BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 04/06 pp196–207
Volume 3, Number 3, September 2020
DOI: 10.26599/BDMA.2020.9020004

Gradient Amplification: An Efficient Way to Train Deep Neural Networks

Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan*

Abstract: Improving performance of deep learning models and reducing their training times are ongoing challenges in deep neural networks. There are several approaches proposed to address these challenges, one of which is to increase the depth of the neural networks. Such deeper networks not only increase training times, but also suffer from vanishing gradients problem while training. In this work, we propose gradient amplification approach for training deep learning models to prevent vanishing gradients and also develop a training strategy to enable or disable gradient amplification method across several epochs with different learning rates. We perform experiments on VGG-19 and Resnet models (Resnet-18 and Resnet-34) , and study the impact of amplification parameters on these models in detail. Our proposed approach improves performance of these deep learning models even at higher learning rates, thereby allowing these models to achieve higher performance with reduced training time.

Key words: deep learning; gradient amplification; learning rate; backpropagation; vanishing gradients

1 Introduction

Deep learning models have achieved state-of-the-art performances in several areas including computer vision[1], automatic speech recognition[2], natural language processing[3], and beyond[4–8]. These models are designed, trained, and tuned to achieve better performance for a given dataset. Their performance increases further with the increase in the depth of the network[9]. The major challenge associated with the increase in the network architecture is the high amount of time required to train the model even on parallel computation resources and also vanishing gradients[9]. Training deep neural networks is time-consuming, which could take days or sometimes weeks depending on the type of the model architecture and size of the dataset. One way to speed up the training process is to increase the learning rate. This will accelerate the training process by quickly converging to optima, but also has the risk of missing the global optima resulting in sub-optimal solutions or sometimes non-convergence[10]. Lower learning rate does not have such a risk and can converge to optima, but increases training speeds. In general, training process with a learning rate scheduler begins with higher learning rates for a few epochs, followed by reduction of learning rates for the next couple of epochs, which is repeated until the desired optima or model performance is achieved. One way to improve the training speed of deep learning models can be to determine ways to achieve optimal model parameters at larger learning rates.

The other important area of research in deep learning models is to prevent vanishing gradient problem[11–13]. The vanishing gradient problem occurs during training of artificial neural networks, specifically during backpropagation. There are several approaches to avoid this problem. One suggested early method was to perform a two-step training process, which

• Sunitha Basodi, Chunyan Ji, and Yi Pan are with the Department of Computer Science, Georgia State University, Atlanta, GA 30302, USA. E-mail: sbasodi1@student.gsu.edu; cji2@student.gsu.edu; yipan@gsu.edu.
• Haiping Zhang is with the Center for High Performance Computing, Joint Engineering Research Center for Health Big Data Intelligent Analysis Technology, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China. E-mail: hp.zhang@siat.ac.cn.
* To whom correspondence should be addressed.
Manuscript received: 2020-04-01; accepted: 2020-04-16

involved network weight initialization followed by fine-tuning using backpropagation method[14]. The other simpler methods that prevent this problem are Rectified Linear Unit (ReLU) activation function[15,16] and Batch Normalization (BN)[17]. Since ReLU activation saturates inputs in only one direction, it has less impact of vanishing gradients. The other recent approach of batch normalization not only improves the performance of the model, but also reduces vanishing gradient problems. Resnet architectures have residual connections, which also overcome vanishing gradient problem to some extent[1]. Lately, due to the improvement of hardware along with the computational abilities of Graphical Processing Units (GPUs), neural networks can be trained without the issue of vanishing gradients.

In this work, we propose a novel gradient amplification approach along with a training strategy which addresses the challenges discussed above. In this method, gradients are dynamically increased for some layers during backpropagation so that significant gradient values are propagated to the initial layers. This process is repeated for a few epochs along with the normal training process with no gradient amplification for the other epochs. When neural networks are trained using this method, we observe that the testing/training accuracies of the models improve and achieve higher accuracies faster, even at higher learning rates, therefore reduce the training time of these deep learning models.

Our contributions include the following:

– We propose a novel way of amplifying gradients during backpropagation for effective training of deep neural networks.

– We suggest a unique training strategy which includes amplification during certain epochs along with normal training with no amplification.

– We perform comprehensive experiments to understand the impact of different parameters used in amplification.

– We perform step-wise analysis of training strategy demonstrating the best strategy with different learning rates.

The remainder of this paper is organized as follows. Related works are briefly described in Section 2. Our proposed approach is presented in Section 3. Experimental setup, results, and their comparisons are covered in Section 4, followed by conclusions in Section 5.

2 Background

In addition to deep learning models, there have been efforts to analyze large datasets using clustering techniques on cloud architectures[18,19]. In this section, we briefly discuss the existing approaches to address vanishing gradient problem and study the impact of learning rates on the training time of deep learning models.

2.1 Vanishing gradient

Vanishing gradient problem[11–13] occurs while training artificial neural networks during backpropagation and can become significant with the increase of depth of the network. In gradient-based learning methods, during backpropagation, network weights are updated proportional to the gradient value (partial derivative of the cost function with respect to the current weights) after each training iteration (epoch). Depending on the type of the activation functions and network architectures, sometimes the gradient value is too small and gets gradually diminished during backpropagation to the initial layers. This prevents the network from updating its weights and also sometimes when the value is too small, the network may be completely stopped from training (updating weights). Though there is no fundamental solution to this problem, some of the approaches help to avoid it[20]. One such approach consists of performing a two-step training process. In the first step, network weights are trained using unsupervised learning methods (such as auto-encoding) and then the weights are fine-tuned using backpropagation method[14]. Other simpler methods that prevent this problem are ReLU activation function[15,16], BN[17] and Resnet networks[1]. ReLU activation zeros the negative values and only considers positive values. As it saturates inputs in only one direction, it has less impact on vanishing gradients. The other approach, batch normalization, also reduces vanishing gradient problems other than boosting the performance of the model. In batch normalization, during every training iteration, the input data are normalized to reduce its variance, so that the data do not have large bounds. Since inputs are normalized, gradients are also regulated[17]. Resnet network architectures have residual networks and residual connections which help to improve on this problem. In addition to these approaches, recent advancement in the hardware has also played a crucial role in solving this issue. Increased computational

abilities and availability of GPUs aid in reducing this problem.

