Decoupled Greedy Learning of CNNs

Eugene Belilovsky ¹  Michael Eickenberg ²  Edouard Oyallon ³

Abstract

A commonly cited inefficiency of neural network training by back-propagation is the update locking problem: each layer must wait for the signal to propagate through the network before updating. We consider and analyze a training procedure, Decoupled Greedy Learning (DGL), that addresses this problem more effectively and at scales beyond those of previous solutions. It is based on a greedy relaxation of the joint training objective, recently shown to be effective in the context of Convolutional Neural Networks (CNNs) on large-scale image classification. We consider an optimization of this objective that permits us to decouple the layer training, allowing for layers or modules in networks to be trained with a potentially linear parallelization in layers. We show theoretically and empirically that this approach converges. In addition, we empirically find that it can lead to better generalization than sequential greedy optimization and even standard end-to-end back-propagation. We show that an extension of this approach to asynchronous settings, where modules can operate with large communication delays, is possible with the use of a replay buffer. We demonstrate the effectiveness of DGL on the CIFAR-10 datasets against alternatives and on the large-scale ImageNet dataset, where we are able to effectively train VGG and ResNet-152 models.

1. Introduction

Training jointly all layers using back-propagation is the standard method for learning neural networks, including the computationally intensive vision models based on Convolutional Neural Networks (CNNs) (Goyal et al., 2017). Due to the sequential nature of the gradient processing, standard back-propagation has several well-known inefficiencies that do not permit the computations of the different constituent modules to be parallelized. (Jaderberg et al., 2017) characterizes these in order of severity as the forward, update, and backward locking problems. Backward unlocking would permit updates of all modules once signals have propagated to all subsequent modules, update unlocking would permit updates of a module before a signal has reached all subsequent modules, and forward unlocking would permit a module to operate asynchronously from its predecessor and dependent modules.

Multiple methods have been proposed, which can deal up to a certain degree with the backward unlocking challenge (Huo et al., 2018b; Choromanska et al., 2018; Nø kland, 2016). Jaderberg et al. (2017); Czarnecki et al. (2017) propose and analyze DNI, a method that addresses the more challenging update locking. The DNI approach uses an auxiliary network to predict the gradient of the backward pass directly from the input. This method is not shown to scale well computationally or in terms of accuracy, especially in the case of CNNs (Huo et al., 2018b;a). Indeed, auxiliary networks must predict a weight gradient that can be very large in dimensionality, which can be inaccurate and challenging to scale when intermediate representations are large, as is the case for larger models and input image sizes.

Recently, several authors have revisited the classic (Ivakhnenko & Lapa, 1965; Bengio et al., 2007) approach of supervised greedy layer-wise training of neural networks (Huang et al., 2018; Marquez et al., 2018). In Belilovsky et al. (2018) it is shown that such an approach, which relaxes the joint learning objective, can lead to high performance deep CNNs on large-scale datasets. Some of these works also consider the use of auxiliary networks with hidden layers as part of the local auxiliary problems which has some analogs to the auxiliary networks of DNI and target propagation (Lee et al., 2014). We will show that the greedy learning objective can be solved with an alternative optimization algorithm, which permits decoupling the computations and achieving update unlocking. This can also be augmented with replay buffers (Lin, 1992) to permit forward unlocking. This strategy can be shown to be a state-of-the-art baseline for parallelizing the training across modules of a neural network.

Our contributions in this work are as follows. We (a) pro-
pose an optimization procedure for a decoupled greedy learning objective that solves the update locking problem. 

(b) Empirically, we show that it exhibits similar convergence rates and generalization as its non-decoupled counterpart. 

(c) We show that it can be extended to an asynchronous setting by use of a replay buffer, providing a step towards addressing the forward locking problem. 

(d) We motivate these observations theoretically, showing that the proposed optimization procedure converges and recovers standard rates of non-convex optimization. Experimentally we (e) design an improved auxiliary network structure for greedy layer-wise training of CNNs that permits to maintain accuracy while having negligible cost for the auxiliary task. We (f) show that the decoupled greedy learning can well outperform competing methods in terms of scalability to larger and deeper models and stability to optimization hyper-parameters, allowing it to be applied to large datasets. We then demonstrate on the ImageNet dataset that we can train the deep models VGG-19 and ResNet-152 with larger degrees of parallelism than other proposals and reduced memory consumption. Code for experiments will be made available.

2. Related work

To the best of our knowledge (Jaderberg et al., 2017) is the only work which directly addresses the update or forward locking problems in deep feed-forward networks. Other works (Huo et al., 2018a; b) consider the backward locking problem, furthermore a number of back-propagation alternatives such as (Choromanska et al., 2018; Lee et al., 2014; Nø kland, 2016) are also able to address this problem. However, update locking is a more severe inefficiency. Consider the case where each layer’s forward processing time is \( T_F \) and is equal across a network of \( L \) layers. Given that the backward pass of back-propagation is a constant multiple in time of the forward pass, in the most ideal case the backward unlocking will only scale as \( O(LT_F) \) with \( L \) parallel nodes, while update unlocking could scale as \( O(T_F) \).

One class of alternatives to standard back-propagation aims to avoid its biologically implausible aspects, most notably the weight transport problem (Bartunov et al., 2018; Nø kland, 2016; Lillicrap et al., 2014; Lee et al., 2014). Some of these methods (Lee et al., 2014; Nø kland, 2016) can also achieve backward unlocking as they permit all parameters to be updated at the same time, but only once the signal has propagated to the top layer. None of them however solve the update locking problem or forward locking problem which we consider. Target propagation uses a local auxiliary network as in our approach, which is used to propagate backward the optimal activations computed from the layer above. Feedback alignment replaces the symmetric weights of the backward pass with random weights. Direct feedback alignment extends the idea of feedback alignment passing errors from the top to all layers, potentially permitting a simultaneous update. These approaches have also not been shown to be scalable to large datasets (Bartunov et al., 2018), obtaining only 17.5% top-5 accuracy on ImageNet (for a reference model that achieves 59.8%). On the other hand a greedy learning strategy has been shown to work well on the same task (Belilovsky et al., 2018).

