MSE-Optimal Neural Network Initialization via Layer Fusion

Ramina Ghods[1], Andrew S. Lan[2], Tom Goldstein[3], and Christoph Studer[4]

[1]Carnegie Mellon University, Pittsburgh, PA; rghods@cs.cmu.edu
[2]University of Massachusetts Amherst, Amherst, MA; andrewlan@cs.umass.edu
[3]University of Maryland, College Park, MD; tomg@cs.umd.edu
[4]Cornell Tech, New York, NY; studer@cornell.edu

***Abstract*—Deep neural networks achieve state-of-the-art performance for a range of classification and inference tasks. However, the use of stochastic gradient descent combined with the nonconvexity of the underlying optimization problems renders parameter learning susceptible to initialization. To address this issue, a variety of methods that rely on random parameter initialization or knowledge distillation have been proposed in the past. In this paper, we propose FuseInit, a novel method to initialize shallower networks by fusing neighboring layers of deeper networks that are trained with random initialization. We develop theoretical results and efficient algorithms for mean-square error (MSE)-optimal fusion of neighboring dense-dense, convolutional-dense, and convolutional-convolutional layers. We show experiments for a range of classification and regression datasets, which suggest that deeper neural networks are less sensitive to initialization and shallower networks can perform better (sometimes as well as their deeper counterparts) if initialized with FuseInit.**

I. Introduction

A prominent approach to improving the performance of artificial neural networks is to increase the number of network parameters [1], [2]. Theoretical and empirical evidence in [3]–[5] suggest that over-parametrization (more parameters in the network than in the training data) enables one to find better minimizers (and often faster) and reduce the generalization error. Furthermore, reference [6] has shown that finding global minimizers can be easier for sufficiently large networks.

Unfortunately, the deployment of deep neural nets with a large number of parameters in resource-constrained systems, such as mobile devices, unmanned aerial vehicles, autonomous cars is extremely challenging in terms of both storage and computation [7], [8]. Fortunately, the parameters of deep networks often exhibit high redundancy and, with appropriate initialization schemes, shallower networks can in many situations be trained to perform as well as their deeper counterparts [9], [10]. For example, reference [11] has demonstrated that one can substantially compress the number of parameters in deep networks, but training of such shallower networks directly, without using a deeper network, is a notoriously difficult task. In many situations, the success or failure of training shallower networks depends on the initialization method—the design of powerful initialization strategies, however, remains an active research area.

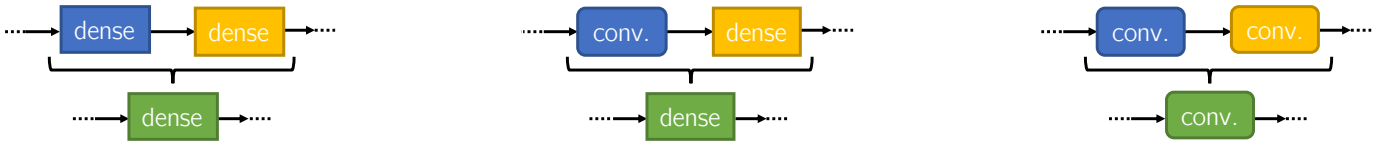
The work of RG and CS was supported in part by Xilinx Inc. and by the US National Science Foundation under grants ECCS-1408006, CCF-1535897, CCF-1652065, CNS-1717559, and ECCS-1824379.

A. Contributions

We propose FuseInit, a principled network initialization method. The key idea of FuseInit is to first train a deeper neural network with initialization methods that rely on random weights—the deeper network is then used to initialize a shallower network by fusing neighboring layers. Using a classical result by Bussgang [12], we develop new theory for mean-square error (MSE)-optimal fusion of neighboring dense-dense, convolutional-dense, convolutional-convolutional layers with arbitrary activation functions. We propose efficient algorithms for FuseInit that scale favorably to deeper neural networks and large datasets. To demonstrate the efficacy of our approach, we show experimental results for a range of classification and regression datasets. Our results suggest that deeper networks are less sensitive to initialization and shallower networks can perform better (sometimes as well as their deeper counterparts) if initialized with FuseInit.

B. Relevant Prior Art

The majority of parameter initialization schemes for neural nets deployed in practice rely on randomly initialized network parameters. A widespread approach to random initialization is the use of zero-mean Gaussian random variables with small variance (e.g., 0.01) [13]. Reference [14] proposed random initialization with a variance that depends on the number of inputs and outputs of the layer to be initialized. Reference [15] improved upon this approach for networks with ReLU activations by using random variables with variance $2/N$, where N stands for the number of inputs to the target layer. Other methods that focus particularly on deep network initialization with random parameters have been proposed in, e.g., [16], [17]. Our focus is on initializing shallow networks. FuseInit combines random initialization with an expansion-and-fusion strategy: To initialize a target network, first add one (or multiple) layers to the network, initialize the deeper network with random parameters, train it, and finally fuse it to the smaller target architecture.