2.2 Learning rate

Learning rate is one of the most important hyperparameters, which controls the performance of deep neural networks. Having higher learning rates cause the model to train faster, but might have sub-optimal solutions. However, lower learning rates take longer time to train the model, but can achieve better optimal solutions[10]. There are several approaches designed to take advantage of them. One such method is learning rate scheduler, where we start with higher learning rates and gradually lower the rates with training epochs[21]. There are several ways in which such a scheduler can be designed, namely, directly assigning the learning rates to the epochs, gradually decaying the learning rate based on the current learning rate, current epoch, and total number of epochs (time-based decay); reducing the learning rate in a step-wise manner after a certain number of epochs (step decay); and exponentially decaying the learning rate based on the initial learning rate and the current epoch (exponential decay). Another approach includes adapting learning rate dynamically based on the performance of the optimization algorithm without need of any scheduling, some of such methods include Adagrad[22], Adadelta[23], RMSprop[24], and Adam[25]. Reference [26] summarized all the above discussed methods in detail. Reference [27] proposed a method to automatically tune the learning rate based on the local gradient variations of the data samples, which has similar performance to other adaptive learning rate methods. Reference [28] showed that models can achieve similar test performance without decaying the learning rate but by increasing the batch size instead. This method not only has fewer parameter updates, but also increases parallelism thereby reducing training times.

In the next section, we discuss our proposed method and training strategy with a fixed learning rate schedule across epochs, which achieves better accuracies even at the higher values of learning rates.

3 Proposed Method

Our proposed approach is to dynamically amplify (increase) the value of the gradients for a selection of layers during backpropagation. This ensures that the gradient values are not diminished while updating weights for the initial network layers and a significant value of the gradients is available during backpropagation even for deep neural networks with large number of layers. Architectures of neural networks have evolved over the years and there are many different layers where such an amplification can be done. The layers on which gradient amplification can be performed during backpropagation are arranged into a group, say G. To determine this group, firstly, the type of layers that needs to be included for gradient amplification should be identified. Each of the layers, such as convolution layers, batch normalization layers, pooling layers, activation function layers, and so on, can be chosen to be included in the group. The type of the layer considered plays a crucial role in the performance of the model. Gradient amplification is done on a subset of the layers from this group G, which we refer as amp layers in the rest of the paper. Selection of the amp layers from a group of layers can be done in various methods. In this work, we determine the amp layers by random selection. To identify which subset size has better performance, we choose a parameter β representing the ratio of amp layers to be selected from all the layers in the group G. Gradients are amplified when they pass through these randomly selected layers during backpropagation. During amplification, value of gradients is increased at run time by multiplying the actual gradient values by a factor Γ. The value of Γ is important as it should not be too small or too large. If the value of Γ is too small, then the increase might not be effective, and if it is too large, it might overfit the data or cause incorrect weight updates. During training, we perform gradient amplification for some epochs and with no gradient amplification for other epochs. Algorithm 1 describes the training process with gradient amplification and Algorithm 2 describes the steps for the selection of layers from G.

4 Experiment & Result

4.1 Setup

Our experiments are performed on CIFAR 10 dataset which consists of 60 000 colored images of 10 classes with 6000 images per class and each image has 32×32 resolution. We implement our algorithms using python and pytorch[29] libraries. In our experiments, we employ several standard deep learning models and train them for 150 epochs. The number of epochs, combination of number of epochs, and learning rates can be chosen as one thinks best. In this work, the first 100 epochs have learning rate of 0.1 and the next 50 epochs have the learning rate of 0.01 (as shown in Fig. 1). The first

Algorithm 1 Training process with gradient amplification

Input: M, params=[$(e_1, \eta_1, \beta_1, \Gamma_1)$, $(e_2, \eta_2, \beta_2, \Gamma_2)$, ...]
Variables:
Γ is gradient amplification factor;
β is ratio of layers to be selected for amplification;
η is the learning rate;
amp the set of layers selected to perform amplification;
M is the neural network model;
params is an array of elements, each in the format (end_epoch, η, β, Γ).

```
start_epoch=0
for (e_i, η_i, β_i, Γ_i) in params do
    update learning rate to η_i
    optimizer=sdg_optimizer(η_i)
    if (β_i > 0) then
        amp = GetGradientAmpLayers(M, β)
    end if
    for k = start_epoch to e_i do
        train the model M
        if (β_i > 0) then
            multiply gradients with Γ_i for layers in amp during
            backpropagation
        else
            perform regular backpropagation without gradient
            amplification
        end if
    end for
    start_epoch=e_i
    reset amp
    evaluate model M with a testing set
end for
return
```

Algorithm 2 Determination of amp layers

Input: M, β
β is the ratio of layers to be selected for amplification;
G is a set consisting of a group of all layers that can be used for gradient amplification;
layer_types= Set indicating the type of layers to be used for amplification.

```
Function GetGradientAmpLayers(M, β)
for all layer in layer_types do
    include layer in G
end for
amp_size = β*size_of(G)
amp = RandomSelect(G, amp_size)
EndFunction
```

50 epochs are trained with learning rate of 0.1 without gradient amplification. This is because for the first few epochs, the model is considered to be in transient phase and the network parameters undergo significant changes. This initial transient can be considered for any number of epochs and in this work, we set it to

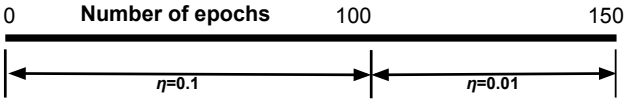
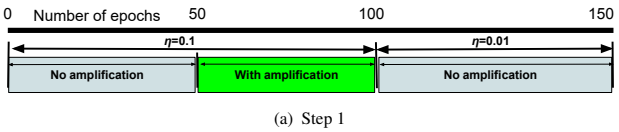


Fig. 1 Experiment setting showing the number of epochs and learning rates corresponding to epochs for training all the models.

