
Lulu Zhang¹, Wei Cai² and Zhi-Qin John Xu¹,*

¹ Institute of Natural Sciences and School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai, 200240, China.
² Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA.

Received 20 May 2023; Accepted 27 May 2023

Abstract. This note provides a correction of a missing weight constant in the MscaleDNN formula and some comments on the performance of the corrected algorithm.

AMS subject classifications: 35Q68, 65N99, 68T07
Key words: Multi-scale DNN, PDE solver, deep learning.

1 Correction on a missing weight constant

Our previous paper on the multi-scale deep neural network (MscaleDNN) in [2] contains an error: a constant \( \alpha_i^d \) or its inverse is missing outside \( f_i(\cdot) \) or \( f_{\theta_i}(\cdot) \) in Eqs. (2.10), (2.11), (2.14) and (2.15). The following text should replace the corresponding paragraphs in [2] to correct this error.

From (2.5), we can apply a simple down-scaling to convert the high frequency region \( A_i \) to a low frequency region. Namely, we define a scaled version of \( f_i(k) \) as

\[
\hat{f}_i^{(\text{scale})}(k) = \hat{f}_i(\alpha_i k), \quad \alpha_i > 1,
\]  

(2.9)

and, correspondingly in the physical space

\[
f_i^{(\text{scale})}(x) = f_i\left(\frac{1}{\alpha_i}x\right) \frac{1}{\alpha_i^d},
\]  

(2.10)

*Corresponding author. Email addresses: zhangl9661@sjtu.edu.cn (L. Zhang), cai@smu.edu (W. Cai), xuzhiqin@sjtu.edu.cn (Z.-Q. J. Xu)
or
\[ f_i(x) = \alpha_i^d f_i^{(\text{scale})}(\alpha_i x). \] (2.11)

We can see the low frequency spectrum of the scaled function \( f_i^{(\text{scale})}(k) \) if \( \alpha_i \) is chosen large enough, i.e.,
\[ \text{supp} f_i^{(\text{scale})}(k) \subset \left\{ k \in \mathbb{R}^d \, | \, \frac{(i-1)K_0}{\alpha_i} \leq |k| \leq \frac{iK_0}{\alpha_i} \right\}. \] (2.12)

Using the F-Principle of common DNNs (Ref. [27] in [2]), with \( iK_0/\alpha_i \) being small, we can train a DNN \( f_{\theta_i^n}(x) \), with \( \theta_i^n \) denoting the DNN parameters, to learn \( f_i^{(\text{scale})}(x) \) quickly
\[ f_i^{(\text{scale})}(x) \sim f_{\theta_i^n}(x), \] (2.13)
which gives an approximation to \( f_i(x) \) immediately
\[ f_i(x) \sim \alpha_i^d f_{\theta_i^n}(\alpha_i x) \] (2.14)
and to \( f(x) \) as well
\[ f(x) \sim \sum_{i=1}^M \alpha_i^d f_{\theta_i^n}(\alpha_i x). \] (2.15)

### 3 Numerical results with the corrected MscaleDNN (2.15)

In this section, we present several numerical tests on approximation and solving PDEs to demonstrate the necessity of the missing factor \( \alpha_i \) in front of the subnetworks \( f_{\theta_i^n}(\cdot) \) in (2.15), which results in faster training and lower generalization errors, as shown in Fig. 1 and later sections.

Three networks will be tested: FNN – fully connected neural network; MscaleDNN – the one missing the \( \alpha_i \) weights; MscaleDNN-corrected – the corrected one with weight factor \( \alpha_i \) included. In the comparison tests, we use the same compact activation functions in [2],
\[ \phi(x) = \text{ReLU}(x)^2 - 3\text{ReLU}(x-1)^2 + 3\text{ReLU}(x-2)^2 - \text{ReLU}(x-3)^2. \] (3.1)

#### 3.1 Approximation of a 2-D oscillatory function

The target function for the fitting problem is
\[ u(x,y) = \frac{1}{N^2} \sum_{n=1}^N \sum_{m=1}^N e^{(2\pi m x)} e^{(2\pi n y)}, \]
\( (x,y) \in [-1,1]^2 \), where \( N = 20 \). 5000 training data at each epoch are randomly sampled from \([-1,1]^2\). DNNs are trained by the Adam optimizer with a learning rate 0.0001 and initialized with a Glorot-normal. We compare the following different network structures: (1) FNN with a size 2-1600-1600-1600-1; (2) MscaleDNN with eight subnetworks with a size 2-200-200-200-1 each and scales \( \{1,2,4,8,16,32,64,128\} \); (3) MscaleDNN-corrected with eight subnetworks with a size 2-200-200-200-1 each and same scales as MscaleDNN.

