Towards Scalable Backpropagation-Free Gradient Estimation

Daniel Wang[0009−0005−1349−1552], Evan Markou[0000−0001−8476−7310], and Dylan Campbell[0000−0002−4717−6850]

Australian National University, Canberra, Australia
`{u7918232, Evan.Markou, Dylan.Campbell}@anu.edu.au`

Abstract. While backpropagation—reverse-mode automatic differentiation—has been extraordinarily successful in deep learning, it requires two passes (forward and backward) through the neural network and the storage of intermediate activations. Existing gradient estimation methods that instead use forward-mode automatic differentiation struggle to scale beyond small networks due to the high variance of the estimates. Efforts to mitigate this have so far introduced significant bias to the estimates, reducing their utility. We introduce a gradient estimation approach that reduces both bias and variance by manipulating upstream Jacobian matrices when computing guess directions. It shows promising results and has the potential to scale to larger networks, indeed performing better as the network width is increased. Our understanding of this method is facilitated by analyses of bias and variance, and their connection to the low-dimensional structure of neural network gradients.

Keywords: Optimisation · Low-rank gradient subspace · Forward-mode differentiation · Automatic differentiation · Deep learning

1 Introduction

While backpropagation [1] is ubiquitous in deep learning, there has been recent interest in alternative memory-efficient methods of optimising a neural network without using backpropagation [2,3,4,5]. One such class of methods is based on estimating the gradient using forward-mode automatic differentiation (AD). Forward-mode AD efficiently evaluates the Jacobian–vector product in a single forward pass and avoids the memory cost of storing intermediate activations required by backpropagation. Additional motivation for backpropagation alternatives arises from the fact that backpropagation is biologically implausible [6].

Using forward-mode AD, an unbiased estimator of the gradient can be constructed by randomly sampling a guess direction and scaling it by the directional derivative in the guess direction. This method is defined as the forward gradient [7]. However, because this estimator requires sampling in a high-dimensional guessing space equal to the number of parameters, high variance prohibits this method from scaling beyond training small networks on toy datasets. For a network with N parameters, the cosine similarity between the estimated and true gradient is $O(\frac{1}{\sqrt{N}})$, which means that the guess direction becomes orthogonal to the true gradient as network size increases. A number of works based on the forward gradient, including state-of-the-art gradient guessing method "$\tilde{W}^\intercal$" [2], have focused on decreasing its variance by reducing the guessing space [7,8,2]. Neural network gradients have also been shown to exhibit a low-dimensional structure [9,10], which if properly exploited has the potential to significantly improve gradient guessing methods.

In this paper, we introduce a novel method $\tilde{W}^\perp$ that extends the $\tilde{W}^\intercal$ approach, reducing both its bias and its variance by considering the low-rank structure of the guessing space. Our experiments show that the $\tilde{W}^\perp$ method performs considerably better than $\tilde{W}^\intercal$ for larger networks. We also observe a possible connection to the low-dimensional structure of the gradient in that it lies in a low-dimensional subspace of the image of the upstream Jacobian matrix, providing additional insight into our gradient guessing methods.

2 Related Work and Technical Background

Forward-mode Automatic Differentiation (AD). Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, forward-mode AD computes the Jacobian–vector product Jv where $J \in \mathbb{R}^{m \times n}$ is the Jacobian matrix and $v \in \mathbb{R}^n$ is an arbitrary vector. In the context of optimising a neural network where the loss function $L : \mathbb{R}^n \to \mathbb{R}$ typically has a scalar output, the Jacobian–vector product is simply the directional derivative in the direction v. In practice, automatic differentiation frameworks such as PyTorch and JAX provide functions to compute this efficiently.

Forward Gradient (Weight Perturbation [3]). For the remainder of this paper we will consider an l-layer MLP with weights W and biases b with layer widths of $d_1, d_2, \ldots, d_l$, ReLU activations, and loss function L. The ith linear layer can be written as

$$s_i = W_i x_i + b_i, \quad s_i \in \mathbb{R}^{d_{i+1}} , \tag{1}$$

where $W_i \in \mathbb{R}^{d_{i+1} \times d_i}$, $b_i \in \mathbb{R}^{d_{i+1}}$, and

$$x_{i+1} = \mathrm{ReLU}(s_i), \quad x_{i+1} \in \mathbb{R}^{d_{i+1}} . \tag{2}$$

We will only consider optimising the weights W_i, but these methods can be easily extended to the biases too. At layer i, the vanilla forward gradient method generates a guess of $\partial L/\partial W_i$ as follows. First, a guess direction y_i of the same dimension as W_i is sampled from the standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Then, the Jacobian–vector product $y^\intercal \frac{\partial L}{\partial W}$—the directional derivative of $\partial L/\partial W$ in the direction y, where W denotes the concatenated W_is and y denotes the concatenated y_is—is computed in the forward pass. The guess for $\partial L/\partial W_i$ is defined as $\frac{\hat{\partial L}}{\partial W_i} = (y_i^\intercal \frac{\partial L}{\partial W_i}) y_i$: the guess direction scaled by the directional derivative in that direction. With this approach, weight updates are performed by replacing the true gradient with the guess $\hat{\partial L}/\partial W_i$. In general, sampling the guess direction from a zero-mean unit-variance distribution with independent components yields an unbiased but high variance estimator of the true gradient.