Another line of related work inspired by optimization methods such as Alternating Direction Method of Multipliers (ADMM) (Taylor et al., 2016; Carreira-Perpinan & Wang, 2014; Choromanska et al., 2018) considers approaches that use auxiliary variables to break optimization into sub-problems. These approaches are fundamentally different from ours as they optimization for the joint training objective, the auxiliary variables providing a link between a layer and it’s successive layers, whereas we consider a different objective where a layer has no dependence on its successors. None of these methods can achieve update or forward unlocking, however some (Choromanska et al., 2018) are able to have a simultaneous weight updates (backward unlocked). Another issue with these methods is that most of the existing approaches except for Choromanska et al. (2018) require standard (“batch”) gradient descent and are thus difficult to scale. They also often involve an inner minimization problem and have thus not been demonstrated to work on realistic large scale datasets. Furthermore, none of these have been combined with CNNs.

Distributed optimization based on data parallelism is a popular area of research in machine learning beyond deep learning models and often studied in the convex setting (Leblond et al., 2018). In deep network optimization the predominant method is distributed synchronous SGD (Goyal et al., 2017) and variants, as well as asynchronous (Zhang et al., 2015) variants. Our approach on the other hand can be seen as closer to exploiting a type of model parallelism vs data parallelism and can be easily combined with many of these methods, particularly distributed synchronous SGD.

3. Parallel Decoupled Greedy Learning

In this section we formally define the greedy objective and parallel optimization which we study in both the synchronous and asynchronous setting. We mainly consider the online setting and assume a stream of samples or mini-batches denoted \( S \triangleq \{(x_i^t, y_i^t)\}_{i \leq T} \), that can be run during \( T \) iterations.

3.1. Optimization for Greedy Objective

Let \( X_0 \) and \( Y \) be the data matrix and labels for the training data. Let \( X_j \) be the output representation for module \( j \). We will denote the per-module objective function \( \mathcal{L}(X_j, Y; \theta_j, \gamma_j) \), where the parameters \( \theta_j \) correspond to the module parameter (i.e. \( X_{j+1} = f_{\theta_j}(X_j) \)), \( \gamma_j \) corre-
We consider an extension of this framework that addresses we can attempt an extension that allows to also remove some or substantial variations in speeds between layers/modules. It by the use of a replay buffer that is shared between adja-
tations of the modules in DGL are only loosely dependent, such that this can operate asynchronously. This is achieved
3.2. Asynchronous Decoupled Greedy Learning with
Replay
We will evaluate one instance of such an algorithm based on the use of a replay buffer of size $M$, shown in Alg. 2. Here each module maintains a buffer to which it writes its output representations, which is read by the module above.

Our minimal distributed setting is as follows. Each worker $j$ has a buffer that it writes to and that worker $j + 1$ can read from. The buffer uses a simple read and write protocol. A buffer $B_j$ permits layer $j$ to write new samples. When it reaches capacity it overwrites the oldest sample. Layer $j + 1$ requests samples from the buffer $B_j$. The sample is selected by a simple last-in-first-out (LIFO) rule, with a precedence for the least-reused samples. The speed of the worker is constant, yet can be potentially different across workers and is also subject to small random fluctuations. Our algorithm does not require a shared buffer across all workers, but only across pairs of workers. Alg. 2 simulates potential delays in such a setup by the use of a probability mass function (pmf) $p(j)$ over workers analogous to typical asynchronous settings such as (Leblond et al., 2017). At each iteration a layer is chosen at random according to $p(j)$ to perform computation. In our experiments we will limit ourselves to pmfs that are uniform over workers except for a single layer which is chosen to be selected less frequently on average. We note even in the case of a uniform pmf, asynchronous behavior will naturally arise requiring the

**Algorithm 1: Decoupled Greedy Learning**

| Input: Stream $S \triangleq \{(x^0_j, y^j)\}_{j \leq T}$ of samples or mini-batches |
|---|
| 1 Initialize Parameters $(\theta_j, \gamma_j)_{j \leq J}$ |
| 2 for $(x^0_j, y^j) \in S$ do |
| 3 for $j \in 1, ..., J$ do |
| 4 $x^j_j \leftarrow f_{\theta_{j-1}}(x_{j-1}^j) |
| 5 Compute $\nabla_{(\gamma_j, \theta_j)} \hat{L}(y^j, x_j^j, \gamma_j, \theta_j)$ |
| 6 $(\theta_j, \gamma_j) \leftarrow$ Update parameters $(\theta_j, \gamma_j)$ |
| 7 end |
| 8 end |

**Algorithm 2: Asynchronous DGL with Replay Buffer**

| Input: Stream $S \triangleq \{(x^0_j, y^j)\}_{j \leq T}$; Distribution of the delay $p = \{p(j)\}_j$; Buffer size $M$ |
|---|
| 1 Initialize: Buffers $\{B_j\}_j$ with size $M$; params $(\theta_j, \gamma_j)_j$ |
| 2 while training do |
| 3 Sample $j \in \{1, ..., J\}$ following $p$ |
| 4 if $j = 1$ then $(x_0, y) \leftarrow S$ else $(x_{j-1}, y) \leftarrow B_{j-1}$ |
| 5 $x_j \leftarrow f_{\theta_{j-1}}(x_{j-1}) |
| 6 Compute $\nabla_{(\gamma_j, \theta_j)} \hat{L}(y, x_j, \gamma_j, \theta_j)$ |
| 7 $(\theta_j, \gamma_j) \leftarrow$ Update parameters $(\theta_j, \gamma_j)$ |
| 8 if $j < J$ then $B_j \leftarrow (x^j, y)$ |
| 9 end |

---

Fig. 1 illustrates the decoupling compared to how samples are processed in standard back-propagation. We observe that once a $x_j^j$ has been computed, processing by subsequent layers can begin. Sec. 3.2 will also consider a version of the algorithm that can be made asynchronous by introducing a replay buffer.

