A RELU DENSE LAYER TO IMPROVE THE PERFORMANCE OF NEURAL NETWORKS

Alireza M. Javid, Sandipan Das, Mikael Skoglund, and Saikat Chatterjee

School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology, Sweden
{almj, sandipan, skoglund, sach}@kth.se

ABSTRACT

We propose ReDense as a simple and low complexity way to improve the performance of trained neural networks. We use a combination of random weights and rectified linear unit (ReLU) activation function to add a ReLU dense (ReDense) layer to the trained neural network such that it can achieve a lower training loss. The lossless flow property (LFP) of ReLU is the key to achieve the lower training loss while keeping the generalization error small. ReDense does not suffer from vanishing gradient problem in the training due to having a shallow structure. We experimentally show that ReDense can improve the training and testing performance of various neural network architectures with different optimization loss and activation functions. Finally, we test ReDense on some of the state-of-the-art architectures and show the performance improvement on benchmark datasets.

***Index Terms*—** Rectified linear unit, random weights, deep neural network

1. INTRODUCTION

Artificial neural networks (ANNs) and deep learning architectures have received enormous attention over the last decade. The field of machine learning is enriched with appropriately trained neural network architectures such as deep neural networks (DNNs) [1] and convolutional neural networks (CNNs) [2], outperforming the classical methods in different classification and regression problems [3,4]. However, very little is known regarding what specific neural network architecture works best for a certain type of data. For example, it is still a mystery that for a given image dataset such as CIFAR-10, how many numbers of hidden neurons and layers are required in a CNN to achieve better performance. Typical advice is to use some rule-of-thumb techniques for determining the number of neurons and layers in a NN or to perform an exhaustive search which can be extremely expensive [5]. Therefore, it frequently happens that even an appropriately trained neural network performs lower than expected on the training and testing data. We address this problem by using a combination of random weights and ReLU activation functions to improve the performance of the network.

There exist three standard approaches to improve the performance of a neural network - avoiding overfitting, hyperparameter tuning, and data augmentation. Overfitting is mostly addressed by ℓ_1 or ℓ_2 regularization [6], Dropout [7], and early stopping techniques [8]. Hyperparameter tuning of the network via cross-validation includes choosing the best number of neurons and layers, finding the optimum learning rate, and trying different optimizers and loss functions [9]. And finally, the data augmentation approach tries to enrich the training set by constructing new artificial samples via random cropping and rotating, rescaling [10], and mixup [11]. However, all of the above techniques require retraining the network from scratch while there is no theoretical guarantee of performance improvement. Motivated by the prior uses of random matrices in neural networks as a powerful and low-complexity tool [12–16], we use random matrices and lossless-flow-property (LFP) [17] of ReLU activation function to theoretically guarantee a reduction of the training loss.

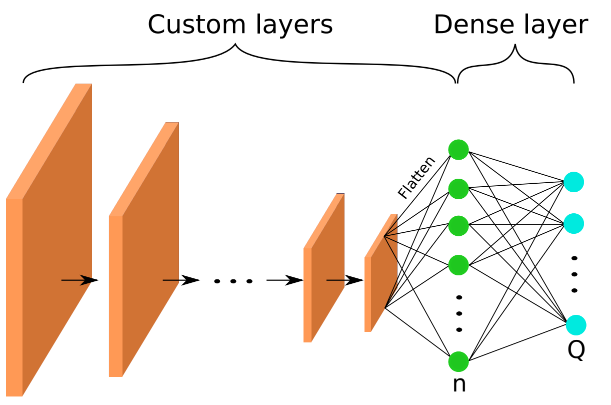
Our Contribution: We propose to add a ReDense layer instead of the last layer of an optimized network to improve the performance as shown in Figure 1. We use a combination of random weights and ReLU activation functions in order to theoretically guarantee the reduction of the training loss. Similar to transfer learning techniques, we freeze all the previous layers of the network and only train the ReDense layer to achieve a lower training loss. Therefore, training ReDense is fast and efficient and does not suffer from local minima or vanishing gradient problem due to its shallow structure. We test ReDense on MNIST, CIFAR-10, and CIFAR-100 datasets and show that adding a ReDense layer leads to a higher classification performance on a variety of network architecture such as MLP [18], CNN [2], and ResNet-50 [19].

