ORTHOGONALISING GRADIENTS TO SPEEDUP NEURAL NETWORK OPTIMISATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The optimisation of neural networks can be sped up by orthogonalising the gradients before the optimisation step, ensuring the diversification of the learned representations. We hypothesize that components in the same layer learn the same representations at the beginning of learning. To prevent this we orthogonalise the gradients of the components with respect to each other. Our method of orthogonalisation allows the weights to be used more flexibly, in contrast to restricting the weights to an orthogonalised sub-space. We tested this method on ImageNet and CIFAR-10 resulting in a large decrease in learning time, and also obtain a speed-up on the semi-supervised learning BarlowTwins. We obtain similar accuracy to SGD without fine-tuning and better accuracy for naïvely chosen hyper-parameters.

1 INTRODUCTION

Neural network layers are made up of several identical, but differently parametrised, components, *e.g.* filters in a convolutional layer, or heads in a multi-headed attention layer. Layers consist of several components so that they can provide a diverse set of intermediary representations to the next layer, however, there is no constraint or bias, other than the implicit bias from the cost function, to learning different parametrisations. This is undesirable since at the start of learning one might expect all the components, *i.e.* convolutional filters, to learn the same parametrisation – the parametrisation that provides the most information to the next layer — and so provide duplicate information. We introduce this diversification bias in the form of orthogonalised gradients and find a resultant speed-up in learning and sometimes improved performance, see fig. 1.

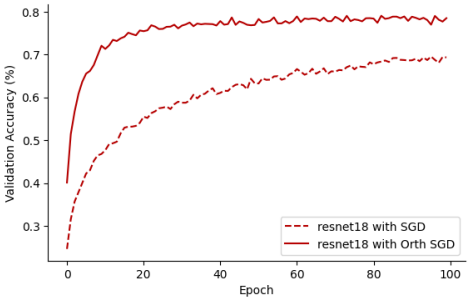


Figure 1: An example of the speed-up obtained by orthogonalising the gradients on CIFAR-10.

Our novel contributions include this new optimisation method, thorough testing on CIFAR-10 and ImageNet, additional testing on a semi-supervised learning method, and experiments to support our hypothesis.

In section 2 we detail the method and results to give an understanding of how this method works and its capabilities. Then, in section 3, we provide experimental justifications and supporting experiments for this method along with finer details of the implementation and limitations.

2 OVERVIEW OF NEW METHOD AND RESULTS

2.1 PROBLEM CONJECTURE

Initially, the components in a neural network layer activate on noise and so, to start, all the components will attempt to learn the same function, the parametrisation with the highest discriminatory

power. If we reduce the layer to have only one component, then this is the one that results in the best performing network. This parametrisation will be learned first since it will reduce the loss significantly, *i.e.* the gradient component will be largest in this direction, this is inefficient as all the components will attempt to co-learn this one parametrisation and will have to pivot and learn a different function later on — wasting the initial updates.

This is analogous to Biehl et al. (1996)'s observation that for a tiny Multi-layered Perceptron (MLP) ($\mathbb{R}^N \to \mathbb{R}^2 \to \mathbb{R}$) trained from an identically-structured teacher MLP that the student model will initially learn hidden nodes each with high correlation to both the teacher nodes (Biehl et al., 1996, Figure 1); "the student vectors are almost identical and have — apart from small deviations — the same overlap with each teacher vector." Then, after some time at this plateau, they diverge to correlate with the teacher's hidden nodes.

We can avoid this co-learning problem by orthogonalising the gradients of the components with respect to each other, speeding up the learning of the lower information components at the beginning of learning, as it is now more difficult for them to learn the same parametrisation. Orthogonalising the gradients may provide benefits in addition to avoiding this co-learning problem since it will continuously bias towards disparate feature detectors. Note that this method does not restrict the features to an orthogonal sub-space since a composition of orthogonal updates is not necessarily orthogonal itself. What we will show is that doing this significantly speeds up learning on several tasks across different data sets.

2.2 Orthogonalising Gradients

Given a neural network, f, with L layers made from components, c,

$$f = \circ_{i=1}^{L} (f_i)\,, \tag{1}$$

$$f_l(x) = [c_{l1}(x), c_{l2}(x), \ldots, c_{lN_l}(x)], \tag{2}$$

where $\circ$ is the composition operator, N_l is the number of components in layer l, $c_l : \mathbb{R}^{S_{l-1} \times N_{l-1}} \to \mathbb{R}^{S_l}$ is a parametrised function and c_{li} denotes c_l parametrised with $\theta_{li} \in \mathbb{R}^{P_l}$ giving $f_l : \mathbb{R}^{S_{l-1} \times N_{l-1}} \to \mathbb{R}^{S_l \times N_l}$ parametrised by $\theta_l \in \mathbb{R}^{P_l \times N_l}$.

