Small-to-Large: Proportional Uniform Distribution Weights Transferring to Reduce Energy Consumption of Training the Models with Fully Connected Layers

Farzad Vazinram(✉)

Department of Computer Science, University of Twente, Enschede, The Netherlands
f.vazinram@utwente.nl

Abstract. Providing sufficient and sustainable energy for users has always been a concern for all related stakeholders. The development of Deep Learning (DL) models with various applications has increased the need for their training, consequently raising their energy consumption drastically. This work has concentrated on reducing energy consumption of training the models with Fully Connected (FC) layers. So, the energy consumption of Multilayer Perceptron (MLP) models with FC layers has been investigated first to come up with a proper solution. The idea is based on the concept that the energy consumption of DL models exponentially rises by increasing their size. So, a small model with much lower energy consumption has been used for good initialization of the large model. As a result, there would be fewer number of epochs and less energy needed to reach a specific accuracy in comparison to randomly initialization of the model. The transferring of weights from the small model to the large model has been carried out by proposing a method called Proportional Uniform Distribution (PUD). The proposed method has been examined in various cases. The results show that training the models using the proposed method has lower energy consumption. Moreover, it can be applied easily to any library. Keras library has been selected in this work as a case.

Keywords: Energy Consumption Reduction · Fully Connected Layers · MLP · Proportional Uniform Distribution (PUD) · Training · Weight Transferring

1 Introduction

By developing the Deep Learning (DL) models and consequently increasing their complexity [1], the required energy for their training and usage is rapidly increasing [2]. So, any effort to reduce their energy consumption would have benefits for all related stakeholders and help to reduce carbon emissions.

Various efforts have been made to reduce the complexity of models and consequently make their training faster and with lower required computation.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15650, pp. 81–97, 2025.
https://doi.org/10.1007/978-3-031-96962-1_6

Any effort to reduce the energy consumption of model training per se consumes energy. The key point is that this consumed energy must always be lower than the final energy consumption reduction (Fig. 1). This case would be critical for Fully Connected (FC) layers in comparison to the convolutional (CONV) layers. The reason is that the energy consumption of CONV layers is much more than FC layers due to the much higher amount of data movement in comparison to pure computation [3,4]. This is the main reason that in the AlexNet model [5] 96% of the total weights belong to the FC layers but they only consume 28% of the total energy consumption [3]. So, in CONV layers by consuming energy to limit the data movement, we will achieve a lot. However, this range of energy consumption in FC layers is so limited. As a result, only considering the removed weights may not be enough to say that the energy consumption of training the models has been reduced [3] and the energy consumption as an independent criterion needs to be considered.

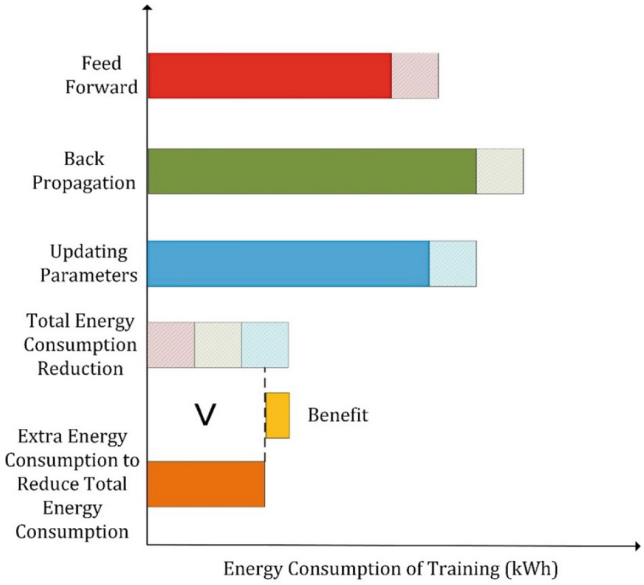


Fig. 1. Comparing the extra energy consumed to reduce the energy consumption of the model and the total energy consumption reduction of the model.

This work has been concentrated on energy consumption reduction of training Multilayer Perceptron (MLP) models with FC layers. The contributions of this work can be summarized as follows:

- Energy-based investigation of MLP models training with FC layers.
- Energy-based investigation of sparsifying technique in training of MLP models.