3.2. Asynchronous Decoupled Greedy Learning with
Replay
We consider an extension of this framework that addresses forward unlocking (Jaderberg et al., 2017). Since the computations of the modules in DGL are only loosely dependent, we can attempt an extension that allows to also remove some dependency of the computations on the previous modules such that this can operate asynchronously. This is achieved by the use of a replay buffer that is shared between adjacent modules and allows modules to reuse older samples. It can be beneficial in scenarios with communication delays or substantial variations in speeds between layers/modules.

**Fig. 1.** We illustrate the signal propagation for three mini-batches processed by standard back-propagation and with decoupled greedy learning. In each case a module can begin processing forward and backward passes as soon as possible. For illustration we assume same speed for forward and backward passes, and discount the auxiliary network computation (negligible in our experiments).
We note although we focus on the distributed learning con-
Although auxiliary network design might seem like an ad-
with data-parallel methods is also natural. For example a
As will be demonstrated in our experiments, the DGL can
Appendix B also provides a more intuitive pseudo-code for
(Goyal et al., 2017; Keskar et al., 2016). This suggests that
(Zhang et al., 2015), the asynchronous DGL does not rely
The DNI method requires an auxiliary network to predict
3.3. Auxiliary and Primary Network Design
The DNI method requires an auxiliary network to predict
the primary network than standard end-to-end trained deep
potentially require slightly different architecture principles for
An auxiliary network in addition to the CNN architecture
relies on an auxiliary network. This requires the design of
Decoupled Greedy Learning of CNNs
We also assume bounded gradient moments:
Assumption 2 (Convergence of the previous layer)
Assumption 3 (Robbins-Monro conditions). The step sizes
We consider the SGD scheme with learning rate \( \{\eta_t\} \):
\[ \Theta_{j+1} = \Theta_j - \eta_t \nabla_{\Theta_j} \ell(Z_{j-1}; \Theta_j) \]  
where \( Z_{j-1} \sim p_{j-1}^t \) (1)
Assumption 3 (Finite variance), There exists \( G > 0 \), for any \( t \) and \( \Theta_j \), \( \mathbb{E}_{p_{j-1}^t} [ || \nabla \ell(Z_{j-1}; \Theta_j) ||^2 ] \leq G. \)
The Assumptions 1, 2 and 3 are standard (Bottou et al., 2018; Huo et al., 2018a), and we show in the following that our proof of convergence leads to similar rates, up to a multiplicative constant. The following assumption is specific to our setting where we consider a time-varying distribution:
Assumption 4 (Convergence of the previous layer). We assume that \( \sum_t c_{j-1} < \infty. \)
Assumption 3 can be extend to $p_j^{t−1}$.

**Lemma 4.1.** Under Assumption 3 and 4, one has: 
\[ \forall \Theta_j, \mathbb{E}_{p_j^{t−1}}[||\nabla \ell(Z_j^{t−1}; \Theta_j)||^2] \leq G. \]

We are now ready to prove the core statement for the convergence results in this setting:

**Lemma 4.2.** Under Assumptions 1, 3 and 4, we have:
\[
\mathbb{E}[\mathcal{L}(\Theta_j^{t−1} + 1)] \leq \mathbb{E}[\mathcal{L}(\Theta_j^t)] - \eta_t \left( \mathbb{E}[||\nabla \mathcal{L}(\Theta_j^t)||^2] - \sqrt{2}Gc_j^{t−1} + \frac{LG}{2} \eta_t^2 \right) \tag{2}
\]

The expectation is taken over each random variable. Also, note that without the temporal dependency (i.e. $c_j^t = 0$), this becomes analogous to Lemma 4.4 in (Bottou et al., 2018).

Naturally it follows, that

**Proposition 4.1.** Under Assumptions 1, 2, 3 and 4, each term of the following equation converges:
\[
\sum_{t=0}^{T} \eta_t \mathbb{E}[||\nabla \mathcal{L}(\Theta_j^t)||^2] \leq \mathbb{E}[\mathcal{L}(\Theta_j^0)] + G \sum_{t=0}^{T} \eta_t \left( \sqrt{2}c_j^{t−1} + \frac{L\eta_t}{2} \right). \tag{3}
\]

Thus the DGL scheme converges in the sense of (Bottou et al., 2018; Huo et al., 2018a). It is also possible to obtain the following rate:

**Corollary 4.1.** The sequence of expected gradient norm accumulates around 0 at the following rate:
\[
\inf_{t \leq T} \mathbb{E}[||\nabla \mathcal{L}(\Theta_j^t)||^2] \leq \mathcal{O}\left( \frac{\sum_{t=0}^{T} c_j^{t−1} \eta_t}{\sum_{t=0}^{T} \eta_t} \right) \tag{4}
\]

Thus compared to the sequential case, the parallel setting adds a delay that is controlled by $\sqrt{c_j^{t−1}}$. We now evaluate DGL empirically.