1.1. Artificial Neural Networks

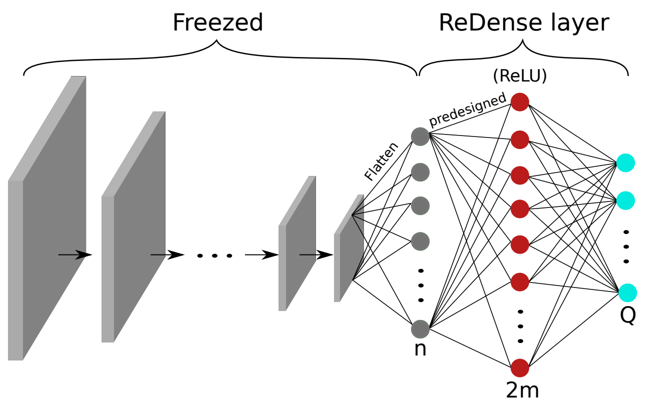
Consider a dataset containing J samples of pair-wise P-dimensional input data $\mathbf{x}_j \in \mathbb{R}^P$ and Q-dimensional target vector $\mathbf{t}_j \in \mathbb{R}^Q$ as $\mathcal{D} = \{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^J$. Let us define the j-th feature vector at second last layer of a ANN as $\mathbf{y}_j = f_{\boldsymbol{\theta}}(\mathbf{x}_j) \in \mathbb{R}^n$, where $\boldsymbol{\theta}$ represents the parameters of all the previous layers of the network. Note that n is the number of hidden neurons in the second last layer of the network. Let us define the weights of the last layer of network by $\mathbf{O} \in \mathbb{R}^{Q \times n}$. We refer to this weight matrix as the 'output weight' in the rest of the manuscript. The training of the network can be done as follows

$$\hat{\mathbf{O}}, \hat{\boldsymbol{\theta}} \in \arg\min_{\mathbf{O},\boldsymbol{\theta}} \sum_{j=1}^{J} \mathcal{L}(\mathbf{t}_j, \mathbf{O} f_{\boldsymbol{\theta}}(\mathbf{x}_j)) + \mathcal{R}(\mathbf{O}, \boldsymbol{\theta}), \qquad (1)$$

where $\mathcal{L}$ is the loss function over a single training pair $(\mathbf{x}_j, \mathbf{t}_j)$, and $\mathcal{R}$ is the regularization term to avoid overfitting, e.g., a simple ℓ_2-norm weight decay. Examples of loss function $\mathcal{L}$ include cross-entropy loss, mean-square loss, and Huber loss. After the training, we have access to the optimized feature vector at the second last layer. We refer to this feature vector as $\hat{\mathbf{y}}_j = f_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_j) \in \mathbb{R}^n$ in the following sections. Note that equation (1) is general in the sense that $f_{\boldsymbol{\theta}}(.)$ can represent any arbitrary NN architecture combined of dropout layer, convolutional layer, pooling layer, skip connections, etc. The popular optimization methods used for solving (1) are Adagrad, ADAM, Adadelta, and their stochastic mini-batch variants [20].



(a) Original network



(b) With ReDense

Fig. 1: Illustraition of ReDense training. ReDense can be seen as a combination of predesigned ReLU layer with $m \geq n$ and a dense layer.

1.2. Lossless Flow Property of ReLU

We briefly discuss lossless flow property (LFP) of ReLU activation function in this section for completeness. Details can be found in [17]. The proposed LFP is the key to design the ReLU layer in order to boost the performance.

Definition 1 (Lossless flow property (LFP))**.** *A non-linear activation function* $\mathbf{g}(\cdot)$ *holds the lossless flow property (LFP) if there exist two linear transformations* $\mathbf{V} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{U} \in \mathbb{R}^{n \times m}$ *such that* $\mathbf{U}\mathbf{g}(\mathbf{V}\mathbf{z}) = \mathbf{z}, \forall \mathbf{z} \in \mathbb{R}^n$.

This property means that the input $\mathbf{z}$ flows through the network without any loss, resulting in $\mathbf{U}\mathbf{g}(\mathbf{V}\mathbf{z}) = \mathbf{z}$. The following proposition illustrates that ReLU activation function holds LFP.