Activation perturbation [7]. This method guesses the gradient with respect to the activations instead of the weights. In deep neural networks, the weight matrix $W_i \in \mathbb{R}^{d_{i+1} \times d_i}$ contains many more entries than the activation vector $s_i \in \mathbb{R}^{d_{i+1}}$. Therefore, guessing the gradient with respect to the activations $\partial L/\partial s_i$ reduces the dimension of guessing space and hence decreases variance. Similar to before, the guess $\frac{\hat{\partial L}}{\partial s_i} = (y^\intercal \frac{\partial L}{\partial s})y_i$, where y denotes the concatenated guess directions y_i and s denotes the concatenated activations s_i at each layer, is computed by sampling a random vector $y_i \in \mathbb{R}^{d_{i+1}}$ and scaling it by the Jacobian–vector product $y^\intercal \frac{\partial L}{\partial s}$. Given the guess $\frac{\hat{\partial L}}{\partial s_i}$, the gradient guess with respect to the weight matrix can be obtained via the chain rule,

$$\frac{\hat{\partial L}}{\partial W_i} = \frac{\hat{\partial L}}{\partial s_i}\frac{\partial s_i}{\partial W_i} = \frac{\hat{\partial L}}{\partial s_i} x_i^\intercal \,. \tag{3}$$

As well as reducing variance, Ren et al. [7] show that activation perturbation is also unbiased. Consequently, it performs better than weight perturbation.

$\tilde{W}^\intercal$ method [2]. The $\tilde{W}^\intercal$ method extends the activation perturbation approach by incorporating terms along the chain rule gradient decomposition up to the next layer's activations into the guess for $\partial L/\partial s_i$. For a network with linear layers and ReLU activations, the chain rule gives

$$\frac{\hat{\partial L}}{\partial W_i} = \frac{\hat{\partial L}}{\partial s_{i+1}}\frac{\partial s_{i+1}}{\partial x_{i+1}}\frac{\partial x_{i+1}}{\partial s_i}\frac{\partial s_i}{\partial W_i} = \epsilon_{i+1}^\intercal W_{i+1} M_i x_i^\intercal \,, \tag{4}$$

where ϵ_{i+1} is a sample from a standard normal distribution that approximates the unknown $\partial L/\partial s_{i+1}$, W_{i+1} is the weight matrix of the next layer, M_i is the diagonal matrix corresponding to the ReLU derivative mask, and x_i is the input to layer i. Using the first three terms in Eq. (4), a guess direction $y_i^\intercal = \epsilon_{i+1}^\intercal W_{i+1} M_i$ for $\partial L/\partial s_i$ is generated by sampling ϵ_{i+1} and multiplying with the next layer weight matrix $W_{i+1}^\intercal$ and the ReLU mask M_i. As before, the Jacobian–vector product is $\frac{\partial L}{\partial s}^\top y$, and for each layer the guess for $\frac{\partial L}{\partial s_i}$ is $\frac{\hat{\partial L}}{\partial s_i} = (y^\intercal \frac{\partial L}{\partial s})y_i$. The weight update for W_i is obtained from the last term of Eq. (4) by taking the outer product $\frac{\hat{\partial L}}{\partial s_i} x_i^\intercal$.

The intuition behind this approach is that it leverages information from the network architecture and narrows the guessing space with each additional term in the chain rule. The next layer weight matrix is typically low-rank and the ReLU mask is sparse, leading to a large variance reduction. Although we focus on an MLP with ReLU activations, in a more general setting the guess direction can be simply written as $y_i = \tilde{W}_{i+1}^\intercal \epsilon_{i+1}$, where $\tilde{W}_{i+1}$ represents the composition of each term in the chain rule from the current layer's activations s_i up to the next layer's activations s_{i+1}. For a ReLU MLP, $\tilde{W}_{i+1} = W_{i+1} M_i$.

However, because $y_i = \tilde{W}_{i+1}^\intercal \epsilon_{i+1}$ is no longer a zero-mean unit-variance random variable with independent components, this estimator is no longer unbiased. Singhal et al. [2] show that the bias at layer i is given by

$$\mathbb{E}\left[\frac{\hat{\partial L}}{\partial s_i}\right] - \frac{\partial L}{\partial s_i} = (\mathrm{Cov}(y_i) - I)\frac{\partial L}{\partial s_i} \,, \tag{5}$$

where $\partial L/\partial s_i$ is the true gradient and y_i is the guess direction with

$$\mathrm{Cov}(y_i) = \tilde{W}_{i+1}^{\top}\tilde{W}_{i+1}\,. \tag{6}$$

Despite this incurred bias, the $\tilde{W}^{\top}$ method significantly outperforms activation perturbation due to a large decrease in variance.

