Journal of Machine Learning Research 23 (2022) 1-28 Submitted 10/20; Revised 6/22; Published 7/22

Interlocking Backpropagation: Improving depthwise model-parallelism

Aidan N. Gomez* AIDAN.GOMEZ@CS.OX.AC.UK
University of Oxford & Cohere

Oscar Key* OSCAR.KEY.20@UCL.AC.UK
University of Oxford

Kuba Perlin KUBA@COHERE.AI
Cohere

Stephen Gou STEPHEN@COHERE.AI
Cohere

Nick Frosst NICK@COHERE.AI
Cohere

Jeff Dean JEFF@GOOGLE.COM
Google

Yarin Gal YARIN@CS.OX.AC.UK
University of Oxford

Editor: Sathiya Keerthi

Abstract

The number of parameters in state of the art neural networks has drastically increased in recent years. This surge of interest in large scale neural networks has motivated the development of new distributed training strategies enabling such models. One such strategy is model-parallel distributed training. Unfortunately, model-parallelism can suffer from poor resource utilisation, which leads to wasted resources. In this work, we improve upon recent developments in an idealised model-parallel optimisation setting: local learning. Motivated by poor resource utilisation in the global setting and poor task performance in the local setting, we introduce a class of intermediary strategies between local and global learning referred to as *interlocking backpropagation*. These strategies preserve many of the compute-efficiency advantages of local optimisation, while recovering much of the task performance achieved by global optimisation. We assess our strategies on both image classification ResNets and Transformer language models, finding that our strategy consistently out-performs local learning in terms of task performance, and out-performs global learning in training efficiency.

∗. Joint first author

Keywords: Model Parallelism, Distributed Optimisation, Large-scale Modelling, Parallel Distributed Processing, Efficient Training

1. Introduction

Modern state-of-the-art language models require billions of parameters. These models are often too large to fit in the memory of a single accelerator, and so the training computation must be distributed across multiple accelerator devices. Training such large models can be accomplished by partitioning the model across several accelerators and communicating the activations and gradients between them. However, this naive approach to training way incurs significant inefficiencies, as each accelerator must wait for all downstream accelerators to compute their forwards and backwards passes before it can begin computation of its own backwards pass. This optimisation setting is referred to as 'global learning', as there is a single global objective that must be evaluated in order to compute updates to the parameters.

An idealised model-parallel optimisation setting would be one where each accelerator need only push data to the next, never waiting for any returning gradient. In order to facilitate this, each accelerator's portion of the model must be able to compute weight updates with which to train itself, without access to any information from downstream accelerators. This idealised setting is referred to as 'local learning' and has seen an uptick in recent interest (Löwe et al., 2019; Belilovsky et al., 2020). However, there remain core limitations to the proposed methods, principal among them: the degradation in modelling performance relative to global learning.

In this work we attempt to improve the efficiency of distributed model training by exploring strategies that strike a middle-ground between local and global learning via backpropagation. We investigate a class of training regimes by training large-scale neural networks with auxiliary classification layers throughout the network, and restricting the gradient flow from each of these classification heads. We refer to these strategies as *interlocking backpropagation*. We find that interlocking backpropagation is significantly more compute efficient than the standard global backpropagation approach, yet it recovers much of its modelling performance, compared to local learning. Our work presents the following contributions:

- We explore modelling limitations of local optimisation.
- We propose a class of optimisation algorithms that aim to *preserve much of the compute efficiency* of local training, while *significantly improving modelling performance*.
- We provide a generic, open-source framework for the study of this class of optimisation algorithms. It is available at `https://github.com/oscarkey/interlocking-backprop`.

2. Related Work

Previous work has attempted to improve the resource utilisation of distributed model-parallel training regimes. GPipe (Huang et al., 2019) addresses the inefficiency of training model-parallel distributed networks by splitting *mini*-batches into *micro*-batches and processing these micro-batches concurrently. GPipe accumulates gradients of all micro-batches before

applying them to the weights. That strategy increases resource utilisation, but it still requires more time overall than a local approach to training.

While GPipe performs weight updates that are equivalent to global learning, there are strategies to speeding up distributed training that do not have this property. One such strategy is called Hogwild (Recht et al., 2011). Instead of performing a forward pass on each accelerator and waiting until the full backwards pass of the network has been computed and communicated to update the parameters, Hogwild starts processing the forward pass of subsequent batches as soon as the first accelerator has completed the forward pass of the first batch. Likewise, as soon as an earlier accelerator receives gradients from the subsequent accelerator, Hogwild applies those gradients and updates the weights. This means that when the gradients for the second batch are applied, they are being applied to different weights than those that were used to compute the corresponding forward pass. This problem is referred to as *stale gradients*, and has been shown to introduce training instabilities. It is visualised in Figure 1. Hogwild overlooks the issue of stale gradients entirely, opting to *greedily* apply gradients to the weights as soon as any arrive. This results in a very fast-executing optimisation strategy, but it can have a dramatic impact on the stability of optimisation when many accelerators are involved, making the method less favourable. The Hogwild approach is orthogonal to interlocking backpropagation, the method we introduce in this paper, as interlocking backpropagation could be run with greedy gradient updates as well. This however would introduce the same issue of stale gradients. In our experiments we find that Hogwild greatly underperforms both global learning and interlocking backpropagation (see Appendix A, Table 4).

Local learning resolves the stale gradient issue by doing away with the communication of gradients entirely. Each accelerator is responsible for using its own training signal to compute and apply parameter updates. This also means each accelerator spends no idle time whatsoever waiting for gradients from other accelerators – which is why it is considered an idealised setting for distributed model optimisation. Local objectives were initially used in early unsupervised methods for training neural networks, such as the wake-sleep algorithm (Hinton et al., 1995). In Inception networks (Szegedy et al., 2015), local objectives were applied in addition to the overall global objective in order to solve the vanishing gradients problem, though this approach has largely been supplanted by residual connections (He et al., 2016). More recently, several variations on the local learning paradigm have been introduced with a focus on parallel training. While highly parallelisable, many of these approaches do not match the accuracy of globally trained models (Mostafa et al., 2018; Jaderberg et al., 2017). Others do achieve performance comparable to that of global learning, by using specially crafted local objective functions. Löwe et al. (2019) use a contrastive predictive loss to achieve excellent performance in an unsupervised setting. Nøkland and Eidnes (2019) consider supervised local learning, and succeed in matching the accuracy of global learning on classification tasks with up to 100 classes. However, the authors note that their *predsim* loss is not suited for classification tasks with larger numbers of classes, due to the similarity matrix becoming sparse. The proposed work-around of limiting the number of classes in each batch was shown to be effective for CIFAR-100, but is not applicable to some tasks such as language modelling. In contrast to the aforementioned approaches, our method is a high-level paradigm for interpolating between end-to-end and local training, applicable

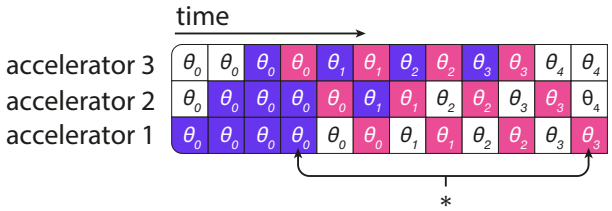


Figure 1: Depiction of the problem of stale gradients in Hogwild-style training. Here we have a model split across three accelerators. Each row shows the evolution of the parameters in the memory of a given accelerator, with each column representing a time step. θ_i denotes the parameters after the i^{th} gradient update. The steps highlighted in purple indicate that the accelerator performed a forward pass on a single batch, while those in pink indicate a backward pass. $(*)$ During the forward pass of the fourth batch of data, the first accelerator computes its activations using the parameters θ_0 (left arrow); however, during the fourth batch's backwards pass the first module has already had its parameters updated three times to θ_3 (right arrow). This mismatch between the weights used to compute the activations and those used to compute the gradient can disrupt optimisation dramatically (see Table 4 in Appendix A).

to both the unsupervised and supervised settings, and possible to combine with specialised loss functions, such as predsim.

One reason why some local learning algorithms fail to match the performance of global algorithms is that there is no mechanism for layers higher in the model to communicate with lower layers of the model. This means that the lower layers may learn representations which are not productive for learning in the higher layers. For example, a lower layer may discard information from which a higher layer could have extracted additional accuracy. Xiong et al. (2020) attempt to solve this problem by allowing the gradient signal to flow between pairs of adjacent accelerators. Importantly, the pairs overlap with each other, meaning that the upper accelerator of one pair is the lower accelerator of the subsequent pair, thus allowing indirect gradient flow through the entire model (see Section 3 for a more detailed description of this structure, as a special case of interlocking backpropagation). Introduced concurrently to our work, this approach is similar to our N-WISE algorithm, detailed later, for the particular case where $N = 2$. Our more general framework allows the practitioner to trade off training speed for increased accuracy, and we provide substantial experimental results investigating this trade-off. We also focus our experiments on supervised rather than unsupervised learning, and we specifically test our algorithm on large Transformer models with hundreds of millions of parameters.