Let

$$G_l = [\nabla c_{l1}, \nabla c_{l2}, \ldots, \nabla c_{lN_l}],$$

be the $P_l \times N_l$ matrix of the components' gradients.

Then the nearest orthonormal matrix, *i.e.* the orthonormal matrix, O_l, that minimises the Frobenius norm of its difference from G_l

$$\min_{O_l} \|O_l - G_l\| \quad \text{subject to } \forall i, j : \langle O_{li}, O_{lj} \rangle = \delta_{ij},$$

where δ_{ij} is the Kronecker delta function, is the product of the left and right singular vector matricies from the Singular Value Decomposition (SVD) of G_l (Trefethen & Bau III, 1997),

$$G_l = U_l \Sigma_l V_l^{\mathsf{T}}, \tag{3}$$

$$O_l = U_l V_l^{\mathsf{T}}. \tag{4}$$

Thus, we can adjust a first-order gradient descent method, such as Stochastic Gradient Descent with Momentum (SGDM) (Polyak, 1964), to make steps where the components are pushed in orthogonal directions,

$$v_l^{(t+1)} = \gamma v_l^{(t)} + \eta O_l^{(t)}, \text{ and} \tag{5}$$

$$\theta_l^{(t+1)} = \theta_l^{(t)} - v_l^{(t+1)}, \tag{6}$$

where v_l is the velocity matrix, $t \in \mathbb{Z}^{0+}$ is the time, γ is the momentum decay term, and η is the step size. We call this method Orthogonal Stochastic Gradient Descent with Momentum (Orthogonal-SGDM). This modification can clearly be applied to any first-order optimisation algorithm by replacing the gradients with $O_l^{(t)}$ before the calculation of the next iterate.

Code for creating orthogonal optimisers in PyTorch is provided at `https://anonymous.4open.science/r/Orthogonal-Optimisers`. And code for the experiments in this work is provided at `https://anonymous.4open.science/r/Orthogonalised-Gradients`

2.3 RESULTS

2.3.1 CIFAR-10

We trained a suite of models on the CIFAR-10 (Krizhevsky et al., 2009) data set with a mini-batch size of 1024, learning rate of 10^{-2}, momentum of 0.9, and a weight decay of 5×10^{-4} for 100 epochs. We then repeated this using Orthogonal-SGDM instead of SGDM and plot the results in figs. 2 and 3 and table 1.

Table 1: Test loss and accuracy across a suite of models on CIFAR-10 comparing normal SGDM with Orthogonal-SGDM, standard error across five runs.

	Test Loss		Test Accuracy (%)	
	SGDM	Orthogonal-SGDM	SGDM	Orthogonal-SGDM
BasicCNN[1]	0.7603 ± 0.0061	0.6808 ± 0.0038	73.60 ± 0.19	**76.67** ± 0.10
resnet20[2]	0.6728 ± 0.0301	0.6766 ± 0.0155	79.14 ± 0.62	**87.12** ± 0.12
resnet44[2]	0.7000 ± 0.0166	0.7600 ± 0.0299	79.81 ± 0.37	**88.12** ± 0.20
resnet18[3]	0.9656 ± 0.0104	0.8427 ± 0.0121	77.01 ± 0.21	**84.68** ± 0.12
resnet34[3]	1.0468 ± 0.0134	0.7087 ± 0.0165	75.86 ± 0.26	**85.42** ± 0.33
resnet50[3]	1.2304 ± 0.0462	0.6797 ± 0.0235	67.99 ± 0.73	**86.51** ± 0.12
densenet121[3]	1.0027 ± 0.0132	0.8669 ± 0.0132	75.26 ± 0.30	**84.34** ± 0.15
densenet161[3]	1.1399 ± 0.0096	1.1688 ± 0.1960	75.81 ± 0.20	**85.51** ± 0.19
resnext50_32x4d[3]	1.2470 ± 0.0254	0.6669 ± 0.0223	68.73 ± 0.30	**86.37** ± 0.24
wide_resnet50_2[3]	1.4141 ± 0.0337	0.7018 ± 0.0091	69.42 ± 0.33	**87.30** ± 0.12

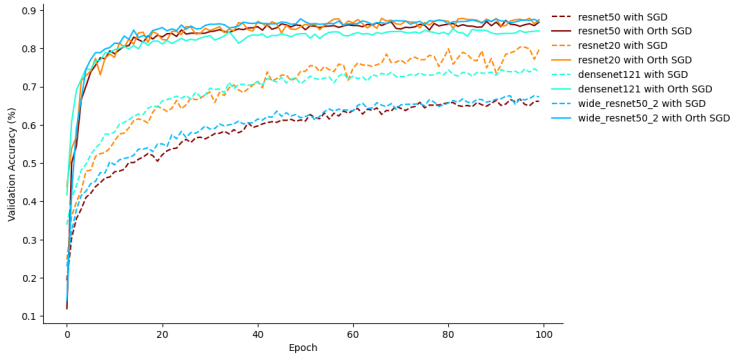


Figure 2: Validation accuracy from one run of SGDM vs Orthogonal-SGDM for a selection of models. Full plot in appendix C. Best viewed in colour.