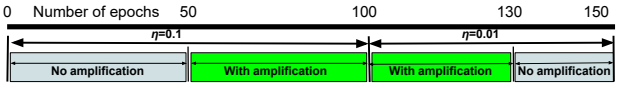
50 epochs. The next 50 epochs have the same learning rate of 0.1 but has gradient amplification applied during backpropagation while training the model (as shown in Fig. 2a). After identifying the best params with gradient amplification for epochs 51–100, using those params for those epochs, we extend amplification for epochs 101–130 to identify the best params and with no amplification for epochs 131–150, as shown Fig. 2b. There are mainly three important parameters while applying gradient amplification method, namely, the type of the layers to be employed for amplification, the ratio of layers (β) to be chosen from selected layers to perform amplification, and gradient amplification factor. The effects of varying each of these parameters are explained in detail in the subsections below. We run our experiments on Resnet and VGG models with different architectures.

Here we perform three phase analysis while evaluating our model.

Phase 1 In this phase, we choose the type of layers to be considered for amplification. There are several types of layers at which amplification can be applied such as activation function layers, pooling layers, batch normalization layers, and convolution layers. Convolution layers apply kernel functions and



(a) Step 1



(b) Step 2

Fig. 2 Two-step training process carried out during performance analysis of deep learning models. Experiments are firstly executed on the models with training steps shown in Step 1. For Step 2, ratio parameters for gradient amplification which have better performance of the models in Step 1 are considered as the parameters for epochs 51–100 and experiments are performed by analyzing ratio parameters for epochs 101–130, with no amplification from epochs 131–150. These settings show the number of epochs and the learning rates corresponding to these epochs while training these models.

extract important features from the data and pooling layers perform accumulation of features over a grid using several strategies, such as retrieving maximum values, minimum values, averaging, fractional pooling, and so on. Since the network parameter tuning while training can be sensitive to these values, in this work, we do not perform amplification on these layers. Batch normalization layers normalize data over a batch of inputs, and activation function layers transform data non-linearly before forwarding it to the succeeding layers. In our work, we perform gradient amplification on batch normalization and activation function layers. ReLU is the activation function used in Resnet and VGG models. From these two types of layers, either one or both of them can be considered for amplification. Once the type of the layers is selected, we now tag all the layers of the selected type to belong to the group G. We now move to the next phase to determine the final amplification layers amp.

Phase 2 Once the set of layers G is determined, the next task is to find the subset of layers, which gives better performance. It requires identifying subset size and selection of those many layers from G. Since the size is unknown, experiments are performed by selecting the size to be a ratio of size of G. This ratio, β, is chosen from the set $\beta \in \{0, 0.1, 0.2, \ldots, 0.9, 1\}$. The actual size of amp is determined by the value $\beta \times \text{size\_of}(G)$. When the value is 0, no layers are chosen and gradient amplification is not performed. When the value is 1, then all the layers in G are considered for amplification. 0 is included to verify whether the model performs better without gradient amplification or vice versa. Random selection is employed to select amp subset of layers from G. We perform experiments with all these sizes and select the model with the best performance.

Phase 3 In this phase, the layers amp on which gradient amplification can be applied is known. The only parameter left to explore is Γ, the factor with which gradient needs to be amplified. To reduce computation complexity in testing all the combinations of parameter values amp, β, and Γ, firstly, experiments are performed on all combinations of amp and β, i.e., until Phase 2, then the best models are chosen from Phase 2 and analyzed by varying Γ. The value of Γ is firstly varied from $\{1, 2, 3, \ldots, 10\}$ to analyze the impact of amplification and then fine-tuned by varying from $\{1.1, 1.2, \ldots, 2.9, 3.0\}$ to determine the value that works best during training.

4.2 Result

In our experiments, we employ Resnet-18, Resnet-34, and VGG-19 models and perform thorough analysis. As the complexity of the model and the depth of the network increase, it takes longer to compute and requires more GPU/Control Processing Unit (CPU) resources. Since we perform many experiments (around hundreds), having models with relatively simpler architectures and less layers would make the computation time faster. Most of our experiments are performed on High Performance Computing (HPC) cluster in Georgia State University (GSU).

While performing experiments, we choose either batch normalization layers or ReLU layers or both and then verify their performance over multiple epochs. We firstly explain the training params, which is important to understand the performance tables. We train our models for 150 epochs and the learning rate of the first 100 epochs is 0.1 and the next 50 is 0.01. We train the models with no gradient amplification for the first 50 epochs as the initial transient and for the next epochs. We aim to identify the pattern to select the epochs, which improve the overall performance of the model. We follow the training steps mentioned in Algorithm 1 and params $= [(e_1, \eta_1, \beta_1, \Gamma_1), (e_2, \eta_2, \beta_2, \Gamma_2), (e_3, \eta_3, \beta_3, \Gamma_3), (e_4, \eta_4, \beta_4, \Gamma_4), \ldots]$ is chosen as $[(50, 0.1, 0, 1), (100, 0.1, 0, 1), (130, 0.01, 0, 1), (150, 0.01, 0, 1)]$ when no gradient amplification is performed. The values in each element represent end epoch, learning rate, ratio of amplified layers, and gradient amplification factor, respectively. For instance, $(50, 0.1, 0, 1)$ means that the model is trained with learning rate 0.1 until we reach 50 epochs, during which 0 layer is selected for gradient amplification and amplification factor is 1.

Performance of original models with no gradient amplification is firstly recorded. Next, models with gradient amplification are experimented in two steps. We firstly set params as [(50, 0.1, 0, 1), (100, 0.1, xx, 2), (130, 0.01, 0, 1), (150, 0.01, 0, 1)]. That is, no gradient amplification is applied for the first and the last 50 epochs, as shown in Fig. 2a. For epochs 51–100, the ratio of selected layers is scanned from $\{0, 0.1, 0.2, \ldots, 1\}$ to identify the best model with the amplification factor 2. For simplicity, we define $S1\_\{mm\}$ to represent the modified ratio mm during epochs 51–100 in Step 1 while performing amplification, and $S2\_\{mm\}\_\{nn\}$ to represent the modified ratio mm during epochs 51–100

and nn during epochs 101–130, respectively, during amplification in Step 2. So, the params defined above will be represented as $S1$_xx, where xx represents the value that is varied. Once we identify the best ratio for 51–100 epochs, say 0.7, we then run the experiments with different ratio values for the next 30 epochs by setting params to be $S2$_0.7_xx, i.e., $[(50, 0.1, 0, 1), (100, 0.1, 0.7, 2), (130, 0.01, xx, 2), (150, 0.01, 0, 1)]$ as shown in Fig. 2b. Note that, the learning rate is decreased to 0.01 after 100 epochs. After these experiments, the best models are chosen to perform experiments in Phase 3 to analyze the impact of gradient amplification factor on its performance. All the phases and various experiments performed are shown in Fig. 3.