In further concurrent work, Laskin et al. (2020) also inspect interlocking backpropagation, referring to it as 'overlapping local updates', and present a cost-performance analysis of 2-WISE, END-TO-END, and alternative training strategies, applied to tasks in image and language domains.

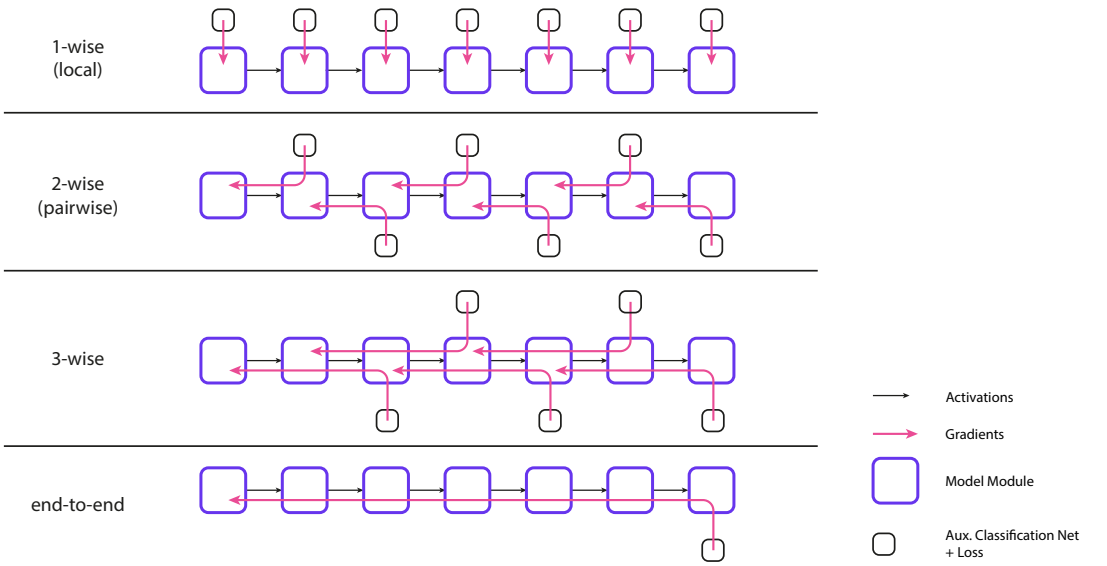


Figure 2: Depiction of the flow of activations and gradients through interlocking backpropagation for different optimisation strategies. Activation flows are shown in black and gradients are shown in red. One extreme is 1-WISE optimisation (local learning), where there is no gradient communication between modules. The other extreme is END-TO-END optimisation, where gradients flow through all modules from a global loss function at the top of the network. The 2-WISE and 3-WISE strategies, as introduced in this paper, strike a middle ground. In 2-WISE, gradients flow from an auxiliary network attached to each module, through the local module, and travel one module boundary before stopping. Similarly, 3-WISE has gradients travel through two module boundaries before stopping.

3. Interlocking Backpropagation

A neural network can be described as a composition of a series of smaller functions; for example $f = f_6 \circ \cdots \circ f_2 \circ f_1$. When the parameterisation of the network exceeds the limit of a single hardware accelerator, contiguous groups of these functions can be placed on individual accelerators. Each of these contiguous groups is referred to as a *module*. Note that in this work we assume that each accelerator holds only one module, as having multiple modules on a single accelerator would introduce unnecessary overheads to training. If an accelerator holds more than one module, these can simply be merged together to form a single module.

The communication between modules can be costly, and so one could attempt to speed up the learning process by performing local learning on each module. Consider a network

composed of three modules of two layers each:

$$\begin{aligned} f &= f_{c_3} \circ f_{c_2} \circ f_{c_1} \\ \text{where, } f_{c_1} &= f_2 \circ f_1, \text{ parameterised by } \theta_{c_1} = (\theta_2, \theta_1) \\ f_{c_2} &= f_4 \circ f_3, \text{ parameterised by } \theta_{c_2} = (\theta_4, \theta_3) \\ f_{c_3} &= f_6 \circ f_5, \text{ parameterised by } \theta_{c_3} = (\theta_6, \theta_5). \end{aligned}$$

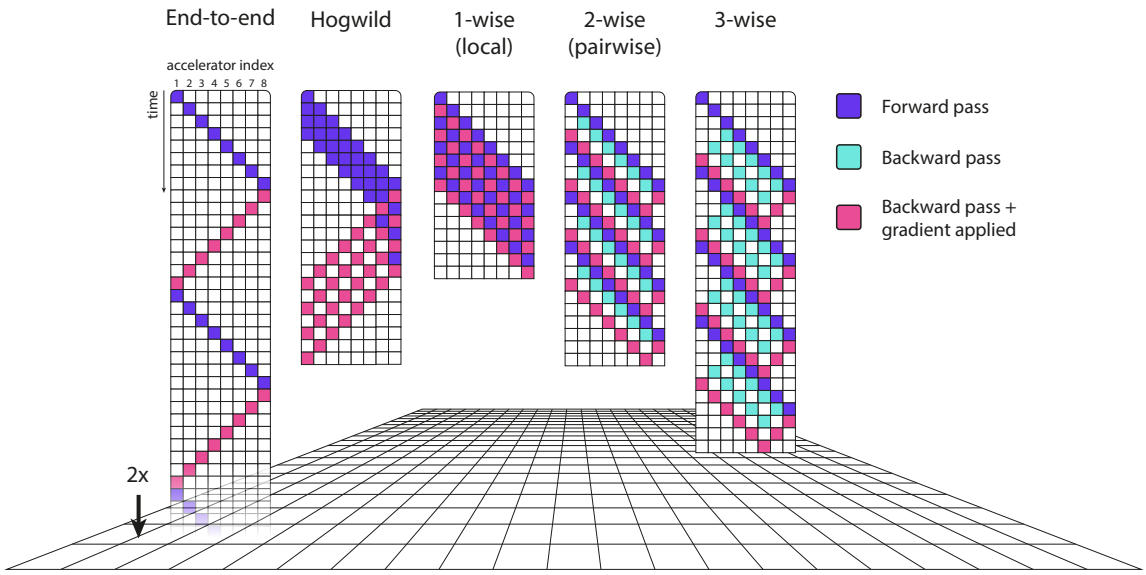
We consider several possible approaches for training this model, which differ in the amount of communication between modules. One extreme, involving the most communication, is END-TO-END training. Here we compute the loss based on the output of the final module, f_{c_3}, and propagate the loss backwards through each module to update their parameters. This approach is depicted in the bottom row of Figure 2 and it achieves identical accuracy to if the model was in a single module on a single accelerator; however, the communication between modules during the backwards pass leads to inefficiencies. For instance, the first module in the model, having completed its forward pass, must sit idle, while it waits for the modules above it to complete their forward and backward passes. Only then, does it receive the gradient signal and is able to perform its own backwards pass.

The other extreme, resulting in the least idleness, is local training, illustrated in the first row of Figure 2. In this setting, we augment each module with a local loss function, $\mathcal{L}_{c_k}$:

$$\begin{aligned} \mathcal{L}_{c_k}(x, y) &= \mathcal{L}(\hat{y}_{c_k}(x), y) \\ \text{where, } \hat{y}_{c_k}(x) &= h_{c_k}(f_{c_k} \circ \cdots \circ f_{c_1}(x)). \end{aligned}$$

Here, x is the training input to the model, y is the target, and $\mathcal{L}$ is a standard loss function, such as the cross-entropy loss. We call h_{c_k} the auxiliary network for module k. It produces predictions for the task directly from the outputs of the k^{th} module. During training, we update the parameters of both the main and auxiliary networks of the k^{th} module based *only* on gradients from $\mathcal{L}_{c_k}$. This means that during the backwards pass no communication is required between modules. Thus, the only communication necessary between the hardware accelerators holding each module is to propagate the activations during the forward pass. This strategy avoids accelerators idling during the backward pass, while they wait to receive gradients from subsequent accelerators.