Gradients lies in a low-dimensional subspace. The gradient guessing methods described above are based on the principle of reducing variance by reducing the dimensionality of the guessing space. Related to the idea of gradients within a low-dimensional subspace is the work of Gur-Ari et al. [9], which observes that for a deep neural network trained on a k-class classification task, the gradient predominantly lies in the low-dimensional subspace spanned by the eigenvectors corresponding to the top-k largest eigenvalues of the Hessian. Interestingly, Song et al. [10] found that performing SGD in the small top-k subspace fails to decrease the loss, but performing SGD in the orthogonal complement of this subspace recovers the original SGD performance, leading to the hypothesis that this phenomenon is caused by noise in stochastic gradient descent as opposed to full-batch gradient descent. In the context of forward gradient guessing methods, this phenomenon reveals the underlying structure of gradients which can be exploited by, as we show in the next section, guessing within a small subspace that aligns with the true gradient.

3 A Low-Bias, Low-Variance Guess Direction

We propose the $\tilde{W}^{\perp}$ method, based on the $\tilde{W}^{\top}$ method, that reduces both the bias and variance of the guess direction by orthogonalising the upstream Jacobian matrix. A variant, $\tilde{W}^{\perp}$-NS, accelerates the orthogonalisation using Newton–Schulz iterations. Finally, to better understand the effect of bias and variance on the performance of gradient guessing methods, we also introduce a preconditioning method for comparison that is unbiased but has higher variance.

3.1 $\tilde{W}^{\perp}$: Orthogonalising the Guess Projection Matrix

In this section, we consider layer i in isolation and drop the layer index i, so the general upstream Jacobian matrix up to the next layer is denoted by $\tilde{W}^{\top}$ and the guess direction is $y = \tilde{W}^{\top}\epsilon$. That is, $\tilde{W}^{\top}$ projects the Gaussian sample ϵ to the guess direction y. Recall that for a ReLU MLP, $\tilde{W}_{i+1} = W_{i+1}M_i$, where W_{i+1} is the weight matrix of the next layer and M_i is the ReLU derivative mask, a diagonal binary matrix. The central idea is to manipulate $\tilde{W}$ into an orthonormal, low-rank matrix $\tilde{W}'$ that further narrows the guessing space while bringing the covariance matrix $\mathrm{Cov}(y) = \tilde{W}'^{\top}\tilde{W}'$ closer to the identity, thus reducing both bias and variance. We expect $\mathrm{rank}(\tilde{W})$ to be relatively low, since for a ReLU network it is at most the number of non-zero activations. Let the reduced SVD of $\tilde{W}$ be

$$\tilde{W} = U_r \Lambda_r V_r^{\top}\,, \tag{7}$$

where $r = \text{rank}(\tilde{W})$. Then, we manipulate $\tilde{W}$ into the orthonormal matrix

$$\tilde{W}' = U_r V_r^\top \,, \tag{8}$$

so the guess direction is $y = \tilde{W}'^\top \epsilon$ with covariance matrix $\text{Cov}(y) = \tilde{W}'^\top \tilde{W}' = V_r U_r^\top U_r V_r^\top = I_r$, being the identity matrix in the first r diagonal elements and 0 everywhere else. To control the bias–variance trade-off, integer values of k from 1 to r can be chosen as a hyperparameter to select only the rows and columns in U_r and $V_r^\top$ corresponding to the k largest singular values so that $\tilde{W}'_k = U_k V_k^\top$. Smaller values of k reduce the dimensionality of the guessing space at the cost of pushing the covariance matrix I_k further from the identity and thus incurring higher bias.

3.2 $\tilde{W}^\perp$-NS: Fast Orthogonalisation via Newton–Schulz Iterations

The Newton–Schulz iteration method [11,12,13] can be used instead of SVD to significantly speed up our $\tilde{W}^\perp$ method. The idea is that, given an odd polynomial, applying it to a matrix $A = USV^\top$ is the same as applying it element-wise to the singular values S. That is,

$$p(A) = \sum_{k=0}^{K} a_k A (A^\top A)^k = U \left(\sum_{k=0}^{K} a_k S^{2k+1} \right) V^\top = U p(S) V^\top \,. \tag{9}$$

Applying this N times yields

$$p^N(A) = U p^N(S) V^\top \,, \tag{10}$$

where on the left hand side $p^N(A)$ is the matrix polynomial applied N times to A and on the right hand side it is the real-valued polynomial applied N times element-wise to the singular values. With carefully chosen coefficients, repeated composition of the polynomial $p^N(A)$ can be made to converge to various functions as $N \to \infty$. Jordan et al. [13] use this method to orthogonalise $p^N(A) \approx UV^\top$ by mapping all singular values to 1. They find a degree five polynomial with $N = 5$ to approximate the function $f(x) = 1$ on $[0, 1]$.