From our initial experiments, we observe that the ratio values $\{0.1, 0.3, 0.5, 0.6\}$ on average provide better results in Step 1, explained below in detail in Phase 2. Instead of running Step 2 only on the best models from Step 1, different models are built with ratio values $\{0.1, 0.3, 0.5, 0.6\}$ for epochs 51–100 where the learning rate is 0.1. We perform our analysis on Phases 1 and 2 for the following amplification params in Step 2 (see Fig. 2b):

- $S2$_0.1_xx: $[(100, 0.1, 0.1, 2), (130, 0.01, xx, 2)]$
- $S2$_0.3_xx: $[(100, 0.1, 0.3, 2), (130, 0.01, xx, 2)]$
- $S2$_0.5_xx: $[(100, 0.1, 0.5, 2), (130, 0.01, xx, 2)]$
- $S2$_0.6_xx: $[(100, 0.1, 0.6, 2), (130, 0.01, xx, 2)]$

4.2.1 Phase 1: Effect of type of layers

In this work, ReLU, BN, or both (ReLU+BN) are the layers used for gradient amplification. We run original models without gradient amplification 5 times and record their training and testing accuracies and compare the corresponding gradient amplified models with the mean of the these accuracies across 5 runs. For each type(s) of layer chosen, experiments are run for params $S2$_0.1_xx, $S2$_0.3_xx, $S2$_0.5_xx, and $S2$_0.6_xx, that is, for each ratio value in $\beta \in \{0.1, 0.3, 0.5, 0.6\}$ during epochs 51–100 ($\eta = 0.1$), we build models by varying ratio values from $\{0, 0.1, \ldots, 1.0\}$ for epochs 101–130 ($\eta = 0.01$), without amplification from 131–150 (see Fig. 2b). The best training and testing accuracies of these models are compared with the average training and testing accuracies of corresponding original model. Since training accuracies of the original models are close to 100%, we emphasize our comparison on testing accuracies.

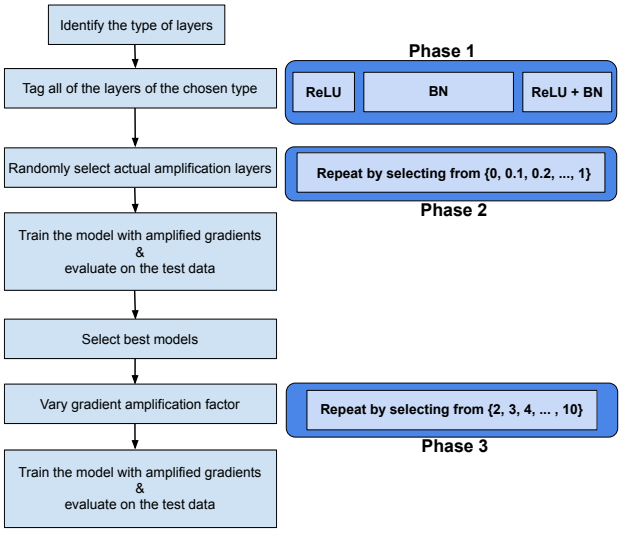


Fig. 3 Overview of all the experiments performed by varying different parameters of gradient amplification.

In VGG-19 model, we perform analysis considering ReLU, BN, or both layers for gradient amplification and provide accuracy improvements for params $S2$_0.1_xx, $S2$_0.3_xx, $S2$_0.5_xx, and $S2$_0.6_xx, respectively. When only ReLU layers are chosen, testing accuracies improve around 1.98%, 1.31%, 1.08%, and 1.25%, respectively for above params. In the case of amplification applied only to BN layers, an improvement of 2.27%, 1.64%, 0.91%, and 0.96% in testing accuracies is observed. When both ReLU and BN are chosen, models have accuracy difference of 0.9%, 0.64%, 0.13%, and 0.3%, respectively. When both layers are used in amplification, the improvements across different models are less than 1%, which become better when only either ReLU or BN is used. Best improvements are seen when amplification is applied on only BN layers.

Resnet models are made of residual blocks, each of which consists of two convolutional units and therefore each block has two ReLU and BN layers. In these models, other than experimenting with all ReLU and BN layers, we additionally perform experiments considering only one of the BN layers from residual blocks. When all the BN layers are considered for amplification in Resnet-18 models, there is an improvement of 1.66%, 1.35%, 0.53%, and 0.34% respectively, for params $S2$_0.1_xx, $S2$_0.3_xx, $S2$_0.5_xx, and $S2$_0.6_xx. When only ReLU layers are used, there is a difference of 1.76%, 1.32%, 0.77%, and 0.26%, respectively. When both BN and ReLU are used, there is an initial improvement of 01.14%, 0.03%, for params $S2$_0.1_xx and $S2$_0.3_xx, but the performance drops for params $S2$_0.5_xx and $S2$_0.6_xx. When only one of the BN layer from a residual block is considered, there is an improvement

of 1.98%, 2.08%, 1.64%, and 1.32% for respective params. When both BN and ReLU are used for amplification, there is an improvement only for params $S2\_0.1\_xx$, which either declines or slightly changes for the remaining params. When either ReLU or BN layers are considered, for params $S2\_0.1\_xx$ and $S2\_0.3\_xx$, performance improvement of more than 1.3% can be observed and for params $S2\_0.5\_xx$ and $S2\_0.6\_xx$, improvements are less than 1%. When one of the BN layers in residual blocks are considered, then the models have accuracy improvements of more than 1.3% for all params and it also achieves the best testing accuracy of 94.57% with an improvement of 2.08% over original model.

