The Hessian of tall-skinny networks is easy to invert

Ali Rahimi

October 16, 2025

1 Introduction

The Hessian of a deep net is the matrix of second-order mixed partial derivatives of its loss with respect to its parameters. Reliance on the Hessian has come in and out of favor over the past few decades. In the 80s and 90s, when deep nets had only thousands of parameters, the Hessian matrix could be inverted to implement second order optimizers that converged must faster than gradient descent [8, 1]. For larger networks, approximating the Hessian as a long rank matrix [9, 5] or as a diagonal matrix [2, 3] became more popular. However, as modern deep nets get larger, these methods have become increasingly less practical: The Hessian of a model with a billion parameters would have a quintillion entries, which far more than can be stored, multiplied, factorized, or inverted, even in the largest data centers. We show that the Hessian of a deep net has a regular structure that makes it amenable to these operations. In fact, with the number of parameters fixed, as the network gets deeper, it becomes easier to compute the product of the Hessian with a fixed vector (the Hessian-vector product) and to solve linear systems involving the Hessian (the Hessian-inverse-vector product).

It has long been known that the Hessian-vector product can be computed without incurring quadratically man FLOPs or storage in the number of parameters. The trick, which is due to Pearlmutter [**?**], is to modify the network so that the gradient of the network with respect to the parameters ends up being the product of the Hessian of the original with the vector. This avoids computing the full Hessian, and takes time linear int the number of parameters. Given a way to compute the Hessian-vector product, we can compute the Hessian-inverse-vector product by any number of Krylov iteration, like Conjugate Gradient. However, quality of these methods depends on the conditioning of the Hessian, which is notoriously bad for deep nets [4]. Unfortunately, there seems to be no variant of Pearlmutter's trick to compute the Hessian-inverse-vector product directly.

Regardless of the specific operations in each layer, the Hessian of a layerwise deep net can be represented as a low-order matrix polynomial that involves the first order and second order mixed derivatives of each layer, and the inverse of a block-bi-diagonal operator that represents the backpropoagatin algorithm. Exploiting this structure offers a different way to compute Hessian-vector products, and a direct way to compute the Hessian-inverse-vector product. without forming or storing the full matrix, and without incurring quadratically (let alone qubically) many FLOPs to perform these operations. For an L-layer deep net where each layer has p parameters and produces a activations, naively storing the Hessian would require $O(L^2p^2)$ memory, and multiplying it by a vector or solving a linear system would require $O(L^2p^2 + L\max(a,p)^3)$ and $O(L^3p^3)$ operations, respectively. In contrast, we will show how to perform these operations using only $O(L\max(a,p)^3)$ computations. The dependence on the parameters on the number of activations and parameters in each layer is still cubic, but the dependence on the number of layers is only linear. This makes operating on the Hessian of tall and skinny networks more efficient than the Hessian of short and fat networks.

2 Overview

Our objective is to efficiently solve linear systems of the form $Hx = g$, where H is the Hessian of a deep neural network. Forming H explicitly is infeasible due to its size, and directly inverting or multiplying by H would require $O(L^3p^3)$ and $O(L^2p^2)$ operations, respectively—both prohibitively expensive for large networks. To overcome this, we employ the following strategy:

1. Write down the gradient of the deep net as a bi-diagonal system of linear equations. Solving this system uses back-substitution, which requires a forward and backward pass similar to those used in backpropagation. In fact, back-substitution and backpropagation are identical in this context.
2. Differentiate the components of this linear system, including the bi-diagonal matrix itself, to obtain the second-order derivatives.
3. Reformulate the resulting expressions so that the Hessian appears as a second-order polynomial in the inverse of the bi-diagonal matrix.
4. Derive a fast algorithm to apply the inverse of such polynomials.

3 Notation

We write a deep net as a pipeline of functions $\ell = 1, \ldots, L$,

$$\begin{aligned} z_1 &= f_1(z_0; x_1) \\ &\cdots \\ z_\ell &= f_\ell(z_{\ell-1}; x_\ell) \\ &\cdots \\ z_L &= f_L(z_{L-1}; x_L) \end{aligned} \tag{1}$$

The vectors $x_1, \ldots, x_L$ are the parameters of the pipeline. The vectors $z_1, \ldots, z_L$ are its intermediate activations, and z_0 is the input to the pipeline. The last layer f_L computes the final activations and their training loss, so the scalar z_L is the loss of the model on the input z_0. To make this loss's dependence on z_0 and the parameters explicit, we sometimes write it as $z_L(z_0; x)$. This formalization deviates slightly from the traditional deep net formalism in two ways: First, the training labels are subsumed in z_0, and are propagated through the layers until they're used in the loss. Second, the last layer fuses the loss (which has no parameters) and the last layer (which does).