The Newton–Schulz iteration can be easily adapted to our $\tilde{W}^\perp$ method, labelled $\tilde{W}^\perp$-NS. We wish to obtain $p^N(\tilde{W}) \approx U_k V_k^\top$ by mapping the singular values larger than σ_k to 1 and the ones smaller than σ_k to 0, where $\sigma_k \in [0, 1]$ is the k^{th} largest singular value. That is, we need to approximate the step function defined by $f_{\sigma_k}(x) = 0$ for $x \in [0, \sigma_k]$ and $f(x) = 1$ for $x \in (\sigma_k, 1]$, where σ_k is constrained to lie in $[0, 1]$ by normalising $\tilde{W}$ to have unit spectral norm. Since the step function is slightly more complex than $f(x) = 1$ on [0, 1], we fit a degree seven polynomial with coefficients obtained via gradient descent.

3.3 $\tilde{W}^{\text{P}}$: Preconditioning the Guess Projection Matrix

Both the $\tilde{W}^\top$ and $\tilde{W}^\perp$ methods reduce variance and increase accuracy over the unbiased activation perturbation approach, but at the cost of introducing bias.

In this section, we develop an unbiased estimator, based on the $\tilde{W}^\top$ method, at the cost of increasing the variance. This allows us to investigate the bias–variance trade-off. Specifically, we pre-multiply $\tilde{W}^\top$ by $(\tilde{W}^\top\tilde{W})^{-1/2}$ so that $\tilde{W}'^\top \approx (\tilde{W}^\top\tilde{W})^{-1/2}\tilde{W}^\top$. Then $\tilde{W}'^\top\tilde{W}' = I$, and $y = \tilde{W}'^\top\epsilon$ has identity covariance. To be precise, let the SVD of $\tilde{W}$ be $\tilde{W} = USV^\top$. Since $\tilde{W}$ is not guaranteed to be full-rank, we introduce a small constant $\sigma = 10^{-5}$ and modify $\tilde{W}$ to be $\tilde{W}_\sigma = U(S^2+\sigma I)^{1/2}V^\top$, so that

$$\tilde{W}_\sigma^\top\tilde{W}_\sigma = V(S^2+\sigma I)V^\top\,, \tag{11}$$

can be diagonalised while keeping $\tilde{W}_\sigma$ close to $\tilde{W}$. Setting

$$\tilde{W}_\sigma'^\top = (\tilde{W}_\sigma^\top\tilde{W}_\sigma)^{-1/2}\tilde{W}_\sigma^\top\,, \tag{12}$$

yields an unbiased estimator, because the covariance matrix is identity:

$$\begin{aligned}\mathrm{Cov}(y) &= \tilde{W}_\sigma'^\top\tilde{W}_\sigma' &(13)\\ &= (\tilde{W}_\sigma^\top\tilde{W}_\sigma)^{-1/2}\tilde{W}_\sigma^\top\tilde{W}_\sigma(\tilde{W}_\sigma^\top\tilde{W}_\sigma)^{-1/2} &(14)\\ &= V(S^2+\sigma I)^{-1/2}V^\top V(S^2+\sigma I)V^\top V(S^2+\sigma I)^{-1/2}V^\top &(15)\\ &= I\,. &(16)\end{aligned}$$

This method has high variance because $(\tilde{W}_\sigma^\top\tilde{W}_\sigma)^{-1/2}$ contains large singular values, due to σ being small. Moreover, unlike the $W^\top$ and $W^\perp$ methods, the dimension of the guessing space is not reduced, because $\tilde{W}_\sigma'$ is a full rank matrix.

4 Experiments

4.1 Experiment Setup

Implementation Details and Dataset. Following Singhal et al. [2], the experiments in this section involve optimising an MLP with three hidden layers of size 128 and ReLU activations. Our methods and existing methods from previous works are implemented in PyTorch, using `functorch.jvp` for forward mode automatic differentiation [1]. The experiments were run on an Apple M1 Pro Macbook with an 8-core CPU and 14-core GPU with 16GB of unified memory on the MNIST-1D dataset [14] for 300 epochs. Note that 300 epochs is not enough for convergence. The choice of optimiser, AdamW [15] with learning rate 10^{-4} and batch size 512, is taken from the best performing configuration in Singhal et al. [2] with no additional hyperparameter tuning.

Metrics: Bias. For a gradient guess direction $y_i^b = \tilde{W}_{i+1}^{b\top}\epsilon_i^b$ at layer i where b indexes over a batch of size B, the bias magnitude is given by

$$\mathrm{Bias}\left(\frac{\partial\hat{L}}{\partial s_i}\right) = \left\|\frac{1}{B}\sum_{b=1}^{B}(\mathrm{Cov}(y_i^b) - I)\frac{\partial L}{\partial s_i^b}\right\|_2\,. \tag{17}$$

[1] The code is available at https://github.com/danielwang0452/Forward-W-Perp.