Proposition 1. *Consider ReLU activation* $\mathbf{g}(\mathbf{z}) = \max(\mathbf{z}, 0)$ *and the identity matrix* $\mathbf{I}_n \in \mathbb{R}^{n \times n}$. *If* $\mathbf{V} \triangleq \mathbf{V}_n = \begin{bmatrix} \mathbf{I}_n \\ -\mathbf{I}_n \end{bmatrix} \in \mathbb{R}^{2n \times n}$ and $\mathbf{U} \triangleq \mathbf{U}_n = [\mathbf{I}_n \ \ -\mathbf{I}_n] \in \mathbb{R}^{n \times 2n}$, *then* $\mathbf{U}_n\mathbf{g}(\mathbf{V}_n\mathbf{z}) = \mathbf{z}$ *holds for every* $\mathbf{z} \in \mathbb{R}^n$.

The above proposition can be easily proved by a simple calculation. It is shown in [17] that LFP property holds for the family of ReLU-based activations such as leaky-ReLU.

2. RELU DENSE LAYER

Consider a trained neural network using the training dataset $\mathcal{D} = \{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^J$. Let us define the optimized output weight as $\hat{\mathbf{O}}$ and the corresponding feature vectors as

$$\hat{\mathbf{y}}_j = f_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_j) \in \mathbb{R}^n, \qquad (2)$$

where $f_{\hat{\boldsymbol{\theta}}}(\cdot)$ represents the optimized preceding layers of the network. The optimized training loss can be written as $\mathcal{L}_o = \sum_{j=1}^J \mathcal{L}(\mathbf{t}_j, \hat{\mathbf{O}}\hat{\mathbf{y}}_j)$, where $\mathcal{L}_o$ stands for old loss that we wish to improve. To this end, we want to add a new layer to the network that can provides us with a lower training loss. Let us construct a new ReLU layer as follows

$$\bar{\mathbf{y}}_j = \mathbf{g}(\mathbf{V}_m\mathbf{R}\hat{\mathbf{y}}_j) \in \mathbb{R}^{2m}, \qquad (3)$$

where $\mathbf{g}(\cdot)$ is ReLU activation function, $\mathbf{V}_m = \begin{bmatrix} \mathbf{I}_m \\ -\mathbf{I}_m \end{bmatrix}$, and $\mathbf{R} \in \mathbb{R}^{m \times n}$, $m \geq n$ is an instance of normal distribution. The training of ReDense is done as follows

$$\mathbf{O}^\star \in \arg\min_{\mathbf{O}} \sum_{j=1}^J \mathcal{L}(\mathbf{t}_j, \mathbf{O}\bar{\mathbf{y}}_j) \ \text{ s.t. } \ \|\mathbf{O}\|_F \leq \epsilon, \qquad (4)$$

where ϵ is the regularization hyperparameter. In the following, we show that by properly choosing the hyperparameter ϵ, we can achieve a lower training loss than $\mathcal{L}_o$.

Proposition 2. *Consider the new training loss achieved by adding the new ReLU layer* $\mathcal{L}_n = \sum_{j=1}^J \mathcal{L}(\mathbf{t}_j, \mathbf{O}^\star\bar{\mathbf{y}}_j)$, *where* $\mathbf{O}^\star$ *is the solution of minimization* (4). *Then, there exists a choice of* ϵ *for which, we have* $\mathcal{L}_n \leq \mathcal{L}_o$.

Proof. Let us define $\mathcal{L}_n(\mathbf{O}) = \sum_{j=1}^J \mathcal{L}(\mathbf{t}_j, \mathbf{O}\bar{\mathbf{y}}_j)$. We first show that there exists a point in the feasible set of (4) that results in the old training loss, i.e., $\exists\bar{\mathbf{O}}, \mathcal{L}_n(\bar{\mathbf{O}}) = \mathcal{L}_o$. Consider $\bar{\mathbf{O}} = \hat{\mathbf{O}}\mathbf{R}^\dagger\mathbf{U}_m$, where $\mathbf{U}_m = [\mathbf{I}_m \ \ -\mathbf{I}_m]$, and $\dagger$ denotes pseudo-inverse of a matrix. Note that when $m \geq n$, the random matrix $\mathbf{R}$ is full-column rank and therefore, its pseudo-inverse exits. Then, by using LFP in Proposition 1 and the feature vector in (3), we have