Orthogonal-SGDM is more efficient and achieves better test accuracy than SGDM for every model we trained on CIFAR-10 without hyper-parameter tuning. More importantly though, we can see that the model learns much faster at the beginning of training, as shown by fig. 2, this means that we do not need as many epochs to get to a well-performing network. This is especially good in light of the large data sets that new models are being trained on, where they are trained for only a few epochs, or even less (Brown et al., 2020).

For SGDM the performance of the residual networks designed for ImageNet (Deng et al., 2009) (18, 34, 50) get worse as the models get bigger. The original ResNet authors, He et al. (2015), note that unnecessarily large networks may over-fit on a small data set such as CIFAR-10. However, when trained with Orthogonal-SGDM, these models do not suffer from this over-parametrisation problem and even slightly improve in performance as the models get bigger, in clear contrast to SGDM. This

[1] As described in appendix B.1
[2] Model same as in He et al. (2015)
[3] From `https://pytorch.org/vision/0.9/models.html`

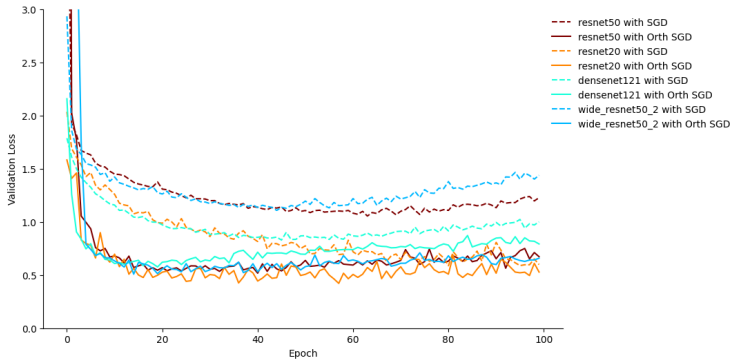


Figure 3: Validation losses from one run of SGDM vs Orthogonal-SGDM for a selection of models. Full plot in appendix C. Best viewed in colour.

agnosticism to over-parametrisation helps alleviate the need for the practitioner to tune a model's architecture to the task at hand to achieve a reasonable performance.

2.3.2 MATCHING RESNET'S PERFORMANCE

Having shown that Orthogonal-SGDM speeds up learning with non-optimised hyper-parameters, we now aim to show that it can achieve state-of-the-art results. To do this we use the same hyper-parameters as the original ResNet paper (He et al., 2015), which have been painstakingly tuned to benefit SGDM, to train using Orthogonal-SGDM.

Table 2: Test loss and accuracy of a resnet20, as in He et al. (2015), on CIFAR-10; hyper-parameter tuned to normal SGDM vs Orthogonal-SGDM, standard error across five runs. Mini-batch size of 128, see section 3.6 for why this hyper-parameter value impedes Orthogonal-SGDM, learning-rate of 0.1, momentum of 0.9, weight-decay of 10^{-4}, and a learning rate schedule of $\times 0.1$ at epochs 100, 150 for 200 epochs.

	Test Loss	Test Accuracy (%)
SGDM (He et al., 2015)	—	**91.25**
SGDM	0.4053 ± 0.0054	91.17 ± 0.28
Orthogonal-SGDM	0.4231 ± 0.0043	90.18 ± 0.30

This also tests the efficacy of Orthogonal-SGDM as a drop-in replacement for SGDM. Orthogonal-SGDM gets close to the original results, table 2, even though the hyper-parameters are perfected for SGDM. It is the authors' belief that with enough hyper-parameter tuning Stochastic Gradient Descent (SGD) or SGDM will be the best optimisation method; however, this experiment shows that Orthogonal-SGDM is robust to hyper-parameter choice and can easily replace SGDM in existing projects. Unfortunately, the authors do not have the compute-power to extensively hyper-parameter tune a residual network for Orthogonal-SGDM, however, it is exceedingly likely that better results would be achieved by doing so.