5. Experiments

We conducted several experiments that empirically show that DGL optimizes the greedy objective well. We compare the DGL method to others, showing it is a state-of-the-art solution for decoupling training of deep network modules. We show that it can still work on a large-scale dataset (ImageNet) and that it can, in some cases, generalize better than standard back-propagation. We also demonstrate positive initial results for the asynchronous variant of the algorithm.

5.1. Other Approaches and Auxiliary Network Designs

This section presents experiments evaluating DGL with the CIFAR-10 dataset (Krizhevsky, 2009) and standard data augmentation. We first use a setup that permits us to compare against the DNI method and which also highlights the generality and scalability of DGL. We then consider the design of a more efficient auxiliary network which we will subsequently use to permit scaling to the ImageNet dataset. We will also show that DGL is effective at optimizing the greedy objective compared to a naive sequential algorithm.

**Comparison to DNI** We reproduce the CIFAR-10 CNN experiment described in (Jaderberg et al., 2017), Appendix C.1. This experiment utilizes a 3 layer network with auxiliary networks of 2 hidden CNN layers. We compare our reproduction to the DGL approach. Instead of the final synthetic gradient prediction for the DGL we apply a final projection to the target prediction space. We follow the prescribed optimization procedure from (Jaderberg et al., 2017) in this comparison, using Adam with a learning rate of $3e−5$. We run training for 1500 epochs and compare standard backpropagation, DNI, cDNI (Jaderberg et al., 2017) and DGL. Results are shown in Fig. 2. Further details are included in the Appendix. We find that the DGL method outperforms DNI and the context DNI by a substantial amount both in test accuracy and training loss. We also find in this setting that the DGL can generalize better than standard backpropagation and obtains a very close final training loss.

We also attempted DNI with the more commonly used optimization settings for CNNs (SGD with momentum and step decay), but found that the DNI would diverge when larger learning rates were used, although DGL sub-problem optimization worked effectively with common CNN optimization strategies. We also note that the prescribed experiment uses a setting where the scalability of our method is not fully exploited. Each layer of the primary network of (Jaderberg et al., 2017) has a pooling operation, which permits the auxiliary network to be small for synthetic gradient prediction. This however severely restricts the architecture choices in the primary network to using a pooling operation at each layer. In DGL we can apply the pooling operations in the auxiliary network thus permitting the auxiliary network to be negligible in cost even for layers without pooling (whereas for synthetic gradient prediction they often have to be as costly as the base network). Overall we find that the DGL approach is not only far more scalable and accurate but also more stable and robust to changes in optimization hyper-parameters than DNI.

|        | FLOPS Aux./ Module | Acc. |
|--------|-------------------|------|
| CNN-aux| 200%              | 92.2 |
| MLP-aux| 0.7%              | 90.6 |
| MLP-SR-aux| 4.0%          | 91.2 |

Table 1. We compare auxiliary networks. CNN-aux applied in previous work is inefficient with respect to the primary module. We report the flop count of the largest module and the corresponding auxiliary network and accuracy for a CIFAR-10. MLP-aux and MLP-SR-aux applied after spatial averaging operations are far more effective with minimal accuracy loss.
Auxiliary Network Design  We consider different auxiliary networks for CNNs. As a baseline we use convolutional auxiliary layers as in (Jaderberg et al., 2017) and (Belilovsky et al., 2018). For distributed training application this approach is sub-optimal as the auxiliary network can be substantial compared to the base network, leading to poorer parallelization gains. We note however that even in those cases (that we don’t study here) where the auxiliary network computation is potentially on the order of the the primary network, it can still give advantages for parallelization for very deep networks and many available workers.

The primary network architecture we use for these experiments is a simple CNN similar to VGG family models (Simonyan & Zisserman, 2014). It consists of 6 convolutional layers with $3 \times 3$ kernels, batchnorm and shape preserving padding, with $2 \times 2$ maxpooling operations at layers 1 and 3. The channel width of the first layer is 128 and is doubled at each downsampling operation. The final layer does not have an auxiliary model, it is learned with a linear spatial averaging followed by a 2-hidden layer constant depth fully connected network, for all experiments. Two alternatives to the CNN auxiliary of (Belilovsky et al., 2018) are explored, which exploit a spatial averaging operation. We re-iterate that this kind of approach and even the simple network structure we consider is not easily applicable in the case of DNI and synthetic gradient prediction. Optimization is done using a standard strategy for CIFAR CNN training. We apply SGD with momentum of 0.9 and weight decay $5e^{-4}$ (Zagoruyko & Komodakis, 2016) and decaying step sizes. For these experiments we use a short schedule of 50 epochs and decay factor of 0.2 every 15 epochs (Belilovsky et al., 2018). Results of comparisons are given in Table 1.

The baseline auxiliary strategy based on (Belilovsky et al., 2018) and (Jaderberg et al., 2017) applies 2 CNN layers followed by a spatial averaging to $2 \times 2$ before a final projection. We denote this CNN-aux. The first alternative we explore is a direct application of the spatial averaging to $2 \times 2$ output shape (regardless of the input resolution) followed by a 3 layer MLP (of constant width). This is denoted MLP-aux and drastically reduces the FLOP count with minimal accuracy loss compared to CNN-aux. Finally we consider applying a staged spatial resolution, first reducing the spatial resolution by $4 \times$ (and total size $16 \times$), then applying $3 \times 1$ convolutions followed by a reduction to $2 \times 2$ and a 3 layer MLP. We denote this approach MLP-SR-aux. These latter two strategies that leverage the spatial averaging produce auxiliary networks that are less than 5% of the FLOP count of the primary network even for large spatial resolutions as in real world image datasets. We will show that MLP-SR-aux is still effective even for the large-scale ImageNet dataset.