Similarly for Resnet-34, in the case of only BN layers, there is an accuracy gain of 1.21% and 0.64% for params $S2\_0.1\_xx$ and $S2\_0.3\_xx$, and then slightly decreases for params $S2\_0.5\_xx$ and $S2\_0.6\_xx$, respectively. When only ReLU layers are considered, there is an improvement of 1.61% and 0.42% for $S2\_0.1\_xx$ and $S2\_0.3\_xx$, and for other params, there is a slight decrease (less than 0.8%) for params $S2\_0.5\_xx$ and $S2\_0.6\_xx$. When both BN and ReLU are used, there is an initial improvement of 1.14% for params $S2\_0.1\_xx$ and then it declines for other params. As we include both BN and ReLU layers, Resnet-34 model performance decreases quickly with the increase in the ratio of amplified layers. When only one of the BN layers is used in a residual block, an improvement of 1.67%, 1.23%, 1.55%, and 1.49% can be seen for respective params. When all the BN layers are only used for amplification, there is an improvement of more than 1% only for params $S2\_0.1\_xx$ and the performance either declines or slightly changes for remaining params. Similar pattern is observed when only ReLU or both ReLU and BN are used for amplification. When one of the BN layers in residual blocks is considered, models have accuracy improvements of more than 1.2% for all params and it also achieves the best testing accuracy of 94.39% with an improvement of 1.67% over original model.

Our experiments show that for VGG-19 models, selecting ReLU improves the performance of the models, but achieves best performance when BN layers are chosen for amplification. In Resnet-18 and Resnet-34, performance of the models improves when BN layers are chosen for amplification and best performance is achieved when one of the BN layers from a residual block is selected for amplification.

4.2.2 Phase 2: Effect of ratio of selected layers β

Here, we discuss the impact of the ratio of selected layers on each of the above types. In our training strategy, gradient amplification is firstly applied in Step 1 (as shown in Fig. 2a) to determine the best performing ratio values for epochs 51–100. The best training and testing accuracies after gradient amplification across all the ratio values are compared with the original baseline models to analyze the overall effectiveness. In VGG-19, for all layer types, as the ratio of amplified layers increases, the performance of the model diminishes compared to original models. The training accuracies decrease at an increased rate compared to testing accuracies. When gradient amplification is performed, training and testing accuracies decrease slightly by -0.3% and -0.14% (for BN only) and increase slightly by 0.02% and 0.03% (in the case of ReLU+BN) and show an improvement of 0.65% and 0.77% (for ReLU only), respectively.

In Resnet-18 and Resnet-34 models, as the ratio of amplified layers increases, training and testing accuracies remain close to the accuracies of the baseline models when only one of the BN layers in a residual block is considered. When either BN or ReLU is considered, as the ratio of amplified layers increases, performance of the models decreases slightly compared to original models. When both BN and ReLU are considered for amplification, as the ratio of layers increases, performance of the models decreases significantly compared to respective baseline models. In Resnet-18, the best training and testing accuracies after gradient amplification across all the ratio values, show an improvement by 0.69% and 0.62% (for BN only), 0.52% and 0.67% (for ReLU only), 0.52% and 0.67% (in the case of ReLU+BN), and 0.79% and 0.92% (in the case when one of BN layers from residual block), respectively. In Resnet-34, the best training and testing accuracies after gradient amplification across all the ratio values show an improvement by 0.54% and 0.68% (for BN only), 0.4% and 0.15% (for ReLU only), 0.42% and 0.81% (in the case of ReLU+BN), and 0.57% and 0.85% (when one of the BN layers from residual blocks are considered), respectively.

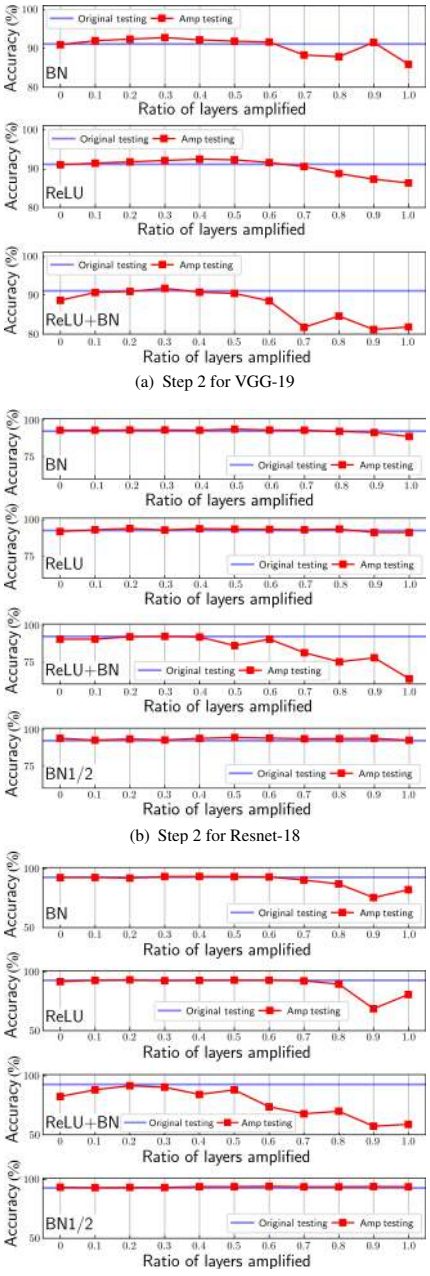
In the case of Step 1, amplification is applied only for epochs 51–100 ($\eta = 0.1$). We also perform experiments by applying amplification for epochs 101–150 ($\eta = 0.01$), by considering all or some of the epochs. We observe that the models perform better when amplification is applied from 51–100 epochs

($\eta = 0.1$) followed by 101–130 epochs ($\eta = 0.01$) as shown in Step 2 (Fig. 2b). To narrow the parameter space, we only consider ratio values for epochs 51–100 where the models perform better. From analysis of model performances in Step 1, ratio values $\{0.1, 0.3, 0.5, 0.6\}$ on average provide better results and therefore, these ratios are used for epochs 51–100 ($\eta = 0.1$) as mentioned earlier and ratio values are varied for 101–130 epochs ($\eta = 0.01$), namely $S2\_0.1\_xx$, $S2\_0.3\_xx$, $S2\_0.5\_xx$, and $S2\_0.6\_xx$. Figure 4 shows the performance of VGG-19, Resnet-18, and Resnet-34 models respectively for these params when different layers are amplified. Performance improvements of these models are discussed in Phase1 in detail. Here we emphasize on the effect on the models as the ratio of amplified layers increases. For VGG-19 models, as the ratio of layers increases, there is an increase in performance initially and then it decreases when the ratios are above 0.7. When both ReLU+BN are amplified, the models have significant decrease with the increase of ratio values. In the case of Resnet-18, models have improved or similar performance even as the ratio increases except when both ReLU+BN are amplified, in which case it decreases. For Resnet-34, models have improved or similar performance even as the ratio increases until 0.8, after which it decreases. But when both ReLU+BN are amplified, the performance declines even for smaller ratios.