While local training is time efficient, without backwards communication between modules it fails to match END-TO-END in test accuracy. In this work we address this problem by introducing new intermediate strategies between END-TO-END and local training, where we allow varying amounts of communication between modules. We refer to this family of strategies as N-WISE interlocking backpropagation. Figure 2 illustrates 2-WISE and 3-WISE. Here the parameters in module k are updated using gradients from $\mathcal{L}_{c_{k+(N-1)}}$, which have been propagated backwards through the intermediate modules. When N is set to 1, this is equivalent to local optimisation (Belilovsky et al., 2020) (i.e. 1-WISE); when N is set to the number of modules, this is equivalent to global optimisation (i.e. END-TO-END). By changing N, we can control the trade-off between performance and accuracy to match our application. A slight variation on this approach is to update the parameters of module k using the mean of the gradients propagated from both the $\mathcal{L}_{c_k}$ and $\mathcal{L}_{c_{k+(N-1)}}$. We use this variant in our Transformer experiments as we find that, in this specific case, it improves modelling performance without affecting step time.



END-TO-END	HOGWILD	1-WISE (local)	2-WISE	3-WISE	N-WISE
$2A$	2	2	4	6	$2N$

Figure 3: (top) Four training steps of different distributed optimisation strategies. Each row represents a time step, and each column an accelerator. 2-WISE and 3-WISE interlocking backpropagation, as introduced in this paper, are far cheaper than END-TO-END in terms of total optimisation time, and offer a natural trade off between speed and optimisation performance. (bottom) Table comparing the scaling of time per batch for different optimisation strategies applied to A accelerators. Hogwild achieves its optimal time per batch when run long enough to fill its pipeline, which is not illustrated above.

The step times of these strategies are visualised in Figure 3, which demonstrates that 2-WISE and 3-WISE are substantially faster than END-TO-END training. In the next section we expand on this figure by presenting a detailed analysis of the step time of each training method. While N-WISE has a time per gradient step that is always at least as fast as END-TO-END, we note that N-WISE will use more total computation because each module in a model trained using N-WISE must perform N backward passes, while END-TO-END performs only one backward pass per module. These additional backward passes do not cause any increase in training time because they occur while the accelerator would otherwise be idle, however it is possible they could affect the total energy used during training. Whether N-WISE will increase or decrease total energy usage will depend on the exact scenario, as it is influenced by the power used by the accelerators when idle, and any reduction in total training time due to N-WISE being faster than END-TO-END to achieve a given accuracy (as demonstrated in Section 4).

During N-WISE testing, we only make predictions using the output of the final module, and this is what we use to compute test accuracy. An alternative approach would be to ensemble the predictions of the auxiliary network at each module. However, experimentally we find that this does not significantly increase performance, likely due to the modules being highly correlated, and in some cases earlier modules having much lower accuracy than later ones (see Table 3 in Appendix A).

3.1 Training Speed of Interlocking Backpropagation

Interlocking backpropagation allows shorter training step times than end-to-end learning because the gradients do not need to pass through the entire network. The step time can be controlled and reduced dramatically by lowering the N parameter of N-WISE. A different model parallelism technique is pipeline parallelism, as used by GPipe (Huang et al., 2019). Here each mini-batch is split into multiple micro-batches, allowing multiple mini-batches to be processed simultaneously by different modules of the network. In fact, interlocking backpropagation can be combined with micro-batching as done in GPipe to yield further reductions in training time. In this section we propose a general algebraic model of *time per batch* for synchronous model-distributed learning. Our model uses three parameters to represent local learning, end-to-end learning, GPipe and interlocking backpropagation, allowing an in-depth comparison of the performance of these methods. Additionally, it allows us to examine the combination of interlocking backpropagation and micro-batching. We fit this model to our experiment results, thus the predictions it makes are supported by practical data from experiments rather than being purely theoretical.

The parameters of the model are as follows:

- A – the number of sequential accelerators (modules).
- M – the number of micro-batches per mini-batch.
- N – the N-WISE parameter, in the range $[1, A]$.

If $M = 1$ then the model corresponds to the training methods without micro-batching described earlier in the paper. $N = 1$ corresponds to local learning, $N = A$ to end-to-end learning, and $1 < N < A$ to N-WISE interlocking backpropagation. If $M \geq 2$ then micro-batching is enabled. In this case $N = A$ corresponds to GPipe as described by Huang et al. (2019), and other values of N correspond to pipeline parallised versions of N-WISE.

As we are modelling a synchronous setting, we split the time domain into segments of equal duration, as visualised in Figure 3. We denote the duration of a single time slot by $c(M)$, and refer to it as the micro-batch processing cost. Our model makes the simplifying assumption that a forward and backward pass are the same duration. Relaxation of this assumption is discussed in Appendix C.

The time per mini-batch (equivalently, per one gradient update) can then be derived as:

$$T(A,M,N) = \begin{cases} (2+A-N)\cdot Mc(M) + 2(2N-A-1)\cdot c(M) & (\text{for } M>2, A<2N-1) \\ (N+1)\cdot Mc(M) & (\text{for } M>2, A\geq 2N-1) \\ (N+1)\cdot Mc(M) & (\text{for } M=2) \\ 2N\cdot Mc(M) & (\text{for } M=1) \end{cases}$$

For end-to-end learning ($N=A$), the formula degenerates to $T(A,M) = 2(M+A-1)\cdot c(M)$. We define the micro-batch processing cost $c(M) := c_0 + c_1/M$, finding that this choice yields a good fit to our experimental data (see Appendix C, Figure 10). This formula accounts for both a constant overhead present for each micro-batch (e.g. waiting times in inter-accelerator communication) and a processing time proportional to the number of examples in a micro-batch. Details of the model derivation, micro-batching timing diagrams, and fit of the c_0, c_1 parameters to experimental timing data, are presented in Appendix C.

First, we note that for N-WISE with $N < 1 + A/2$ the time per batch is completely independent of the total number of modules of the network. This demonstrates that the timing benefits of N-WISE are most pronounced for big models, distributed across a large number of accelerators. Next, we compare N-WISE with micro-batching to GPipe. For $c(M)$ as defined above, the benefits of micro-batching are maximised at $M \propto \sqrt{A}$ for GPipe, while $M=2$ is optimal for N-WISE (assuming $N < A/2+1$). For the optimal choice of M in each case, Figure 4 shows the predicted speed-up of N-WISE with micro-batching over GPipe. For optimal choices of M, and the cost function parameters ($c_0 = 0.025\text{s}, c_1 = 1.279\text{s}$) tuned to results of our Transformer experiments, the model predicts a 50% speed-up of time per batch (compared to end-to-end) for 2-WISE at 15 or more accelerators. This demonstrates that interlocking backpropagation and GPipe are complimentary techniques that are suitable to being combined. Having demonstrated this, in the rest of the paper we consider the variant of interlocking backpropagation without micro-batching.

3.2 Information Flow in Interlocking Backpropagation

Unlike local training, in N-WISE interlocking backpropagation the gradient from the loss at the final layer affects all the parameters of the network, it just does so indirectly. In 2-WISE training, the first module's parameters are updated to optimise the second module's loss. The second module's loss depends on its parameters, which are updated to optimise the third module's loss, and so on, until the final loss. As a result, there is indirect communication from modules at the head of the model to previous modules. Note that a similar argument about indirect gradient communication in this type of model was made concurrently by Xiong et al. (2020).

Consider a network with n modules trained using 2-WISE:

$$f = f_{c_n} \circ \cdots \circ f_{c_1}$$

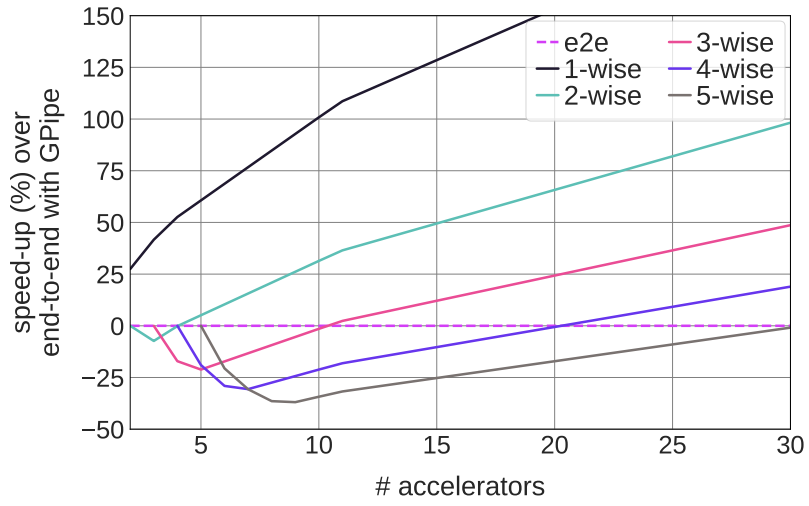


Figure 4: Modelled time per batch speed-up for N-WISE, compared to END-TO-END with GPipe, for varying numbers of accelerators. The optimal number of micro-batches is assumed at each point of the plot (see Appendix C Table 7). Micro-batch cost $c(M) = 0.025\text{s} + 1.279\text{s}/M$, tuned to our Transformer experiments, is used. Experimental measurements validating the timing model are presented in Appendix C, Figure 10.