Sequential vs. Parallel Optimization of Greedy Objective  We briefly compare the sequential optimization of the greedy objective (Belilovsky et al., 2018; Bengio et al., 2007) to the DGL (Alg. 1). We use a 4 layer CIFAR-10 network with an MLP-SR-aux auxiliary model and a final layer attached to a 2 layer MLP. We use the same optimization settings with 50 epochs as in the last experiment. In the sequential training we train each layer for 50 epochs before moving to the subsequent one. Thus the difference to DGL lies only in the input received at each layer (fully converged

Figure 2. Comparison of DNI, context DNI (cDNI), and DGL in terms of training loss and test accuracy for experiment from (Jaderberg et al., 2017). DGL converges better than cDNI and DNI with the same auxiliary net. and generalizes better than backprop for this case.

Figure 3. Comparison of sequential and parallel training. We observe parallel training catches up rapidly to sequential.
previous layer versus not fully converged previous layer). The rest of the optimization settings are identical. Figure 3 shows comparisons of the learning curves for sequential training and DGL at layer 4 (layer 1 is the same for both as the input representation is not varying over the training period). We observe that the DGL quickly catches up with the sequential training scheme and appears to sometimes generalize better. We additionally visualize the dynamics of training per layer in Fig. 4, which demonstrates that after just a few epochs the individual layers build a dynamic of progressive improvement with depth. Additional visualizations are included in the supplementary materials.

**Multi-Layer modules** Although we have so far considered the setting of layer-wise decoupling, this approach can easily be applied to generic modules. Indeed approaches such as DNI (Jaderberg et al., 2017) often consider decoupling entire multi-layer modules. Furthermore the propositions for backward unlocking (Huo et al., 2018b,a) also rely on and report they can often only decouple 100 layer networks into 2 or 4 blocks before observing optimization issues or performance losses and require that the number of parallel modules is much lower than the network depth for the theoretical guarantees to hold. As in those cases, using multi-layer decoupled modules can improve performance and is natural in the case of deeper networks. We now use such a multi-layer approach to directly compare to the backward unlocking of (Huo et al., 2018b) and then subsequently we will apply this on deep networks for ImageNet. We will denote from here-on the number of total modules a network is split into as $K$.

**Comparison to DDG** (Huo et al., 2018b) proposes a solution to the backward locking (less efficient than solving update-locking, see discussion above) We show that even in this situation the DGL method can provide a strong baseline for work on backward unlocking. We take the example from (Huo et al., 2018b), which considers a ResNet-110 parallelized into $K = 2$ blocks. We use the auxiliary network

| Backprop | DDG    | DGL     |
|----------|--------|---------|
| 93.53    | 93.41  | 93.5 ± 0.1 |

Table 2. ResNet-110 ($K = 2$) for Backprop and DDG method reported from (Huo et al., 2018b). DGL is run for 3 trials to compute variance. The approaches give the same accuracy with DGL being update unlocked and DDG only backward unlocked. DNI is reported to not work in this setting (Huo et al., 2018b).

MLP-SR-aux which has less than 0.1% the FLOP count of the primary network. We use the exact optimization and network split points as in (Huo et al., 2018b). To assess variance in the accuracy for CIFAR-10 we perform 3 trials. We observe in Tab. 2 that the accuracy is the same across the DDG method, backprop, and our approach. DDG achieves better parallelization because it also splits the forward pass.

**5.2. Large-scale Experiments**

Existing methods considering update or backward locking have not been evaluated on large image datasets as they are often unstable or already show large losses in accuracy on smaller datasets. Here we study the optimization of several well-known architectures, mainly the VGG family (Simonyan & Zisserman, 2014) and the ResNet (He et al., 2016). In all our experiments we use the MLP-SR-aux auxiliary network which scales well from the smaller CIFAR-10 images to the larger ImageNet ones. The final module does not have an auxiliary model.

For all optimization of auxiliary problems and for end-to-end optimization of reference models we use the optimization schedule prescribed in (Xiao et al., 2019). It consists of training for 50 epochs with mini-batches size 256, uses SGD with momentum of 0.9, weight decay of 1e−4, and a learning rate of 0.1 reduced by a factor 10 every 10 epochs. Results are shown in Tab. 3. For several of the models DGL can perform as well and sometimes better than the end-to-end trained model, while permitting parallel training. For the VGG-13 architecture we also evaluate the case where the model is trained layer by layer ($K = 10$). Although performance is degraded by this split we find its performance surprising considering that no backward communication is performed. We conjecture that improved auxiliary models and combinations with methods such as (Huo et al., 2018a; Jaderberg et al., 2017) to allow feedback on top of the local model, may further improve performance. Also as mentioned for the settings with larger potential parallelization, slower but more performant auxiliary models could potentially be considered as well.