We'll assume the first and partial derivatives of each layer with respect to its parameters and its inputs exist. This poses some complications with ReLU activations and other non-differentiable operations in modern networks. Notably, for the Hessian to be symmetric,

some We'll largely ignoring this complication and assume that differentiable approximations to these operations are used.

At the end of each section, we'll count the number of floating point operations required to compute various expressions. While the derivations do not impose any restrictions on the shape of the layers, for the purpose of this accounting, we'll assume all but the last L layers have a-dimensional activations ($z_\ell \in \mathbb{R}^a$) and p-dimensional parameters ($x_\ell \in \mathbb{R}^p$).

4 Backpropagation, the matrix way

We would like to fit the vector of parameters $x = (x_1, \ldots, x_L)$ given a training dataset, which we represent by a stochastic input z_0 to the pipeline. Training the model proceeds by gradient descent steps along the stochastic gradient $\partial z_L(z_0; x)/\partial x$. The components of this direction can be computed by the chain rule with a backward recursion:

$$\frac{\partial z_L}{\partial x_\ell} = \underbrace{\frac{\partial z_L}{\partial z_\ell}}_{b_\ell} \underbrace{\frac{\partial z_\ell}{\partial x_\ell}}_{\nabla_x f_\ell} \tag{2}$$

$$\frac{\partial z_L}{\partial z_\ell} = \underbrace{\frac{\partial z_L}{\partial z_{\ell+1}}}_{b_{\ell+1}} \underbrace{\frac{\partial z_{\ell+1}}{\partial z_\ell}}_{\nabla_z f_{\ell+1}}. \tag{3}$$

The identification $b_\ell \equiv \frac{\partial z_L}{\partial z_\ell}$, $\nabla_x f_\ell \equiv \frac{\partial z_\ell}{\partial x_\ell}$, and $\nabla_z f_\ell \equiv \frac{\partial z_\ell}{\partial z_{\ell-1}}$ turns this recurrence into

$$\begin{aligned} \frac{\partial z_L}{\partial x_\ell} &= b_\ell \cdot \nabla_x f_\ell && (4) \\ b_\ell &= b_{\ell+1} \cdot \nabla_z f_{\ell+1}, && (5) \end{aligned}$$

with the base case $b_L = 1$, a scalar. These two equations can be written in vector form as

$$\frac{\partial z_L}{\partial x} = \begin{bmatrix} \frac{\partial z_L}{\partial x_1} & \cdots & \frac{\partial z_L}{\partial x_L} \end{bmatrix} = \underbrace{\begin{bmatrix} b_1 & \cdots & b_L \end{bmatrix}}_{\equiv b} \underbrace{\begin{bmatrix} \nabla_x f_1 & & \\ & \ddots & \\ & & \nabla_x f_L \end{bmatrix}}_{\equiv D_x}, \tag{6}$$

and

$$\begin{bmatrix} b_1 & b_2 & b_3 & \cdots & b_{L-1} & b_L \end{bmatrix} \underbrace{\begin{bmatrix} I & & & & \\ -\nabla_z f_2 & I & & & \\ & -\nabla_z f_3 & I & & \\ & & \ddots & \ddots & \\ & & & -\nabla_z f_L & 1 \end{bmatrix}}_{\equiv M} = \underbrace{\begin{bmatrix} 0 & \cdots & 1 \end{bmatrix}}_{\equiv e_L}. \tag{7}$$

Solving for b and substituting back gives

$$\frac{\partial z_L}{\partial x} = e_L M^{-1} D_x. \tag{8}$$

The matrix M is block bi-diagonal. Its diagonal entries are identity matrices, and its off-diagonal matrices are the gradient of the intermediate activations with respect to the layer's parameters. The matrix D_x is block diagonal, with the block as the derivative of each layer's activations with respect to its inputs. M is invertible because the spectrum of a triangular matrix can be read off its diagonal, which in this case is all ones.

The number of operations required to compute

5 The Hessian

The gradient we computed above is the unique vector v such that $dz_L \equiv z_L(x+dx) - z_L(dx) \to v(x) \cdot dx$ as $dx \to 0$. In this section, we compute the Hessian H of z_L with respect to the parameters. This is the unique matrix $H(x)$ such that $dv^\top \equiv v^\top(x+dx) - v^\top(x) \to H(x)\ dx$ as $dx \to 0$. We use the facts that $dM^{-1} = -M^{-1}(dM)M^{-1}$ and $b = e_L M^{-1}$ to write

$$\begin{aligned} dv &= d(e_L M^{-1} D_x) \\ &= e_L M^{-1}(dD_x) + e_L \left(dM^{-1}\right) D_x \\ &= b \cdot dD_x - e_L M^{-1}(dM)M^{-1} D_x \\ &= b \cdot dD_x - b \cdot (dM) M^{-1} D_x \end{aligned} \tag{9}$$