When amplification is applied using approach in Step 1 (Fig. 2a), the models perform better when only ReLU is amplified in the case of VGG-19, and Resnet-18 and Resnet-34 models perform best when only one of the BN layers from a residual block is used for amplification. When amplification is done as in Step 2, all models achieve higher accuracies than baseline models for most of the ratio values except when ReLU+BN are amplified, in which case only some of the smaller ratio values have better models. This shows that a small ratio of amplified layers are sufficient to improve the performance of original models.

4.2.3 Phase 3: Effect of gradient amplification factor

Figure 5 shows the performance of models as Γ is varied. params of the best models after analysis Phases 1 and 2 is taken and gradients are amplified by varying the value of Γ from $\{1, 2, 3, \ldots, 10\}$. For VGG-19, the best model is achieved while amplifying only BN layers for params $S2\_0.1\_0.3$. For Resnet-18 and Resnet-34, the



(a) Step 2 for VGG-19

(b) Step 2 for Resnet-18

(c) Step 2 for Resnet-34

Fig. 4 Performance of the models after training with Step 2 strategy with gradient amplification (red) applied from epochs 51–100 compared to mean accuracies of the original models (blue) with no gradient amplification. In each plot, blue horizontal line shows the average testing accuracy of the original models without gradient amplification. Amp testing refers to testing accuracies of models with gradient amplification. The type of the layer is shown in each subplot; horizontal and vertical axes correspond to the ratio of amplified layers and accuracies, respectively. These experiment plots correspond to params $S2\_0.3\_xx$, where the ratio (0.3) of layers are amplified for epochs 51–100. The other params $S2\_0.1\_xx$, $S2\_0.5\_xx$, and $S2\_0.6\_xx$ also have similar performance patterns.

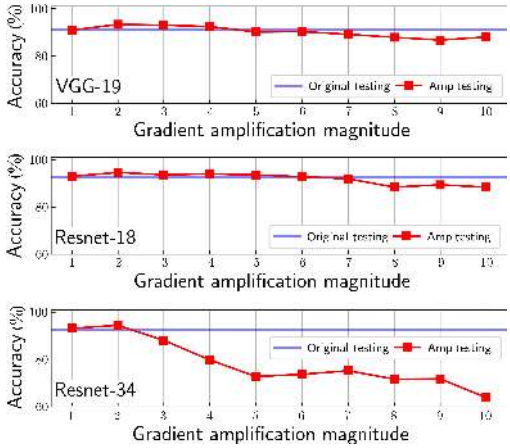


Fig. 5 Performance comparison of amplified models (red) as Γ is varied from 1 to 10 (horizontal axis) vs. original models (blue).

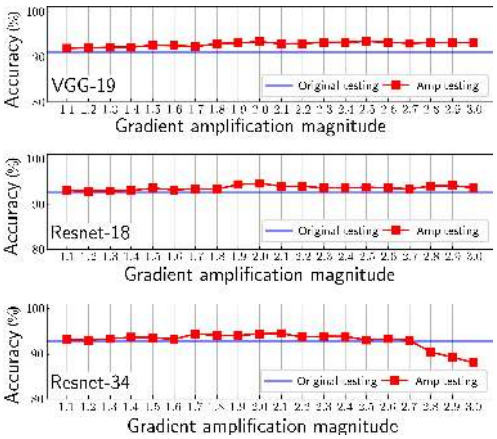


Fig. 6 Performance comparison of amplified models (red) as Γ is varied in small steps from 1 to 3 (horizontal axis) vs. original models (blue).

best models are achieved while amplifying only one of the BN layers in residual units and for params $S2\_0.3\_0.5$ and $S2\_0.1\_0.7$, respectively. While changing values of Γ, as the factor of amplification Γ increases, the performance of the models declines. To generalize, we can say that when Γ is more than 5, the models do not perform better or sometimes perform worse than the corresponding baseline models. Effect of amplification factor Γ also depends on the ratio of layers being amplified. If the ratio is close to 1, then Γ value less than 5 can also decrease performance of the models.

We also perform experiments by fine-tuning the amplification factor from 1 to 3 in steps of 0.1, i.e., by varying $\Gamma \in \{1.1, 1.2.1.3, 1.4, \ldots, 3\}$. Figure 6 shows the performance of these models as Γ is varied in small steps from 1 to 3. In the case of VGG-19 and Resnet-18, the model always performs better than the baseline models both during training and testing and for Resnet-34, the model performs better until 2.7 and declines after that. In all these models, it can be observed that the best accuracy is around the value 2， which justifies our experiment analysis in the above phases.

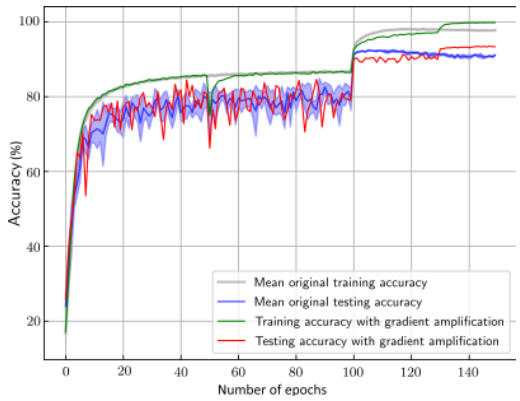
4.3 Best model

The best performance of all the models is shown in Table 1. Performance improvements can be observed both training and testing accuracies. The rows with "original" in params column show the performance of the original model with no gradient amplification. The following grayed row shows the performance of the corresponding model with gradient amplification. The params that achieves these best models is also shown. We can observe that gradient amplification increases both training and testing accuracies. Though training accuracies are very close to 100% in the original model, gradient amplification improves them further. It can be noted that Resnet models comprise of residual blocks architecture (an extra connection to the preceding layer), which already overcome vanishing gradients by passing the current gradients directly to the previous layers without modification using residual connection, and therefore an improvement of 1.67% can be assumed to be significant.