2.3.3 IMAGENET

Orthogonal-SGDM also works on a large data set such as ImageNet (Deng et al., 2009) — fig. 4. Using a resnet34, mini-batch size of 1024, learning rate of 10^{-2}, momentum of 0.9, and a weight decay of 5×10^{-4}, for 100 epochs. SGDM achieves a test accuracy of 61.9% and a test loss of 1.565 while Orthogonal-SGDM achieves 67.5% and 1.383 respectively. While these results are a way off the capabilities of the model they still demonstrate a significant speed-up and improvement from using Orthogonal-SGDM, especially at the start of learning, and further reinforces how a dearth of hyper-parameter tuning impedes performance.

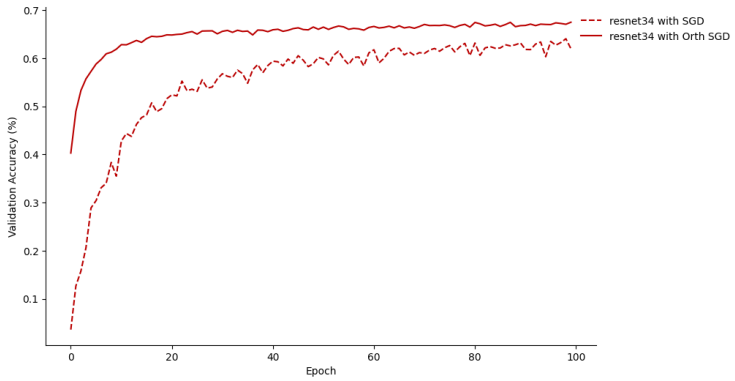


Figure 4: Validation accuracy of SGDM vs Orthogonal-SGDM on ImageNet

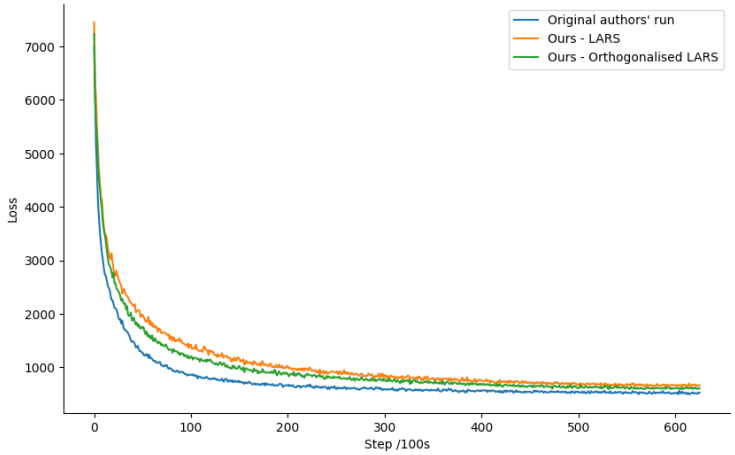


Figure 5: Barlow Twins loss during the unsupervised phase using LARS and Orthogonal LARS on ImageNet

2.3.4 BARLOW TWINS

Barlow Twins (Zbontar et al., 2021) is a semi-supervised method that uses "the cross-correlation matrix between the outputs of two identical networks fed with distorted versions of a sample" to avoid collapsing to trivial solutions. While the authors do provide code, we could not replicate their results by running it. To train within our compute limitations we used a mini-batch size of 1024 instead of 2048 however this should not affect the results since "Barlow Twins does not require large batches" (Zbontar et al., 2021). Additionally, Barlow Twins uses the Layer-wise Adaptive Rate Scaling (LARS) algorithm (You et al., 2017), which is designed to adjust the learning rate based on the ratio between the magnitudes of the gradients and weights, there should be no significant slow-down, or speed-up, in learning due to the mini-batch size. We do not orthogonalise the gradients for the dense layers (see section 3.5).

Comparing our own runs, we establish that orthogonalising the gradients before the LARS algorithm does speed up learning as shown in fig. 5, in agreement with previous experiments. This is evidence that orthogonalising gradients is also beneficial for semi-supervised learning and, moreover, that optimisation algorithms other than SGDM can be improved in this way.

3 DISCUSSION OF PROBLEM AND METHOD

3.1 SIMILARITY OF COMPONENT PARAMETRISATION

We hypothesised in section 1 that neural networks have components with parametrisations resulting in disproportionally large discriminatory powers and that the components would co-learn as a result of this. To see if there is evidence of the filers co-learning, we calculate the pair-wise cosine similarity of the weights.

Specifically, we look at the mean of C_l,

$$C_l = \{\left|\langle \theta_{l\,i}, \theta_{l\,j} \rangle_2\right| \mid i < j\},$$

that is, the absolute cosine of all distinct pairs of different components' parameters. We confirm, in fig. 6, that the similarity between the filters of the first convolutional layer in a resnet20 on CIFAR-10 is significant (see appendix A). However, there is no initial spike in similarity that would indicate multiple components co-learning the same parametrisation. Indeed, when we look at the same graph for Orthogonal-SGDM we see that the components' parameters are more similar than they were with SGDM and even include the initial spike. The reason behind this counter-intuitive result will be studied in the future. One initial theory is that components require some basic similarities – reflecting the importance of initialisation — and that it is actually beneficial to have several alike components *e.g.* when visualising GoogLeNet Olah et al. (2018) find several different "floppy ear" detectors which are clearly similar.

