$

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

$$\boldsymbol{W}^l = \arg\min_{\boldsymbol{W}} \left\{ \hat{f}^l_t(\boldsymbol{W}) \equiv \mu \sum_{i=1}^{t} ||\boldsymbol{c}^l_i - \boldsymbol{W}\sigma_{l-1}(\boldsymbol{c}^{l-1}_i)||^2_2 \right\}. \tag{7}$$

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 2 in **memoryW**, 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. 7 using the following:

$$\sum_{i=1}^{t} ||\boldsymbol{c}^l_i - \boldsymbol{W}\boldsymbol{a}^l_i||^2_2 = Tr(\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{A}^l_t) - 2Tr(\boldsymbol{W}^T\boldsymbol{B}^l_t), \tag{8}$$

where $\boldsymbol{A}^l_t = \sum_{i=1}^t \boldsymbol{a}^{l-1}_i(\boldsymbol{a}^{l-1}_i)^T$ and $\boldsymbol{B}^l_t = \sum_{i=1}^t \boldsymbol{c}^l_i(\boldsymbol{a}^{l-1}_i)^T$ are the "memory" matrices (i.e. *co-activation memories*), compactly representing the accumulated strength of co-activations in each layer (matrices $\boldsymbol{A^l_t}$, i.e. covariances) and across consecutive layers (matrices $\boldsymbol{B^l_t}$, 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 $\boldsymbol{A_t} \leftarrow \boldsymbol{A_t} + \boldsymbol{a}^{l-1}_t(\boldsymbol{a}^{l-1}_t)^T$ and $\boldsymbol{B} \leftarrow \boldsymbol{B_t} + \boldsymbol{c}^l_t(\boldsymbol{a}^{l-1}_t)^T$.

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. 8, 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 till convergence, we used the number of such iterations as an additional hyperparameter.

3. Theoretical analysis

We next provide a theoretical convergence analysis for general alternating minimization (AM) schemes. Under certain assumptions that we will discuss, the proposed AM algorithm(s) falls into the category of approaches that comply with these guarantees, though the theory itself is more general and novel. *To the best of our knowledge, we provide the first theoretical convergence guarantees of AM in the stochastic setting*.

Setting. Let in general $\hat{f}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$ denote the function to be optimized using AM, where in the i^{th} step of the algorithm, we optimize $\hat{f}$ 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 $\hat{f}$ 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 $\hat{f}$ 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 $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K$.[2] Let $\nabla_{\boldsymbol{\theta}_i} \hat{f}^1$ denote the gradient of $\hat{f}$ computed for a single data sample $(\boldsymbol{x}, \boldsymbol{y})$ or code $\boldsymbol{c}$. In the next section, we refer to $\nabla_{\boldsymbol{\theta}_i} \hat{f}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$ as the gradient of $\hat{f}$ 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, the AM algorithm performs the update:

$$\boldsymbol{\theta}_i = \Pi_i(\boldsymbol{\theta}_i - \eta^\tau \nabla_{\boldsymbol{\theta}_i} \hat{f}^1(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)), \tag{9}$$

where 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 $i = 1, 2, \ldots, K$.

This scheme is *much more difficult* to prove theoretically and 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 more advantageous (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 dictionary. 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 run multiple times, we analyze more noisy version of parameter updates.

[2]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 $L(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K) = -\hat{f}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$ 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 $L_d^*(\boldsymbol{\theta}_d)$ and 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*

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

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*

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

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*

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

Next, we introduce the *population gradient AM operator*, $\mathcal{G}_i(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$, where $i = 1, 2, \ldots, K$, defined as

$$\mathcal{G}_i(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K) \coloneqq \boldsymbol{\theta}_i + \eta \nabla_{\boldsymbol{\theta}_i} \hat{f}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K),$$

where η is the step size. We also define the following bound σ on the expected value of the gradient of our objective function (a common assumption made in stochastic gradient descent convergence theorems as well). Define

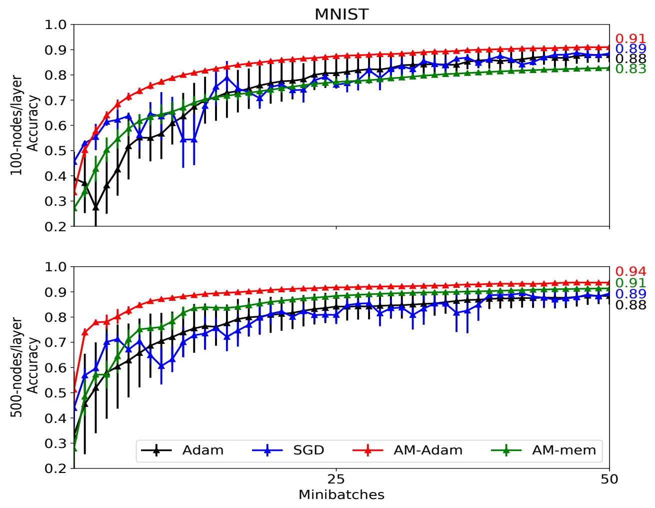

Figure 1. MNIST (fully-connected nets, 2 layers): online methods, first epoch; 50 mini-batches, 200 samples each.

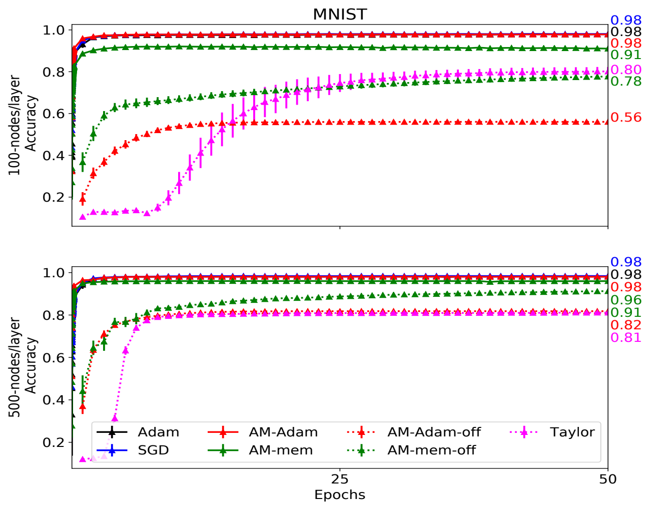

Figure 2. MNIST (fully-connected nets, 2 layers): online vs. offline methods vs. Taylor's ADMM, 50 epochs.

$\sigma = \sqrt{\sum_{d=1}^{K} \sigma_d^2}$ where

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

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 Algorithm 1 with decaying step size $\{\eta^t\}_{t=0}^{\infty}$, for any $d = 1, 2, \ldots, K$ and $\gamma < \frac{2\xi}{3(K-1)}$ the error $\boldsymbol{\Delta}_d^{t+1} := \boldsymbol{\theta}_d^{t+1} - \boldsymbol{\theta}_d^*$ at iteration $t+1$ satisfies recursion*

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

where $q^t = 1 - \frac{1-2\eta^t\xi+2\eta^t\gamma(K-1)}{1-(K-1)\eta^t\gamma}$.