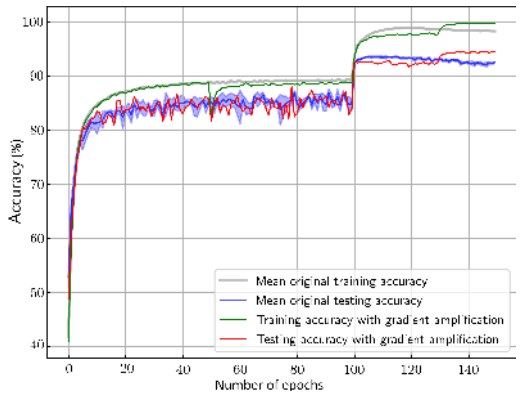
Table 1 Accuracy comparison of models with gradient amplification vs. mean accuracies of corresponding original model across 5 runs.

Model	params	Mean/best accuracy (%)		Improved accuracy (%)	
		Training	Testing	Training	Testing
VGG-19	Original	97.87	91.08	–	–
	Ours (VGG-19 with amplification)	99.764	93.35	1.9	**2.27**
Resnet-18	Original	98.371	92.488	–	–
	Ours (Resnet-18 with amplification)	99.878	94.57	1.51	**2.08**
Resnet-34	Original	98.444	92.716	–	–
	Ours (Resnet-34 with amplification)	99.774	94.39	1.25	**1.67**

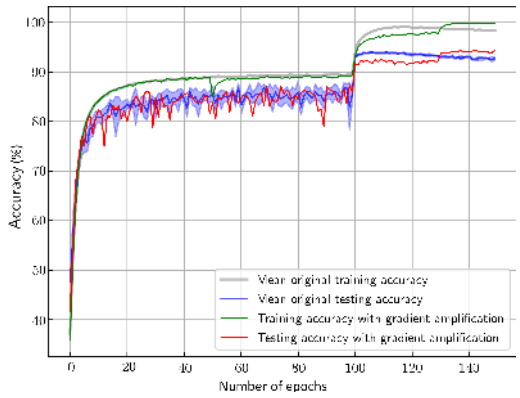
Figure 7 shows the performance of the each of the models for the params listed in Table 1. These plots demonstrate that the models trained with gradient amplification do not cause overfitting problem. In the case of VGG-19, the best model is achieved when



(a) VGG-19



(b) Resnet-18



(c) Resnet-34

Fig. 7 Performance of the best models with gradient amplification over 150 epochs compared to original model with no gradient amplification. Original training (gray) and testing (blue) accuracies including their mean accuracies are plotted along with amplified training (green) and testing (red) accuracies. These plots demonstrate that the models do not overfit while training with amplification.

amplification is applied only on BN layers and for Resnet models, the best models are achieved when only one of the BN layers from a residual block is considered for amplification. Gradient amplification model surpasses the performance of all the original models. Accuracies achieved by amplified models not only exceed the mean average accuracies across 5 runs of the original models, but also outperform the best accuracy among these 5 runs of the original models.

To analyze the impact of training time, we perform experiments on the original models without amplification. Even the best models out of 5 runs do not achieve performance close to the corresponding amplified models. We run all the original models for 50 more epochs with the learning rate of 0.01 (i.e., 151–200 with $\eta = 0.01$) and notice that these models do not reach the performance of amplified models. We need to reduce the learning rate further for the next epochs to improve performance of the original models and therefore there is no direct way of comparing them. But clearly, our proposed method of training with amplified gradients can train the deep learning models at higher learning rates to achieve better performance. Original models take more time to achieve similar accuracy (at the same settings) as amplified models.

5 Conclusion & Future Work

In this work, we propose a novel gradient amplification method to dynamically increase gradients during backpropagation. We also provide a training strategy consisting of set of epochs with switching between gradient amplification and without amplification. Detailed experiments are performed on VGG-19, Resnet-18, and Resnet-34 models to analyze the impact of gradient amplification with different amplification parameters. We learn that only a proportion of layers are sufficient to attain such a performance gain. It can also be observed that BN layers give the best improvement while performing amplification, followed by ReLU layers, whereas the performance quickly diminishes when both ReLU+BN are used. All these experiments show that our proposed amplification method and training strategy increase the performance of the original models and achieve better accuracies even at higher learning rates.

In future work, we would like to perform the experiments on the larger datasets having more output classification classes like CIFAR-100 and ImageNet. In our current experimental models, there were no

exploding gradients. However, we would also like to explore other models having such an issue and experiment gradient diminishing dynamically, similar to graident amplification, to address gradient exploding problem. We are also interested in designing an adaptive algorithm, which will intelligently amplify layers during backpropagation without manually tuning the amplification parameters. Fuzzy logic systems have been used in many applications, such as wireless network routing[30]. We will introduce fuzzy logic into our learning and prediction models in the future.

Acknowledgment

The authors would like to thank Georgia State University (GSU) for providing us access to High Performance Computing (HPC) cluster on which most of the experiments are performed. This research was supported in part by an NVIDIA Academic Hardware Grant.

References

[1] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, Deep residual learning for image recognition, in *Proc. 2016 IEEE Conf. on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 770–778.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Proc. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[3] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] A. Kamilaris and F. X. Prenafeta-Boldú, Deep learning in agriculture: A survey, *Comput. Electron. Agric.*, vol. 147, pp. 70–90, 2018.

[5] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. W. M. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, A survey on deep learning in medical image analysis, *Med. Image Anal.*, vol. 42, pp. 60–88, 2017.

[6] Q. C. Zhang, L. T. Yang, Z. K. Chen, and P. Li, A survey on deep learning for big data, *Inf. Fusion*, vol. 42, pp. 146–157, 2018.

[7] L. Zhang, S. Wang, and B. Liu, Deep learning for sentiment analysis: A survey, *Wiley Interdiscip. Rev.*, vol. 8, no. 4, p. e1253, 2018.

[8] J. D. Wang, Y. Q. Chen, S. J. Hao, X. H. Peng, and L. S. Hu, Deep learning for sensor-based activity recognition: A survey, *Pattern Recognit. Lett.*, vol. 119, pp. 3–11, 2019.

[9] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, Deep networks with stochastic depth, in *Proc. 14^{th} European Conf. on Computer Vision*, Amsterdam, Netherlands, 2016, pp. 646–661.

[10] J. Brownlee, Understand the impact of learning rate on neural network performance, https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks, 2019.