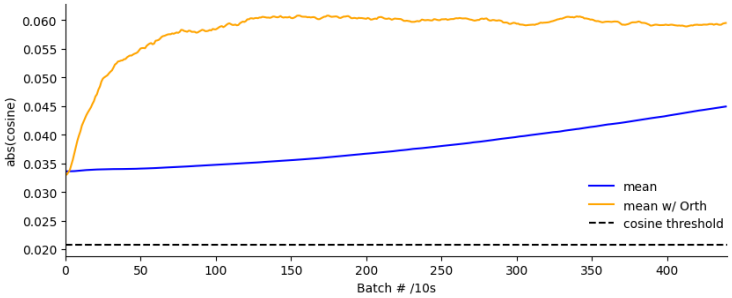


Figure 6: Mean of the absolute cosine of all distinct pairs of different components, $\mathbb{E}[C_l]$, for the second convolutional layer of the third block in layer three of a resnet20 trained on CIFAR-10 with Orthogonal-SGDM as in section 2.3.1. See appendix A for details on the cosine threshold.

3.2 DIVERSIFIED INTERMEDIARY REPRESENTATIONS

Along with different parametrisations we also desire different intermediary representations, a model will perform better if its layers output N different representations as opposed to N similar ones.

Given x_l are the resulting representations from the intermediary layers,

$$x_l = \left(\circ_{i=1}^{l} f_i\right)(x_0)$$

where x_0 is the input and x_l is the intermediary representation after layer l. Then $x_{l\,i}$ is the representation provided by $c_{l\,i}$.

We now look at the statistics of the absolute cosine of all distinct pairs of different latent features,

$$R_l = \left\{\left|\langle x_{l\,i}, x_{l\,j} \rangle_2\right| \mid i < j\right\}.$$

The representations have smaller cosines when using Orthogonal-SGDM versus SGDM — fig. 7. In addition, Orthogonal-SGDM shows a steady decline in cosine similarity throughout training. This indicates that more information is being passed to the next layer as the network is trained.

3.3 DEAD PARAMETERS

Dead parameters occur when the activation function has a part with zero gradient, *e.g.* a Rectified Linear Unit (ReLU). If the result of the activation remains in this part, then the gradients of the

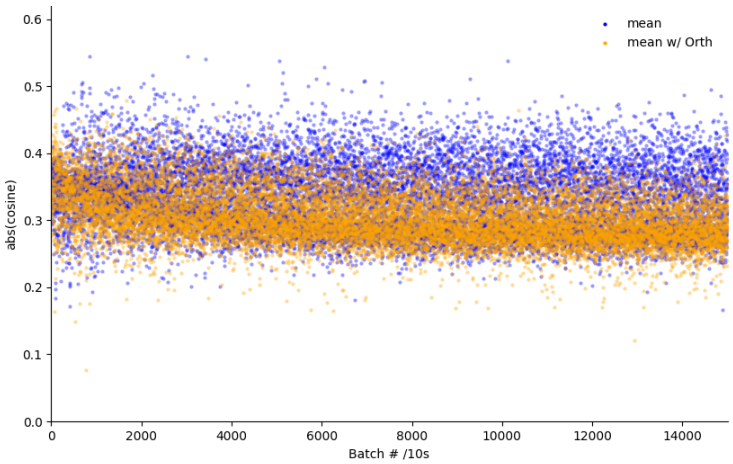
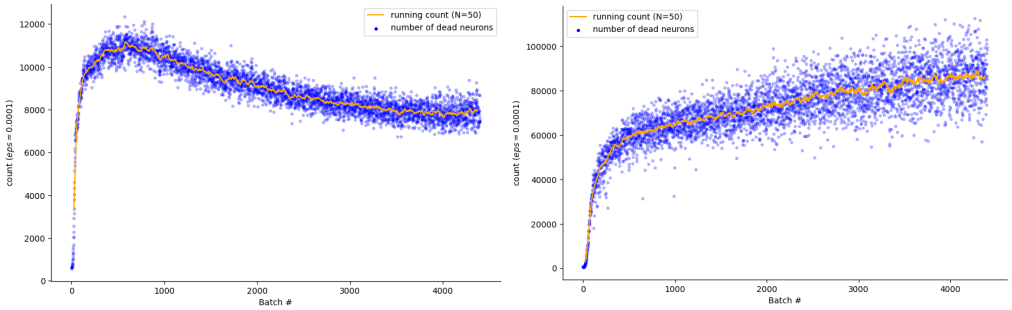