In Fig. 2, we show the target function and the DNN solutions on fixed \( x = -0.6 \) and \( y = 0.2 \). The MscaleDNN-corrected performs better than the MscaleDNN and FNN.
3.2 Solving a 2-D Poisson-Boltzmann equation with DeepRitz

In this section, we use a variational loss, i.e., Ritz loss \cite{1}, as the training loss. We consider a 2-D Poisson-Boltzmann equation in \( \Omega = [-1,1]^2 \),

\[-\Delta u(x,y) + \lambda^2 u(x,y) = f(x,y), \tag{3.2}\]

where the exact solution is given by \( u(x,y) = \frac{1}{N^2} \sum_{m=1}^{N} \sum_{n=1}^{N} \frac{m^2+n^2}{1+m^2+n^2} e^{2\pi m x} e^{2\pi n y}. \)
Here, we set $N = 20$, $\lambda = 10$. In each training epoch, we uniformly sample 5000 points inside the domain. We choose the penalty coefficient for the boundary term $\beta = 5000$. DNNs are trained by the Adam optimizer with a learning rate $1e^{-5}$ and initialized with a Glorot-normal. We examine the following different network structures: (1) FNN with a size 2-1600-1600-1600-1; (2) MscaleDNN with eight subnetworks with a size 2-200-200-200-1 each and scales $\{1,2,4,8,16,32,64,128\}$; (3) MscaleDNN-corrected with eight subnetworks with a size 2-200-200-200-1 each and same scales as MscaleDNN. In Fig. 3, we show the exact solution and DNN solutions on fixed $x = 0.33$ and $y = -0.47$. The MscaleDNN-corrected performs better than the MscaleDNN and FNN.

### 3.3 Solving a 2-D Poisson equation with DeepRitz

Lastly, we use the variational loss for the training loss for a 2-D Poisson equation in $\Omega = [-1,1]^2$, i.e., $\lambda = 0$ in (3.2), where the exact solution is $u(x,y) = \sum_{m=1}^{N} \sum_{n=1}^{N} e^{2\sin(\pi mx)} e^{2\cos(\pi ny)}$.

Still we choose $N = 20$. In each training epoch, we uniformly sample 5000 points inside the domain. We choose the penalty coefficient for the boundary term $\beta = 1000$. DNNs are trained by the Adam optimizer with a learning rate $1e^{-5}$ and initialized with a Glorot-normal. We examine the following different network structures: (1) FNN with a size 2-3200-3200-3200-1; (2) MscaleDNN with eight subnetworks with a size 2-400-400-400-1.
Figure 4: Different network structures in solving a 2-D Poisson equation with a Ritz loss. Comparison between analytic solution and DNN solutions at $x = 0.33$ and $y = -0.47$.

each and scales $\{1, 2, 4, 8, 16, 32, 64, 128\}$; (3) MscaleDNN-corrected with eight subnetworks with a size 2-400-400-400-1 each and same scales as MscaleDNN.

In Fig. 4, we show the exact solution and DNN solutions on fixed $x = 0.33$ and $y = -0.47$. The MscaleDNN-corrected performs better than the MscaleDNN and FNN.

4 Conclusion

The weight factors $\alpha^d_i$ in front of sub-networks $f_{\theta_i}(\alpha; x)$ in (2.15) ensure better learning of higher frequency components. The weight factors are equivalent to sampling the network parameters in the outermost layer of the sub-networks in a larger range. Numerical results confirm a noticeable improvement of the performance with the corrected MscaleDNN.

References