(a) dense-dense → dense *(b) conv.-dense → dense* *(c) conv.-conv. → conv.*

Fig. 1: *The three considered scenarios of fusing neighboring dense and/or convolutional layers.*

A prominent approach to train shallow neural networks from deep networks is knowledge distillation [18]. This approach builds upon the idea of imposing the outputs of a deeper teacher network to the outputs of the shallower student network. FuseInit differs from such methods as it starts directly from a deeper network and successively fuses neighboring layers to initialize the parameters of the shallower network instead of training the shallower (student) network with the outputs of the deeper (teacher) network from scratch. FuseInit can be combined with such methods by initializing the student network, which can then be trained via knowledge distillation.

ExpandNet is a recent initialization method for shallow networks [19]. The idea is to learn shallow nets by expanding each layer into multiple linear layers and training the expanded network. FuseInit differs from this approach in the following ways. While ExpandNet is using linear layers, FuseInit is able to *optimally* fuse *nonlinear* layers. FuseInit also uses the MSE-optimal fusion weights as a starting point to retrain the shallower network. Our experiments indicate that this re-training step significantly improves the performance of the shallower network. While ExpandNet only relies on experiments, we provide theory for MSE-optimal fusion of neighboring layers and use experiments to demonstrate the effectiveness of FuseInit. We furthermore provide an MSE analysis for the fused layers, which provides a metric that can be used to determine which layers to fuse.

Slightly less related to FuseInit is the plethora of network simplification methods that aim at reducing the number of parameters of deep neural nets; see, e.g., [20], [21] and the references therein. Pruning methods are among the most prominent ones and remove network parameters based on their magnitude [22] or the cost function [23]–[25]. Other network simplification methods include quantization [26]–[28], sparsity [29], and low-rank structure [30]. The concept of FuseInit can be generalized for a range of network architectures, including networks with sparse and low-rank structure.

II. FuseInit: MSE-Optimal Neural Network Initialization via Layer Fusion

We now detail FuseInit for the three cases illustrated in Figure 1: (a) Two dense layers are fused into one dense layer, (b) one convolutional layer and one dense layer are fused into one dense layer, and (c) two convolutional layers are fused into a convolutional layer. We first summarize the notation and then present theoretical results for MSE-optimal fusion of neighboring layers. Finally, we show an efficient FuseInit algorithm that scales to deep neural networks and large datasets.

A. Notation

Lowercase and uppercase boldface letters represent column vectors and matrices, respectively. For a matrix $\mathbf{A}$, the transpose is $\mathbf{A}^T$, and the ith row and jth column entry is $\mathbf{A}[i,j]$. For a vector $\mathbf{a}$, the ith entry is $\mathbf{a}[i]$, and the sub-vector containing the ith to jth entries is $\mathbf{a}[i:j] = \mathbf{a}_{i:j}$; furthermore, $\mathbf{a}[i:j:k] = \mathbf{a}_{i:j:k}$ stands for a vector consisting of one entry every other k entries taken from the ith to the jth entries of vector $\mathbf{a}$; $\sum_{i=1,i+=s}^{L} \mathbf{a}[i]$ denotes summation of $\mathbf{a}[i]$ starting from index 1 to L with strides of s. The ℓ_2-norm of $\mathbf{a}$ is $\|\mathbf{a}\|_2$; $\text{flip}(\mathbf{a})$ denotes a vector $\mathbf{a}$ with its entries in reverse order.

B. FuseInit for Dense-Dense and Convolutional-Dense Layers

Consider the following model for two consecutive layers of a neural network, with $\mathbf{a}_0 \in \mathbb{R}^{L_0}$ as the input to the first layer and $\mathbf{a}_2 \in \mathbb{R}^{L_2}$ as the output of the second layer. Note that these can be any two neighboring layers in a deep neural network, as long as the second layer is a dense, fully-connected layer. As it will be clear later, there are no restrictions on the first layer since we only need its empirical moments. The function $H_1(\cdot)$ fully characterizes the input-output relation of the first layer. Let the second layer use activation function $f_2(\cdot)$, weight matrix $\mathbf{W}_2 \in \mathbb{R}^{L_2 \times L_1}$, and bias vector $\mathbf{b}_2 \in \mathbb{R}^{L_2}$. The following model describes the end-to-end input-output relation of the two neighboring layers:

$$\mathbf{a}_2 = f_2(\mathbf{W}_2\mathbf{a}_1 + \mathbf{b}_2) \quad \text{and} \quad \mathbf{a}_1 = H_1(\mathbf{a}_0). \tag{1}$$