Figure 7: Mean of the absolute cosine of all distinct pairs of different intermediary representations, $\mathbb{E}[R_l]$, $l \in \{1, 2, 3\}$, for all layers of a BasicCNN trained on CIFAR-10 as in section 2.3.1.

preceding parameters will be zero and prevented from learning. This limits the model's capacity based on a parameterisation, however temporarily dead parameters can be beneficial and act as a regulariser, similar to dropout. To detect temporarily dead parameters, we simply look for parameters with zero gradient. Comparing the amount of dead parameters produced by SGDM versus Orthogonal-SGDM, figs. 8a and 8b respectively, shows that Orthogonal-SGDM ends with around and order of magnitude more temporarily dead parameters. This implies a much higher regularisation which helps to explain Orthogonal-SGDM's insensitivity to over-parametrisation.



(a) With SGDM.

(b) With Orthogonal-SGDM.

Figure 8: Number of temporarily dead parameters in layer2[1].conv2 of a resnet50 trained as in section 2.3.1.

3.4 Implementation Details

While the QR decomposition is the most common orthogonalisation method, it is, in practice, less stable as the gradients are rank deficient (Demmel, 1997, Section 3.5), *i.e.* they have at least one small singular value. Orthogonal-SGDM has a longer wall time than SGDM because of the added expense of the SVD which has non-linear time complexity in the matrix size. In practice, we have found that the calculation of the SVD is either a trivial or prohibitively expensive cost, with dense layers being the largest and so most problematic. While there exist methods for computing an approximate SVD which are faster, we have used PyTorch's default implementation since we are more concerned with Orthogonal-SGDM's performance and efficiency in iterates and not in wall time. It is doubtful that convergence of SVD is needed, so a custom matrix orthogonalisation algorithm, that has the required stability but remains fast and approximate, will reduce the computation overhead significantly and may allow previously infeasible networks to be optimised using Orthogonal-

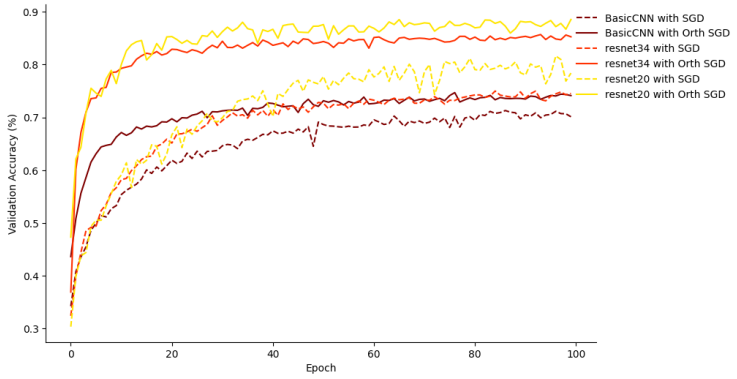


Figure 9: Orthogonalising just the convolutional filters vs both the convolutional layers and final dense layer on CIFAR-10; trained as in section 2.3.1.

SGDM. However, we note that even with a more suitable implementation, this method would still bias towards many smaller layers for a deeper, thinner network.

3.5 Fully connected layers

Fully connected or dense layers also fit our component model from eq. (2) where the components are based on the inner product of the input and the parametrisation,

$$c_{li}(x) = \sigma(\langle \text{flatten}(x), \theta_{li} \rangle),$$

where σ is an activation function, $S_l = 1$ giving $f_l : \mathbb{R}^{S_{l-1} \times N_{l-1}} \to \mathbb{R}^{N_l}$ and $\theta_l \in \mathbb{R}^{S_{l-1} \cdot N_{l-1} \times N_l}$ as desired. Intuitively, each column of the weight matrix acts as a linear map resulting in one item in the output vector. Thus, the gradients of fully connected layers can also be orthogonalised.

Table 3: Test accuracy and loss for Orthogonal-SGDM on CIFAR-10 when orthogonalising all layers vs orthogonalising just the convolutional layers. Trained as in section 2.3.1, standard error across five runs.