Each auxiliary network approximates the composition of the modules above it:

$$h_{c_k} \approx f_{c_n} \circ \cdots \circ f_{c_{k+1}}$$

For training to work effectively, this approximation should be as close as possible, so that the gradient computed from $\mathcal{L}_{c_k}$ encourages f_{c_k} to learn a representation which is useful for the modules above. This observation points to a trade-off that arises between modelling quality and compute efficiency, controlled by the size of the auxiliary networks.

In 1-WISE training, under the particular supervised learning setting we consider in this paper, several factors discourage h_{c_k} from being a good approximation of the remainder of the modules. As h_{c_k} has much lower capacity than the rest of the network in the modules above, it may encourage f_{c_k} to greedily learn a simpler representation which is more amenable to immediately computing the logits, rather than feeding into the subsequent modules. This simpler representation may throw away information which the subsequent modules could use to achieve higher test accuracy.

In 2-WISE training, we hope that the communication between modules allows upstream modules to learn a representation that is useful for the downstream modules. We argue that this could happen by starting at the head and walking down the model. The penultimate module $f_{c_{n-1}}$ is updated using gradients which have propagated from the true loss at the head of the network and through f_{c_n}. Thus, $f_{c_{n-1}}$ is incentivised to learn the most useful representation for $f_{c_{n-1}}$, rather than learning a representation which improves the performance of $h_{c_{n-1}}$. Now we examine the updates of the $f_{c_{n-2}}$. This module is updated with gradients

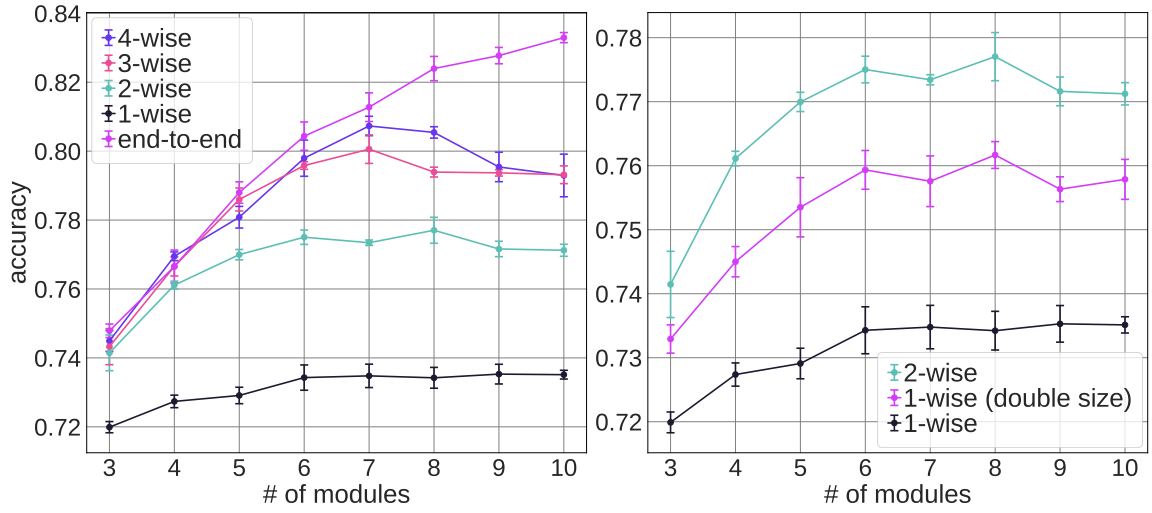


Figure 5: (left) Test accuracy of small convolutional models on CIFAR-10, comparing across depth and training method. (right) Comparison demonstrating that simply doubling the size of modules trained locally does not recover the full benefits of 2-wise training. In both cases the error bars show one standard deviation over four random seeds.

that propagate from $\mathcal{L}_{c_{n-1}}$, and so depend on $h_{c_{n-1}}$. If $h_{c_{n-1}}$ is a close approximation to f_{c_n}, then these gradients push $f_{c_{n-2}}$ towards a function which outputs a useful representation for both $f_{c_{n-1}}$ and f_{c_n}. As $h_{c_{n-1}}$ and f_{c_n} both have the same inputs and targets we hope that $h_{c_{n-1}}$ should become a close approximation to f_{c_n}, as close as possible given the limited capacity of $h_{c_{n-1}}$. Thus, $f_{c_{n-2}}$ should learn a useful representation for $f_{c_{n-1}}$ and f_{c_n}. We can continue to extend this reasoning down the model.

4. Experiments

Here we present results of experiments in both the image and language domains. We compare our n-wise strategies to both extremes of local and end-to-end training.

4.1 Validation of Approach Using a Small Convolutional Network

We investigate the behaviour of our method when training a small convolutional network on the CIFAR-10 dataset (Krizhevsky and Hinton, 2009). We consider models of 3 to 10 convolutional layers. Each module of the model contains a single convolutional layer and batch norm, with the final two modules also containing max pool layers. The auxiliary networks used for the local losses are comprised of a single linear layer. Appendix B gives full details of the experiment configuration. We train the model using several approaches: 1-wise, 2-wise, 3-wise, 4-wise, and end-to-end. Figure 5 shows how the different approaches to training perform, as we increase the number of modules in each model. Unsurprisingly, we see that end-to-end optimisation results in the best test accuracy, and local optimisation results in the worst. Increasing the n-wise N parameter consistently improves performance, validating our view of interlocking backpropagation as an interpolation between local and

		END-TO-END	1-WISE	2-WISE	3-WISE
CIFAR-10	ResNet-32	95.20 (0.11)	94.20 (0.09)	95.05 (0.09)	**95.42** (0.06)
CIFAR-100	ResNet-32	76.71 (0.14)	75.02 (0.09)	**78.09** (0.13)	77.84 (0.04)
ImageNet	ResNet-50	75.60	72.05	74.45	**76.27**

Table 1: Accuracy of ResNet-32 and ResNet-50. For CIFAR we give the accuracy on the test set, for ImageNet we give the accuracy on the validation set. One standard error over three seeds is given in brackets for CIFAR. Bold indicates the best performing strategy. The auxiliary networks used are larger than in the toy experiments illustrated in Figure 5, explaining the decreased gap between local and end-to-end learning performance.

global training. For models with 6 or fewer modules, we find that 3-WISE and 4-WISE methods achieve comparable accuracy to END-TO-END training, within standard deviation. A performance gap between END-TO-END and N-WISE for small values of N appears, as we increase the number of modules. This is likely a result of increasing and accumulating approximation errors of the auxiliary networks. However, interlocking backpropagation is shown to remain a viable middle ground between local and global training for all model sizes, as it consistently outperforms local learning, while offering increasing speed-ups over END-TO-END. While this is only a toy setup, these conclusions are also supported by our examination of ResNets and Transformer networks later in the paper.

In order to understand the interplay between the number and size of the modules in the network and the optimisation strategy, we compare our 2-WISE training scheme to an alternative approach with similar compute efficiency in Figure 5 (right). This approach, labelled '1-wise (double size)', is equivalent to performing 1-WISE training, but with adjacent pairs of modules merged to give half the number of modules, each of which is twice the size. As merging modules is not possible in practice – each module would be large enough to fill an entire accelerator – we could implement this method by grouping modules into blocking pairs. We are interested in a comparison with this method because it is similar to 2-WISE, except there is no possible indirect communication between modules which are not directly adjacent. For 2-WISE, we have proposed that the fact that the pairs of modules are overlapping induces indirect communication further down the model than just the adjacent module that the gradients are passed to. Figure 5 shows that '1-wise (double size)' performs better than 1-WISE, but not as well as 2-WISE, which suggests that there is indirect end-to-end information flow happening in 2-WISE.

4.2 Image Domain Results Using ResNet Models

Having investigated our method in a toy setting, in this section we demonstrate that it continues to lead to improved performance with a more realistic model architecture. In particular, we consider ResNets (He et al., 2016) on CIFAR-10, CIFAR-100, and ImageNet (Deng et al., 2009). While a ResNet is usually sufficiently small to fit on a single accelerator, these results suggest that our method would work with significantly larger vision models

which require multiple accelerators. In the next section we consider a language model, a Transformer, which is is too large to fit on a single accelerator.