We compute each of these terms separately. As part of this agenda, we will rely on the gradient of tensor-valued functions $g : \mathbb{R}^d \to \mathbb{R}^{o_1 \times \cdots \times o_k}$. Define this gradient $\nabla_x g(x) \in \mathbb{R}^{(o_1 \cdots o_k) \times d}$ as the unique matrix-valued function that satisfies

$$\operatorname{vec}\left(g(x+dx) - g(x)\right) \to \nabla_x g(x) \cdot dx \tag{10}$$

as $dx \to 0$. This convention readily implies the Hessian of a vector-valued function: If $g : \mathbb{R}^d \to \mathbb{R}^o$, then $\nabla_{xx} g(x) \in \mathbb{R}^{o \times d^2}$ is the unique matrix such that $\operatorname{vec}\left(\nabla_x g(x+dx) - \nabla_x g(x)\right) \to \nabla_{xx} g(x)\ dx$. This convention also readily accommodates the chain rule. For example, the gradient of $h(x) \equiv f(g(x))$ for matrix-valued f and g can be written as $\nabla f \nabla g$ as expected. It also implies partial derivatives like $\nabla_{yz} g$ for $g : \mathbb{R}^{|x|} \to \mathbb{R}^{|g|}$. If $y \in \mathbb{R}^{|y|}$ and $z \in \mathbb{R}^{|z|}$ are restrictions of $x \in \mathbb{R}^{|x|}$ to some $|y|$ and $|z|$-dimensional subsets, then $\nabla_z g(x) \in \mathbb{R}^{|g| \times |z|}$, and $\nabla_{yz} g(x) = \nabla_y \nabla_z g(x) \in \mathbb{R}^{|g||z| \times |y|}$. See Chapter 6 of Magnus & Neudecker [6] for a deeper treatment of higher order derivatives of vector-valued functions.

5.1 The term involving dD_x

The matrix D_x is block-diagonal with its ℓth diagonal block containing the matrix $D_\ell \equiv \nabla_x f_\ell$. Using the facts that $\operatorname{vec}(ABC) = \left(C^\top \otimes A\right) \operatorname{vec}(B)$, and $(A \otimes B)^\top = A^\top \otimes B^\top$, we get

$$\begin{aligned}
b \cdot (dD_x) &= \begin{bmatrix} b_1 & \cdots & b_L \end{bmatrix} \begin{bmatrix} dD_1 & & \\ & \ddots & \\ & & dD_L \end{bmatrix} \\
&= \begin{bmatrix} b_1 \cdot dD_1 & \cdots & b_L \cdot dD_L \end{bmatrix} \\
&= \begin{bmatrix} \operatorname{vec}(dD_1)^\top \left(I \otimes b_1^\top\right) & \cdots & \operatorname{vec}(dD_L)^\top \left(I \otimes b_L^\top\right) \end{bmatrix} \\
&= \begin{bmatrix} \operatorname{vec}(dD_1) \\ \vdots \\ \operatorname{vec}(dD_L) \end{bmatrix}^\top \begin{bmatrix} I \otimes b_1^\top & & \\ & \ddots & \\ & & I \otimes b_L^\top \end{bmatrix} \qquad (11)
\end{aligned}$$

Observe that $\operatorname{vec}(dD_\ell) = d \operatorname{vec} \nabla_x f_\ell(z_{\ell-1}; x_\ell)$ varies with dx through both its arguments x_ℓ and $z_{\ell-1}$. Using mixed partials of vector-valued functions described above, we get

$$\operatorname{vec}(dD_\ell) = d \operatorname{vec}(\nabla_x f_\ell) = (\nabla_{xx} f_\ell)\ dx_\ell + (\nabla_{zx} f_\ell)\ dz_{\ell-1}. \qquad (12)$$

Stacking these equations gives

$$\begin{bmatrix} \operatorname{vec}(dD_1) \\ \vdots \\ \operatorname{vec}(dD_L) \end{bmatrix} = \begin{bmatrix} \nabla_{xx} f_1 & & \\ & \ddots & \\ & & \nabla_{xx} f_L \end{bmatrix} dx + \begin{bmatrix} \nabla_{zx} f_1 & & \\ & \ddots & \\ & & \nabla_{zx} f_L \end{bmatrix} \begin{bmatrix} dz_0 \\ \vdots \\ dz_{L-1} \end{bmatrix}. \qquad (13)$$