We also remark that the synchronous DGL has favorable memory usage compared to DDG and to the DNI method. DNI requiring to store larger activations and DDG having memory. Although not the focus of this work in the single worker version of DGL has favorable memory usage compared to standard end-to-end backpropagation training. For example the ResNet-152 DGL $K = 2$ setting we have con-
We now study the stability of Alg. 2 w.r.t. the buffers. We consider models trained with all other architecture and optimization settings as in the auxiliary network experiments of Sec. 5.1. Each layer is equipped with a buffer of size $M$. At each iteration, a layer is chosen according to the pmf $p(j)$, and a batch selected from buffer $B_{j-1}$. One layer is slowed down by decreasing its selection probability in the pmf $p(j)$ by a factor $S$. We evaluate different slowdown factors (up to $S = 2.0$). Accuracy versus slowdown factor is shown in Fig. 5. For this experiment we use a buffer of size $M = 50$. We run separate experiments with the slowdown applied at each layer of the network as well as 3 random seeds for each of these settings (for a total of 18 experiments per data point). We show the evaluations for 10 values of $S$. To ensure a fair comparison we also stop updating layers once they have completed the iterations for 50 epochs, thus assuring identical number of gradient updates for all layers in all experiments compared. In practice one could continue updating until all layers have completed training. In Fig. 5 and compare it to the synchronous case (standard DGL). We first observe that the accuracy of the synchronous algorithm is maintained in the setting where $S = 1$ and the pmf is uniform. Note that even this is a non-trivial case, as it will mean that layers inherently have random delays (as compared to the synchronous Alg. 1). We see that accuracy is maintained until approximately $1.2\times$ and accuracy losses after that still remain small. Our maximum slowdown factor of $2\times$ is somewhat drastic – it means that for the 50 epochs of training, the slowed-down layer is only on epoch $25$ while those following it are at epoch $50$.

In a second experiment we evaluate performance with respect to the buffer size. Results are shown in Fig. 6. For this experiment we fix the slowdown factor to $1.2\times$. We observe that even when a very small buffer size can yield only a slight loss in performance accuracy. Indeed building on this demonstration there are multiple direction to improve Async DGL with replay. For example improving the efficiency of the batch, by including data augmentation in feature space (Verma et al., 2018), mixing samples in batches, and improved batch sampling among other directions.

### 6. Conclusion

We have analyzed and introduced a simple and strong baseline for parallelizing per layer and per module computations in CNN training. Our approach is shown to match or exceed state of the art approaches addressing these problems and shown able to scale to much large datasets than others. Future work can develop improved auxiliary problem objectives and combinations with delayed feedback.
Acknowledgements

EO and EB acknowledges NVIDIA for its GPU donation. We would like to thank John Zarka, Louis Thiry, Georgios Exarchakis, Fabian Pedregosa, Maxim Berman, Amal Rannen, Aaron Courville, and Nicolas Pinto for helpful discussions.

References

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. 9389–9399, 2018.

Belilovsky, E., Eickenberg, M., and Oyallon, E. Greedy layerwise learning can scale to imagenet, 2018. URL http://arxiv.org/abs/1812.11446.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In Advances in neural information processing systems, pp. 153–160, 2007.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. SIAM Review, 60(2):223–311, 2018.

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

Choromanska, A., Kumaravel, S., Luss, R., Rish, I., Kingsbury, B., Tejwani, R., and Bouneffouf, D. Beyond backprop: Alternating minimization with co-activation memory. arXiv preprint arXiv:1806.09077, 2018.

Czarnecki, W. M., Swirszcz, G., Jaderberg, M., Osindero, S., Vinyals, O., and Kavukcuoglu, K. Understanding synthetic gradients and decoupled neural interfaces. CoRR, abs/1703.00522, 2017. URL http://arxiv.org/abs/1703.00522.

Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: training imagenet in 1 hour. CoRR, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. Latent space policies for hierarchical reinforcement learning. In Dy, J. and Krause, A. (eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 1851–1860, Stockholmsmsson, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/haarnoja18a.html.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

Huang, F., Ash, J., Langford, J., and Schapire, R. Learning deep resnet blocks sequentially using boosting theory. International Conference on Machine Learning(ICML), 2018.

Huo, Z., Gu, B., and Huang, H. Training neural networks using features replay. Advances in Neural Information Processing Systems, 2018a.

Huo, Z., Gu, B., qian Yang, and Huang, H. Decoupled parallel backpropagation with convergence guarantee. In Dy, J. and Krause, A. (eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 2098–2106, Stockholmsmsson, Stockholm Sweden, 10–15 Jul 2018b. PMLR. URL http://proceedings.mlr.press/v80/huo18a.html.

Ivakhnenko, A. G. and Lapa, V. G. Cybernetic Predicting Devices. CCM Information Corporation., 1965.

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., and Kavukcuoglu, K. Decoupled neural interfaces using synthetic gradients. International Conference of Machine Learning, 2017.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836, 2016.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. Asaga: Asynchronous parallel saga. In 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, 2017.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. Improved asynchronous parallel optimization analysis for stochastic incremental methods. Journal of Machine Learning Research, 19(81):1–68, 2018. URL http://jmlr.org/papers/v19/17-650.html.

Lee, D., Zhang, S., Biard, A., and Bengio, Y. Target propagation. CoRR, abs/1412.7525, 2014. URL http://arxiv.org/abs/1412.7525.

Lian, X., Zhang, W., Zhang, C., and Liu, J. Asynchronous decentralized parallel stochastic gradient descent. arXiv preprint arXiv:1710.06952, 2017.
Decoupled Greedy Learning of CNNs

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. Random feedback weights support learning in deep neural networks. *CoRR*, abs/1411.0247, 2014. URL http://arxiv.org/abs/1411.0247.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Marquez, E. S., Hare, J. S., and Niranjan, M. Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5475–5485, Nov 2018. ISSN 2162-237X. doi: 10.1109/TNNLS.2018.2805098.

Nøkland, A. Direct feedback alignment provides learning in deep neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 1037–1045. 2016.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. Training neural networks without gradients: A scalable admn approach. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2722–2731, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/taylor16.html.

Verma, V., Lamb, A., Beckham, C., Najafi, A., Courville, A., Mitliagkas, I., and Bengio, Y. Manifold mixup: Learning better representations by interpolating hidden states. 2018.