$$\begin{aligned} \mathcal{L}_n(\bar{\mathbf{O}}) &= \sum_{j=1}^J \mathcal{L}(\mathbf{t}_j, \hat{\mathbf{O}}\mathbf{R}^\dagger\mathbf{U}_m\bar{\mathbf{y}}_j) \\ &= \sum_{j=1}^J \mathcal{L}(\mathbf{t}_j, \hat{\mathbf{O}}\mathbf{R}^\dagger\mathbf{U}_m\mathbf{g}(\mathbf{V}_m\mathbf{R}\hat{\mathbf{y}}_j)) \\ &= \sum_{j=1}^J \mathcal{L}(\mathbf{t}_j, \hat{\mathbf{O}}\mathbf{R}^\dagger\mathbf{R}\hat{\mathbf{y}}_j)) \\ &= \sum_{j=1}^J \mathcal{L}(\mathbf{t}_j, \hat{\mathbf{O}}\hat{\mathbf{y}}_j)) = \mathcal{L}_o. \end{aligned} \qquad (5)$$

Therefore, by choosing $\epsilon = \|\hat{\mathbf{O}}\mathbf{R}^\dagger\mathbf{U}_m\|_F$, we make sure to have the smallest feasible set possible to avoid overfitting while including $\bar{\mathbf{O}}$ in the search space. On the other hand, we know that $\mathcal{L}_n(\mathbf{O}^\star) \leq \mathcal{L}_n(\bar{\mathbf{O}})$ by definition in (4), which concludes the proof. $\square$

```
Algorithm 1 : ReDense Training
Input:
 1: $\hat{\mathbf{O}}$: output weight matrix of the old trained neural network
 2: $\{(\hat{\mathbf{y}}_j, \mathbf{t}_j)\}_{j=1}^{J}$: training features of the output layer and their corresponding target vectors
 3: $\eta_0$: initial learning rate of the gradient descent optimizer
 4: $\tau$: number of epochs of the gradient descent optimizer
Initialization:
 1: Construct the ReLU layer as $\bar{\mathbf{y}}_j = \mathbf{g}(\mathbf{V}_m \mathbf{R} \hat{\mathbf{y}}_j)$
 2: $\epsilon \leftarrow \|\hat{\mathbf{O}}\mathbf{R}^{\dagger}\mathbf{U}_m\|_F$
 3: $\mathbf{O}_0 \leftarrow \hat{\mathbf{O}}\mathbf{R}^{\dagger}\mathbf{U}_m$
 4: $t \leftarrow 0$
Training:
 1: repeat
 2:     $\mathbf{G}_t \leftarrow \sum_{i=1}^{N} \nabla_{\mathbf{O}} \mathcal{L}(\mathbf{t}_i, \mathbf{O}_t \bar{\mathbf{y}}_i)$
 3:     $\mathbf{Z} \leftarrow \mathbf{O}_t - \eta_t \mathbf{G}_t$
 4:     if $\|\mathbf{Z}\|_F \geq \epsilon$ then
 5:         $\mathbf{O}_{t+1} \leftarrow \mathbf{Z}.(\frac{\epsilon}{\|\mathbf{Z}\|_F})$
 6:     end if
 7:     $t \leftarrow t + 1$
 8: until $t = \tau$
```

Note that the optimization in (4) is not computationally complex as it only minimizes the total loss with respect to the output weight $\mathbf{O}$. The weight matrix of the ReLU layer in (3) is fixed before training, and there is no need for error backpropagation. Therefore, by solving (4), we have added a ReLU dense layer to a trained network to reduce its training loss.

Detailed steps of training ReDense are outlined in Algorithm 1. Note that we have not included the update step of learning rate η_t in Algorithm 1 for the sake of simplicity. In practice, training of ReDense can be done via any of the common adaptive gradient descent variants such as ADAM, Adagrad, etc. An illustration of ReDense training is shown in Figure 1, where the grayscale layers indicate being freezed during the training. Therefore, ReDense is not prone to vanishing gradient issue and is computationally fast. Note that the original network can be any arbitrary neural network architecture, making ReDense a universal technique to reduce the training loss.