Note that the inputs to the first and second layers may not be vectors; in this case, we vectorize $\mathbf{a}_0$ and $\mathbf{a}_1$. In order to fuse two neighboring layers into one, we use the following three-step procedure. In the first step, we train the parameters of the entire network by random initialization using a standard training method, e.g., stochastic gradient descent. In the second step, we use the trained parameters to fuse the first and second layer into a single dense layer with input-output relation

$$\mathbf{a}_2 = f_2(\tilde{\mathbf{W}}\mathbf{a}_0 + \tilde{\mathbf{b}}), \tag{2}$$

where $\tilde{\mathbf{W}} \in \mathbb{R}^{L_2 \times L_0}$ is a new weight matrix and $\tilde{\mathbf{b}} \in \mathbb{R}^{L_2}$ a new bias vector; we keep the activation function $f_2(\cdot)$ of the second layer. We select the new weight matrix and bias vector to minimize the MSE between the output of the initial two layers (1) and the output of the new fused dense layer (2). Mathematically, we solve the following optimization problem:

$$\{\tilde{\mathbf{W}}^\star, \tilde{\mathbf{b}}^\star\} = \underset{\tilde{\mathbf{W}} \in \mathbb{R}^{L_2 \times L_0}, \tilde{\mathbf{b}} \in \mathbb{R}^{L_2}}{\arg\min} \; \textit{MSE}. \tag{3}$$

Here, the MSE is defined as

$$MSE = \mathbb{E}\left[\left\|\left(\tilde{\mathbf{W}}\mathbf{a}_0 + \tilde{\mathbf{b}}\right) - \left(\mathbf{W}_2 H_1(\mathbf{a}_0) + \mathbf{b}_2\right)\right\|_2^2\right], \quad (4)$$

where the expectation $\mathbb{E}[\cdot]$ is over the distribution of the input vector $\mathbf{a}_0$. In the third step, we retrain the entire fused neural network (including other layers) by initializing the fused layer with the new weight matrix $\tilde{\mathbf{W}}^\star$ and new bias vector $\tilde{\mathbf{b}}^\star$ obtained from solving (3). We note that while minimizing the MSE is not necessarily optimal in terms of classification or regression performance, it yields analytical expressions and efficient algorithms (see Section II-D).

The following result for MSE-optimal weights and biases builds upon the nonlinear signal decomposition by J. J. Bussgang in [12]. See Appendix A for the proof.

Theorem 1. *Let (1) be the input-output relation of two neighboring layers of a trained neural net. Define the vectors* $\bar{\mathbf{a}}_0 = \mathbb{E}[\mathbf{a}_0]$ *and* $\bar{\mathbf{a}}_1 = \mathbb{E}[\mathbf{a}_1] = \mathbb{E}[H_1(\mathbf{a}_0)]$*, where expectation is over the distribution of* $\mathbf{a}_0$*. Define the covariance matrix*

$$\mathbf{C}_{\mathbf{a}_0} = \mathbb{E}\left[(\mathbf{a}_0 - \bar{\mathbf{a}}_0)(\mathbf{a}_0 - \bar{\mathbf{a}}_0)^T\right], \quad (5)$$

and the cross-covariance matrix

$$\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0} = \mathbb{E}_{\mathbf{a}_0}\left[(\mathbf{a}_1 - \bar{\mathbf{a}}_1)(\mathbf{a}_0 - \bar{\mathbf{a}}_0)^T\right]. \quad (6)$$

By assuming that the covariance matrix $\mathbf{C}_{\mathbf{a}_0}$ *is full rank, the new weight matrix* $\tilde{\mathbf{W}}^\star$ *and bias vector* $\tilde{\mathbf{b}}^\star$ *of the equivalent layer (2) that minimizes MSE in (4) are given by*

$$\tilde{\mathbf{W}}^\star = \mathbf{W}_2\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}\mathbf{C}_{\mathbf{a}_0}^{-1} \text{ and } \tilde{\mathbf{b}}^\star = \mathbf{W}_2\bar{\mathbf{a}}_1 + \mathbf{b}_2 - \tilde{\mathbf{W}}^\star\bar{\mathbf{a}}_0. \quad (7)$$

The only assumption required in Theorem 1 is that the matrix $\mathbf{C}_{\mathbf{a}_0}$ has full rank; a more general condition is to use any new weight matrix $\tilde{\mathbf{W}}^\star$ for which $\tilde{\mathbf{W}}^\star\mathbf{C}_{\mathbf{a}_0} = \mathbf{W}_2\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}$. In our experiments with the algorithm detailed in Section II-D, we have not observed this matrix to be rank deficient. Furthermore, we emphasize that the method in Theorem 1 can also be used to fuse more than two consecutive layers and more general network structures—in this case, the function $H_1(\cdot)$ simply represents the effect of multiple layers.

From Theorem 1, we can obtain the following compact expression for the MSE incurred by layer fusion; a short derivation is given in Appendix B.

Corollary 1. *The MSE of the fused layer in (4) obtained by Theorem 1 is given by*

$$MSE = \mathrm{trace}\left(\mathbf{W}_2\left(\mathbf{C}_{\mathbf{a}_1} - \mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}\mathbf{C}_{\mathbf{a}_0}^{-1}\mathbf{C}_{\mathbf{a}_0\mathbf{a}_1}\right)\mathbf{W}_2^T\right). \quad (8)$$

We note that this result can be used to determine which layers in a network to fuse. A detailed study on methods that select the best layers to fuse is left for future work.