[11] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies, In *A Field Guide to Dynamical Recurrent Networks*, J. F. Kolen and S. C. Kremer, eds. Wiley-IEEE Press, DOI: 10.1109/9780470544037.ch14.

[12] G. B. Goh, N. O. Hodas, and A. Vishnu, Deep learning for computational chemistry, *J. Comput. Chem.*, vol. 38, no. 16, pp. 1291–1307, 2017.

[13] B. Hanin, Which neural net architectures give rise to exploding and vanishing gradients? in *Proc. Advances in Neural Information Processing Systems 31*, Montréal, Canada, 2018, pp. 582–591.

[14] J. Schmidhuber, Learning complex, extended sequences using the principle of history compression, *Neural Comput.*, vol. 4, no. 2, pp. 234–242, 1992.

[15] V. Nair and G. E. Hinton, Rectified linear units improve restricted Boltzmann machines, in *Proc. 27^{th} Int. Conf. on Machine Learning*, Haifa, Israel, 2010, pp. 807–814.

[16] X. Glorot, A. Bordes, and Y. Bengio, Deep sparse rectifier neural networks, in *Proc. 14^{th} Int. Conf. on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, USA, 2011, pp. 315–323.

[17] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv: 1502.03167, 2015.

[18] Y. Yang and H. Wang, Multi-view clustering: A survey, *Big Data Mining and Analytics*, vol. 1, no. 2, pp. 83–107, 2018.

[19] S. Kumar and M. Singh, A novel clustering technique for efficient clustering of big data in hadoop ecosystem, *Big Data Mining and Analytics*, vol. 2, no. 4, pp. 240–247, 2019.

[20] J. Schmidhuber, Deep learning in neural networks: An overview, *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[21] C. Darken, J. Chang, and J. Moody, Learning rate schedules for faster stochastic gradient search, in *Proc. Neural Networks for Signal Processing II Proc. of the 1992 IEEE Workshop*, Helsingoer, Denmark, 1992, pp. 3–12.

[22] J. Duchi, E. Hazan, and Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[23] M. D. Zeiler, Adadelta: An adaptive learning rate method, arXiv preprint arXiv: 1212.5701, 2012.

[24] A. Graves, Generating sequences with recurrent neural networks, arXiv preprint arXiv: 1308.0850, 2013.

[25] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv: 1412.6980, 2014.

[26] S. Lau, Learning rate schedules and adaptive learning rate methods for deep learning, https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1, 2019.

[27] T. Schaul, S. X. Zhang, and Y. LeCun, No more pesky learning rates, in *Proc. 30^{th} Int. Conf. on Machine Learning*, Atlanta, GA, USA, 2013, pp. 343–351.

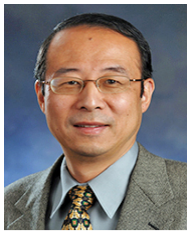
[28] S. L. Smith, P. J. Kindermans, C. Ying, and Q. V. Le, Don't decay the learning rate, increase the batch size, arXiv preprint arXiv: 1711.00489, 2017.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. M. Lin, A. Desmaison, L. Antiga, and A. Lerer, Automatic differentiation in PyTorch, in *Proc. 31^{st} Conf. on Neural Information Processing Systems*, Long Beach, CA, USA, 2017.

[30] H. Liu, J. Li, Y. Q. Zhang, and Y. Pan, An adaptive genetic fuzzy multi-path routing protocol for wireless ad-hoc networks, in *Proc. 6th Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and 1st ACIS Int. Workshop on Self-Assembling Wireless Network*, Towson, MD, USA, 2005, pp. 468–475.

Sunitha Basodi is currently a senior PhD student at the Department of Computer Science, Georgia State University. Prior to that, she worked in Amazon Inc., India as a developer from 2009 to 2014. Her current research focuses on machine learning and deep learning methods, and their applications in bioinformatics and smart grid domains.

Chunyan Ji received the BSc degree from East China Normal University, Shanghai, China in 1995 and the MSc degree in computer science from Georgia State University (GSU), Atlanta, USA in 2001. She worked as a software developer at Atlanta from 2001 to 2008 and worked as a senior lecturer at BNU-HKBU United International College, Zhuhai, China from 2008 to 2018. Since August 2018, she has been with GSU, where she currently works toward the PhD degree. Her main research interests include sound event detection and deep learning.



Yi Pan is currently a Regents' Professor and chair of computer science at Georgia State University, USA. He has served as an associate dean and chair of Biology Department during 2013–2017 and chair of Computer Science during 2006–2013. He joined Georgia State University in 2000, was promoted to full professor in 2004, named a distinguished university professor in 2013, and designated a Regents' Professor (the highest recognition given to a faculty member by University System of Georgia) in 2015. He received the BEng and MEng degrees in computer engineering from Tsinghua University, China in 1982 and 1984, respectively, and the PhD degree in computer science from University of Pittsburgh, USA, in 1991. His profile has been featured as a distinguished alumnus in both Tsinghua Alumni Newsletter and University of Pittsburgh CS Alumni Newsletter. His current research interests mainly include bioinformatics and health informatics using big data analytics, cloud computing, and machine learning technologies. He has published more than 450 papers including over 250 journal papers with more than 100 papers published in IEEE/ACM Transactions journals. In addition, he has edited/authored 43 books. His work has been cited more than 12 800 times based on Google Scholar and his current h-index is 57. He has served as an editor-in-chief or editorial board member for 20 journals including 7 IEEE Transactions. Currently, he is serving as an associate editor-in-chief of *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. He is the recipient of many awards including one IEEE Transactions Best Paper Award, five IEEE and other international conference or journal Best Paper Awards, 4 IBM Faculty Awards, 2 JSPS Senior Invitation Fellowships, IEEE BIBE Outstanding Achievement Award, IEEE Outstanding Leadership Award, NSF Research Opportunity Award, and AFOSR Summer Faculty Research Fellowship. He has organized numerous international conferences and delivered keynote speeches at over 60 international conferences around the world.

Haiping Zhang received the PhD degree in computational biophysics and biology from Nanyang Technological University in 2018. He is currently working as a postdoc in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His recent work focuses on applying deep learning techniques to explore protein-ligand interactions and exploring strategies to aid drug development by combinations of MD method, deep learning, and molecular docking together.