Each vector dz_ℓ in turn varies with dx via $dz_\ell = (\nabla_x f_\ell) dx_\ell + (\nabla_z f_\ell) dz_{\ell-1}$, with the base case $dz_0 = 0$, since the input z_0 does not vary with dx. Stacking up this recurrence gives

$$\begin{bmatrix} I & & & \\ -\nabla_z f_2 & I & & \\ & & \ddots & \\ & & -\nabla_z f_L & 1 \end{bmatrix} \begin{bmatrix} dz_1 \\ \vdots \\ dz_{L-1} \\ dz_L \end{bmatrix} = \begin{bmatrix} \nabla_x f_1 & & \\ & \ddots & \\ & & \nabla_x f_L \end{bmatrix} dx. \qquad (14)$$

We can solve for the vector $\begin{bmatrix} dz_1 \\ \vdots \\ dz_L \end{bmatrix} = M^{-1} D_x dx$ and use the downshifting matrix

$$P \equiv \begin{bmatrix} 0 & & & \\ I & 0 & & \\ & & \ddots & \\ & & I & 0 \end{bmatrix} \qquad (15)$$

to plug back the vector $\begin{bmatrix} dz_0 \\ \vdots \\ dz_{L-1} \end{bmatrix} = P M^{-1} D_x dx$:

$$\begin{bmatrix} \operatorname{vec}(dD_1) \\ \vdots \\ \operatorname{vec}(dD_L) \end{bmatrix} = \left(\begin{bmatrix} \nabla_{xx} f_1 & & \\ & \ddots & \\ & & \nabla_{xx} f_L \end{bmatrix} + \begin{bmatrix} \nabla_{zx} f_1 & & \\ & \ddots & \\ & & \nabla_{zx} f_L \end{bmatrix} P M^{-1} D_x \right) dx. \qquad (16)$$

5.2 The term involving dM

The matrix dM is lower-block-diagonal with $dM_2, \dots, dM_L$, and $dM_\ell \equiv d\nabla_z f_\ell$. Similar to the above, we can write

$$
\begin{aligned}
b \cdot (dM) M^{-1} D_x &= \begin{bmatrix} b_1 & \cdots & b_{L-1} & b_L \end{bmatrix} \begin{bmatrix} 0 & & & \\ -dM_2 & 0 & & \\ & & \ddots & \\ & & -dM_L & 0 \end{bmatrix} M^{-1} D_x && (17) \\
&= -\begin{bmatrix} b_2 \cdot dM_2 & \cdots & b_L \cdot dM_L & 0 \end{bmatrix} M^{-1} D_x && (18) \\
&= -\begin{bmatrix} \operatorname{vec}(dM_2)^\top \left(I \otimes b_2^\top \right) & \cdots & \operatorname{vec}(dM_L)^\top \left(I \otimes b_L^\top \right) & 0 \end{bmatrix} M^{-1} D_x && (19) \\
&= -\begin{bmatrix} \operatorname{vec}(dM_1) \\ \vdots \\ \operatorname{vec}(dM_L) \end{bmatrix}^\top \begin{bmatrix} 0 & & & \\ I \otimes b_2^\top & 0 & & \\ & & \ddots & \\ & & I \otimes b_L^\top & 0 \end{bmatrix} M^{-1} D_x && (20) \\
&= -\begin{bmatrix} \operatorname{vec}(dM_1) \\ \vdots \\ \operatorname{vec}(dM_L) \end{bmatrix}^\top \begin{bmatrix} I \otimes b_1^\top & & \\ & \ddots & \\ & & I \otimes b_L^\top \end{bmatrix} P M^{-1} D_x. && (21)
\end{aligned}
$$

Each matrix $dM_\ell = d\nabla_z f_\ell(z_{\ell-1}; x_\ell)$ varies with dx through both x_ℓ and $z_{\ell-1}$ as $d\operatorname{vec}(M_\ell) = (\nabla_{xz} f_\ell)\, dx_\ell + (\nabla_{zz} f_\ell)\, dz_{\ell-1}$. Following the steps of the previous section gives

$$
\begin{bmatrix} \operatorname{vec}(dM_1) \\ \vdots \\ \operatorname{vec}(dM_L) \end{bmatrix} = \left(\begin{bmatrix} \nabla_{xz} f_1 & & \\ & \ddots & \\ & & \nabla_{xz} f_L \end{bmatrix} + \begin{bmatrix} \nabla_{zz} f_1 & & \\ & \ddots & \\ & & \nabla_{zz} f_L \end{bmatrix} P M^{-1} D_x \right) dx. \qquad (22)
$$