C. FuseInit of Convolutional-Convolutional Layers

Consider the following model for two consecutive convolutional layers of a neural network. For the sake of simplicity, we detail the 1-dimensional case. The first layer has M input channels, each of length L_0, i.e., $\{\mathbf{a}_0^1, \ldots, \mathbf{a}_0^M\}$, and N output channels, each of length L_1, i.e., $\{\mathbf{a}_1^1, \ldots, \mathbf{a}_1^N\}$. The second layer has P output channels, each of length L_2, i.e., $\{\mathbf{a}_2^1, \ldots, \mathbf{a}_2^P\}$. In what follows, we assume that the the following zero-padding strategy is implemented.

Definition 1. *If the vector* $\mathbf{x}$ *is convolved with a filter of length* k*, then we pad the first and last entries of* $\mathbf{x}$ *with* $\lfloor \frac{k}{2} \rfloor$ *and* $\lfloor \frac{k-1}{2} \rfloor$ *zeros, respectively. This zero-padding operation is denoted by* $\mathcal{Z}^S(\mathbf{x})$*.*

The following model describes the input-output relation of the two neighboring convolutional layers:

$$\mathbf{a}_1^n = f_1\left(\textstyle\sum_{m=1}^M \mathbf{h}_1^{m,n} * \mathbf{a}_0^m + \mathbf{b}_1^n\right), \quad n = 1, \ldots, N \quad (9)$$

$$\mathbf{a}_2^p = f_2\left(\textstyle\sum_{n=1}^N \mathbf{h}_2^{n,p} * \mathbf{a}_1^n + \mathbf{b}_2^p\right), \quad p = 1, \ldots, P. \quad (10)$$

Here, the superscripts for the filters $\mathbf{h}_1^{m,n}$ and $\mathbf{h}_2^{n,p}$ denote the input and output channel index, respectively. We assume that the convolutions performed with the filters $\mathbf{h}_1^{m,n}$ and $\mathbf{h}_2^{n,p}$ have stride s_1 and s_2, respectively. The functions $f_1(\cdot)$ and $f_2(\cdot)$ describe each layer's activation function and a max-pool of stride r_1 and r_2; these functions can also represent batch normalization or dropout.

To fuse two neighboring convolutional layers into one, we use a three-step procedure similar to that in Section II-B. In the first step, we train the parameters of the entire network using random initialization. In the second step, we use the trained parameters to fuse the two layers in (9) and (10) into a single convolutional layer with input-output relation:

$$\mathbf{a}_2^p = f_2\left(\textstyle\sum_{m=1}^M \tilde{\mathbf{h}}^{m,p} * \mathbf{a}_0^m + \tilde{\mathbf{b}}^p\right), \quad p = 1, \ldots, P. \quad (11)$$

Here, $\tilde{\mathbf{h}}^{m,p}$ are new filter coefficients and $\tilde{\mathbf{b}}^p$ new bias vectors; we keep the activation function $f_2(\cdot)$ of the second layer. Note that the convolution has stride $\tilde{s}$ and uses the same zero-padding strategy as defined above. As in Section II-B, we propose to select the new filter coefficients and bias vectors to minimize the MSE per output channel p between the output of the initial two layers, denoted by *C-MSE*^{p}. Put simply, we seek the quantities $\tilde{\mathbf{h}}^{m,p}$, $m = 1, \ldots, M$, and $\tilde{\mathbf{b}}^p$ that minimize

$$\text{C-MSE}^p = \quad (12)$$
$$\mathbb{E}\left[\left\|\left(\textstyle\sum_{n=1}^N \mathbf{h}_2^{n,p} * \mathbf{a}_1^n + \mathbf{b}_2^p\right) - \left(\textstyle\sum_{m=1}^M \tilde{\mathbf{h}}^{m,p} * \mathbf{a}_0^m + \tilde{\mathbf{b}}^p\right)\right\|_2^2\right],$$

for $p = 1, \ldots, P$, where expectation is over the distribution of the input vectors $\mathbf{a}_0^m$, $m = 1, \ldots, M$. In the third step, we retrain the entire fused neural net (including the other layers) by initializing the filters of the fused layer with the new filter coefficients and bias vectors obtained by minimizing (12).

We obtain the following result for MSE-optimal filters and bias vectors. The proof of the following theorem is included in a slightly longer arXiv version of this paper; see [31, App. C]. In contrast to the derivation in Appendix A, the proof for convolutional layers is more involved considering that they include input, output channels, and zero-padding.