Xiao, W., Chen, H., Liao, Q., and Poggio, T. A. Biologically-plausible learning algorithms can scale to large datasets. *International Conference on Learning Representations*, 2019.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, S., Choromanska, A. E., and LeCun, Y. Deep learning with elastic averaging sgd. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 685–693. Curran Associates, Inc., 2015.
A. Proofs

**Lemma 4.1.** **Under Assumption 3 and 4, one has:** \( \forall \Theta_j, \mathbb{E}_{p^t_{j-1}} [||\nabla \ell(Z_{j-1}; \Theta_j)||^2] \leq G. \)

**Proof.** First of all, observe that under Assumption 4 and via Fubini’s theorem:

\[
\sum c_{j-1}^t = \sum_t \int |p_{j-1}^t(z) - p_{j-1}^*| \, dz = \sum_t \int |p_{j-1}^t(z) - p_{j-1}^*| \, dz < \infty
\]

thus, \( \sum |p_j^t - p_j^*| \) is convergent a.s. and \( |p_j^t - p_j^*| \to 0 \) a.s as well. From Fatou’s lemma, one has:

\[
\int p_{j-1}^* \|\nabla_{\Theta_j} \ell(z; \Theta_j)\|^2 \, dz = \int \liminf_t p_{j-1}^* \|\nabla_{\Theta_j} \ell(z; \Theta_j)\|^2 \, dz \\
\leq \liminf_t \int p_{j-1}^* \|\nabla_{\Theta_j} \ell(z; \Theta_j)\|^2 \, dz \leq G
\]

\[\square\]

**Lemma 4.2.** **Under Assumptions 1, 3 and 4, we have:**

\[
\mathbb{E}[\mathcal{L}(\Theta_{j+1}^t)] \leq \mathbb{E}[\mathcal{L}(\Theta_j^t)] - \eta_t (\mathbb{E}[\|\nabla \mathcal{L}(\Theta_j^t)\|^2] - \sqrt{2}Gc_{j-1}^t) + \frac{LG}{2} \eta_t^2,
\]

**Observe the expectation is taken over each random variable.**

**Proof.** By \( L \)-smoothness:

\[
\mathcal{L}(\Theta_j^{t+1}) \leq \mathcal{L}(\Theta_j^t) + \nabla \mathcal{L}(\Theta_j^t)^T (\Theta_j^{t+1} - \Theta_j^t) + \frac{L}{2} \|\Theta_j^{t+1} - \Theta_j^t\|^2
\]

Substituting \( \Theta_j^{t+1} - \Theta_j^t \) on the right:

\[
\mathcal{L}(\Theta_j^{t+1}) \leq \mathcal{L}(\Theta_j^t) - \eta_t \nabla \mathcal{L}(\Theta_j^t)^T \nabla \Theta_j \ell(Z_{j-1}; \Theta_j^t) + \frac{L \eta_t^2}{2} \|\nabla \Theta_j \ell(Z_{j-1}; \Theta_j^t)\|^2
\]

Taking the expectation w.r.t. \( Z_{j-1}^t \) which has a density \( p_{j-1}^t \), we get:

\[
\mathbb{E}_{p_{j-1}^t} [\mathcal{L}(\Theta_{j+1}^t)] \leq \mathcal{L}(\Theta_j^t) - \eta_t \mathbb{E} [\nabla \mathcal{L}(\Theta_j^t)^T \nabla \Theta_j \ell(Z_{j-1}; \Theta_j^t)] \leq \frac{1}{2} \mathbb{E}_{p_{j-1}^t} [||\nabla \Theta_j \ell(Z_{j-1}; \Theta_j^t)||^2]
\]

From Assumption 3, we obtain that:

\[
\frac{L \eta_t^2}{2} \mathbb{E}_{p_{j-1}^t} [||\nabla \Theta_j \ell(Z_{j-1}; \Theta_j^t)||^2] \leq \frac{L \eta_t^2 G}{2}
\]

Then, as a side computation, observe that:

\[
||\mathbb{E}_{p_{j-1}^t} [\nabla \Theta_j \ell(Z_{j-1}; \Theta_j^t)] - \nabla \mathcal{L}(\Theta_j^t)|| = || \int \nabla \ell(z, \Theta_j^t) p_{j-1}^t(z) \, dz - \int \nabla \ell(z, \Theta_j^t) p_{j-1}^* \, dz ||
\]

\[
\leq \int ||\nabla \ell(z, \Theta_j^t)|| |p_{j-1}^t(z) - p_{j-1}^*| \, dz
\]

\[
= \int (||\nabla \ell(z, \Theta_j^t)|| \sqrt{|p_{j-1}^t(z) - p_{j-1}^*|}) \sqrt{|p_{j-1}^t(z) - p_{j-1}^*|} \, dz
\]
Let us apply the Cauchy-Schwartz inequality, we obtain:
\[
\|E_{p_{j-1}^*} \left[ \nabla \ell_j(\ell(Z_{j-1}; \Theta_j^j)) \right] - \nabla \mathcal{L}(\Theta_j^j) \| \leq \sqrt{\int \| \nabla \ell(z, \Theta_j^j) \|^2 \| p_{j-1}^* - p_{j-1}^* \| dz} \sqrt{\int |p_{j-1}^* - p_{j-1}^*| dz} 
\]
\[
= \sqrt{\int \| \nabla \ell(z, \Theta_j^j) \|^2 \| p_{j-1}^* - p_{j-1}^* \| dz} \sqrt{\epsilon_j^2} 
\]
Then, observe that:
\[
\int \| \nabla \ell(z, \Theta_j^j) \|^2 \| p_{j-1}^* - p_{j-1}^* \| dz \leq \int \| \nabla \ell(z, \Theta_j^j) \|^2 (p_{j-1}^* + p_{j-1}^*) dz 
\]
\[
= E_{p_{j-1}^*} \| \nabla \ell(Z_{j-1}, \Theta_j^j) \|^2 + E_{p_{j-1}^*} \| \nabla \ell(Z_{j-1}, \Theta_j^j) \|^2 
\]
\[
\leq 2G 
\]
The last inequality follows from Lemma 4.1 and Assumption 3.