Table 1: Training accuracies after 300 epochs for different values of k in the $\tilde{W}^{\perp}$ method. The $\tilde{W}^{\perp}$ accuracies are comparable to $\tilde{W}^{\mathsf{T}}$, with smaller values $k = 10$ and $k = 25$ reaching the highest accuracy. In general, the trend is that smaller k perform better due to larger variance reduction, despite increased bias.

Method	k	**Accuracy**
$\tilde{W}^{\mathsf{T}}$		0.312
$\tilde{W}^{\perp}$	rank($\tilde{W}$)	0.311
$\tilde{W}^{\perp}$	25	0.323
$\tilde{W}^{\perp}$	10	**0.324**
$\tilde{W}^{\perp}$	1	0.275

Metrics: Variance. We compute the variance magnitude at layer i via the decomposition of MSE into bias and variance using the empirical mean squared error (MSE) and squared bias, given by

$$\widehat{\mathrm{Var}}\left(\frac{\partial \hat{L}}{\partial s_i}\right) = \Bigg\| \underbrace{\frac{1}{B}\sum_{b=1}^{B}\left(\frac{\partial L}{\partial s_i^b} - \frac{\partial \hat{L}}{\partial s_i^b}\right)^{\circ 2}}_{\text{MSE}} - \underbrace{\left(\frac{1}{B}\sum_{b=1}^{B}\left(\mathrm{Cov}(y_i^b) - I\right)\frac{\partial L}{\partial s_i^b}\right)^{\circ 2}}_{\text{Squared Bias}} \Bigg\|_2, \tag{18}$$

where $\circ 2$ is the element-wise square.

Metrics: Covariance Frobenius Norm. Another way to measure bias is to consider $\mathrm{Cov}(y_i) - I$ independent of the true gradient $\frac{\partial L}{\partial s}$. We quantify this with the Frobenius norm $||(\mathrm{Cov}(y) - I)||_F$, averaged over the batch dimension. Intuitively, a smaller Covariance Frobenius Norm means that the covariance matrix is closer to identity, resulting in lower bias.

4.2 Results

$W^{\perp}$: Optimal Choice of k. We evaluate $W^{\perp}$ with different values of k, which controls the bias–variance trade-off, against the baseline W^{T} method to examine its effect on training accuracy, shown in Tab. 1. Smaller values of k reduce variance by reducing the dimensionality of the guessing space. However, reducing k also increases the bias since the covariance matrix I_k is further from the identity. For $k = \mathrm{rank}(W)$ the final training accuracy is almost identical to W^{T}, and as smaller values of k are chosen the accuracy increases. This result shows that a lower-dimensional guessing space with reduced variance counteracts the additional bias, except for the extreme low variance, high bias $k = 1$ case where there is a sharp decrease in accuracy.

Why Small Values of k Perform Best. The phenomenon that gradients lie in a low-dimensional subspace offers an explanation as to why small values of k attain the highest accuracy. Given the idea behind the $\tilde{W}^{\mathsf{T}}$ method, that

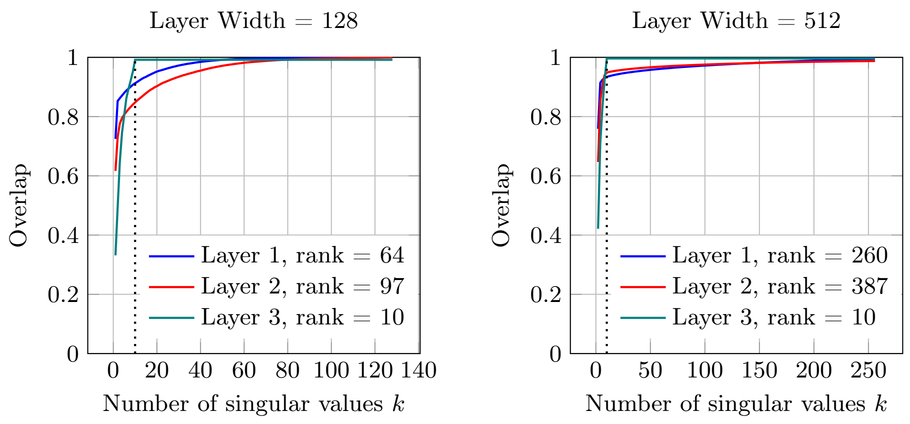

Fig. 1: Overlap of the true gradient $\frac{\partial L}{\partial s}$ onto the k-dimensional subspace of $\mathrm{Im}(\tilde{W}^\top)$ corresponding to the k largest singular values of W, for layer widths of 128 and 512. In both cases, $\frac{\partial L}{\partial s}$ predominantly lies in a subspace of much lower dimension than $\mathrm{rank}(W)$; around $k = 10$ captures most of the gradient, reducing the guessing space while introducing minimal bias. Note that the Layer 3 weight matrix is low rank since it is connected to the output layer of width 10.