The recursion in Theorem 3.1 is expanded in the Supplementary Material to prove the final convergence theorem for Algorithm 1 which states the following:

Theorem 3.2. *Given the stochastic AM gradient iterates of Algorithm 1 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 $\|\boldsymbol{\Delta}^{t+1}\|_2 := \sqrt{\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{t+1}\|_2^2} = \sqrt{\sum_{d=1}^{K} \|\boldsymbol{\theta}_d^{t+1} - \boldsymbol{\theta}_d^*\|_2^2}$ at iteration $t+1$ satisfies*

$$\mathbb{E}\left[\|\boldsymbol{\Delta}^{t+1}\|_2^2\right] \leq \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right]\left(\frac{2}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \frac{9}{[2\xi-3\gamma(K-1)]^2(t+3)}. \quad (11)$$

4. Experiments

We compared 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 the code provided by the authors[3], and against the two offline versions of our methods, *AM-Adam-off* and *AM-mem-off*, which simply treated the training dataset as a single minibatch, i.e. one AM iteration was 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-Perpinan & Wang, 2014; Taylor et al., 2016; Zhang et al., 2016; Zhang & Brand, 2017; Zeng et al., 2018; Askari et al., 2018), while *we also experimented with RNNs and CNNs, as well as with discrete (nondifferentiable) networks.*

Fully-connected nets: MNIST, CIFAR10, HIGGS. We experimented with fully-connected networks on the standard MNIST (LeCun, 1998) dataset, consisting of 28×28

[3]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-Perpinan & Wang, 2014) trained autoencoders, (Zhang et al., 2016) learned hashing.

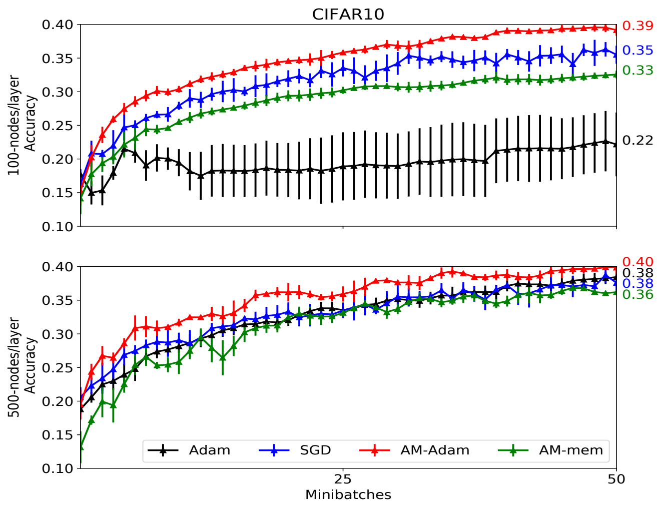

Figure 3. CIFAR10 (fully-connected nets): online methods, 1st epoch. 2 hidden layers with 100 (top) and 500 (bottom) units each; 250 mini-batches, 200 samples each.

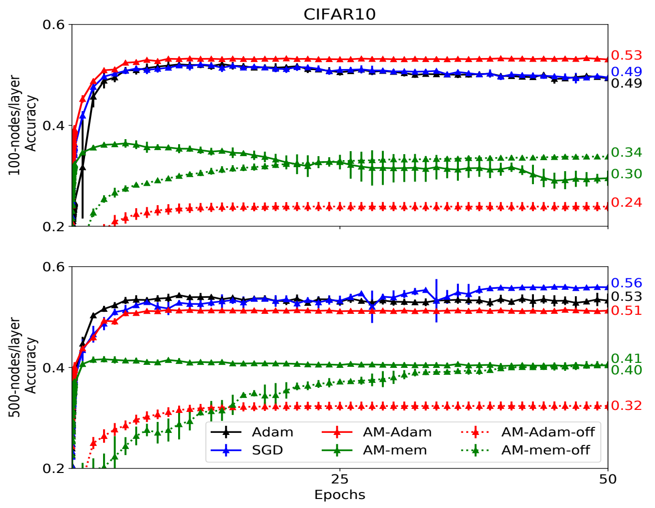

Figure 4. CIFAR10 (fully-connected networks): online vs. offline, 50 epochs. Similar experiments to Figure 2.

gray-scale images of hand-drawn digits, with 50K samples, and a test set of 10K samples. We evaluated two different 2-hidden-layer architectures, with equal hidden layer sizes of 100 and 500, and ReLU activations. Figure 1 zoomed-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, our *AM-Adam clearly dominates both 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* 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 100-node network; it is still inferior to all other methods on 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* outperforms both SGD and Adam on the first 50 minibatches (same size 200 as before), and even on 50 epochs for the 1-100 architecture, 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 vs0.53 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 will 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 again, vs. 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 minibatch of size 200, so one epoch over the 10.5M dataset corresponds to 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* version quickly reached 0.6 together with *AM-Adam*, but then slowed down, reaching only 0.61 on the 1st epoch).

In summary, our online *AM-Adam* on HIGGS greatly outperforms Taylor's offline ADMM (0.70 vs 0.64) on less than half of the 1st epoch, quickly reaching 0.64 benchmark on a tiny fraction (less than 0.01%) of the 10.5M dataset; moreover, our algorithm learns faster than SGD and on par/slightly faster than Adam initially (inset in Figure 5).

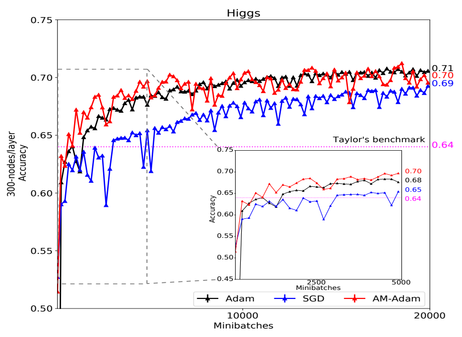

Figure 5. HIGGS dataset.

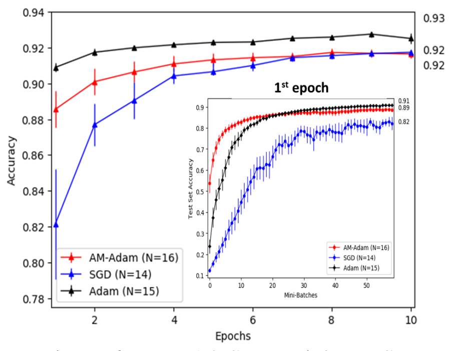

Figure 6. RNN-15, Sequential MNIST.

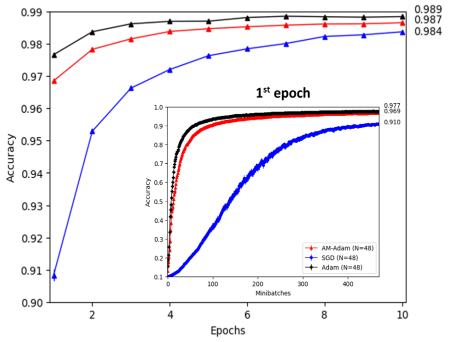

Figure 7. CNN: LeNet5, MNIST.