5.3 Putting it all together

We have just shown that the Hessian of the deep net has the form

$$
\begin{aligned}
H \equiv \frac{\partial^2 z_L}{\partial x^2} &= D_D \left(D_{xx} + D_{zx} P M^{-1} D_x \right) + D_x^\top M^{-T} P^\top D_M \left(D_{xz} + D_{zz} P M^{-1} D_x \right) && (23) \\
&= D_D D_{xx} + D_D D_{zx} P M^{-1} D_x + D_x^\top M^{-\top} P^\top D_M D_{xz} + D_x^\top M^{-\top} P^\top D_M D_{zz} P M^{-1} D_x. && (24)
\end{aligned}
$$

The various matrices involved are recapitulated below:

$$
D_D \equiv \begin{bmatrix} \underbrace{I \otimes b_1}_{p \times ap} & & \\ & \ddots & \\ & & I \otimes b_L \end{bmatrix}, \qquad D_M \equiv \begin{bmatrix} \underbrace{I \otimes b_1}_{a \times a^2} & & \\ & \ddots & \\ & & I \otimes b_L \end{bmatrix},
$$

$$
P \equiv \begin{bmatrix} 0 & & & \\ I & 0 & & \\ & \ddots & & \\ & & I & 0 \end{bmatrix}, \qquad M \equiv \begin{bmatrix} I & & & & \\ \underbrace{-\nabla_z f_2}_{a \times a} & I & & & \\ & -\nabla_z f_3 & I & & \\ & & \ddots & \ddots & \\ & & & -\nabla_z f_L & 1 \end{bmatrix},
$$

$$
D_x \equiv \begin{bmatrix} \underbrace{\nabla_x f_1}_{a \times p} & & \\ & \ddots & \\ & & \nabla_x f_L \end{bmatrix}, \qquad D_{xx} \equiv \begin{bmatrix} \underbrace{\nabla_{xx} f_1}_{ap \times p} & & \\ & \ddots & \\ & & \nabla_{xx} f_L \end{bmatrix}, \qquad D_{xz} \equiv \begin{bmatrix} \underbrace{\nabla_{xz} f_1}_{a^2 \times p} & & \\ & \ddots & \\ & & \nabla_{xz} f_L \end{bmatrix},
$$

$$
D_{zx} \equiv \begin{bmatrix} \underbrace{\nabla_{zx} f_1}_{ap \times a} & & \\ & \ddots & \\ & & \nabla_{zx} f_L \end{bmatrix}, \qquad D_{zz} \equiv \begin{bmatrix} \underbrace{\nabla_{zz} f_1}_{a^2 \times a} & & \\ & \ddots & \\ & & \nabla_{zz} f_L \end{bmatrix}.
$$

5.4 Mulitplying a vector by the Hessian

Given a vector $g \in \mathbb{R}^{Lp}$, the formula above allows us to compute Hg in $O\left(Lap^2 + La^2p + La^3\right)$ operations without forming H. This cost is dominated by multiplying by the D_{xx}, D_{zx}, and D_{zz} matrices. Appendix A shows that these operations are exactly the operations involved in Pearlmutter's trick to compute the Hessian-vector product.

It's tempting to use this insight to use Krylov methods to solve systems of the form $Hx = b$ without forming H. This would require compute Hg some number of times that depends on the condition number of H. However, the next section shows how to compute $H^{-1}b$ with roughly only as many operations as are needed to comptue Hg.

6 The inverse of the Hessian

The above shows that the Hessian is a second order matrix polynomial in M^{-1}. While M itself is block-bidiagonal, M^{-1} is dense, so H is dense. Nevertheless, this polynomial can be lifted into a higher order object whose inverse is easy to compute:

$$
\begin{aligned}
H &= D_D D_{xx} + D_D D_{zx} P M^{-1} D_x + D_x^\top M^{-\top} P^\top D_M D_{xz} + D_x^\top M^{-\top} P^\top D_M D_{zz} P M^{-1} D_x \\
&= \begin{bmatrix} M^{-1} D_x \\ I \end{bmatrix}^\top \begin{bmatrix} P^\top D_M D_{zz} P & P^\top D_M D_{xz} \\ D_D D_{zx} P & D_D D_{xx} \end{bmatrix} \begin{bmatrix} M^{-1} D_x \\ I \end{bmatrix} \\
&= I + \begin{bmatrix} D_x \\ I \end{bmatrix}^\top \underbrace{\begin{bmatrix} M^{-\top} & \\ & I \end{bmatrix}}_{\hat{M}^{-\top}} \underbrace{\begin{bmatrix} P^\top D_M D_{zz} P & P^\top D_M D_{xz} \\ D_D D_{zx} P & D_D D_{xx} - I \end{bmatrix}}_{\equiv Q} \underbrace{\begin{bmatrix} M^{-1} & \\ & I \end{bmatrix}}_{\hat{M}^{-1}} \begin{bmatrix} D_x \\ I \end{bmatrix}.
\end{aligned}
$$