multiplying by the upstream Jacobian matrix $\tilde{W}^\top$ constrains the gradient guess to lie in its image, denoted by $\mathrm{Im}(\tilde{W}^\top)$, it is reasonable to speculate that an analogous phenomenon occurs here where the true gradient lies not only in $\mathrm{Im}(\tilde{W}^\top)$ but in a low-dimensional subspace of $\mathrm{Im}(\tilde{W}^\top)$. This can be empirically verified by projecting the true gradient $\partial L/\partial s$ onto $\mathrm{Im}(\tilde{W})$, similar to Gurari et al. [9]. In the reduced SVD of $\tilde{W} = U_k \Lambda_k V_k^\top$, U_k contains an orthonormal basis of $\mathrm{Im}(\tilde{W}^\top)$, and the overlap o_{top} of the true gradient in $\mathrm{Im}(\tilde{W}^\top)$ can be computed by projecting $\frac{\partial L}{\partial s}$ onto $\mathrm{Im}(\tilde{W}^\top)$, given by

$$o_{\mathrm{top}} = \frac{\left\| U_k U_k^\top \frac{\partial L}{\partial s} \right\|}{\left\| \frac{\partial L}{\partial s} \right\|} . \tag{19}$$

Fig. 1 shows that the true gradient does in fact mostly lie in in the small k-dimensional subspace spanned by V_k. Intuitively, choosing the smallest value of k that is just large enough for the k-dimensional subspace to contain most of the true gradient will incur minimal additional bias while greatly reducing variance by narrowing the guessing space. For the network architecture and dataset in our case, $k = 10$ is a good choice.

$\tilde{W}^\perp$: Understanding the Bias Reduction. Next, we observe that for appropriately chosen k, our $\tilde{W}^\perp$ method outperforms $\tilde{W}^\top$ in both variance *and* bias. This is shown in Tab. 2. The variance reduction is achieved by constraining the guesses to a low-dimensional subspace, while the bias reduction is achieved by aligning the guesses with the true gradient, which is less obvious. Recall that the

Table 2: Variance, bias, and covariance Frobenius norm for different methods.

Method	**Variance** (18)	**Bias** (17)	**Cov. Norm**
$\tilde{W}^\top$	1.50×10^{-4}	2.75×10^{-4}	**0.084**
$\tilde{W}^\perp$-bottom	1.30×10^{-5}	2.83×10^{-4}	0.085
$\tilde{W}^\perp$ $(k = 10)$	$\mathbf{1.03 \times 10^{-5}}$	$\mathbf{2.86 \times 10^{-5}}$	0.085

bias is zero when the covariance matrix is identity. For $\tilde{W}^\perp$, the purpose of orthogonalisation is to zero out off-diagonal entries in the covariance matrix I_k to bring it closer to identity. However, for small values of k, I_k ends up being equally distant from identity (in the Frobenius norm) as $\tilde{W}^\intercal \tilde{W}$, the covariance matrix of the $\tilde{W}^\intercal$ method. This might suggest that we have not actually decreased the bias in $\tilde{W}^\perp$. However, Tab. 2 shows that this is not the case.

To illustrate that the bias reduction comes from aligning the guessing space with the true gradient, consider the following variant of $\tilde{W}^\perp$ which we refer to as $\tilde{W}^\perp$-bottom. In the SVD of $\tilde{W} = USV^\intercal$, we take $\tilde{W}' = U_k V_k^\intercal$ corresponding to the $k = 10$ *smallest* (in this case 0) singular values, where we know the true gradient does not lie. Both $\tilde{W}^\perp$ and $\tilde{W}^\perp$-bottom have the same covariance matrix I_k, so according to Eq. (5), the difference in bias comes from the difference in the true gradients $\frac{\partial L}{\partial s}$. In other words, the optimisation trajectory of $\tilde{W}^\perp$ has smaller gradient norm $\left\|\frac{\partial L}{\partial s}\right\|_2$ than that of $\tilde{W}^\perp$-bottom, and hence lower bias. The conclusion is that $\tilde{W}^\perp$ reduces bias by aligning the guessing space with the underlying low-rank structure of the true gradient.

$\tilde{W}^\perp$: Scaling to Larger Networks. Compared to previous methods, the $\tilde{W}^\perp$ method shows promising results for larger networks. Fig. 2 and Tab. 3 show that the gap in training accuracy between $\tilde{W}^\perp$ with $k = 10$ and $\tilde{W}^\intercal$ increases as the hidden layer width increases from 64 to 128 and 512. This behaviour can be explained by the variance reduction from narrowing the guessing space. Recall the idea behind $\tilde{W}^\intercal$, that multiplying a randomly sampled ϵ with $\tilde{W}^\intercal$ restricts the guess direction $y = \tilde{W}^\intercal \epsilon$ to lie in a lower dimensional $\mathrm{Im}(\tilde{W}^\intercal)$. However, as layer width increases, so does the rank of the next layer weight matrix and hence the dimensionality of $\mathrm{Im}(\tilde{W}^\intercal)$. $\tilde{W}^\perp$ does not suffer from this problem, because the dimensionality of the guessing space can be controlled through the hyperparameter k. In our experiments, the value of $k = 10$ which was observed to work well in the width 128 experiments still performs well for width 512 because the 10-dimensional subspace still contains a large portion of the true gradient. In general, these results indicate that for a layer of width n, most of the true gradient will be contained in a k-dimensional subspace with $k << n$, and that k grows much slower than n. In practice, this means that the guessing space and hence variance of the $\tilde{W}^\perp$ method can be kept almost constant as layer width increases, enabling $\tilde{W}^\perp$ to be scaled to larger networks than previous methods.