– A novel method to exploit the lower energy consumption of smaller models for better initialization of larger models to achieve the desired accuracy faster and reduce the energy consumption of MLP model training.

2 Related Works

There are so many attempts to make the training of DL models efficient with low or no effect on their accuracy. Most of the methods consider the training time, model parameters, Floating Point (FLOPs) Operations, Multiply-Accumulate (MACs) Operations, or a combination of them as criteria of efficiency. These methods are included but not limited to pruning, sparse training, model compression [6], quantization [7], efficient architecture design [8], and optimally set hyper-parameters [9]. However, as discussed in the introduction section, they may not be enough from viewpoint of energy consumption. In this work, the CodeCarbon package [10] has been used to measure the energy consumption of the training part of the models. This package is a project designed to measure the emission approximation and energy consumption of running codes.

Among the various methods, pruning and sparse training have been extensively studied in recent years. [11] has been provided with a good survey of various pruning methods. Various methods have been proposed to make models sparse. [12] covers the most recent works in this area. However, exploiting sparsity needs extra effort which has been an incentive for other researchers to concentrate on exploiting sparsity in the level of matrix multiplication. [13] focused on parallel multiplication of sparse and dense matrices. [14] mentioned a method for masked matrix multiplication by avoiding unnecessary computations. On the other side, some research avoids using masks and disregards all zero elements during training using sparse techniques with some overheads [15]. So, sparsity will be discussed shortly from the viewpoint of energy consumption in the next section.

Some other approaches include increasing the accuracy of the model which makes it faster to reach a specific accuracy and some of them can be similar to the mentioned directions. In [16], a new optimization model based on the Genetic Algorithm has been used for tuning the hyper-parameters of the MLP model with the aim of accuracy improvement. Averaging the prediction of various models trained in one dataset is another method used for increasing the performance of models [17]. In the same research path, [18] proposed a compression technique to distill the knowledge with the aim of computation reduction. In [19] the MLP projector has been used to better transfer the learning of unsupervised pretrained models. There are various other methods available for learning to transfer; however, they used another dataset or model type for transferring. Moreover, the author could not find any similar work carried out on FC layers specifically.

3 Methodology

The main- and sub-research questions have been defined as follows to systematically analyze the problem and their optimum possible solutions.

- **Main_Q**: How to reduce the energy consumption of MLP networks with FC layers with less impact on their accuracy?
- ***Sub_Q1:*** What are the energy consumption of different parts of the MLP models training with FC layers and what factors affect them?
- ***Sub_Q2:*** What are the energy consumption of different layers of the MLP model training?
- ***Sub_Q3:*** What is the relation among sparsity, its difficulty to be exploited, and final energy consumption of training MLP models with FC layers?

3.1 Sub_Q1: Discussion on Energy Consumption of MLP Models

This part aims to analyze the energy consumption of different parts of the training MLP model. So, the energy consumption of the different parts of an MLP model including Feed Forward, Back Propagation, and updating the parameters during training through the MNIST dataset with 784 input features and 10 different output classes was measured (Fig. 2). The input and output sizes are 784 and 10, respectively. The batch size of all training was chosen as 128. The hidden layer sizes are (128×200), (1280×2000), and (12800×20000) for small, medium and large models, respectively. The energy consumption of the model is divided into different parts of the training phase including Feed Forward, Back Propagation, and updating the parameters. The range of energy consumptions in small and medium models are close to each other but the differences are prominent in a large model. It is observed that the most energy-consuming part of the MLP model is Back Propagation and the updating the parameters part is in the second rank. These findings show that there is more potential to reduce the energy consumption of training MLP models rather than only focusing on weight pruning. The general solutions can be reducing the models' dimension or reducing the total number of required iterations by keeping the accuracy in an acceptable range.

Although the Feed Forward part had the lowest energy consumption in our case, increasing the batch size can increase considerably the energy consumption and has also effect on the Back Propagation part of the MLP model during the training but it does not have too much effect on update parameters part (Fig. 3). Increasing the batch size will reduce our required iterations and total energy consumption but with the cost of losing our accuracy. So, choosing the appropriate batch size has a direct effect on the time and energy consumption of the MLP model during training. In Fig. 3, the total energy and accuracy reduced from 0.002 kWh and 0.95 to 0.0009 kWh and 0.85, respectively when batch size increased from 128 to 1280.