Theorem 2. *Let (9) and (10) describe the input-output relation of two consecutive 1-dimensional convolutional layers of a trained deep neural network. Define* $\bar{\mathbf{a}}_0^m = \mathbb{E}[\mathbf{a}_0^m]$*,*

$m = 1, \ldots, M$, *and* $\overline{\mathbf{a}}_1^n = \mathbb{E}[\mathbf{a}_1^n]$, $n = 1, \ldots, N$. *Furthermore, define the auxiliary quantities*

$$\mathbf{v}^p = \sum_{n=1}^N \mathbf{h}_2^{n,p} * (\mathbf{a}_1^n - \overline{\mathbf{a}}_1^n), \quad (13)$$

$$\mathbf{u}^m = \text{flip}[\mathcal{Z}^s(\mathbf{a}_0^m - \overline{\mathbf{a}}_0^m)], \quad (14)$$

and assume that input vectors $\mathbf{a}_0^m$ *from different channels* m *are uncorrelated, i.e.*

$$\mathbb{E}\left[\left(\mathbf{a}_0^m - \overline{\mathbf{a}}_0^m\right)\left(\mathbf{a}_0^{m'} - \overline{\mathbf{a}}_0^{m'}\right)\right] = 0 \quad \textit{for} \quad m \neq m'. \quad (15)$$

Select a new filter length $\tilde{k}$*. Then, the filter and bias vectors that minimize (12) of the convolutional layer in (11) for input and output channel indices* $m' = 1, \ldots, M$ *and* $p' = 1, \ldots, P$ *are given by*

$$\tilde{\mathbf{h}}^{m',p'} = \left(\mathbf{U}^{m'}\right)^{-1} \mathbf{z}^{m',p'}, \quad (16)$$

$$\tilde{\mathbf{b}}^{p'} = \sum_{n=1}^N \mathbf{h}_2^{n,p'} * \overline{\mathbf{a}}_1^n + \mathbf{b}_2^{p'} - \sum_{m=1}^M \tilde{\mathbf{h}}^{m,p'} * \overline{\mathbf{a}}_0^m, \quad (17)$$

with the two auxiliary quantities

$$\mathbf{U}^{m'} = \mathbb{E}\left[\sum_{i=1, i+=\tilde{s}}^{L_0} \left(\mathbf{u}^{m'}_{L_0-i+1:L_0-i+\tilde{k}}\right)\left(\mathbf{u}^{m'}_{L_0-i+1:L_0-i+\tilde{k}}\right)^T\right] \quad (18)$$

$$\mathbf{z}^{m',p'} = \mathbb{E}\left[\sum_{i=1, i+=\tilde{s}}^{L_0} \mathbf{v}^{p'}\left[\tfrac{i-1}{\tilde{s}} + 1\right] \mathbf{u}^{m'}_{L_0-i+1:L_0-i+\tilde{k}}\right], \quad (19)$$

where the filter $\tilde{\mathbf{h}}$ *has stride* $\tilde{s} = s_1 r_1 s_2$.

Note that the above result requires the matrices $\mathbf{U}^{m'}$ to be full rank; in all our experiments in Section III, we have not observed this matrix to be rank deficient. Furthermore, the assumption in (15) may not hold in practice, especially if the number of channels is large. In our experiments, however, different channels were approximately uncorrelated. Similar to Theorem 1, the above result can be used to fuse multiple convolutional layers into one convolutional layer. In addition, a generalization to two or more dimensional convolutions follows analogously, but results in arduous expressions.

D. FuseInit in Practice

While the results in Theorems 1 and 2 enable compact analytical expressions for MSE-optimal layer fusion, explicit results for the first and second moments are often unavailable. In fact, one would need to have knowledge of the data distribution. In addition, even if the distribution were known perfectly, analytically computing the first and second moment is often difficult, even for simple distributions. Since a vast amount of training data is available in most applications, we can replace the exact moments with empirical moments computed with training data. Algorithm 1 summarizes a practical approach to FuseInit for the case of fusing a neural net into a dense layer—the algorithm for fusing convolutional layers is analogous.

In Step 1, one can use any of the existing random initialization methods. In our experiments, we will use zero-mean Gaussian random variables with variance 0.05. Another widely used initializer is He-initializer in [15], where we sample from a truncated zero-mean Gaussian distribution with variance $2/N$ (N is the number of inputs). We have excluded results for the He-initializer as they are indistinguishable to our current results. In Step 2, we only need to sample T vectors in the neural network that correspond to T training samples to calculate the necessary empirical moments (which we all compute in parallel). As shown by [32], T only needs to be slightly larger than the number of input dimensions of the layers (L_0 and L_1) for the empirical moments to be accurate estimates of the true covariance matrices $\mathbf{C}_{\mathbf{a}_0}$ and $\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}$. Hence, the computational complexity of FuseNet is dominated by neural network inference for the T training samples and empirical computation of the two matrices $\mathbf{C}_{\mathbf{a}_0}$ and $\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}$. In situations where the layers contain thousands of nodes, the inversion of $\mathbf{C}_{\mathbf{a}_0}$ in (7) can be done implicitly using conjugate gradient methods. Furthermore, for such large networks, storage of $\mathbf{C}_{\mathbf{a}_0}$ and $\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}$ becomes the major bottleneck. In Step 3, the network is retrained using the same T training samples. As we will show next, far fewer epochs are required to retrain the network to achieve good performance.