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 mini-batches 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. Again, *our Am-Adam learns faster than Adam (about half of the 1st epoch) and much faster than SGD up to epoch 6, and matches the latter afterwards.*

CNN (LeNet-5), MNIST. Next, we experimented with CNNs, using LeNet-5 (LeCun et al., 1998) on MNIST (Figure 7). Similarly to RNN result, our AM-Adam greatly outperforms SGD, while falling slightly behind Adam but catching up with it at 10 epochs, where our method and Adam reach 0.987 and 0.989 accuracy, respectively.

Binary nets (nondifferentiable activations), MNIST. Finally, to investigate the ability of our method to handle non-differentiable networks, we considered 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 is endowed with the non-differentiable $sign$ transfer function (while the second hidden layer uses $\tanh$). Target propagation family of approaches was motivated by the goal of finding more biologically plausible mechanisms for credit assignment in 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 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 it took about 200 epochs for DTP to reach 0.2 error, matching the STE performance (Figure 3 in (Lee et al., 2015)), our AM-Adam with binary activations reached same error within less than 20 epochs (Figure 4).*

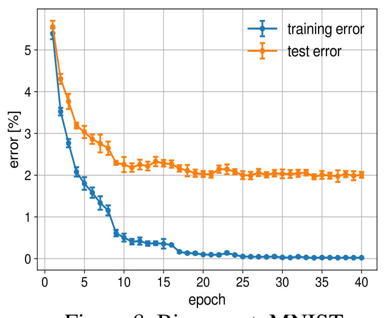

Figure 8. Binary net, MNIST.

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 thus vanishing gradients, allow weight update parallelization, handle non-differentiable nonlinearities, and are a step closer than backprop to a biologically plausible learning mechanism.

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-art backprop methods, AM-Adam learns faster initially, consistently outperforming SGD and Adam, and then matches, or nearly matches (and sometimes improves) their performance over multiple epochs. It also converges 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-Perpinan, 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.

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.

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. 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.

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.

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.

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

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

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.

Supplemental Material

A. Proofs

Proof of Theorem 3.2 relies on Theorem 3.1, which in turn relies on Theorem A.1 and Lemma A.1, both of which are stated below. Proofs of the lemma and theorems follow in the subsequent subsections.

The next result is a standard result from convex optimization (Theorem 2.1.14 in (Nesterov, 2014)) and is used in the proof of Theorem A.1 below.

Lemma A.1. *For any $d = 1, 2, \dots, K$, the gradient operator $\mathcal{G}_d(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \dots, \boldsymbol{\theta}_{d-1}^*, \boldsymbol{\theta}_d, \boldsymbol{\theta}_{d+1}^*, \dots, \boldsymbol{\theta}_{K-1}^*, \boldsymbol{\theta}_K^*)$ under Assumption 3.1 (strong concavity) and Assumption 3.2 (smoothness) with constant step size choice $0 < \eta \leq \frac{2}{\mu_d + \lambda_d}$ is contractive, i.e.*

$$\|\mathcal{G}_d(\boldsymbol{\theta}_1^*, \dots, \boldsymbol{\theta}_{d-1}^*, \boldsymbol{\theta}_d, \boldsymbol{\theta}_{d+1}^*, \dots, \boldsymbol{\theta}_K^*) - \boldsymbol{\theta}_d^*\|_2 \leq \left(1 - \frac{2\eta\mu_d\lambda_d}{\mu_d + \lambda_d}\right) \|\boldsymbol{\theta}_d - \boldsymbol{\theta}_d^*\|_2 \tag{12}$$

for all $\boldsymbol{\theta}_d \in B_2(r_d, \boldsymbol{\theta}_d^*)$.

The next theorem also holds for any d from 1 to K. Let $r_1, \dots, r_{d-1}, r_{d+1}, \dots, r_K > 0$ and $\boldsymbol{\theta}_1 \in B_2(r_1, \boldsymbol{\theta}_1^*), \dots, \boldsymbol{\theta}_{d-1} \in B_2(r_{d-1}, \boldsymbol{\theta}_{d-1}^*), \boldsymbol{\theta}_{d+1} \in B_2(r_{d+1}, \boldsymbol{\theta}_{d+1}^*), \dots, \boldsymbol{\theta}_K \in B_2(r_k, \boldsymbol{\theta}_K^*)$.

Theorem A.1. *For some radius $r_d > 0$ and a triplet $(\gamma_d, \lambda_d, \mu_d)$ such that $0 \leq \gamma_d < \lambda_d \leq \mu_d$, suppose that the function $L(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \dots, \boldsymbol{\theta}_{d-1}^*, \boldsymbol{\theta}_d, \boldsymbol{\theta}_{d+1}^*, \dots, \boldsymbol{\theta}_{K-1}^*, \boldsymbol{\theta}_K^*)$ is λ_d-strongly concave (Assumption 3.1) and μ_d-smooth (Assumption 3.2), and that the GS (γ_d) condition of Assumption 3.3 holds. Then the population gradient AM operator $\mathcal{G}_d(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_K)$ with step η such that $0 < \eta \leq \min_{i=1,2,\dots,K} \frac{2}{\mu_i + \lambda_i}$ is contractive over a ball $B_2(r_d, \boldsymbol{\theta}_d^*)$, i.e.*

$$\|\mathcal{G}_d(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_K) - \boldsymbol{\theta}_d^*\|_2 \leq (1 - \xi\eta)\|\boldsymbol{\theta}_d - \boldsymbol{\theta}_d^*\|_2 + \eta\gamma \sum_{\substack{i=1 \\ i \neq d}}^{K} \|\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^*\|_2 \tag{13}$$

where $\gamma := \max_{i=1,2,\dots,K} \gamma_i$, and $\xi := \min_{i=1,2,\dots,K} \frac{2\mu_i\lambda_i}{\mu_i + \lambda_i}$.

A.1. Proof of Theorem A.1

$\|\mathcal{G}_d(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_K) - \boldsymbol{\theta}_d^*\|_2 = \|\boldsymbol{\theta}_d + \eta\nabla_{\boldsymbol{\theta}_d} L(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \dots, \boldsymbol{\theta}_K) - \boldsymbol{\theta}_d^*\|_2$
by the triangle inequality we further get
$\leq \|\boldsymbol{\theta}_d + \eta\nabla_{\boldsymbol{\theta}_d} L(\boldsymbol{\theta}_1^*, \dots, \boldsymbol{\theta}_{d-1}^*, \boldsymbol{\theta}_d, \boldsymbol{\theta}_{d+1}^*, \dots, \boldsymbol{\theta}_K^*) - \boldsymbol{\theta}_d^*\|_2$
$+\eta\|\nabla_{\boldsymbol{\theta}_d} L(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_d, \dots, \boldsymbol{\theta}_K)$
$-\nabla_{\boldsymbol{\theta}_d} L(\boldsymbol{\theta}_1^*, \dots, \boldsymbol{\theta}_{d-1}^*, \boldsymbol{\theta}_d, \boldsymbol{\theta}_{d+1}^*, \dots, \boldsymbol{\theta}_K^*)\|_2$
by the contractivity of T from Equation 12 from Lemma A.1 and GS condition

$$\leq \left(1 - \frac{2\eta\mu_d\lambda_d}{\mu_d + \lambda_d}\right) \|\theta_d - \theta^*\|_2 + \eta\gamma_d \sum_{\substack{i=1 \\ i \neq d}}^{K} \|\theta_i - \theta_i^*\|_2.$$