It is also observed that (Fig. 2) the energy consumption of MLP model training will increase exponentially by increasing the neurons in their hidden layers

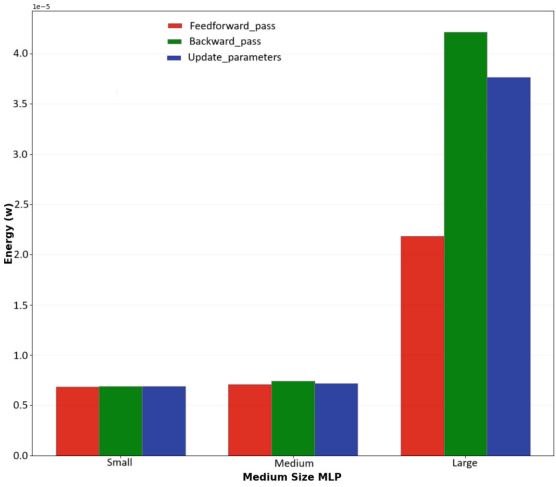


Fig. 2. The energy consumption of a model with two hidden layers with three different sizes.

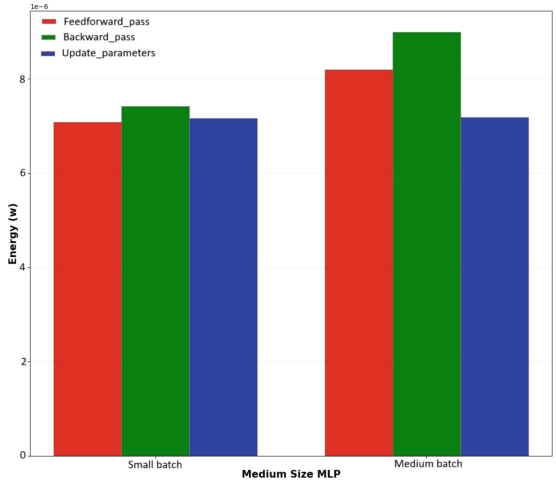


Fig. 3. Comparison of average energy consumption of different parts of an MLP model including two hidden layers with 1280 and 2000 neurons during training by MNIST dataset with 784 input features and 10 different output classes and two batch sizes of 128 and 1280.

which is compatible with the findings of [3]. So, the large models are more attractive to reduce their energy consumption and obviously choosing the appropriate size for the model is important from the viewpoint of energy consumption. Moreover, it is a great incentive for this paper to use small models for large model initialization.

3.2 Sub_Q2: Energy Consumption of Different Layers of the MLP Model

One idea to reduce the total energy consumption of models is to start reducing the layers that have the highest energy consumption. Because in FC layers, computation is a considerable portion of energy consumption unlike CONV layers which data movement is the key factor [3,4], it was observed that the energy consumption of FC layers is approximately related to the required computation. In other words, in FC layers, the energy consumption is related to the layer sizes that make our matrices. The results of observations are presented in Table 1. The energy consumption of each layer is measured after 10 iterations and the average is calculated.

Table 1. The energy consumption of different layers of MLP models with different sizes.

Model	Layer1_2 (kWh)	Layer2_3 (kWh)	Layer3_4 (kWh)
input_size = 784 hidden_size1 = 2000 hidden_size2 = 2000 output_size = 10	6.9495e-06	7.1233e-06	6.9043e-06
input_size = 784 hidden_size1 = 200 hidden_size2 = 200 output_size = 10	6.9298e-06	6.9151e-06	6.8985e-06
input_size = 784 hidden_size1 = 1280 hidden_size2 = 2000 output_size = 10	6.9396e-06	7.0510e-06	6.9002e-06

3.3 Sub_Q3: Discussion About Exploiting Sparsity to Reduce Energy Consumption and Speed of Training MLP Models with FC Layers

To reduce training model computation, there are various methods available that revolve around sparsity (zero parameters). However, the currently available hardware is fully optimized for dense computations [14]. Hence, using suitable hardware to exploit sparsity, e.g., Sputnik [20] and NVIDIA cuSPARSE [21] is one

solution. The more common solution is using standard hardware along with some software techniques. The main idea is to only save the non-zero elements and disregard the zero elements from computations to save memory and computation. To do so, there are some libraries available e.g. SciPy [22], TensorFlow [23], etc.