The Woodbury formula gives

$$
\begin{aligned}
H^{-1} &= I - \begin{bmatrix} D_x \\ I \end{bmatrix}^\top \left(\left(\hat{M}^{-\top} Q \hat{M}^{-1} \right)^{-1} + \begin{bmatrix} D_x \\ I \end{bmatrix} \begin{bmatrix} D_x \\ I \end{bmatrix}^\top \right)^{-1} \begin{bmatrix} D_x \\ I \end{bmatrix} \\
&= I - \begin{bmatrix} D_x \\ I \end{bmatrix}^\top \Bigg(\underbrace{\hat{M} Q^{-1} \hat{M}^\top + \begin{bmatrix} D_x D_x^\top & D_x \\ D_x^\top & I \end{bmatrix}}_{\equiv A} \Bigg)^{-1} \begin{bmatrix} D_x \\ I \end{bmatrix}. \qquad (25)
\end{aligned}
$$

The matrix Q^{-1} can be computed explicitly using the partitioned matrix inverse formula. Define the Schur complement $S = Q_{11} - Q_{12} Q_{22}^{-1} Q_{21}$, where Q_{ij} denote the i, jth block of Q as defined above. Then

$$
Q^{-1} = \begin{bmatrix} S^{-1} & -S^{-1} Q_{12} Q_{22}^{-1} \\ -Q_{22}^{-1} Q_{21} S^{-1} & Q_{22}^{-1} + Q_{22}^{-1} Q_{21} S^{-1} Q_{12} Q_{22}^{-1} \end{bmatrix}. \qquad (26)
$$

The matrices Q_{11}, Q_{12}, Q_{21}, and Q_{22} are all block-diagonal. S is also block diagonal because Q_{11} and $Q_{12} Q_{22}^{-1} Q_{21}$ are both block-diagonal. Since all the terms involved in the blocks of Q^{-1} are block-diagonal, Q^{-1} has the same banded structure as Q.

The inverse of $A \equiv \hat{M} Q^{-1} \hat{M}^\top + \begin{bmatrix} D_x D_x^\top & D_x \\ D_x^\top & I \end{bmatrix}$ can be applied efficiently. Instead of applying the Woodbury formula again, we compute its $LDL^\top$ decomposition and apply the inverse of that decomposition. The $LDL^\top$ decomposition of A is

$$
\begin{aligned}
A &= \begin{bmatrix} I & A_{12} A_{22}^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11} - A_{12} A_{22}^{-1} A_{12}^\top & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} I & A_{12} A_{22}^{-1} \\ 0 & I \end{bmatrix}^\top \\
A_{11} &= M \left[Q^{-1} \right]_{11} M^\top + D_x D_x^\top \\
A_{12} &= M \left[Q^{-1} \right]_{12} + D_x \\
A_{22} &= \left[Q^{-1} \right]_{22} + I. \qquad (27)
\end{aligned}
$$

so

$$
A^{-1} = \begin{bmatrix} I & -A_{12} A_{22}^{-1} \\ 0 & I \end{bmatrix}^\top \begin{bmatrix} A_{11} - A_{12} A_{22}^{-1} A_{12}^\top & 0 \\ 0 & A_{22} \end{bmatrix}^{-1} \begin{bmatrix} I & -A_{12} A_{22}^{-1} \\ 0 & I \end{bmatrix} \qquad (28)
$$

Since A_{11} is block tri-diagonal, A_{12} is block-bidiagonal, and A_{22} is block-diagonal, applying A^{-1} to a vector is fast.

Summary: Algorithm to compute $H^{-1} g$

Given a vector $g \in \mathbb{R}^{Lp}$, compute $H^{-1} g$ as follows:

1. **Compute the auxiliary vector:**

   $$
   g' \in \overset{La+Lp}{\mathbb{R}} \equiv \begin{bmatrix} D_x \\ I \end{bmatrix} g.
   $$

   D_x has L $a \times p$ blocks on its diagonal, so it takes Lap multiplications to compute v.

2. **Form the banded matrix:**

   $$
   A \in \overset{L(a+p) \times L(a+p)}{\mathbb{R}} \equiv \hat{M} Q^{-1} \hat{M}^\top + \begin{bmatrix} D_x D_x^\top & D_x \\ D_x^\top & I \end{bmatrix}.
   $$

   To compute Q^{-1}, we first compute the blocks of Q. These take La^3 multiplications for $Q_{11} \in \mathbb{R}^{La \times La}$, $La^2 p$ for $Q_{12} \in \mathbb{R}^{La \times Lp}$ and $Q_{21} \in \mathbb{R}^{Lp \times La}$, and $La^2 p$ for $Q_{22} \in \mathbb{R}^{Lp \times Lp}$. Computing $S \in \mathbb{R}^{La \times La}$ takes Lp^3 to compute Q_{22}^{-1}, and $2Lap^2$ to compute the product $Q_{12} Q_{22}^{-1} Q_{21}$. Given these quantities, for the blocks of Q^{-1}, it takes an additional La^3 to compute the upper left block, $L(a^2p+ap^2)$ to compute the off-diagonal blocks, and somewhat less than that to compute the bottom diagonal block since the matrices involved have already been computed. In all, it takes less than $9L\max(a,p)^3$ multiplications to compute Q^{-1}.