Then, using again Cauchy-Schwarz inequality:
\[
\left| \| \nabla \mathcal{L}(\Theta_j^j) \|^2 - \nabla \mathcal{L}(\Theta_j^j)^T E_{p_{j-1}^*} \left[ \nabla \ell_j(\ell(Z_{j-1}; \Theta_j^j)) \right] \right| 
\]
\[
= \left| \nabla \mathcal{L}(\Theta_j^j)^T \left( \nabla \mathcal{L}(\Theta_j^j) - E_{p_{j-1}^*} \left[ \nabla \ell_j(\ell(Z_{j-1}; \Theta_j^j)) \right] \right) \right| 
\]
\[
\leq \| \nabla \mathcal{L}(\Theta_j^j) \| \| \nabla \mathcal{L}(\Theta_j^j) \|^2 - \nabla \mathcal{L}(\Theta_j^j)^T E_{p_{j-1}^*} \left[ \nabla \ell_j(\ell(Z_{j-1}; \Theta_j^j)) \right] \right| 
\]
\[
\leq \| \nabla \mathcal{L}(\Theta_j^j) \| \sqrt{2Gc_j^2} 
\]
Then, taking the expectation leads to
\[
\left| E \left[ \| \nabla \mathcal{L}(\Theta_j^j) \|^2 - \nabla \mathcal{L}(\Theta_j^j)^T E_{p_{j-1}^*} \left[ \nabla \ell_j(\ell(Z_{j-1}; \Theta_j^j)) \right] \right] \right| 
\]
\[
\leq E \left[ \| \nabla \mathcal{L}(\Theta_j^j) \|^2 - \nabla \mathcal{L}(\Theta_j^j)^T E_{p_{j-1}^*} \left[ \nabla \ell_j(\ell(Z_{j-1}; \Theta_j^j)) \right] \right] \right| 
\]
\[
\leq E \left[ \| \nabla \mathcal{L}(\Theta_j^j) \|^2 \right] \sqrt{2Gc_j^2} 
\]
\[
\leq \sqrt{E \left[ \| \nabla \mathcal{L}(\Theta_j^j) \|^2 \right]} \sqrt{2Gc_j^2} 
\]
However, observe that by Lemma 4.1 and Jensen inequality:
\[
\| \nabla \mathcal{L}(\Theta_j^j) \|^2 = \| E_{p_{j}^*} [\nabla \ell(Z, \Theta_j^j)] \|^2 \leq E_{p_{j}^*} [\| \nabla \ell(Z, \Theta_j^j) \|^2] \leq G 
\]
Combining this inequality and Assumption 3, we get:
\[
E[\mathcal{L}(\Theta_j^{j+1})] \leq E[\mathcal{L}(\Theta_j^j)] - \eta_t (E[\| \nabla \mathcal{L}(\Theta_j^j) \|^2] - \sqrt{2Gc_j^2}) + \frac{LG}{2} \eta_t^2 
\]
\[
\text{Proposition 4.2. Under Assumptions 1, 2, 3 and 4, each term of the following equation converges:} 
\]
\[
\sum_{t=0}^{T} \eta_t E[\| \nabla \mathcal{L}(\Theta_j^j) \|^2] \leq E[\mathcal{L}(\Theta_j^{0})] + G \sum_{t=0}^{T} \eta_t (\sqrt{2c_j^t} + \frac{L}{2} \eta_t) 
\]
We briefly illustrate the sizes of auxiliary networks. Let’s take as an example the ImageNet experiments for VGG-13. At worse than those reported in (Jaderberg et al., 2017), which could be due to minor differences in the implementation. Although the baseline accuracies for backprop and cDNI are close to those reported in the original work, those of DNI are comparably worse than those reported in (Jaderberg et al., 2017), which could be due to minor differences in the implementation. We utilize a popular pytorch DNI implementation available and source code will be provided.

### B. Additional pseudo-code

To illustrate the parallel implementations of the Algorithms we show a different pseudocode implementation with an explicit behavior for each worker specified. The following Algorithm 3 is equivalent to Algorithm 1 in terms of output but directly illustrates a parallel implementation. Similarly, 4 illustrates a parallel implementation of the algorithm described in Algorithm 2. The probabilities used in Algorithm 4 are not included here as they are derived from communication and computation speed differences.

### C. Additional Descriptions of Experiments

Here we provide some additional details of the experiments. Code for experiments is provided along with the supplementary materials.

#### Comparisons to DNI

The comparison to DNI attempts to directly replicate the the Appendix C.1 (Jaderberg et al., 2017). Although the baseline accuracies for backprop and cDNI are close to those reported in the original work, those of DNI are worse than those reported in (Jaderberg et al., 2017), which could be due to minor differences in the implementation. We utilize a popular pytorch DNI implementation available and source code will be provided.

### C.1. Auxiliary Network Sizes

We briefly illustrate the sizes of auxiliary networks. Let’s take as an example the ImageNet experiments for VGG-13. At the first layer the output is $224 \times 224 \times 64$. The MLP-aux here would be applied after averaging to $2 \times 2 \times 64$, and...
would consists of 3 fully connected layers of size 256 (2 * 2 * 64) followed by a projection to 1000 image categories. The MLP-SR-aux network used would first reduce to 56 × 56 × 64 and then apply 3 layers of 1 × 1 convolutions of size 64. This is followed by reduction to 2 × 2 and 3 FC layers as in the MLP-aux network.