Algorithm 1 Practical FuseInit algorithm for fusing dense-dense and convolutional-dense layers

Let the architecture in (1) describe two consecutive fully-connected layers and let the assumptions in Theorem 1 hold. Then, FuseInit is given by the following 3-step process:

1) Train the original neural network using random initialization with T training data samples.
2) Using the trained parameters, compute the fusion weight matrix $\tilde{\mathbf{W}}^\star$ and bias vector $\tilde{\mathbf{b}}^\star$ in (7) by first and second empirical moments using the T training data samples.
3) Replace the two fused layers in (1) with the single dense layer $\mathbf{a}_2 = f_2(\tilde{\mathbf{W}}\mathbf{a}_0 + \tilde{\mathbf{b}})$. Retrain the fused network by initializing the fused layer with $\tilde{\mathbf{W}}^\star$ and $\tilde{\mathbf{b}}^\star$ and the remaining layers with the trained parameters from Step 1.

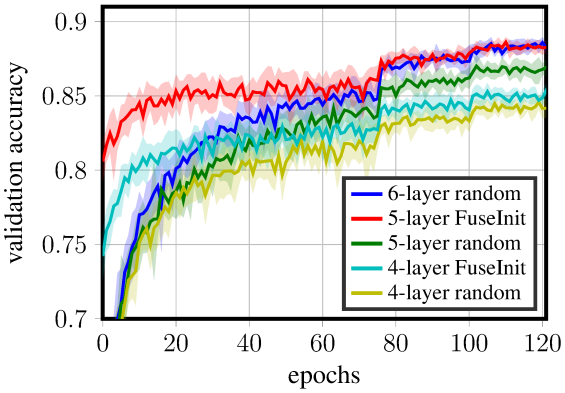


Fig. 2: *Comparison of validation accuracy for FuseInit and random initialization for different convolutional nets on CIFAR-10. FuseInit outperforms random initialization for the considered scenario; the 5-layer FuseInit net achieves the same accuracy as the randomly-initialized 6-layer net.*

III. Experimental Results

We now demonstrate the efficacy of FuseInit on five datasets. Tables I to V summarize the validation accuracy (or loss)

TABLE I: *Validation accuracy of convolutional-dense layers on CIFAR-10 dataset [33].*

Algorithm	FuseInit	Random
6-layer: 32-32-64-64-128-128	–	0.8825 ± 0.0040
5-layer: 32-32-64-64-128	**0.8826** ± 0.0041	0.8691 ± 0.0056
4-layer: 32-32-64-64	**0.8535** ± 0.0046	0.8417 ± 0.0060

TABLE II: *Validation accuracy of convolutional-dense layers on Fashion-MNIST dataset [34].*

Algorithm	FuseInit	Random
4-layer: 2-4-8-16	–	0.9107 ± 0.0024
3-layer: 2-4-8	**0.9120** ± 0.0025	0.9104 ± 0.0017
2-layer: 2-4	**0.9010** ± 0.0019	0.8971 ± 0.0024
1-layer: 2	**0.8803** ± 0.0030	0.8756 ± 0.0043

TABLE III: *Validation accuracy of convolutional-convolutional layers on HAR dataset [35].*

Algorithm	FuseInit	Random
2-layer: 18-36	–	0.962 ± 0.002
1-layer: 36	**0.958** ± 0.005	**0.958** ± 0.002

TABLE IV: *Validation accuracy of convolutional-convolutional layers on speech commands dataset [36].*

Algorithm	FuseInit	Random
4-layer: 32-32-64-64	–	0.887 ± 0.005
3-layer: 32-64-64	**0.880** ± 0.006	0.868 ± 0.003

TABLE V: *Validation mean-absolute error (smaller is better) of dense-dense layers on wireless positioning dataset [37].*

Algorithm	FuseInit	Random
3-layer: 16-128-2	–	7.426 ± 0.112
2-layer: 16-2	**7.221** ± 0.336	7.277 ± 0.472
1-layer: 2	12.273 ± 0.001	**12.262** ± 0.0053

of FuseInit on CIFAR-10 [33], Fashion-MNIST [34], human activity recognition (HAR) [35], speech commands [36] and wireless positioning [37] dataset. For each row of each table, we fuse one-by-one the layers of the network using FuseInit. We then report the mean and standard deviation of the achieved validation accuracy (or loss) over 10 trials in comparison to a randomly initialized network. The left column lists the number of nodes (channels) used per layer of the corresponding dense (convolutional) network. Furthermore, we carry out a sufficiently large number of epochs for all experiments so that the validation accuracy (or loss) settles to a stable value.

To further illustrate the efficacy of FuseInit, we provide Figure 2. This figure shows the mean and standard deviation of the validation accuracy over training epochs for CIFAR-10. Clearly, FuseInit provides a high-quality starting point for the network parameters, which helps the network to converge to an accuracy that is superior to that of randomly-initialized networks with the same topology. (The accuracy jump at epoch 75 is due to reduction of learning rate which is used to improve performance.) Overall, our results indicate that neural networks that are initialized with FuseInit perform better that their randomly initialized counterparts.