Table 1 compares the performance of different training schemes for a ResNet-32 and ResNet-50. In both cases the model is split into four modules by grouping layers with the same feature map size, with the first module containing the input layers, and the last the output layers. For example, the ResNet-50 we use for the ImageNet experiments is split as follows. Referring to He et al. (2016, Table 1), the first module contains the layers labelled 'conv1' and 'conv2_x', the second module contains 'conv3_x', the third 'conv4_x', and the forth 'conv5_x' and the output layers. For the auxiliary classification network we use two convolutional layers with batch norm, global average pooling and a single linear layer. The full experiment configuration is given in Appendix B. These results show that N-WISE training substantially closes the performance gap between local and END-TO-END training. The results also show that 3-WISE training does not consistently offer better performance than 2-WISE training which, given the results in Section 4.1, is what we would expect for a model with only 4 modules.

In fact we see that for both CIFAR-10 and CIFAR-100, interlocking backpropagation outperforms END-TO-END training. This is a surprising finding, as one would expect that a model in which all features were trained to optimise the single global loss would outperform a model in which modules were optimised for local losses. Our results indicate that 2-WISE training of large scale neural networks could outperform training equivalent models with END-TO-END backpropagation. This surprising result may be explained by the success of Inception Nets (Szegedy et al., 2015), which found that adding auxiliary classification losses into the model improved training performance, leading to state of the art results at the time they were first published. We have argued that interlocking backpropagation may be able to propagate information from the top level loss to the initial layers, and Inception Nets showed that auxiliary losses improved performance on CIFAR-10. The success of interlocking backpropagation in this setting may be understood as the consequence of these two findings.

We also use these ResNet experiments to further investigate the behaviour of each module in the network. Figure 6 shows the test accuracy of the outputs of the auxiliary network of each module in a network trained using 1-WISE, compared against a network trained using 2-WISE. Examining the accuracy of the model trained using 1-WISE, we can see that the accuracy of module 2 is close to the accuracy of the overall model. This suggests that the model is encountering information loss, as later layers are unable to improve on the representations learned by earlier layers. The lower modules of the model learn a representation which is suited for the low capacity auxiliary network to map to the logits, throwing away useful information which the subsequent modules could otherwise use to improve accuracy further. In contrast, if we examine the 2-WISE performance we notice that the training accuracy increases more gradually with each module, indicating that that this training regime is able to make use of the additional modules to improve performance.

Finally, we compare 2-WISE to an alternative training method which we refer to as FINE-TUNING, which is another way of interpolating between local and end-to-end learning. In this method, we first train for T_1 epochs using 1-WISE, then continue training the model using END-TO-END for T_2 epochs. To compare this to the other training methods, we consider a setting where we have a small time budget within which we wish to achieve the best accuracy possible. Note that, if a large time budget is available, one can achieve good performance

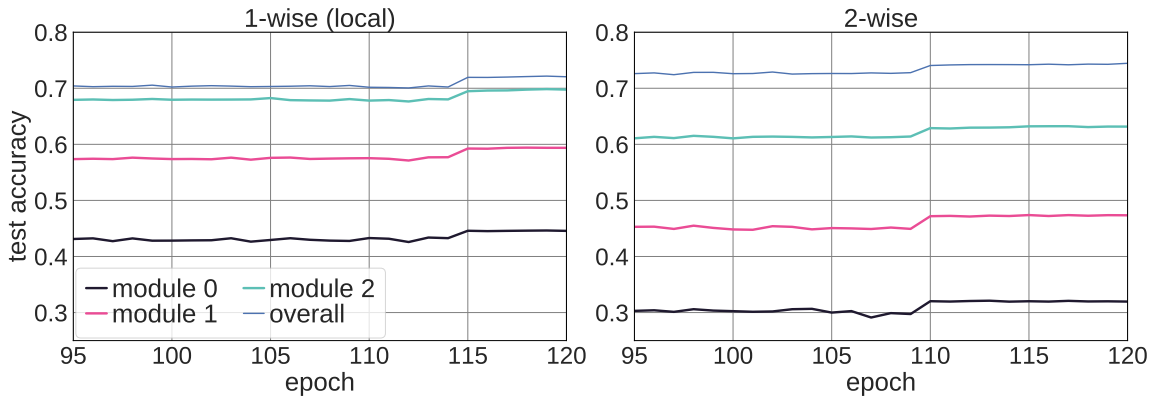


Figure 6: Test accuracy of each auxiliary classification head from a ResNet-50 model trained on ImageNet, trained with 1-wise (left) and 2-wise (right). 1-wise training leads to each module attempting to solve the entire problem on its own; this causes earlier modules to out-perform 2-wise, but ultimately, the final performance of 2-wise's more incremental solution strategy is better than 1-wise's.

with fine-tuning by simply setting $T_1 = 0$ and training fully with end-to-end. The specific scenario we consider is training a ResNet-32, split into four modules as above, on CIFAR-10 and CIFAR-100. On CIFAR-10 we consider a time budget corresponding to 80 epochs of 2-wise training, and on CIFAR-100 a budget of 100 epochs of 2-wise training. We convert this into training budgets for the other methods using the table at the bottom of Figure 3. For example, for CIFAR-100 we run 50 epochs of end-to-end training and 200 epochs of 1-wise training. In the case of fine-tuning, the total number of epochs is $T_1 + T_2$, where T_1 is a hyperparameter and $T_2 = (2 \cdot (\text{num 2-wise epochs}) - T_1)/4$. In addition to T_1, another important hyperparameter for fine-tuning is the learning rate used at the start of the end-to-end phase, which we denote by η_2. We select both T_1 and η_2 using a grid search, the results of which are given in Table 5 in Appendix A. Table 2 compares the test accuracy of fine-tuning to 1-wise, 2-wise, and end-to-end. It reveals that, in these time constrained scenarios, both 2-wise and fine-tuning are able to outperform 1-wise and end-to-end training. Taking account of the standard error, 2-wise and fine-tuning perform similarly, though fine-tuning has a higher mean accuracy on the less complex task (CIFAR-10) and smaller time budget, and 2-wise a higher mean accuracy on the more complex task (CIFAR-100) and slightly larger time budget. However, achieving this performance with fine-tuning requires a large grid search to choose T_1 and η_2, which makes it difficult to apply to large models that require a large amount of compute to train. In particular, we were not able to evaluate fine-tuning on the language models featured later in this paper due to the computational cost. Full details of the fine-tuning algorithm, and other experiment configuration, are given in Appendix B.

	end-to-end	2-wise	1-wise	fine-tuning
CIFAR-10	92.86 (0.15)	94.06 (0.12)	93.98 (0.07)	**94.21 (0.07)**
CIFAR-100	73.98 (0.13)	**76.85 (0.14)**	74.33 (0.09)	76.40 (0.16)

Table 2: Test accuracy when training under a fixed time budget, including results using fine-tuning. In brackets we give the standard error over 4 random seeds, and bold indicates the highest mean.

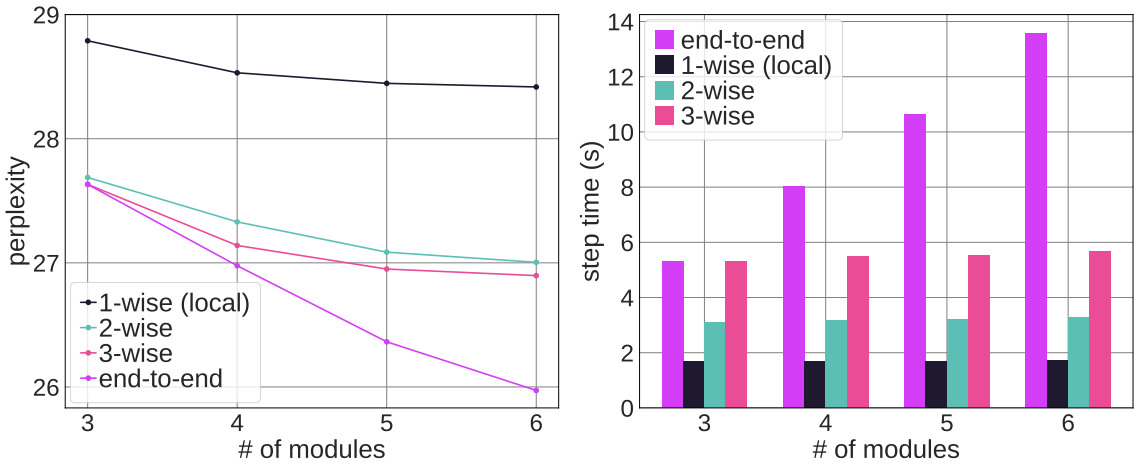


Figure 7: (left) Comparison of test perplexity of Transformer models across depth and training method, for a single simulation in each configuration. Models are run for a *fixed number of steps*. 2-wise recovers a large amount of model performance lost in local training. (right) Comparison of training step time (in wall-clock seconds) of Transformer-based models across depth and training method.