To compute $\hat{M}Q^{-1}\hat{M}^\top$ requires an additional $2La^3$ operations for a total of $11L\max(a,p)^3$ multiplications.

Finally, computing and adding the second term requires La^2p multiplications, bringing the tally to at most $12L\max(a,p)^3$ multiplications to compute A.

3. **Apply A^{-1} to g':**

$$g'' = \begin{bmatrix} I & -A_{12}A_{22}^{-1} \\ 0 & I \end{bmatrix}^\top \begin{bmatrix} A_{11} - A_{12}A_{22}^{-1}A_{12}^\top & 0 \\ 0 & A_{22} \end{bmatrix}^{-1} \begin{bmatrix} I & -A_{12}A_{22}^{-1} \\ 0 & I \end{bmatrix} g'.$$

This computation requires $2L\max(a,p)^3$ multiplications to compute A_{22}^{-1} and $\left[A_{11} - A_{12}A_{22}^{-1}A_{12}\right]^{-1}$. The remaining operations are matrix multiplications that take at most $3L\max(a,p)^2$, which is smaller than Lp^3 when $p > 3$. This brings the tally to at most $15L\max(a,p)^3$ multiplications.

4. **Compute the final result:**

$$y = g - \begin{bmatrix} D_x \\ I \end{bmatrix}^\top g''.$$

These are again matrix-vector multiplications that take at most $L\max(a,p)^2$ when $p > 1$, bringing the tally to at most $16L\max(a,p)^3$.

References

[1] Etienne Barnard. Optimization for training neural nets. *IEEE Transactions on Neural Networks*, 3(2):232–240, March 1992.

[2] Suzanna Becker and Yann LeCun. Improving the convergence of back-propagation learning with second order methods. *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37, 1989.

[3] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[4] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. *CoRR*, abs/1901.10159, 2019.

[5] Quoc V. Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y. Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 265–272, Bellevue, Washington, USA, 2011. Omnipress.

[6] Jan R Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. John Wiley & Sons, 3rd edition, 2019.

[7] Barak A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160, 1994.

[8] Richard L. Watrous. Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. In *Proceedings of the First IEEE International Conference on Neural Networks*, volume 2, pages 619–627, San Diego, CA, June 1987. IEEE.

[9] Andrew R. Webb and David Lowe. A hybrid optimisation strategy for adaptive feed-forward layered networks. Technical Report RSRE Memorandum 4193, Royal Signals and Radar Establishment (RSRE), Malvern, UK, 1988.

A Pearmutter's Hessian-vector multiplication algorithm

We showed that Equation (24) makes it possible to compute Hessian-vector product Hv in time linear in L. These operations are equivalent to Pearlmutter's [7] algorithm, a framework to compute Hessian-vector products in networks with arbitrary topologies. This section specializes that machinery to the pipeline topology and derives the procedure in an elementary way.

Consider a set of vectors $v_1, \dots, v_L$ that match the dimensions of the parameter vectors $x_1, \dots, x_L$. Just as $z_L(x_1, \dots, x_L)$ denotes the loss under the parameters w, we'll consider the perturbed loss $z_L(x_1 + \alpha v_1, \dots, x_L + \alpha v_L)$ with a scalar α. By the chain rule,

$$\frac{\partial}{\partial \alpha} z_L(x_1 + \alpha_1 v_1, \dots, x_L + \alpha_L v_L)\bigg|_{\alpha=0} = \nabla_x z_L(x_1, \dots, x_L) \cdot v. \tag{29}$$

Applying ∇_x to both sides gives

$$\nabla_x \frac{\partial}{\partial \alpha} z_L(x_1 + \alpha_1 v_1, \dots, x_L + \alpha_L v_L)\bigg|_{\alpha=0} = \nabla_x^2 z_L(x_1, \dots, x_L) \cdot v. \tag{30}$$

In other words, to compute the Hessian-vector product $\nabla_x^2 z_L \cdot v$, it suffices to compute the gradient of $\frac{\partial z_L}{\partial \alpha}$ with respect to x. We can do this by applying standard backpropoagation to $\frac{\partial z_L}{\partial \alpha}$. At each stage ℓ during its backward pass, backpropagation produces $\frac{\partial}{\partial x_\ell} \frac{\partial z_L}{\partial \alpha} = \nabla_{x_\ell, x} z_L \cdot v$, yielding a block of rows in in $\nabla_x^2 z_L \cdot v$.