A.2. Proof of Theorem 3.1

Let $\boldsymbol{\theta}_d^{t+1} = \Pi_d(\tilde{\boldsymbol{\theta}}_d^{t+1})$, where $\tilde{\boldsymbol{\theta}}_d^{t+1} := \boldsymbol{\theta}_d^t + \eta^t\nabla_{\boldsymbol{\theta}_d} L^1(\boldsymbol{\theta}_1^{t+1}, \boldsymbol{\theta}_2^{t+1}, \dots, \boldsymbol{\theta}_{d-1}^{t+1}, \boldsymbol{\theta}_d^t, \boldsymbol{\theta}_{d+1}^t, \dots, \boldsymbol{\theta}_K^t)$, where $\nabla_{\boldsymbol{\theta}_d} L^1$ is the gradient computed with respect to a single data sample, is the update vector prior to the projection onto a ball $B_2(\frac{r_d}{2}, \boldsymbol{\theta}_d^0)$. Let $\boldsymbol{\Delta}_d^{t+1} := \boldsymbol{\theta}_d^{t+1} - \boldsymbol{\theta}_d^*$ and $\tilde{\boldsymbol{\Delta}}_d^{t+1} := \tilde{\boldsymbol{\theta}}_d^{t+1} - \boldsymbol{\theta}_d^*$. Thus

$$\begin{aligned}
\|\boldsymbol{\Delta}_d^{t+1}\|_2^2 - \|\boldsymbol{\Delta}_d^t\|_2^2 &\leq \|\tilde{\boldsymbol{\Delta}}_d^{t+1}\|_2^2 - \|\boldsymbol{\Delta}_d^t\|_2^2 \\
&= \|\tilde{\boldsymbol{\theta}}_d^{t+1} - \boldsymbol{\theta}_d^*\| - \|\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*\| \\
&= \left\langle \tilde{\boldsymbol{\theta}}_d^{t+1} - \boldsymbol{\theta}_d^t, \tilde{\boldsymbol{\theta}}_d^{t+1} + \boldsymbol{\theta}_d^t - 2\boldsymbol{\theta}_d^* \right\rangle.
\end{aligned}$$

Let $\hat{\boldsymbol{W}}_d^t \coloneqq \nabla_{\boldsymbol{\theta}_d} L^1(\boldsymbol{\theta}_1^{t+1}, \boldsymbol{\theta}_2^{t+1}, \ldots, \boldsymbol{\theta}_{d-1}^{t+1}, \boldsymbol{\theta}_d^t, \boldsymbol{\theta}_{d+1}^t, \ldots, \boldsymbol{\theta}_K^t)$. Then we have that $\tilde{\boldsymbol{\theta}}_d^{t+1} - \boldsymbol{\theta}_d^t = \eta^t \hat{\boldsymbol{W}}_d^t$. We combine it with Equation 14 and obtain:

$$\begin{aligned}
& \|\boldsymbol{\Delta}_d^{t+1}\|_2^2 - \|\boldsymbol{\Delta}_d^t\|_2^2 \\
\leq \;& \left\langle \eta^t \hat{\boldsymbol{W}}_d^t, \eta^t \hat{\boldsymbol{W}}_d^t + 2(\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*) \right\rangle \\
= \;& (\eta^t)^2 (\hat{\boldsymbol{W}}_d^t)^\top \hat{\boldsymbol{W}}_d^t + 2\eta^t (\hat{\boldsymbol{W}}_d^t)^\top (\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*) \\
= \;& (\eta^t)^2 \|\hat{\boldsymbol{W}}_d^t\|_2^2 + 2\eta^t \left\langle \hat{\boldsymbol{W}}_d^t, \boldsymbol{\Delta}_d^t \right\rangle .
\end{aligned}$$

Let $\boldsymbol{W}_d^t \coloneqq \nabla_{\boldsymbol{\theta}_d} L(\boldsymbol{\theta}_1^{t+1}, \boldsymbol{\theta}_2^{t+1}, \ldots, \boldsymbol{\theta}_{d-1}^{t+1}, \boldsymbol{\theta}_d^t, \boldsymbol{\theta}_{d+1}^t, \ldots, \boldsymbol{\theta}_K^t)$. Recall that $\mathbb{E}[\hat{\boldsymbol{W}}_d^t] = \boldsymbol{W}_d^t$. By the properties of martingales, i.e. iterated expectations and tower property:

$$\mathbb{E}[\|\boldsymbol{\Delta}_d^{t+1}\|_2^2] \quad \leq \quad \mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2] + (\eta^t)^2 \mathbb{E}[\|\hat{\boldsymbol{W}}_d^t\|_2^2] + 2\eta^t \mathbb{E}[\langle \boldsymbol{W}_d^t, \boldsymbol{\Delta}_d^t \rangle] \tag{14}$$

Let $\boldsymbol{W}_d^* \coloneqq \nabla_{\boldsymbol{\theta}_d} L(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \ldots, \boldsymbol{\theta}_K^*)$. By self-consistency, i.e. $\boldsymbol{\theta}_d^* = \arg\max_{\boldsymbol{\theta}_d \in \Omega_d} L(\boldsymbol{\theta}_1^*, \ldots, \boldsymbol{\theta}_{d-1}^*, \boldsymbol{\theta}_d, \boldsymbol{\theta}_{d+1}^*, \ldots, \boldsymbol{\theta}_K^*)$ and convexity of Ω_d we have that

$$\langle \boldsymbol{W}_d^*, \boldsymbol{\Delta}_d^t \rangle = \langle \nabla_{\boldsymbol{\theta}_d} L(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \ldots, \boldsymbol{\theta}_K^*), \boldsymbol{\Delta}_d^t \rangle \leq 0.$$

Combining this with Equation 14 we have

$$\mathbb{E}[\|\boldsymbol{\Delta}_d^{t+1}\|_2^2] \quad \leq \quad \mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2] + (\eta^t)^2 \mathbb{E}[\|\hat{\boldsymbol{W}}_d^t\|_2^2] + 2\eta^t \mathbb{E}[\langle \boldsymbol{W}_d^t - \boldsymbol{W}_d^*, \boldsymbol{\Delta}_d^t \rangle].$$