4.3 Language Domain Results Using Transformer Models

Finally we investigated the performance of our method for training Transformer based models on language modelling tasks. The architecture we used largely follows the decoder only Transformer described in OpenAI's GPT-2 model (Radford et al., 2019), with the addition of the auxiliary classification networks used for calculating the local losses. See Appendix B.3 for details. In these experiments each module is made out of 6 Transformer blocks. We run experiments with networks comprised of 3 to 6 modules. Each module was trained on a v3-8 TPU. We trained and evaluated the models with the One Billion Word Benchmark for Language Modelling (Chelba et al., 2013). Each Transformer block module has a dimensionality of 1024. We train with a max sequence length of 128, and a batch size of 1024. For the experiments in Figure 7, we train for one epoch with the Adam optimiser; for the experiments in Figure 8 we train for 192 hours (eight days). We measure the performance of the models using perplexity, for which a lower value indicates better performance. Figure 7 shows the test set perplexity of models trained with 1-wise,

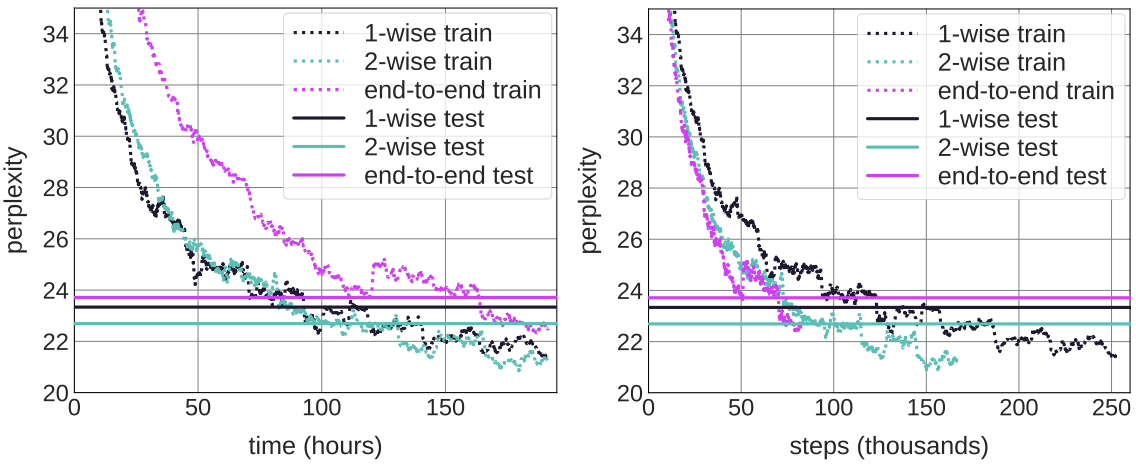


Figure 8: While Figure 7-(left) may suggest that end-to-end training always out-performs local training, it is important to keep in mind that this is still in the 'fixed steps' perspective; i.e we fix the number of steps each model is allowed to take and ignore the fact that running times may differ wildly. This figure shows the training (dashed lines) and test (solid lines) perplexities (measured at the end of training), of Transformer models comprised of four modules and trained for eight days. The results are reported for a single simulation in each configuration. It demonstrates the importance of considering the 'fixed time' perspective: (left) depicts the training curves of 1-WISE, 2-WISE, and END-TO-END in a 'per time' perspective; (right) depicts the same training curves in a 'per step' perspective. The difference between these two perspectives is quite extreme – from the 'per step' perspective, end-to-end training is best at any given point; however, when a 'per time' perspective is considered, 1-WISE and 2-WISE are best at any given point. Given a fixed time constraint, the logical decision to obtain the best possible model is to opt for an N-WISE strategy.

2-WISE interlocking, 3-WISE interlocking and, END-TO-END backpropagation. We can see that, unsurprisingly, END-TO-END greatly outperforms 1-WISE training and the gap between them widens as we increase the size of the model. Interlocking backpropagation is able to make up much of the gap between 1-WISE and END-TO-END, but unlike our observations with CIFAR-10 and CIFAR-100, there is still a test perplexity gap between interlocking and END-TO-END backpropagation for a fixed number of training steps.

In this real world setting we are able to substantially decrease the training time of these large models. The time per step for models of this size varies considerably based on the optimisation strategy used. Figure 7 visualises the test set perplexity and the train step time for models of various sizes trained with END-TO-END, 1-WISE, and 2-WISE interlocking backpropagation. 2-WISE interlocking backpropagation requires less than half the training step time of standard END-TO-END training, to achieve similar test perplexity for models with 4 modules.

Figure 8 demonstrates the importance of considering a 'fixed time' perspective of training. Instead of fixing the number of optimiser steps, we fix the total elapsed training time to

a set number of hours. The result is that methods, which take gradient steps quicker, see many more weight updates relative to slower methods. When running for a fixed amount of time, the performance of interlocking backpropagation improves dramatically relative to end-to-end methods.

5. Conclusion

We have explored a variety of optimisation strategies, for large scale, distributed neural networks, that strike a middle-ground between local and end-to-end training. These strategies, referred to as n-wise training, introduce intermediate auxiliary classification heads and losses into the network and train with interlocking backpropagation, which restricts the gradient flow between accelerators. We have shown that 2-wise interlocking backpropagation significantly reduces training time of large scale distributed neural networks, and recovers much of the test accuracy that is lost in local training. 2-wise training even outperforms end-to-end on ResNets trained on CIFAR-10 and CIFAR-100, though it does not outperform baselines in language modelling tasks with Transformers. We have provided evidence that varying the N parameter of n-wise can be seen as interpolating between 1-wise and end-to-end training, in terms of compute efficiency and modelling performance. Interlocking backpropagation has been shown to be a practical approach to training large scale distributed neural networks.

Acknowledgments

Oscar Key acknowledges funding from the Engineering and Physical Sciences Research Council (EPSRC), grant number EP/S021566/1.

References

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pages 736–745. PMLR, 2020.

Ciprian Chelba, Tomás Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013.

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

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

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32:103–112, 2019.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International Conference on Machine Learning*, pages 1627–1635. PMLR, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Michael Laskin, Luke Metz, Seth Nabarrao, Mark Saroufim, Badreddine Noune, Carlo Luschi, Jascha Sohl-Dickstein, and Pieter Abbeel. Parallel training of deep networks with local updates. *CoRR*, abs/2012.03837, 2020. URL `https://arxiv.org/abs/2012.03837`.

Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, pages 3033–3045, 2019.

Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. *Frontiers in neuroscience*, 12:608, 2018.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters. *CoRR*, abs/2104.04473, 2021.

Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In *International Conference on Machine Learning*, pages 4839–4850. PMLR, 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Yuwen Xiong, Mengye Ren, and Raquel Urtasun. Loco: Local contrastive representation learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11142–11153. Curran Associates, Inc., 2020.

Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake A. Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, Ruoming Pang, Noam Shazeer, Shibo Wang, Tao Wang, Yonghui Wu, and Zhifeng Chen. GSPMD: general and scalable parallelization for ML computation graphs. *CoRR*, abs/2105.04663, 2021.

Supplementary Material

A. Additional Results

	ImageNet		CIFAR-10	
	2-wise	1-wise	2-wise	1-wise
top 1	**74.45**	**72.05**	95.12	94.78
top 2	73.46	71.70	95.10	**94.89**
top 3	72.46	71.34	95.18	94.76
top 4	71.75	70.92	**95.26**	94.82

Table 3: Accuracy when ensembling the predictions made by the top n modules in the model, when using 2-wise interlocking backprop model and a 1-wise local training. On CIFAR-10 we report accuracy on the test set, on ImageNet we report validation on validation set. We present this as a negative result, demonstrating that ensembling the classification heads does not lead to noticeably better performance. In the case of ImageNet, ensembling actually hurt performance.

# Modules	3	4	5
Perplexity	82.682	100.183	110.278

Table 4: The extremely poor task performance of Hogwild optimisation due to gradient staleness (see Fig. 1). The model is the same Transformer used in Figure 7 with the same optimisation parameters and number of training epochs. Increasing the number of modules used causes the average staleness for modules to increase, leading to extremely unstable training dynamics that makes optimisation difficult. Despite Hogwild improving training speeds to a similar effect as local learning, it is ultimately too poor at model optimisation to be considered a viable alternative to end-to-end learning.