3. EXPERIMENTAL RESULTS

In this section, we carry out several experiments on a variety of neural network architectures to show the performance capabilities of ReDense. First, we construct a simple MLP and investigate the effects of changing the number of random hidden neurons m in ReDense. Then, we illustrate the performance of ReDense on more complicated models such as convolutional neural networks with various types of loss functions and output activations. Finally, we choose some of the state-of-the-art networks on benchmark classification datasets and try improving their performance via ReDense. Our open-sourced code can be found at https://github.com/alirezamj14/ReDense.

3.1. Multilayer Perceptrons

In this section, we use a simple two-layer feedforward neural network with $n = 500$ hidden neurons to train on the MNIST dataset. The network is trained for 100 epochs with respect to softmax cross-entropy loss by using ADAM optimizer with a batch size equal to 128 and a learning rate of 10^{-6}. This combination leads to test classification accuracy of 93.1% on MNIST. By using the trained network, we construct the feature vector $\bar{\mathbf{y}}_j$ according to (3) for different choices of $m = 500, 1000, 1500, 2000$ and add a ReDense layer to the network. Note that here we must have $m \geq n = 500$.

Figure 2 shows the learning curves of ReDense for different values of m. Note that the initial performance at epoch 0 is, in fact, the final performance of the above MLP with 93.1% testing accuracy. We use full-batch ADAM optimizer with a learning rate of 10^{-5} for ReDense in all cases. By comparing the curves of training and testing loss, we observe that ReDense consistently reduces both training and testing loss while maintaining a small generalization error. Interestingly, ReDense achieves a higher testing accuracy for $m = 500$ and improves the testing accuracy by 2.4%. The reason for this behaviour is that increasing m, in fact, reduces the value of $\epsilon = \|\hat{\mathbf{O}}\mathbf{R}^{\dagger}\mathbf{U}_m\|_F$ in ReDense, leading to a tighter overfitting constraint as the dimensions of the feature vectors increases. One can, of course, manually choose larger values of ϵ for ReDense but we found in our experiments that a looser ϵ leads to poor generalization. Therefore, we use the smallest possible value of $m = n$ in the rest of the experiments.

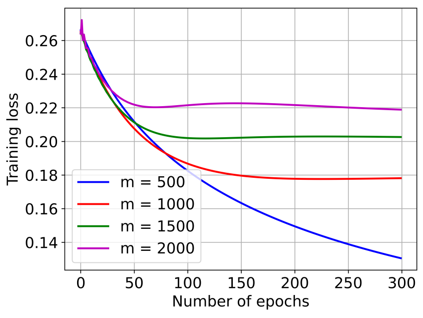
3.2. Convolutional Neural Networks

In this section, we use a convolutional neural network with the following architecture: two Conv2D layer with 32 filters of size 3×3 followed by a MaxPooling2D layer of size 3×3, dropout layer with $p = 0.25$, Conv2D layer with 64 filters of size 3×3, Conv2D layer with 32 filters of size 3×3, MaxPooling2D layer of size 3×3, dropout layer with $p = 0.25$, flatten layer, dense layer of size 512, dropout layer with $p = 0.5$, and dense layer of size 10. We train this network on the CIFAR-10 dataset using ADAM optimizer with a batch size of 128 for 15 epochs with respect to different loss functions. We use 10 percent of the training sample as a validation set. Note that in this case $n = 512$ is the dimension of the feature vector $\hat{\mathbf{y}}_j$. We choose the smallest possible value for the number of random hidden neurons of ReDense $m = 512$ as it is shown to be the best choice in Section 3.1.

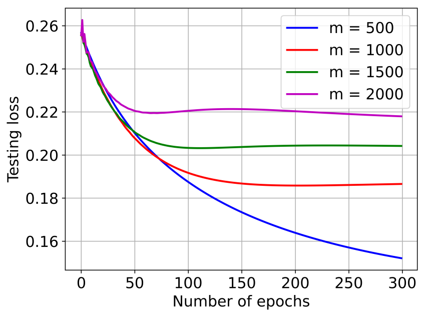
Figure 3 shows the learning curves of ReDense for different choices of CNN loss functions, namely, cross-entropy (CE) loss, mean-square-error (MSE) loss, Poisson loss, and Huber loss. Note that the initial performance at epoch 0 is, in fact, the final performance of the above CNN in each case. We use full-batch ADAM optimizer with a learning rate of 10^{-4} for ReDense in all cases. By comparing the curves of training and testing loss, we observe that ReDense consistently reduces both training and testing loss while maintaining a small generalization error. It can be seen that ReDense approximately improves testing accuracy by 1-2% in all cases. This experiment shows that ReDense can improve the performance of neural networks regardless of the choice of training loss. Note that we have used softmax cross-entropy loss in all the experiments for training ReDense.