Define $\mathcal{G}_d^t \coloneqq \boldsymbol{\theta}_d^t + \eta^t \boldsymbol{W}_d^t$ and $\mathcal{G}_d^{t*} \coloneqq \boldsymbol{\theta}_d^* + \eta^t \boldsymbol{W}_d^*$. Thus

$$\begin{aligned}
& \eta^t \langle \boldsymbol{W}_d^t - \boldsymbol{W}_d^*, \boldsymbol{\Delta}_d^t \rangle \\
= \;& \langle \mathcal{G}_d^t - \mathcal{G}_d^{t*} - (\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*), \boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^* \rangle \\
= \;& \langle \mathcal{G}_d^t - \mathcal{G}_d^{t*}, \boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^* \rangle - \|\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*\|_2^2
\end{aligned}$$

by the fact that $\mathcal{G}_d^{t*} = \boldsymbol{\theta}_d^* + \eta^t \boldsymbol{W}_d^* = \boldsymbol{\theta}_d^*$ (since $\boldsymbol{W}_d^* = 0$):

$$= \; \langle \mathcal{G}_d^t - \boldsymbol{\theta}_d^*, \boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^* \rangle - \|\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*\|_2^2$$

by the contractivity of $\mathcal{G}^t$ from Theorem A.1:

$$\begin{aligned}
\leq \;& \left\{ (1 - \eta^t \xi) \|\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*\| + \eta^t \gamma \left(\sum_{i=1}^{d-1} \|\boldsymbol{\theta}_i^{t+1} - \boldsymbol{\theta}_i^*\|_2 \; + \sum_{i=d+1}^{K} \|\boldsymbol{\theta}_i^t - \boldsymbol{\theta}_i^*\|_2 \right) \right\} \|\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*\|_2 - \|\boldsymbol{\theta}_d^t - \boldsymbol{\theta}_d^*\|_2^2 \\
\leq \;& \left\{ (1 - \eta^t \xi) \|\boldsymbol{\Delta}_d^t\|_2 + \eta^t \gamma \left(\sum_{i=1}^{d-1} \|\boldsymbol{\Delta}_i^{t+1}\|_2 + \sum_{i=d+1}^{K} \|\boldsymbol{\Delta}_i^t\|_2 \right) \right\} \cdot \|\boldsymbol{\Delta}_d^t\|_2 - \|\boldsymbol{\Delta}_d^t\|_2^2
\end{aligned}$$

Combining this result with Equation 15 gives

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{\Delta}_d^{t+1}\|_2^2] \quad \leq \quad & \mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2] + (\eta^t)^2 \mathbb{E}[\|\hat{\boldsymbol{W}}_d^t\|_2^2] + 2\mathbb{E}\left[\left\{ (1 - \eta^t \xi) \|\boldsymbol{\Delta}_d^t\|_2 + \eta^t \gamma \left(\sum_{i=1}^{d-1} \|\boldsymbol{\Delta}_i^{t+1}\|_2 + \sum_{i=d+1}^{K} \|\boldsymbol{\Delta}_i^t\|_2 \right) \right\} \right. \\
& \left. \cdot \|\boldsymbol{\Delta}_d^t\|_2 - \|\boldsymbol{\Delta}_d^t\|_2^2 \right] \\
\leq \; & \mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2] + (\eta^t)^2 \sigma_d^2 + 2\mathbb{E}\left[\left\{ (1 - \eta^t \xi) \|\boldsymbol{\Delta}_d^t\|_2 + \eta^t \gamma \left(\sum_{i=1}^{d-1} \|\boldsymbol{\Delta}_i^{t+1}\|_2 + \sum_{i=d+1}^{K} \|\boldsymbol{\Delta}_i^t\|_2 \right) \right\} \right. \\
& \left. \cdot \|\boldsymbol{\Delta}_d^t\|_2 - \|\boldsymbol{\Delta}_d^t\|_2^2 \right], \quad \text{where}
\end{aligned}$$

$$\sigma_d^2 = \sup_{\substack{\boldsymbol{\theta}_1 \in B_2(r_1, \boldsymbol{\theta}_1^*) \\ \cdots \\ \boldsymbol{\theta}_K \in B_2(r_K, \boldsymbol{\theta}_K^*)}} \mathbb{E}[\|\nabla_{\boldsymbol{\theta}_d} L^1(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)\|_2^2].$$

After re-arranging the terms we obtain

$$\mathbb{E}[\|\boldsymbol{\Delta}_d^{t+1}\|_2^2] \leq (\eta^t)^2\sigma_d^2 + (1-2\eta^t\xi)\mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2] + 2\eta^t\gamma\mathbb{E}\left[\left(\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^{t+1}\|_2 + \sum_{i=d+1}^{K}\|\boldsymbol{\Delta}_i^t\|_2\right)\|\boldsymbol{\Delta}_d^t\|_2\right]$$

apply $2ab \leq a^2+b^2$

and define $\mathbb{1}(x)=1$ for $x>0$ and $\mathbb{1}(x)=0$ otherwise:

$$\begin{aligned}
&\leq && (\eta^t)^2\sigma_d^2 + (1-2\eta^t\xi)\mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2] + \eta^t\gamma\mathbb{E}\left[\sum_{i=1}^{d-1}\left(\|\boldsymbol{\Delta}_i^{t+1}\|_2^2 + \mathbb{1}(d-1)\|\boldsymbol{\Delta}_d^t\|_2^2\right)\right]\\
&+ && \eta^t\gamma\mathbb{E}\left[\sum_{i=1}^{d-1}\left(\|\boldsymbol{\Delta}_i^t\|_2^2 + \mathbb{1}(K-d)\|\boldsymbol{\Delta}_d^t\|_2^2\right)\right]\\
&= && (\eta^t)^2\sigma_d^2 + \mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2]\cdot\left[1-2\eta^t\xi + \eta^t\gamma\left(\sum_{i=1}^{d-1}\mathbb{1}(d-1) + \sum_{i=d+1}^{K}\mathbb{1}(K-d)\right)\right]\\
&+ && \eta^t\gamma\mathbb{E}\left[\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^{t+1}\|_2^2\right] + \eta^t\gamma\mathbb{E}\left[\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^t\|_2^2\right]
\end{aligned}$$

We obtained

$$\mathbb{E}[\|\boldsymbol{\Delta}_d^{t+1}\|_2^2] \leq (\eta^t)^2\sigma_d^2 + [1-2\eta^t\xi+\eta^t\gamma(K-1)]\mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2] + \eta^t\gamma\mathbb{E}\left[\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^{t+1}\|_2^2\right] + \eta^t\gamma\mathbb{E}\left[\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^t\|_2^2\right]$$

we next re-group the terms as follows

$$\mathbb{E}[\|\boldsymbol{\Delta}_d^{t+1}\|_2^2] - \eta^t\gamma\mathbb{E}\left[\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^{t+1}\|_2^2\right] \leq [1-2\eta^t\xi+\eta^t\gamma(K-1)]\mathbb{E}[\|\boldsymbol{\Delta}_d^t\|_2^2] + \eta^t\gamma\mathbb{E}\left[\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^t\|_2^2\right] + (\eta^t)^2\sigma_d^2$$

and then sum over d from 1 to K

$$\begin{aligned}
& && \mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] - \eta^t\gamma\mathbb{E}\left[\sum_{d=1}^{K}\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^{t+1}\|_2^2\right]\\
&\leq && [1-2\eta^t\xi+\eta^t\gamma(K-1)]\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^t\|_2^2\right] + \eta^t\gamma\mathbb{E}\left[\sum_{d=1}^{K}\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^t\|_2^2\right] + (\eta^t)^2\sum_{d=1}^{K}\sigma_d^2
\end{aligned}$$