IV. Conclusions

We have proposed FuseInit, a novel method to fuse neighboring layers in multi-layer neural networks. FuseInit can be used to initialize shallower networks by first training deeper dense or convolutional networks with random weight initialization strategies, followed by layer fusion and retraining. For MSE-optimal layer fusion, we have developed analytical results and efficient algorithms. Our experiments on five datasets have shown that FuseInit is able to consistently outperform random weight initialization methods. Furthermore, our results reveal that shallower networks can sometimes perform as well as their deeper counterparts if initialized with FuseInit.

There are many avenues for future work. FuseInit can be modified to train and initialize networks with special structure, such as residual or sparse networks—a corresponding study is part of ongoing work. The MSE expression in Corollary 1 can potentially be used to identify the best layers that should be fused in deep network architectures. Furthermore, since FuseInit builds upon ideas from Bussgang's theorem, one could study lower-bounds on the information flow of neural networks.

Appendix A
Proof of Theorem 1

We wish to minimize post-fusion MSE in (4). Our approach builds upon a generalization of the nonlinear, scalar signal decomposition by [12] to an affine vector decomposition. Specifically, we first compute the new MSE-optimal bias vector $\tilde{\mathbf{b}}^\star$. Since (4) is a quadratic form, we can take the derivative with respect to $\tilde{\mathbf{b}}$ and setting it to zero, which yields

$$\frac{\partial}{\partial \tilde{\mathbf{b}}} \mathbb{E}\left[\left\| \left(\tilde{\mathbf{W}}\mathbf{a}_0 + \tilde{\mathbf{b}} \right) - \left(\mathbf{W}_2\mathbf{a}_1 + \mathbf{b}_2 \right) \right\|_2^2 \right] = 0 \tag{20}$$

$$\frac{\partial}{\partial \tilde{\mathbf{b}}} \mathbb{E}\left[\|\tilde{\mathbf{b}}\|_2^2 + 2\tilde{\mathbf{b}}^T \left(\tilde{\mathbf{W}}\mathbf{a}_0 - \left(\mathbf{W}_2\mathbf{a}_1 + \mathbf{b}_2 \right) \right) \right] = 0. \tag{21}$$

Here, expectation is over the distribution of the input data $\mathbf{a}_0$. Basic matrix-vector calculus yields

$$\tilde{\mathbf{b}}^\star = \mathbf{W}_2\bar{\mathbf{a}}_1 + \mathbf{b}_2 - \tilde{\mathbf{W}}\bar{\mathbf{a}}_0, \tag{22}$$

where $\bar{\mathbf{a}}_1 = \mathbb{E}[\mathbf{a}_1] = \mathbb{E}_{\mathbf{a}_0}[H_1(\mathbf{a}_0)]$ and $\bar{\mathbf{a}}_0 = \mathbb{E}[\mathbf{a}_0]$. Next, we replace $\tilde{\mathbf{b}}$ in MSE expression and take the derivative with respect to the new weight matrix $\tilde{\mathbf{W}}$ and set it to zero:

$$\begin{aligned} &\frac{\partial}{\partial \tilde{\mathbf{W}}} \mathbb{E}\left[\left\| \tilde{\mathbf{W}} \left(\mathbf{a}_0 - \bar{\mathbf{a}}_0 \right) - \mathbf{W}_2 \left(\mathbf{a}_1 - \bar{\mathbf{a}}_1 \right) \right\|_2^2 \right] \\ &\quad = \mathbb{E}\left[\frac{\partial}{\partial \tilde{\mathbf{W}}} \| \tilde{\mathbf{W}} \left(\mathbf{a}_0 - \bar{\mathbf{a}}_0 \right) \|_2^2 + \| \mathbf{W}_2 \left(\mathbf{a}_1 - \bar{\mathbf{a}}_1 \right) \|_2^2 \right. \\ &\quad \left. -2 \left(\mathbf{a}_1 - \bar{\mathbf{a}}_1 \right)^T \mathbf{W}_2^T \tilde{\mathbf{W}} \left(\mathbf{a}_0 - \bar{\mathbf{a}}_0 \right) \right] \qquad (23) \\ &\quad = \tilde{\mathbf{W}}\mathbf{C}_{\mathbf{a}_0} - \mathbf{W}_2\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0} = 0. \qquad (24) \end{aligned}$$

This expression results in the one provided in (7). Note that even if $\mathbf{C}_{\mathbf{a}_0}$ is not invertible, the result in (24) can be used to find an MSE-optimal weight matrix by computing a matrix $\tilde{\mathbf{W}}$ that satisfies the following condition:

$$\tilde{\mathbf{W}}\mathbf{C}_{\mathbf{a}_0} = \mathbf{W}_2\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}. \tag{25}$$