B. Experiment Details

B.1 Figure 5 (small convolutional network experiments)

The configuration of an n-module main network is as follows, where each box represents a module:

T_1	η_2	val. accuracy	
		CIFAR-10	CIFAR-100
60.0	0.1	93.78 (0.10)	-
	0.01	93.31 (0.09)	-
80.0	0.1	**93.86 (0.13)**	**77.00 (0.21)**
	0.01	93.40 (0.12)	75.66 (0.19)
100.0	0.1	93.70 (0.10)	76.71 (0.27)
	0.01	93.56 (0.11)	75.46 (0.16)
120.0	0.1	93.53 (0.13)	76.78 (0.22)
	0.01	93.45 (0.06)	75.42 (0.17)
	0.001	93.46 (0.14)	-
140.0	0.1	-	76.10 (0.13)
	0.01	93.26 (0.10)	75.24 (0.07)
	0.001	93.34 (0.06)	-
160.0	0.1	-	76.16 (0.17)
	0.01	-	75.32 (0.17)
	0.001	-	74.57 (0.21)
180.0	0.01	-	74.72 (0.18)
	0.001	-	74.50 (0.02)

Table 5: Results of a grid search to choose the hyperparameters for the FINE-TUNING results in Table 2. We search over T_1 – the number of epochs of 1-WISE training – and η_2 – the learning rate at the start of the END-TO-END phase of training. Bold indicates the optimal configuration, and we give the standard error over four random seeds in brackets.

logits
linear
maxpool2d kernel=2x2 stride=1
conv2d kernel=3x3 filters=64 padding=1 stride=1 (batch norm, ReLU)
maxpool2d kernel=2x2 stride=1
conv2d kernel=3x3 filters=64 padding=1 stride=1 (batch norm, ReLU)
conv2d kernel=3x3 filters=32 padding=1 stride=1 (batch norm, ReLU)
(this module repeats $n-3$ times)
...
conv2d kernel=3x3 filters=32 padding=1 stride=1 (batch norm, ReLU)
input image (32×32)

For the 1-WISE and N-WISE configurations, the auxiliary network is a single linear layer.

We use the Adam optimizer with a learning rate of 0.0001. We train for 100 epochs. We do not use learning rate decay, weight decay, or data augmentation. We train on the entire

training set, and report results on the test set. We normalize the inputs based on the mean and standard deviation of the training set.

B.2 Table 1, Table 2, and Figure 5 (ResNet experiments)

We use the CIFAR-10, CIFAR-100 and ImageNet ResNet architectures as given by He et al. (2016). For the 1-wise and n-wise configurations, the auxiliary network is:

approximated logits
linear
global average pool
conv2d kernel=3x3 filters=64 padding=1 stride=1 (batch norm, ReLU)
conv2d kernel=3x3 filters=128 padding=1 stride=1 (batch norm, ReLU)
main network output

The training configuration shared between experiments in this section is as follows:

	CIFAR	**ImageNet**
optimizer	SGD	
initial learning rate	0.1	
weight decay	0.0002	
momentum	0.9	
data augmentation	random horizontal flip, 32×32 crop with padding 4	random resized crop of 224×224, random horizontal flip
test data preprocessing	-	resize so that shortest side has length 224, take 224×224 center crop
batch size	128	256

We normalize the inputs based on the mean and standard deviation of the training set.

B.2.1 Table 1 and Figure 5

In addition to the training configuration above, for these experiments we use the following configuration:

	CIFAR	**ImageNet**
learning rate schedule	divide by 10 at epochs $[91, 136, 182]$	divide by 10 if the validation loss does not improve for 10 epochs
total epochs	200	120

For CIFAR we train on the entire training set, and report results on the test set. For ImageNet we train on the test set, and report results on the validation set.

B.2.2 Table 2 (Fine-tuning experiments)

The fine-tuning algorithm is

1. Train for T_1 epochs using 1-wise.
2. Discard the auxiliary networks, set the learning rate to η_2, and reset any optimizer state.
3. Train for T_2 epochs using end-to-end.

As discussed in the main paper body, for CIFAR-10 we train for the equivalent of 80 2-wise epochs, and for CIFAR-100 the equivalent of 100 2-wise epochs. The number of epochs for each method is then as follows:

	CIFAR-10	CIFAR-100
end-to-end	40	50
2-wise	80	100
1-wise	160	200
fine-tuning	$T_1 + T_2$	

For fine-tuning, T_1 is a hyperparameter, with $T_2 = (2 \cdot 80 - T_1)/4$ for CIFAR-10 and $T_2 = (2 \cdot 100 - T_1)/4$ for CIFAR-100.

One difficulty with obtaining the results in Table 2 is selecting the learning rate schedule for each configuration, because the large number of configurations means it is too expensive to optimize it in each case. Thus we use a heuristic approach to set the schedule. For end-to-end, 2-wise and 1-wise, we start with a learning rate of 0.1 and reduce by a factor of 10 at epochs $0.7n$, $0.85n$ and $0.95n$, where n is the total number of epochs. For fine-tuning we follow the same learning rate schedule as 1-wise for the first phase. For the second phase we start at a learning rate of η_2, and reduce the learning rate by a factor of 10 at approximately epochs $T_1 + 0.7T_2$, $T_1 + 0.85T_2$ and $T_1 + 0.95T_3$. The specific epochs at which we reduce the learning rate are given in Table 6.

For these experiments we withheld 10% of the training set to create a validation set, which we used to perform the grid search.

B.3 Transformer Experiments

The main network's Transformer blocks have 4 attention heads, an embedding dimension of 1024, and sequence length of 128. Each module is composed of 6 such Transformer blocks, amounting to 152M parameters per module, including the auxiliary networks. The first and last modules additionally contain input and output projection layers, respectively.

The architecture of the auxiliary network for each module in the Transformer experiments is:

approximated logits
embedding matrix Transformer block(attention heads=4, embedding dim=1024, max sequence length=128) x 2
main network output

	T_1	η_2	lr reduced at T_1+
CIFAR-10	60	0.010	19, 23
		0.100	18, 21, 24
	80	0.010	14, 18
		0.100	12, 16, 18
	100	0.010	10, 13
		0.100	9, 12, 14
	120	0.001	8
		0.010	6, 8
		0.100	5, 7, 9
	140	0.001	4
		0.010	3
CIFAR-100	80	0.010	22, 27
		0.100	20, 26, 28
	100	0.010	19, 23
		0.100	18, 21, 24
	120	0.010	14, 18
		0.100	12, 16, 18
	140	0.010	10, 13
		0.100	9, 12, 14
	160	0.001	8
		0.010	6, 8
		0.100	5, 7, 9
	180	0.001	4
		0.010	3, 4

Table 6: Epochs at which we reduce the learning rate by a factor of 10 during the END-TO-END phase of FINE-TUNING.

The training configuration is as follows:

optimizer	$\text{Adam}(\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9})$
learning rate	$1024^{-0.5} \times \min(\text{step}^{-0.5}, \text{step} \times \text{warmup\_steps}^{-1.5})$
lr warmup steps	4000
total epochs	1

For N-WISE Transformer experiments, we additionally use the gradient signal from the auxiliary network local to each module to update the parameters. Specifically, when updating the parameters of the k^{th} module, we take the mean of the gradients propagated from both the auxiliary network in the k^{th} module, and from the auxiliary network of the $(k+N-1)^{\text{th}}$ module. We found that this improved modelling performance, without affecting the step time.

C. The Timing Model

C.1 Deriving the time per batch formula

Recall the formula for time per batch, given the number of accelerators A, number of micro-batches M, the N-WISE parameter N, and the cost of processing a single micro-batch $c(M)$:

$$T(A,M,N) = \begin{cases} (2+A-N)\cdot Mc(M) + 2(2N-A-1)\cdot c(M) & (\text{for } M>2, A<2N-1) \\ (N+1)\cdot Mc(M) & (\text{for } M>2, A\geq 2N-1) \\ (N+1)\cdot Mc(M) & (\text{for } M=2) \\ 2N\cdot Mc(M) & (\text{for } M=1) \end{cases}$$

In reference to the timing grid diagrams, such as pictured in Figure 9, the quantity $T(A,M,N)/c(M)$ is the smallest period of the timing diagram, expressed in number of grid cells.

Case 1: $(M=1)$**.** Before the first accelerator can start processing the next mini-batch, the previous micro-batch must undergo N forward passes, and then N backward passes, where the last one triggers the gradient update on the first accelerator.

Cases 2 & 3: $(M=2)$ **or** $(M>2, A\geq 2N-1)$**.** The middlemost accelerator needs to process the forward pass and N different backward passes for each micro-batch, adding up to $(N+1)M$ time slots.