Let $\sigma = \sqrt{\sum_{d=1}^{K}\sigma_d^2}$. Also, note that

$$\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] - \eta^t\gamma(K-1)\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \leq \mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] - \eta^t\gamma\mathbb{E}\left[\sum_{d=1}^{K}\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^{t+1}\|_2^2\right]$$

and

$$\begin{aligned}
& [1-2\eta^t\xi+\eta^t\gamma(K-1)]\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^t\|_2^2\right] + \eta^t\gamma\mathbb{E}\left[\sum_{d=1}^{K}\sum_{i=1}^{d-1}\|\boldsymbol{\Delta}_i^t\|_2^2\right] + (\eta^t)^2\sigma^2\\
\leq\;\; & [1-2\eta^t\xi+\eta^t\gamma(K-1)]\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^t\|_2^2\right] + \eta^t\gamma(K-1)\mathbb{E}\left[\sum_{d=1}^{K}\|\Delta_d^t\|_2^2\right] + (\eta^t)^2\sigma^2
\end{aligned}$$

Combining these two facts with our previous results yields:

$$\begin{aligned}
&[1-(K-1)\eta^t\gamma]\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t+1}\|_2^2\right] \\
&\leq [1-2\eta^t\xi+\eta^t\gamma(K-1)]\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t}\|_2^2\right] + \eta^t\gamma(K-1)\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t}\|_2^2\right] + (\eta^t)^2\sigma^2 \\
&= [1-2\eta^t\xi+2\eta^t\gamma(K-1)]\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t}\|_2^2\right] + (\eta^t)^2\sigma^2
\end{aligned}$$

Thus:

$$\begin{aligned}
\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t+1}\|_2^2\right] \leq \frac{1-2\eta^t\xi+2\eta^t\gamma(K-1)}{1-(k-1)\eta^t\gamma}\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t}\|_2^2\right] \\
+ \frac{(\eta^t)^2}{1-(K-1)\eta^t\gamma}\sigma^2.
\end{aligned}$$

Since $\gamma < \frac{2\xi}{3(K-1)}$, $\frac{1-2\eta^t\xi+2\eta^t\gamma(K-1)}{1-(K-1)\eta^t\gamma} < 1$.

A.3. Proof of Theorem 3.2

To obtain the final theorem we need to expand the recursion from Theorem 3.1. We obtained

$$\begin{aligned}
&\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t+1}\|_2^2\right] \\
&\leq \frac{1-2\eta^t[\xi-\gamma(K-1)]}{1-(K-1)\eta^t\gamma}\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t}\|_2^2\right] + \frac{(\eta^t)^2}{1-(K-1)\eta^t\gamma}\sigma^2 \\
&= \left(1-\frac{\eta^t[2\xi-3\gamma(K-1)]}{1-(K-1)\eta^t\gamma}\right)\mathbb{E}\left[\sum_{d=1}^K \|\mathbf{\Delta}_d^{t}\|_2^2\right] + \frac{(\eta^t)^2}{1-(K-1)\eta^t\gamma}\sigma^2
\end{aligned}$$

Recall that we defined q^t in Theorem 3.1 as

$$q^t = 1-\frac{1-2\eta^t\xi+2\eta^t\gamma(K-1)}{1-(K-1)\eta^t\gamma} = \frac{\eta^t[2\xi-3\gamma(K-1)]}{1-(K-1)\eta^t\gamma}$$

and denote

$$\beta^t = \frac{(\eta^t)^2}{1-(K-1)\eta^t\gamma}.$$

Thus we have

$$\begin{aligned}
&\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \le (1-q^t)\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t}\|_2^2\right] + \beta^t\sigma^2 \\
\le\ & (1-q^t)\left\{(1-q^{t-1})\mathbb{E}\left[\sum_{d=1}^{K}\|\Delta_d^{t-1}\|_2^2\right] + \beta^{t-1}\sigma^2\right\} + \beta^t\sigma^2 \\
=\ & (1-q^t)(1-q^{t-1})\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t-1}\|_2^2\right] + (1-q^t)\beta^{t-1}\sigma^2 + \beta^t\sigma^2 \\
\le\ & (1-q^t)(1-q^{t-1})\left\{(1-q^{t-2})\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t-2}\|_2^2\right] + \beta^{t-2}\sigma^2\right\} + (1-q^t)\beta^{t-1}\sigma^2 + \beta^t\sigma^2 \\
=\ & (1-q^t)(1-q^{t-1})(1-q^{t-2})\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t-2}\|_2^2\right] \\
& +(1-q^t)(1-q^{t-1})\beta^{t-2}\sigma^2 + (1-q^t)\beta^{t-1}\sigma^2 + \beta^t\sigma^2
\end{aligned}$$

We end-up with the following

$$\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \le \mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{0}\|_2^2\right]\prod_{i=0}^{t}(1-q^i) + \sigma^2\sum_{i=0}^{t-1}\beta^i\prod_{j=i+1}^{t}(1-q^j) + \beta^t\sigma^2.$$

Set $q^t = \frac{\frac{3}{2}}{t+2}$ and

$$\begin{aligned}
\eta^t &= \frac{q^t}{2\xi - 3\gamma(K-1) + q^t(K-1)\gamma} \\
&= \frac{\frac{3}{2}}{[2\xi - 3\gamma(K-1)](t+2) + \frac{3}{2}(K-1)\gamma}.
\end{aligned}$$

Denote $A = 2\xi - 3\gamma(K-1)$ and $B = \frac{3}{2}(K-1)\gamma$. Thus

$$\eta^t = \frac{\frac{3}{2}}{A(t+2)+B}$$

and

$$\beta^t = \frac{(\eta^t)^2}{1-\frac{2}{3}B\eta^t} = \frac{\frac{9}{4}}{A(t+2)[A(t+2)+B]}.$$

$$\begin{aligned}
&\mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \\
\le\ & \mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{0}\|_2^2\right]\prod_{i=0}^{t}\left(1-\frac{\frac{3}{2}}{i+2}\right) + \sigma^2\sum_{i=0}^{t-1}\frac{\frac{9}{4}}{A(i+2)[A(i+2)+B]}\prod_{j=i+1}^{t}\left(1-\frac{\frac{3}{2}}{j+2}\right) \\
& +\sigma^2\frac{\frac{9}{4}}{A(t+2)[A(t+2)+B]} \\
=\ & \mathbb{E}\left[\sum_{d=1}^{K}\|\boldsymbol{\Delta}_d^{0}\|_2^2\right]\prod_{i=2}^{t+2}\left(1-\frac{\frac{3}{2}}{i}\right) + \sigma^2\sum_{i=2}^{t+1}\frac{\frac{9}{4}}{Ai[Ai+B]}\prod_{j=i+1}^{t+2}\left(1-\frac{\frac{3}{2}}{j}\right) + \sigma^2\frac{\frac{9}{4}}{A(t+2)[A(t+2)+B]}
\end{aligned}$$