3.3. State-of-the-art Networks

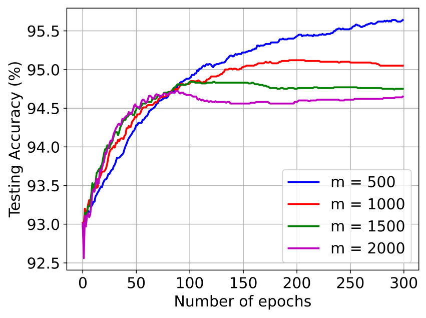
We apply ReDense on the best performer architecture on CIFAR datasets to observe the improvement in classification accuracy. To the best of our knowledge, Big Transfer (BiT) [21] models are the top performer on CIFAR datasets in the literature. BiT uses a combination of fine-tuning techniques and transfer learning heuristics to achieve state-of-the-art performance on a variety of datasets. BiT-



(a) Training loss

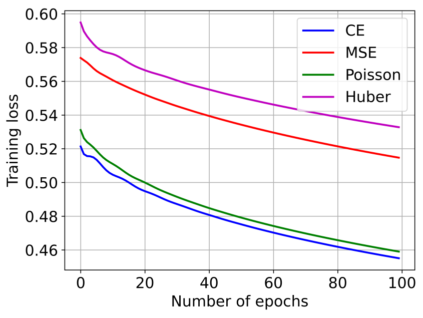


(b) Testing loss

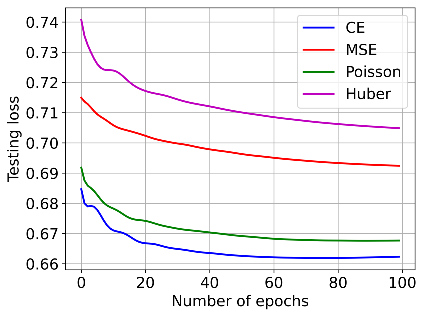


(c) Testing accuracy

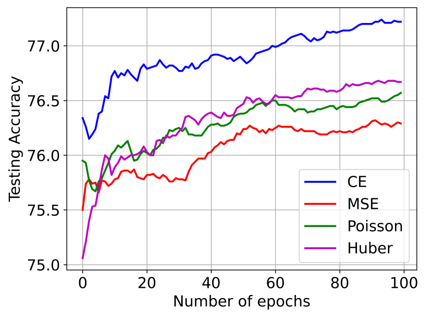
Fig. 2: Performance improvement versus number of epochs for a MLP trained on MNIST by using softmax cross-entropy loss.



(a) Training loss



(b) Testing loss



(c) Testing accuracy

Fig. 3: Performance improvement versus number of epochs for a CNN trained on CIFAR-10 by using different loss functions.

Table 1: Testing accuracy (%) with and without ReDense

Dataset	BiT-M-R50x1	
	with ReDense	without
CIFAR-10	**96.61**	95.35
CIFAR-100	**84.28**	82.64

M-R50x1 model uses a ResNet-50 architecture, pretrained on the public ImageNet-21k dataset, as the baseline model and by applying the BiT technique, achieves testing accuracy of 95.35% and 82.64% on CIFAR-10 and CIFAR-100, respectively.

We add a ReDense layer with $m = 2048$ random hidden neurons since the dense layer of BiT-M-R50x1 model has a dimension of $n = 2048$. We use full-batch ADAM optimizer with a learning rate of 10^{-4} and train ReDense layer with respect to softmax cross-entropy loss for 100 epochs. We observe that BiT-M-R50x1 + ReDense achieves testing accuracy of 96.61% and 84.28% on CIFAR-10 and CIFAR-100, respectively, as shown in Table 1. This experiment illustrates the capability of ReDense in improving the performance of highly optimized and deep networks. It is worth noting that ReDense is trained within a few minutes on a laptop while training BiT-M-R50x1 required several hours in a GPU cloud.