Case 4: $(M>2, A<2N-1)$**.** There are $A-(N-1)<N$ accelerators that need to compute their local loss. Consider any of those $A-(N-1)$ accelerators that's not the final one; it needs to compute exactly one more backward pass than its successor, and that makes it have to delay the computation of the first backward pass it receives from its successor by $M-2$ time slots. Decreasing A by 1 eliminates one of those delays and thus decreases the total time per batch by $M-2$ time slots. For $M>2, A<2N-1$, and using the formula for (end-to-end) $T(N,M,N)$ easily derivable from the diagrams, we get:

$$\begin{aligned} T(A,M,N) &= (A-N)(M-2)c(M) + T(N,M,N) \\ &= (A-N)(M-2)c(M) + 2(M+N-1)\cdot c(M) \\ &= (2+A-N)\cdot Mc(M) + 2(2N-A-1)\cdot c(M) \end{aligned}$$

C.2 Comparison of timing model with experimental data

To validate our timing model against empirical data, we measure time per batch during end-to-end, 2-WISE, and 3-WISE training of Transformer models, distributed over 4 or 6 accelerators. The architecture is as described in Section 4.3, and the results are shown in Figure 10. The micro-batch cost parameters c_0, c_1 are fitted to match the timing data, as they inherently depend on the hardware and model architecture.

The ratio c_0/c_1 controls the speedups offered by N-WISE – the higher it is, the less effective fine micro-batching is, and the more beneficial N-WISE is.

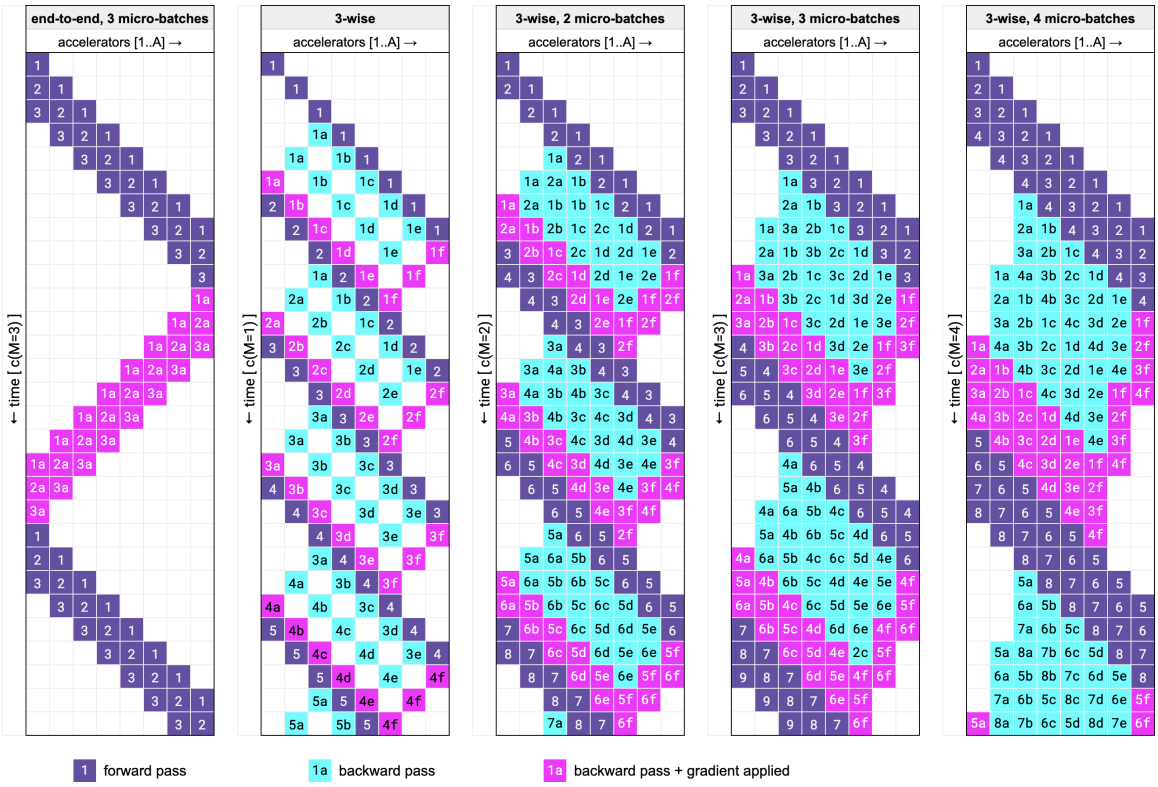


Figure 9: End-to-end and 3-WISE training with GPipe micro-batching. The number in each square indexes the micro-batches, the letter indexes different backward passes of a given micro-batch. The vertical time axis in each diagram is scaled by the micro-batch cost $c(M)$. The larger the M, the shorter the duration of a single grid cell in seconds.

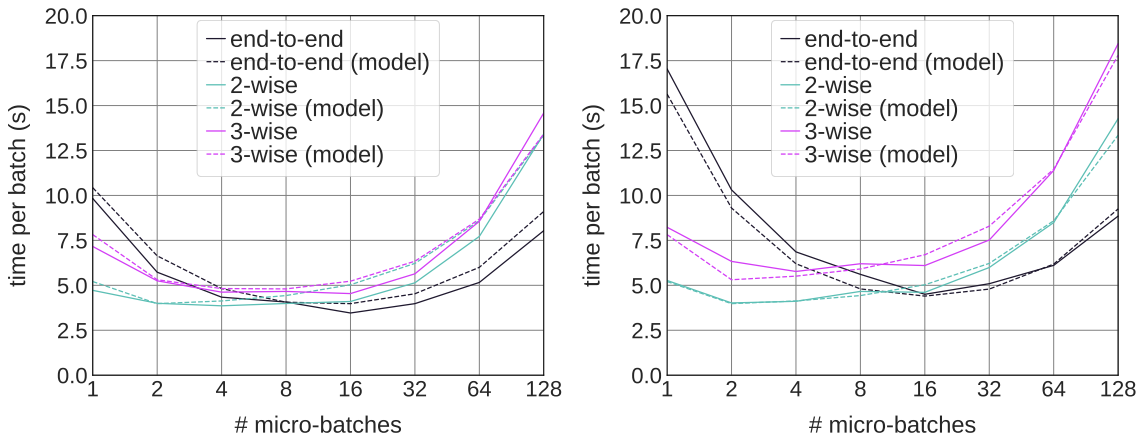


Figure 10: Comparison of experimental and modelled training time per batch, across different micro-batching granularities. (left) $A = 4$ accelerators, (right) $A = 6$. The micro-batch cost parameters were fit to $c_0 = 0.025\text{s}, c_1 = 1.279\text{s}$.

Number of accelerators A	2	3	4	5	6 – 11	12 – 30
Optimal M for end-to-end	8	8	16	16	16	32
Optimal M for 1-WISE	1	1	1	1	1	1
Optimal M for 2-WISE	8	2	2	2	2	2
Optimal M for 3-WISE	8	4	2	2	2	2
Optimal M for 4-WISE	8	8	4	2	2	2
Optimal M for 5-WISE	16	8	8	4	2	2

Table 7: Optimal number of micro-batches (restricted to powers of 2) per mini-batch for a range of training strategies, and increasing numbers of accelerators, given micro-batch cost parameters $c_0 = 0.025\text{s}, c_1 = 1.279\text{s}$.

Finally, it is worth noting that forwards passes might be significantly faster than backwards passes, which is not modelled by our timing model. For instance, in a setting where forwards passes are two times faster than backwards passes, the speed of end-to-end learning is underestimated by the model by 25%. However, that setting still allows Interlocking Backpropagation to achieve large step time improvements for deep enough models.

C.3 Optimal number of micro-batches

Assuming our timing model, the optimal number of micro-batches to use with N-WISE is 2 (when $N < 1 + A/2$), while for end-to-end training, it grows as $\Theta(\sqrt{A})$. Figure 4 assumes the optimal micro-batch size at each point of the plot, only constraining M to be a power of 2. Those optimal values are listed in Table 7.

C.4 Other forms of model-parallelism

The term 'model parallelism' has been used in literature to refer to two distinct ideas – pipeline parallelism (an example of which is GPipe, discussed in this work) and tensor parallelism, also known as tensor sharding, introduced in the Megatron framework (Shoeybi et al., 2019). Tensor parallelism pertains to splitting large tensors, and computations involving them, between multiple accelerators.

Tensor parallelism, pipeline parallelism, and interlocking backpropagation can all be used or disabled independently for any model. The interaction of tensor parallelism and interlocking backpropagation is less complex than that of GPipe and interlocking backpropagation, as the former would simply affect the time of processing a micro-batch, $c(M)$, in our timing model, at the cost of consuming extra accelerators.

Some timing analysis – akin to ours in its goals – of how pipeline and tensor parallelism interact, can be found in Narayanan et al. (2021). Together with our timing analysis, it could provide insight on how to optimally use interlocking backpropagation, GPipe, and tensor sharding all together. Furthermore, an automated framework for model parallelism, compatible with TPU accelerators, has been recently released as GSPMD (Xu et al., 2021), and could act as a starting point for practitioners interested in implementing interlocking backpropagation combined with other forms of model parallelism.