	SGDM		Orthogonal-SGDM		Conv Orthogonal-SGDM	
	Loss	Acc (%)	Loss	Acc (%)	Loss	Acc (%)
BasicCNN	0.7603 $\pm$ 0.0061	73.60 $\pm$ 0.19	0.6808 $\pm$ 0.0038	76.67 $\pm$ 0.10	0.6732 $\pm$ 0.0041	**76.80** $\pm$ 0.18
resnet34	1.0468 $\pm$ 0.0134	75.86 $\pm$ 0.26	0.7087 $\pm$ 0.0165	85.42 $\pm$ 0.33	0.6268 $\pm$ 0.0105	**85.68** $\pm$ 0.21
resnet20	0.6728 $\pm$ 0.0301	79.14 $\pm$ 0.62	0.6766 $\pm$ 0.0155	87.12 $\pm$ 0.12	0.4824 $\pm$ 0.0225	**87.70** $\pm$ 0.40

As noted in section 3.4 the extra wall time is dominated by the largest parameter, this is often the dense layer; table 3 shows that for CIFAR-10 there is no impact on the error rate from not orthogonalising the final dense layer, and the training curves are the same shape — fig. 9. While both the error rates and losses decrease when not orthogonalising the dense layer we hesitate to say that orthogonalising dense layers is detrimental since these networks only have a dense final classification layer which is qualitatively different from intermediary dense layers.

3.6 Limitations due to mini-batch size

Orthogonal-SGDM does not perform as well as SGDM when the mini-batch size is extremely small, fig. 10, due to the increased levels of noise for the SVD. This is the most likely reason that the resnet20 from section 2.3.2 fails to match the original performance.

A mini-batch size of 16 is where Orthogonal-SGDM starts to outperform SGDM on a resnet18 for CIFAR-10. Few models need such small mini-batch sizes, but if they do then SGDM would be a more suitable optimisation algorithm. In addition to the learning collapse, the time taken by SVD is only dependent on the parameter size and not the mini-batch size, so increasing the number of mini-batches per epoch also increases the wall time to train. The reason for the collapse in training with small mini-batch sizes will be subject to further research.

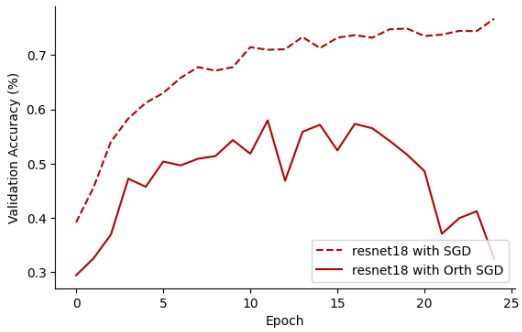


Figure 10: CIFAR-10 with mini-batch size=4 trained as in section 2.3.1.

4 Conclusion

In this work we have laid out a new optimisation method, tested it on different models and data sets, showing close to state-of-the-art results out of the box and robustness to hyper-parameter choice and over-parametrised models. Orthogonal-SGDM also has practical application in problems such as object detection and semantic segmentation since they make use of a pre-trained image classification backbone.

SGDM with a vast amount of hyper-parameter tuning still reigns supreme, but Orthogonal-SGDM is an excellent method for quick verification of models or for prototyping — when we want decent results fast, but do not need the absolute best performing model. However, as more data set sizes are growing more models are being trained on fewer to less than one epoch of data (Brown et al., 2020) leading to an extremely limited ability to tune the hyper-parameters.

Lastly, we mentioned briefly in section 1 how attention heads fit our model but, since they are beyond the scope of this work, we will explore the potential gain in using Orthogonal-SGDM with them in future work, and expect a similarly exceptional gain will be obtained.

References

Michael Biehl, Peter Riegler, and Christian Wöhler. Transient dynamics of on-line learning in two-layered neural networks. *Journal of Physics A: Mathematical and General*, 29(16):4769, 1996.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

James W Demmel. *Applied numerical linear algebra*. SIAM, 1997.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks, 2017.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction, 2021.

A COSINE THRESHOLD

We use the cosine metric

$$\langle x, y \rangle_2 = \frac{x \cdot y}{\|x\|_2 \|y\|_2} \tag{7}$$

in this work since it allows the comparison of the directions of high-dimensional vectors, and so obtain insight about the surface we are optimising on. However, we note that the "significance" of the cosine between two random vectors depends on their size.

Assuming both vectors' components are random variables

$$x = [x_1, x_2, \ldots, x_N] \,,$$

where $x_i \sim \mathcal{N}(0,\, \sigma^2)$, then the components of their dot product

$$\langle x, y \rangle = [x_1 \cdot y_1, x_2 \cdot y_2, \ldots x_N \cdot y_N] \,,$$

have variance σ^4. Now, from the central limit theorem, the dot product has variance $\frac{\sigma^4}{N}$ where N is the size of the vectors. Finally, dividing by the magnitude of the vectors gives a variance of $\bar{\sigma} = \frac{\sigma^2}{N}$ for the cosine metric.