APPENDIX B
PROOF OF COROLLARY 1

As an immediate consequence of Bussgang's decomposition in [12], and with the optimal quantities $\tilde{\mathbf{W}}^\star$ and $\tilde{\mathbf{b}}$ obtained above, the MSE in (4) can be expressed as follows:

$$\begin{aligned} MSE &= \mathbb{E}\left[\|\tilde{\mathbf{W}}\left(\mathbf{a}_0 - \bar{\mathbf{a}}_0\right)\|_2^2 + \|\mathbf{W}_2\left(\mathbf{a}_1 - \bar{\mathbf{a}}_1\right)\|_2^2\right. \\ &\quad \left. -2\left(\mathbf{a}_1 - \bar{\mathbf{a}}_1\right)^T\mathbf{W}_2^T\tilde{\mathbf{W}}\left(\mathbf{a}_0 - \bar{\mathbf{a}}_0\right)\right] \qquad (26) \\ &= \mathrm{trace}\Big(\tilde{\mathbf{W}}\mathbf{C}_{\mathbf{a}_0}\tilde{\mathbf{W}}^T + \mathbf{W}_2\mathbf{C}_{\mathbf{a}_1}\mathbf{W}_2^T - 2\tilde{\mathbf{W}}\mathbf{C}_{\mathbf{a}_0\mathbf{a}_1}\mathbf{W}_2^T\Big) \qquad (27) \\ &= \mathrm{trace}\big(\mathbf{W}_2\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}\mathbf{C}_{\mathbf{a}_0}^{-T}\mathbf{C}_{\mathbf{a}_0\mathbf{a}_1}\mathbf{W}_2^T + \mathbf{W}_2\mathbf{C}_{\mathbf{a}_1}\mathbf{W}_2^T \\ &\quad -2\mathbf{W}_2\mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}\mathbf{C}_{\mathbf{a}_0}^{-1}\mathbf{C}_{\mathbf{a}_0\mathbf{a}_1}\mathbf{W}_2^T\big) \qquad (28) \\ &= \mathrm{trace}\big(\mathbf{W}_2\left(\mathbf{C}_{\mathbf{a}_1} - \mathbf{C}_{\mathbf{a}_1\mathbf{a}_0}\mathbf{C}_{\mathbf{a}_0}^{-1}\mathbf{C}_{\mathbf{a}_0\mathbf{a}_1}\right)\mathbf{W}_2^T\big). \qquad (29) \end{aligned}$$

Note that this expression requires invertibility of $\mathbf{C}_{\mathbf{a}_1}$.

REFERENCES

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

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 1–9.

[3] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," *arXiv preprint: 1811.03962*, 2018.

[4] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv preprint: 1611.03530*, Nov. 2016.

[5] S. Arora, N. Cohen, and E. Hazan, "On the optimization of deep networks: Implicit acceleration by overparameterization," *arXiv preprint: 1802.06509*, Feb. 2018.

[6] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2014, pp. 855–863.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint: 1704.04861*, Apr. 2017.

[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018, pp. 4510–4520.

[9] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2013, pp. 2148–2156.

[10] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2014, pp. 2654–2662.

[11] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang, "Stronger generalization bounds for deep nets via a compression approach," *arXiv preprint: 1802.05296*, Feb. 2018.

[12] J. J. Bussgang, "Crosscorrelation functions of amplitude-distorted Gaussian signals," Technical Report, M.I.T., Cambridge, MA, Tech. Rep. 216, Mar. 1952.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2012, pp. 1097–1105.

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, May 2010, pp. 249–256.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Visionon (ICCV)*, June 2015, pp. 1026–1034.

[16] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint: 1511.06422*, Nov. 2015.

[17] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint: 1312.6120*, Dec. 2013.

[18] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint: 1503.02531*, Mar. 2015.

[19] S. Guo, J. M. Alvarez, and M. Salzmann, "ExpandNet: Training compact networks by linear expansion," *arXiv preprint: 1811.10495v3*, May 2019.

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv preprint: 1510.00149*, Oct. 2015.

[21] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2016, pp. 1379–1387.

[22] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, Dec. 2015, pp. 1135–1143.

[23] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 1993, pp. 164–171.

[24] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 1989, pp. 177–185.

[25] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 1990, pp. 598–605.

[26] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint: 1412.6115*, Dec. 2014.

[27] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2015, pp. 3123–3131.

[28] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning (ICML)*, July 2015, pp. 1737–1746.

[29] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 806–814.

[30] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2014, pp. 1269–1277.

[31] R. Ghods, A. S. Lan, T. Goldstein, and C. Studer, "MSE-optimal neural network initialization via layer fusion," *arXiv preprint: 2001.10509*, 2020.

[32] R. Vershynin, "How close is the sample covariance matrix to the actual covariance matrix?" *Journal of Theoretical Probability*, vol. 25, no. 3, pp. 655–686, Sep. 2012.

[33] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Technical Repot, University of Toronto, Tech. Rep., Apr. 2009.

[34] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint: 1708.07747*, Aug. 2017.

[35] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. CIACO, Apr. 2013, pp. 437–442.

[36] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint: 1804.03209*, Apr. 2018.

[37] C. Studer, S. Medjkouh, E. Gönültaş, T. Goldstein, and O. Tirkkonen, "Channel charting: Locating users within the radio environment using channel state information," *IEEE Access*, vol. 6, pp. 47 682–47 698, Aug. 2018.