Exploiting sparsity rooted in matrix multiplication which depends on the utilized method can be normal or element-wise. Sometimes, depending on the used method, the overhead of using sparse techniques is higher than we can achieve in the end. So, the multiplication of two matrices with various sizes, sparsity levels, values, distribution of non-zero elements, and data type of elements are examined using the Compressed Sparse Row (CSR) method of SciPy library [22]. The results show that having a dispersed pattern of non-zero elements, different values, and low sparsity levels in a sparse matrix can increase the energy consumption of matrix multiplication. The data type (e.g. float or integer) and matrix size have a direct effect on the energy consumption of matrix multiplication but cannot make a general rule for it. For example, by using the sparse technique and sparsity level of 99%, the energy consumption of multiplication of sparse and dense matrices (both sizes 5000×5000) was reduced from 0.08 kWh to 0.00002 kWh. However, by changing the matrix dimension, using the sparse technique increases the energy consumption in comparison to the normal multiplication.

All in all, lots of methods have been proposed to have sparse matrices. However, exploiting sparsity needs extra effort which makes these methods challenging and sometimes containing high overhead.

3.4 Main_Q: How to Reduce the Energy Consumption of MLP Networks

According to the discussion over research sub-questions, the proposed method to reduce the energy consumption of training the FC layers is increasing the model accuracy, with low overhead, to reduce the total number of iterations and consequently energy consumption. To do so, a small model with low energy consumption would be used to train over the same data and transfer its weights to the larger model with high energy consumption with the aim of good initialization. In this method, we will not lose any information or accuracy; however, the model would only be efficient for the training phase and not the inference phase. Based on the author's knowledge, there is no similar method with the aim of energy consumption reduction of training FC layers.

With consideration of having an MLP network with FC layers and two hidden layers as follows:

$$\begin{aligned} X_{(\text{input})} \times W^{(1)} + b^{(1)} &= a^{(1)} \\ Z^{(1)} &= \text{Relu}(a^{(1)}) \\ Z^{(1)} \times W^{(2)} + b^{(2)} &= a^{(2)} \\ Z^{(2)} &= \text{Relu}(a^{(2)}) \\ Z^{(2)} \times W^{(3)} + b^{(3)} &= a^{(3)} \\ Y_{(\text{output})} &= \text{Relu}(a^{(3)}) \end{aligned} \tag{1}$$

that the equations can be summarized as:

$$z_j = f(XW^{(L)} + b^{(L)})$$

where $X \in \mathbb{R}^{n \times x}$ is input features for the batch of n samples, $W^{(L)} \in \mathbb{R}^{m \times n}$ is weights of hidden layer L, and $b^{(L)} \in \mathbb{R}^{1 \times n}$ is biases of layer L, this method can be interpreted as:

$$\mathbf{W}^{(L)}_{j \times i} \& \mathbf{b}^{(L)}_{1 \times j} \mapsto \mathbf{W}'^{(L)}_{m \times n} \& \mathbf{b}'^{(L)}_{1 \times m} \text{ where } \begin{cases} m > j \\ n > i \end{cases} \quad m, n \in \mathbb{N} \tag{2}$$

where $W^{(L)} \in \mathbb{R}^{m \times n}$ is weights of hidden layer L, and $b^{(L)} \in \mathbb{R}^{1 \times n}$ is biases of layer L. $(W^{'},b^{'})$ and (W, b) are weights and biases of large and small models, respectively.

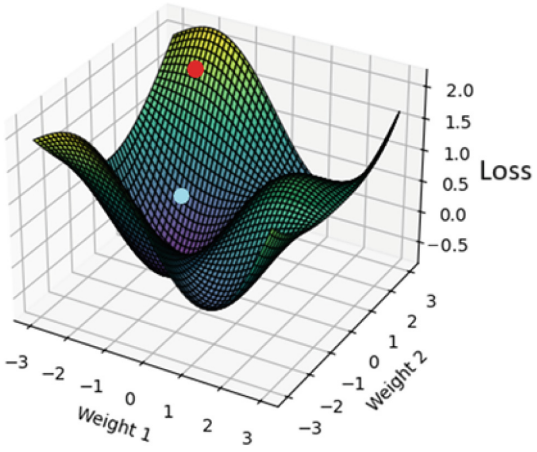


Fig. 4. The solution domain of weights and corresponding loss. Random initialization of the model (red dot). The transferred weights initialization (blue dot). (Color figure online)