Since $A > 0$ and $B > 0$ thus

$$\begin{aligned}
& \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \\
\leq\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \prod_{i=2}^{t+2}\left(1-\frac{\frac{3}{2}}{i}\right) + \sigma^2 \sum_{i=2}^{t+1} \frac{\frac{9}{4}}{Ai[Ai+B]} \prod_{j=i+1}^{t+2}\left(1-\frac{\frac{3}{2}}{j}\right) + \sigma^2 \frac{\frac{9}{4}}{A(t+2)[A(t+2)+B]} \\
\leq\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \prod_{i=2}^{t+2}\left(1-\frac{\frac{3}{2}}{i}\right) + \sigma^2 \sum_{i=2}^{t+1} \frac{\frac{9}{4}}{(Ai)^2} \prod_{j=i+1}^{t+2}\left(1-\frac{\frac{3}{2}}{j}\right) + \sigma^2 \frac{\frac{9}{4}}{[A(t+2)]^2}
\end{aligned}$$

We can next use the fact that for any $a \in (1, 2)$:

$$\prod_{i=\tau+1}^{t+2}\left(1-\frac{a}{i}\right) \leq \left(\frac{\tau+1}{t+3}\right)^a .$$

The bound then becomes

$$\begin{aligned}
& \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \\
\leq\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \prod_{i=2}^{t+2}\left(1-\frac{\frac{3}{2}}{i}\right) + \sigma^2 \sum_{i=2}^{t+1} \frac{\frac{9}{4}}{(Ai)^2} \prod_{j=i+1}^{t+2}\left(1-\frac{\frac{3}{2}}{j}\right) + \sigma^2 \frac{\frac{9}{4}}{[A(t+2)]^2} \\
\leq\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \left(\frac{2}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \sum_{i=2}^{t+1} \frac{\frac{9}{4}}{(Ai)^2} \left(\frac{i+1}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \frac{\frac{9}{4}}{[A(t+2)]^2} \\
=\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \left(\frac{2}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \sum_{i=2}^{t+2} \frac{\frac{9}{4}}{(Ai)^2} \left(\frac{i+1}{t+3}\right)^{\frac{3}{2}}
\end{aligned}$$

Note that $(i+1)^{\frac{3}{2}} \leq 2i$ for $i = 2, 3, \ldots$, thus

$$\begin{aligned}
& \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{t+1}\|_2^2\right] \\
\leq\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \left(\frac{2}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \frac{\frac{9}{4}}{A^2(t+3)^{\frac{3}{2}}} \sum_{i=2}^{t+2} \frac{(i+1)^{\frac{3}{2}}}{i^2} \\
\leq\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \left(\frac{2}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \frac{\frac{9}{2}}{A^2(t+3)^{\frac{3}{2}}} \sum_{i=2}^{t+2} \frac{1}{i^{\frac{1}{2}}} \\
& \text{finally note that} \sum_{i=2}^{t+2} \frac{1}{i^{\frac{1}{2}}} \leq \int_1^{t+2} \frac{1}{x^{\frac{1}{2}}} dx \leq 2(t+3)^{\frac{1}{2}}. \text{ Thus} \\
\leq\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \left(\frac{2}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \frac{9}{A^2(t+3)} \\
& \text{substituting } A = 2\xi - 3\gamma(K-1) \text{ gives} \\
=\ & \mathbb{E}\left[\sum_{d=1}^{K} \|\boldsymbol{\Delta}_d^{0}\|_2^2\right] \left(\frac{2}{t+3}\right)^{\frac{3}{2}} + \sigma^2 \frac{9}{[2\xi - 3\gamma(K-1)]^2(t+3)}
\end{aligned}$$

This leads us to the final theorem.

B. CNNs experiments: details

We compare SGD, Adam, and AM-Adam on the LeNet-5(LeCun et al., 1998) architecture on both MNIST and Fashion-MNIST (Xiao et al., 2017) datasets.

Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

We fix the batchsize to 128, and run a hyperparameter grid search for each algorithm and dataset using the following values: weight-learning rates of 2e-M for M=2,3,4,5; batch-wise mu-increments of 1e-2,1e-5, 1e-7; epoch-wise mu-multipliers of 1, 1,1; code learning-rates of 0.1, 1 (note: only weight learning rates are varied for SGD and Adam). SGD was allowed a standard epoch-wise learning rate decay of 0.9. AM-Adam used only one subproblem iteration (both codes and weights) for each minibatch, an initial μ value of 0.01, and a maximum μ value of 1.5. In total, six total grid searches were performed.

For each hyperparameter combination, each algorithm was run on at least 5 initializations, training for 10 epochs on 5/6 of the training dataset. The mean final accuracy on the validation set (the remaining 1/6 of the training dataset) was used to select the best hyperparameters.

Finally, each algorithm with its best hyperparameters on each dataset was used to re-train Lenet-? with N intializations, this time evaluated on the test set. The mean performances are plotted in Figures ?? for MNIST and Figures ?? for Fashion-MNIST.

The winning hyperparameters for Fashion-MNIST are: Adam: LR=0.002 SGD: LR=0.02 AM: weight-LR= 0.002; code-LR= 1.0; batchwise μ-increment=1e-5; epochwise μ-multiplier=1.1

The winning hyperparameters for MNIST are: Adam: LR=0.002 SGD: LR=0.02 AM: weight-LR= 0.002; code-LR= 1.0; batchwise μ-increment=1e-7; epochwise μ-multiplier=1.1

C. RNN experiments: details

C.1. Architecture and AM Adaptation

We also compare SGD, Adam, and AM-Adam on a standard Elman RNN architecture. That is a recurrent unit that, at time t, yields an output z^t and hidden state h^t based on a combination of input x^t and the previous hidden state h^{t-1}, for $t = 1, ..., T$. The equations for the unit are:

$$h^t = \sigma\{\mathbf{U}x^t + \mathbf{W}h^{t-1} + \mathbf{b}\} \tag{15}$$

$$z^t = \mathbf{V}h^t, \tag{16}$$

where $\mathbf{b}$ is a bias, σ is a *tanh* activation function, and $\mathbf{U} \in \mathbb{R}^{d\times 1}, \mathbf{W} \in \mathbb{R}^{d\times d}$, and $\mathbf{V} \in \mathbb{R}^{1\times d}$ are learnable parameter matrices that do not vary with t. Denote with m the length of one sequence element, so $x^t, z^t \in \mathbb{R}^m$. Then let d be the number of hidden units, so $h^t \in \mathbb{R}^d$.

We train this architecture to classify MNIST digits where each image is vectorized and fed to the RNN as a sequence of $T = 784$ pixels (termed "Sequential MNIST" in (Le et al., 2015)). Thus for each t, the input x^t is a single pixel. A final matrix $\mathbf{C}$ is then used to classify the output sequence z^t using the same multinomial loss function as before:

$$\sum_n \mathcal{L}(y_n, ReLU(\mathbf{z}_n), \mathbf{C}), \tag{17}$$

where $\mathbf{z}_n = [z_n^1, ..., z_n^{784}]^{\mathrm{T}}$ is the output sequence for the n^{th} training sample, and $\mathbf{C} \in \mathbb{R}^{10\times 784}$. In summary, the prediction is made only after processing all 784 pixels.