To see that this generates the same operations as applying Equation (24) to v, we'll write the backprop operations from Equation (2) against $\frac{\partial z_L}{\partial \alpha}$ explicitly. We'll use again the fact that z_ℓ depends on α through both $z_{\ell-1}$ and $x_\ell + \alpha v_\ell$ to massage the backward recursion of for $\frac{\partial z_L}{\partial \alpha}$ into a format that matches Equation (2):

$$b'_\ell \equiv \frac{\partial}{\partial z_\ell} \frac{\partial z_L}{\partial \alpha} = \frac{\partial}{\partial \alpha} \frac{\partial z_L}{\partial z_\ell} = \frac{\partial}{\partial \alpha} b_\ell = \frac{\partial}{\partial \alpha} \left[b_{\ell+1} \cdot \nabla_z f_{\ell+1} \right] \tag{31}$$

$$= b'_{\ell+1} \cdot \nabla_z f_{\ell+1} + \left[(I \otimes b_{\ell+1}) \frac{\partial}{\partial \alpha} \mathrm{vec}\left(\nabla_z f_{\ell+1} \right) \right]^\top \tag{32}$$

$$= b'_{\ell+1} \cdot \nabla_z f_{\ell+1} + \left[(I \otimes b_{\ell+1}) \left(\nabla_{zz} f_{\ell+1} \cdot \frac{\partial z_\ell}{\partial \alpha} + \nabla_{xz} f_{\ell+1} \cdot v_{\ell+1} \right) \right]^\top. \tag{33}$$

During the backward pass, from b_ℓ and b'_ℓ, we compute

$$\frac{\partial}{\partial x_\ell} \frac{\partial z_L}{\partial \alpha} = (\nabla_{x_\ell, x} z_L) \cdot v = \frac{\partial}{\partial \alpha} \frac{\partial z_L}{\partial x_\ell} = \frac{\partial}{\partial \alpha} \left[\frac{\partial z_L}{\partial z_\ell} \frac{\partial z_\ell}{\partial x_\ell} \right] = \frac{\partial}{\partial \alpha} \left[b_\ell \cdot \nabla_x f_\ell \right] \tag{34}$$

$$= b'_\ell \cdot \nabla_x f_\ell + \left[(I \otimes b_\ell) \frac{\partial}{\partial \alpha} \mathrm{vec}\left(\nabla_x f_\ell \right) \right]^\top \tag{35}$$

$$= b'_\ell \cdot \nabla_x f_\ell + \left[(I \otimes b_\ell) \left(\nabla_{zx} f_\ell \cdot \frac{\partial z_{\ell-1}}{\partial \alpha} + \nabla_{xx} f_\ell \cdot v_\ell \right) \right]^\top. \tag{36}$$

Stacking these backward equations horizontally with $b' \equiv [\, b'_1 \cdots b'_L \,]$, $g^\alpha_\ell \equiv \frac{\partial z_\ell}{\partial \alpha}$ and $g^\alpha \equiv \begin{bmatrix} g^\alpha_1 \\ \vdots \\ g^\alpha_L \end{bmatrix}$, then transposing, gives

$$\begin{aligned} M^\top \left(b'\right)^\top &= P D_M \left(D_{zz} g^\alpha + D_{xz} v \right) \\ \nabla_x^2 z_L \cdot v &= D_x^\top \left(b'\right)^\top + D_D \left(D_{zx} P g^\alpha + D_{xx} v \right). \end{aligned} \tag{37}$$

g^α_ℓ can be computed during the forward pass via

$$g^\alpha_\ell \equiv \frac{\partial z_\ell}{\partial \alpha} = \nabla_z f_\ell \cdot \frac{\partial z_{\ell-1}}{\partial \alpha} + \nabla_x f_\ell \cdot v_\ell = \nabla_z f_\ell \cdot g^\alpha_{\ell-1} + \nabla_x f_\ell \cdot v_\ell, \tag{38}$$

which when stacked up, gives $M g^\alpha = D_x v$. Plugging g^α back into Equation (37) and solving for b' gives

$$\nabla_x^2 z_L \cdot v = D_x^\top M^{-\top} P D_M \left(D_{zz} M^{-1} D_x v + D_{xz} v \right) + D_D \left(D_{zx} P M^{-1} D_x v + D_{xx} v \right). \tag{39}$$

This coincides with Equation (24), showing that the two algorithms are equivalent.