To gain some understanding of the significance of a cosine distance, we define a four-sigma threshold on the distribution of cosines, so, assuming $\sigma = 1, \mu = 0$, we get a threshold value of

$$\begin{aligned} \mu \pm 4\bar{\sigma} &= 0 \pm \frac{4\sigma}{\sqrt{N}} \\ &= \pm \frac{4}{\sqrt{N}} \end{aligned} \tag{8}$$

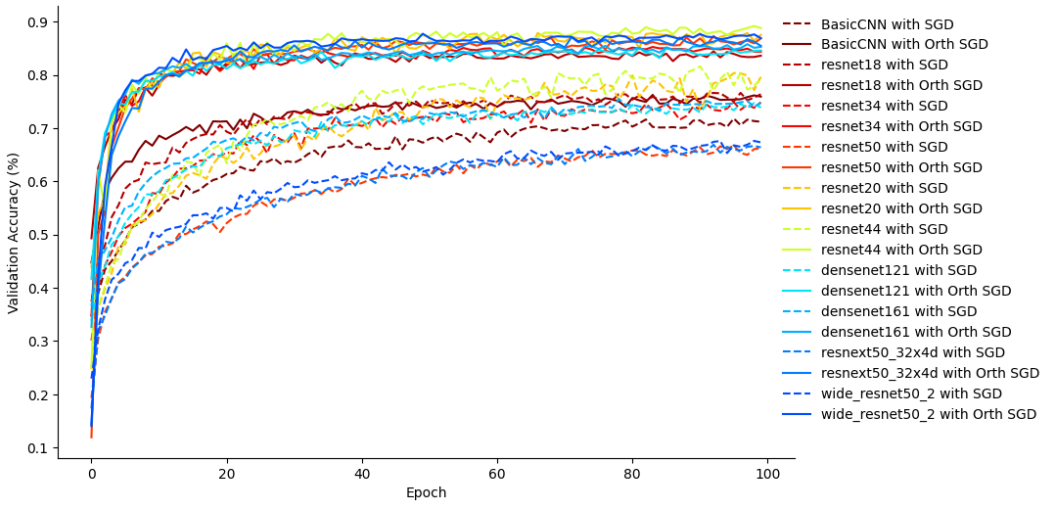
This is important because a distance of 0.1 might seem small, but for $10,000$-dimensional vectors, it easily clears our significance threshold.

B MODEL SUMMARIES

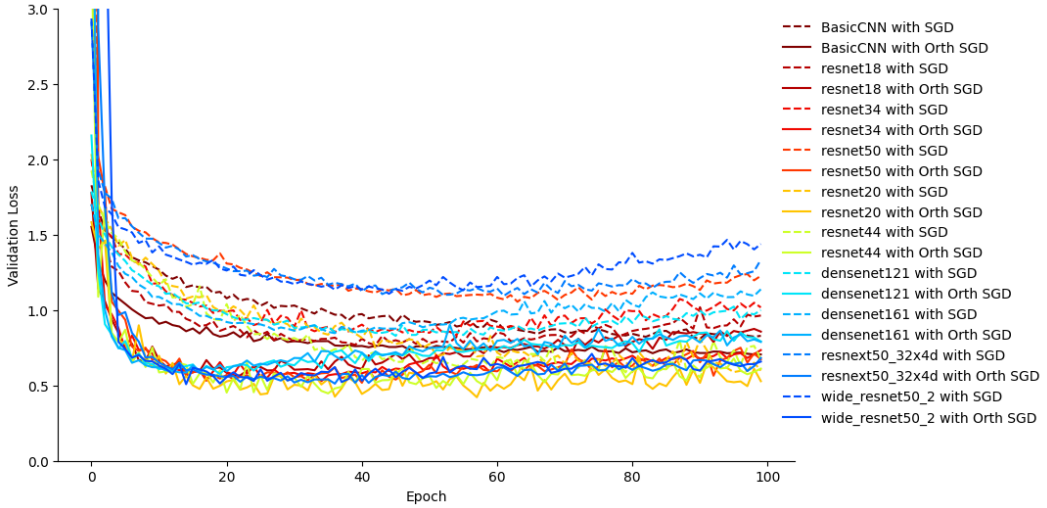
B.1 BASICCNN

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 32, 16, 16]             896
       BatchNorm2d-2            [-1, 32, 16, 16]              64
            Conv2d-3             [-1, 32, 8, 8]            9,248
       BatchNorm2d-4             [-1, 32, 8, 8]              64
            Conv2d-5             [-1, 32, 4, 4]            9,248
       BatchNorm2d-6             [-1, 32, 4, 4]              64
          Linear-7                    [-1, 10]            5,130
        BasicCNN-8                    [-1, 10]                0
================================================================
Total params: 24,714
Trainable params: 24,714
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.16
Params size (MB): 0.09
Estimated Total Size (MB): 0.27
----------------------------------------------------------------
```

C FULL RESULTS PLOT



(a) Validation Accuracy



(b) Validation Loss

Figure 11: SGDM vs Orthogonal SGDM