To train this family of architectures using Alt-Min, we introduce two sets of auxiliary variables (codes). First, we introduce a code for each element of the sequence just before input to the activation function:

$$c^t = \mathbf{U}x^t + \mathbf{W}h^{t-1} + \mathbf{b} \tag{18}$$

where c^t is the internal RNN code at time t. Using the "unfolded" interpretation of an RNN, we have introduced a code between each repeated "layer". Second, we treat the output sequence $\mathbf{z}$ as an auxiliary variable in order to break the gradient chain between the loss function and the recurrent unit.

C.2. Experiments

We compare SGD, Adam, and AM-Adam on the Elman RNN architecture with hidden sizes $d = 15$ and $d = 50$ on the Sequential MNIST dataset. We fix the batchsize to 1024, and run a hyperparameter grid search for each algorithm using the following values: weight-learning rates of 5e-M, for M=1,2,3,4,5 (all methods); weight sparsity = 0, 0.01, 0.1 (SGD and Adam); batch-wise mu-increment 1e-M for M=2,3,4; epoch-wise mu-multiplier for 1, 1.1, 1.25, 1.5; mu-max=1, 5. SGD was allowed a standard learning-rate-decay of 0.9. AM-Adam used an initial μ value of 0.01, and used 5 subproblem iterations for both code and weight optimization subproblems.

Note: in an offline hand-tuning search, we determined that weight-sparsity only hurt Alt-Min, so it was not included in official the grid search. Also note that a larger batchsize is used for the RNN experiments because of the relatively strong dependence of the training time on batchsize. This dependence is because for each minibatch, a series of loops though $t = 1, ..., 784$ are required.

For each hyperparameter combination, each algorithm was run on at least 3 initializations, training for 10 epochs on 5/6 of the training dataset. The mean final accuracy on the validation set (the remaining 1/6 of the training dataset) was used to select the best hyperparameters.

Finally, each algorithm with its best hyperparameters on each dataset was used to re-train the Elman RNN with N intializations, this time evaluated on the test set.

The winning hyperparameters for *d=15* are: Adam: learning rate = 0.005, L1=0; SGD: learning rate = 0.05, L1=0; AM-Adam: learning rate = 0.005, max-mu=1, mu-multiplier=1.1, mu-increment=0.01. Results are depicted in Figure 6

The winning hyperparameters for *d=50* are: Adam: learning rate = 0.005, L1=0.01; SGD: learning rate = 0.005, L1=0; AM-Adam: learning rate = 0.005, max-mu=1, mu-multiplier=1.0, mu-increment=0.0001. Results are depicted in Figure 9.

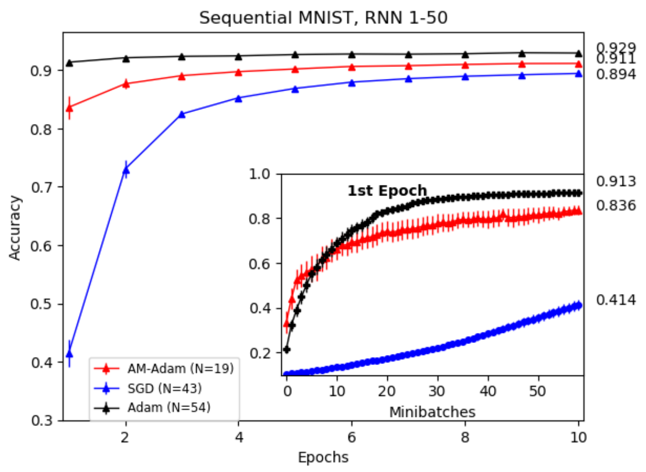

Figure 9. RNN-50, Sequential MNIST.

D. Fully connected networks: details

Performance of the online (i.e., SGD, Adam, AM-Adam, AM-mem) and offline (i.e., AM-Adam-off, AM-mem-off, Taylor) methods are compared on the MNIST and CIFAR-10 datasets for two fully connected network architectures with two identical hidden layers of 100 and 500 units each. We also consider a different architecture with one hidden layer of 300 units for the larger HIGGS dataset. Optimal hyperparameters are reported below for each set of experiments.

D.1. MNIST Experiments

The standard MNIST training dataset is split into a reduced training set (first 50,000 samples) and a validation set (last 10,000 samples) for hyperparameter optimization. More specifically, an iterative bayesian optimization scheme is used to find the optimal learning rates (lr) maximizing classification accuracy on the validation set after 50 epochs of training. Rather than learning rates, for Taylor's method we optimize the γ_{prod} and γ_{nonlin} parameters. The procedure is repeated for five different weight initializations and for both architectures considered. Table 1 reports hyperparameters yielding the highest accuracy among the 5 weight initializations.

Algorithm	Hidden units per layer	lr	γ_{prod}	γ_{nonlin}
Adam	100	0.0210		
Adam	500	0.0005		
SGD	100	0.2030		
SGD	500	0.1497		
AM-Adam	100	0.1973		
AM-Adam	500	0.1171		
AM-mem	100	0.1737		
AM-mem	500	0.1376		
AM-Adam-off	100	0.5003		
AM-Adam-off	500	0.4834		
AM-mem-off	100	0.4664		
AM-mem-off	500	0.2503		
Taylor	100		582.8	54.15
Taylor	500		444.2	111.7

Table 1. Optimal hyperparameters for fully connected networks on MNIST

D.2. CIFAR-10 Experiments

Similary to what done for the MNIST dataset, we split the standard CIFAR-10 training dataset into a reduced training set (first 40,000 samples) and a validation set (last 10,000 samples) used to evaluate accuracy for hyperparameter optimization. Table 2 reports hyperparameters for all the methods yielding the highest accuracy among the 5 weight initializations. Since not included in the original publication, we do not consider Taylor's method on this dataset.

Algorithm	Hidden units per layer	lr
Adam	100	0.0029
Adam	500	0.0002
SGD	100	0.1500
SGD	500	0.1428
AM-Adam	100	0.1974
AM-Adam	500	0.1011
AM-mem	100	0.1746
AM-mem	500	0.1016
AM-Adam-off	100	0.5000
AM-Adam-off	500	0.4844
AM-mem-off	100	0.2343
AM-mem-off	500	0.2277

Table 2. Optimal hyperparameters for fully connected networks on CIFAR-10

D.3. HIGGS Experiments

For the Higgs experiment, we compare only our best performing AM-Adam online method to Adam and SGD. Also, due to the increased computational costs associated to this dataset, we consider only one weight initialization and replace the bayesian optimization scheme with a simpler grid search. Table 3 reports the hyperparameters yielding the highest accuracy.

Algorithm	Hidden units per layer	lr
Adam	300	0.001
SGD	300	0.050
AM-Adam	300	0.001

Table 3. Hyperparameters used for fully connected networks on HIGGS