$\tilde{W}^\perp$-NS Results. Here, we verify that the $\tilde{W}^\perp$-NS approximation to $\tilde{W}^\perp$ with $k = 10$ does not harm performance. First, we describe a few implementation

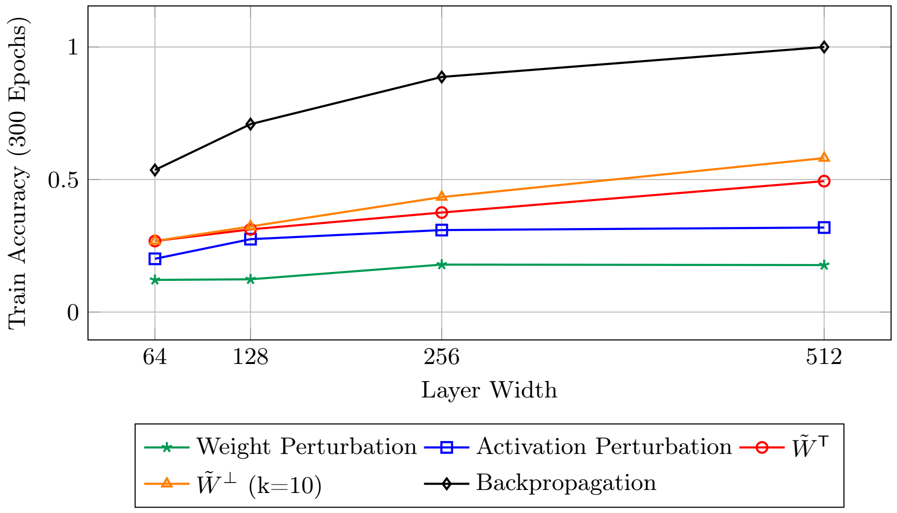

Fig. 2: Training accuracy as the hidden layer width increases.

Table 3: Train accuracies across methods and hidden layer widths. As layer width increases from 64 to 512, the accuracy gap between $\tilde{W}^{\intercal}$ and $W^{\perp}$ increases from 0% up to almost 10%. This result indicates that $\tilde{W}^{\perp}$ will scale to larger networks much better than previous methods. However, there is still a significant gap between gradient guessing methods and backpropagation.

Layer Width	Weight Perturbation	Activation Perturbation	$\tilde{W}^{\intercal}$	$\tilde{W}^{\perp}$ (k=10)	Backprop
64	0.121	0.201	0.268	0.268	0.536
128	0.124	0.275	0.312	0.323	0.709
256	0.179	0.309	0.376	0.434	0.887
512	0.177	0.319	0.494	0.581	1.000

details that further speed up $\tilde{W}^{\perp}$-NS. To avoid having to repeatedly determine the polynomial coefficients that approximate f_{σ_k} (since σ_k varies across layers and over the course of training), we pre-determine a set of coefficients that approximate f_{σ_k} at intervals $\sigma_k = 0.1, 0.2, ..., 0.9$ via gradient descent and use the set of coefficients that σ_k is closest to at the current training iteration. Computing the spectral norm and σ_k can be done efficiently via the Implicitly Restarted Lanczos Method [16]. This can be further sped up by observing that the singular values change slowly over the course of training, allowing us to get away with computing σ_k once every 50 iterations. Tab. 4 shows that after 300 epochs $\tilde{W}^{\perp}$-NS attains comparable accuracies to $\tilde{W}^{\perp}$, and in some cases even surpasses it while being significantly faster than $W^{\perp}$. Our $\tilde{W}^{\perp}$ method and its $\tilde{W}^{\perp}$-NS variant surpasses the baseline $\tilde{W}^{\intercal}$ in all cases. For the MLP with a layer width of 512, $W^{\perp}$-NS has around 10× smaller memory usage than $W^{\perp}$.

Table 4: Train and test accuracies and training times (300 epochs) for our unoptimised[†] implementations of different methods across layer sizes of 128 and 512. [†]PyTorch MPS backend on MacOS laptop; SVD on the CPU.