However, the power of ReDense is limited to scenarios where the network is not overfitted with a 100% training accuracy since ReDense tries to reduce the training loss. For example, we also tested BiT-M-R101x3 model which uses a ResNet-101 as the baseline model, and achieves 98.3% testing accuracy on CIFAR-10 but with a training accuracy of 100%. After adding the ReDense layer, we observed no tangible improvement as the network was already overfitted and there was no room for improvement. Early stopping of the baseline model, including dropout, data augmentation, and unfreezing some of the previous layers of the network are among the possible options to avoid such extreme overfitting in future works.

4. CONCLUSION

We showed that by adding a ReDense layer, it is possible to improve the performance of various types of neural networks including the state-of-the-art if the network is not overfitted. ReDense is a universal technique in the sense that it can be applied to any neural network regardless of its architecture and loss function. Training of ReDense is simple and fast due to its shallow structure and does not require retraining the network from scratch. A potential extension of this work would be to apply ReDense along with other performance improvement techniques such as dropout and data augmentation to further increase the predictive capabilities of neural networks. Besides, ReDense can also be applied in any of the intermediate layers of a network to reduce the training loss.

5. REFERENCES

[1] Christian Szegedy, Alexander Toshev, and Dumitru Erhan, “Deep neural networks for object detection,” in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, Red Hook, NY, USA, 2013, NIPS’13, p. 2553–2561, Curran Associates Inc.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, “Imagenet large scale visual recognition challenge,” *Intl. J. Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015.

[4] Samuel Dodge and Lina Karam, “A study and comparison of human and deep learning recognition performance under visual distortions,” *26th International Conference on Computer Communications and Networks, ICCCN 2017*, Sept. 2017.

[5] A. J. Thomas, M. Petridis, S. D. Walters, S. M. Gheytassi, and R. E. Morgan, “On predicting the optimal number of hidden nodes,” in *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2015, pp. 565–570.

[6] Jan Kukačka, Vladimir Golkov, and Daniel Cremers, “Regularization for deep learning: A taxonomy,” *ArXiv e-prints*, 2017.

[7] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[8] Lutz Prechelt, *Early Stopping — But When?*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[9] Tong Yu and Hong Zhu, “Hyper-parameter optimization: A review of algorithms and applications,” *ArXiv e-prints*, 2020.

[10] Connor Shorten and Taghi M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” *Journal of Big Data*, vol. 6, no. 1, pp. 60, 2019.

[11] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz, “mixup: Beyond empirical risk minimization,” in *International Conference on Learning Representations*, 2018.

[12] A. M. Javid, A. Venkitaraman, M. Skoglund, and S. Chatterjee, “High-dimensional neural feature using rectified linear unit and random matrix instance,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 4237–4241.

[13] R. Giryes, G. Sapiro, and A. M. Bronstein, “Deep neural networks with random gaussian weights: A universal classification strategy?,” *IEEE Trans. Signal Process.*, vol. 64, no. 13, pp. 3444–3457, July 2016.

[14] R. Vidal, J. Bruna, R. Giryes, and S. Soatto, “Mathematics of Deep Learning,” *ArXiv e-prints*, Dec. 2017.

[15] X. Liang, A. M. Javid, M. Skoglund, and S. Chatterjee, “A low complexity decentralized neural net with centralized equivalence using layer-wise learning,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[16] Saikat Chatterjee, Alireza M. Javid, Mostafa Sadeghi, Partha P. Mitra, and Mikael Skoglund, “Progressive learning for systematic design of large neural networks,” *ArXiv e-prints*, 2017.

[17] Saikat Chatterjee, Alireza M. Javid, Mostafa Sadeghi, Shumpei Kikuta, Dong Liu, Partha P. Mitra, and Mikael Skoglund, “SSFN – self size-estimating feed-forward network with low complexity, limited need for human intervention, and consistent behaviour across trials,” *ArXiv e-prints*, 2020.

[18] Jürgen Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[20] Sebastian Ruder, “An overview of gradient descent optimization algorithms,” *ArXiv e-prints*, 2017.

[21] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby, “Big transfer (bit): General visual representation learning,” *ArXiv e-prints*, 2020.