Method	**Layer Size**	**Train Time**	**Train Accuracy**	**Test Accuracy**
$\tilde{W}^{\perp}$-NS	512	6h	**0.594**	0.477
$\tilde{W}^{\perp}$, $k = 10$	512	40h	0.581	**0.498**
$\tilde{W}^{\top}$	512	26m	0.486	0.438
$\tilde{W}^{\perp}$-NS	128	9m	**0.326**	**0.291**
$\tilde{W}^{\perp}$, $k = 10$	128	1h	0.324	0.283
$\tilde{W}^{\top}$	128	5m	0.312	0.277

Table 5: Layer 1 variance and bias, and accuracy at 300 epochs for an MLP with layer width of 256 trained on our methods and the baseline $\tilde{W}^{\mathsf{T}}$. $\tilde{W}^{\perp}$, with lower variance and bias than $\tilde{W}^{\mathsf{T}}$, reaches the highest accuracy by a wide margin while $\tilde{W}^{\mathrm{P}}$ (preconditioning), with the lowest bias but highest variance, reaches lower accuracy than $\tilde{W}^{\mathsf{T}}$.

Method	**Variance** (18)	**Bias** (17)	**Train Accuracy**	**Test Accuracy**
$\tilde{W}^{\perp}$ ($k = 10$)	$\mathbf{6.33 \times 10^{-6}}$	3.63×10^{-5}	**0.433**	**0.383**
$\tilde{W}^{\mathrm{P}}$	1.75×10^{-4}	$\mathbf{1.35 \times 10^{-8}}$	0.367	0.314
$\tilde{W}^{\top}$	7.04×10^{-5}	1.68×10^{-4}	0.375	0.316

$\tilde{W}^{\mathrm{P}}$ Preconditioning Results. While the earlier weight perturbation and activation perturbation methods emphasise unbiasedness, the baseline $\tilde{W}^{\mathsf{T}}$ and our $\tilde{W}^{\perp}$ methods emphasise reducing variance but at the cost of incurring bias. To gauge the effectiveness of lowering variance versus lowering bias, we compare our $\tilde{W}^{\mathrm{P}}$ preconditioning method with $\tilde{W}^{\perp}$ and the $\tilde{W}^{\mathsf{T}}$ baseline in Tab. 5. We see that lower variance correlates with higher accuracy, while the high variance preconditioning method, despite being unbiased, achieves the lowest accuracy. This result shows that removing bias at the expense of increasing variance is ineffective, and therefore it is better to have low variance even if it incurs bias. Based on the accuracy gains of $\tilde{W}^{\perp}$ as layer widths increase (Fig. 2), we hypothesise that this principle becomes even more relevant for larger networks.

5 Conclusion

In this paper, we introduced the $\tilde{W}^{\perp}$ method to addresses the central problem of high variance in gradient estimation methods. Our $\tilde{W}^{\perp}$ method exploits the low-dimensional structure of gradients by orthogonalising the upstream Jacobian matrix, thereby constraining the guessing space and reducing bias and variance.

Experiments show promising results with $\tilde{W}^{\perp}$ as network width increases, while the $\tilde{W}^{\mathrm{P}}$ preconditioning method demonstrates that removing bias at the cost of increased variance is ineffective. Furthermore, we greatly speed up $\tilde{W}^{\perp}$ by using Newton–Schulz iterations for approximate orthogonalisation. In future work, these ideas can be expanded to more practical architectures, such as networks with normalisation layers or residual connections, or approaches that better consider the underlying structure of gradients in deep neural networks.

References

1. David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
2. Utkarsh Singhal, Brian Cheung, Kartik Chandra, Jonathan Ragan-Kelley, Joshua B Tenenbaum, Tomaso A Poggio, and Stella X Yu. How to guess a gradient. *arXiv preprint arXiv:2312.04709*, 2023.
3. Atılım Güneş Baydin, Barak A. Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.
4. David Silver, Anirudh Goyal, Ivo Danihelka, Matteo Hessel, and Hado van Hasselt. Learning by directional gradient descent. In *International Conference on Learning Representations*, 2022.
5. Kartik Chandra. An unexpected challenge in using forward-mode automatic differentiation for low-memory deep learning. Stanford Digital Repository, 2021.
6. Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, January 1989.
7. Mengye Ren, Simon Kornblith, Renjie Liao, and Geoffrey Hinton. Scaling forward gradient with local losses. In *The Eleventh International Conference on Learning Representations*, 2023.
8. Louis Fournier, Stéphane Rivaud, Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Can forward gradient match backpropagation? In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, 2023.
9. Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace, 2018. arXiv preprint arXiv:1812.04754.
10. Minhak Song, Kwangjun Ahn, and Chulhee Yun. Does SGD really happen in tiny subspaces? In *The Thirteenth International Conference on Learning Representations*, 2025.
11. Zdislav Kovarik. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 7(3):386–389, 1970.
12. Åke Björck and Clazett Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 8(2):358–364, Jun 1971.
13. Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.
14. Sam Greydanus and Dmitry Kobak. Scaling down deep learning with MNIST-1D. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.
15. Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.
16. Daniela Calvetti, L Reichel, and And Sorensen. An implicitly restarted lanczos method for large symmetric eigenvalue problems. *Electronic Trans. Numer. Anal.*, 2:1–21, 04 1994.