The goal is to transfer the weights of a small model to the desired large model to have a good initialization instead of random initialization. As illustrated in

Fig. 4, the aim is starting optimization from blue point instead of red point with random initialization. This initialization is based on the input dataset and desired outputs. The random initialization of the model (red dot) will start from high loss (low accuracy) and need more iterations to reach the optimum point. The transferred weights initialization (blue dot) will start with better loss (accuracy) and need fewer iterations and energy consumption. To transfer weights to a larger model various methods have been tried. For example, transferring the weights of the smaller model and keeping the remaining weights zero or using various types of data distribution. The best method was using proportional uniform distribution (PUD) which the large model at initialization had close results to the small model after training.

The procedure of transferring weights from a small model to a large model $W \mapsto \mathbb{W}$ can be implemented by mapping each element of the small matrix to the corresponding segment of the large matrix as follows:

$$\mathbf{W}_{i\times j} \mapsto \mathbb{W}^{(i,j)}_{i'\times j'}, \text{where } \begin{cases} i' : row_scale \\ j' : col_scale \end{cases} \tag{3}$$

where $\mathbf{W}_{i\times j}$ is the element of the small matrix (Fig. 5 dark blue) and $\mathbb{W}^{(i,j)}_{i'\times j'}$ is the corresponding segment of the large matrix (Fig. 5 Light blue) with the size of $i' \times j'$, i' and j' are row and column scaled-up sizes based on the ratio of large to small models. So, each element of the corresponding segment in the large matrix is defined as follows:

$$\mathbb{W}^{(i,j)}_{i,j} = \frac{\mathbf{w}_{i,j}}{i' \times j'} \times \alpha \tag{4}$$

where $\mathbb{W}^{(i,j)}_{i,j}$ is the i, j element of the segment in the large matrix corresponding to the (i, j) element in the small matrix and α is a hyper-parameter that would apply to the transferred values with the aim of better efficiency.

So, all elements will be changed during the transferring procedure. It should be noted that this method is entirely different from what we call Progressive Training (or Progressive Growth) like the research carried out in [24] and our focus is on FC layers during training. Moreover, the training data are used for initialization not using general pretrained models which are trained with other data.

This method starts with training a small model size for a few epochs. Due to the small size, it consumes less energy and time for training. No need to have a large amount of epochs in this phase because the accuracy of the model will exponentially increase. Moreover, we are not interested in very high accuracy in this phase because we will lose some accuracy due to transferring our learning to the larger model in the next phase. Moreover, we are not willing to consume too much extra energy in this phase because it is an extra effort than just training a normal large model. The general concept of this method has been presented in Fig. 6 and the corresponding Pseudocode has been presented in Algorithm 1.

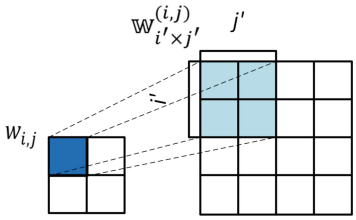


Fig. 5. Transferring weights from small matrix (left) to large matrix (right) corresponding to small and large models, respectively. The element of the small matrix (dark blue) will transfer to the larger matrix as a corresponding segment (light blue) with a size equal to the ratio of large to small matrix (model). (Color figure online)

Algorithm 1. Small-to-large: Weight Transfer

%Random initialization
Initialize the small MLP model
Set hyper-parameters
%Training
Initialize the large MLP model
Set hyper-parameters
%Transferlearning
    Transfer weight and bias values of small model to large matrices of large model with proportional distribution
%Training
    Training the large model
end

4 Experiments

The results of the proposed method have been presented in this section. All experiments have been executed using a standard laptop with the Hardware configuration: CPU Intel Core i7-13700H, 2.40 GHz, RAM 32 GB, Hard disk 1 TB. The MLP model was used for all experiments and MNIST [26] and CIFAR10 [25] datasets for training the models. To have a fair comparison among different cases, the same random initialization is used with the same base code, hyper-parameters, hardware, and no accelerator. The other random initializations were also tested to make sure the results were not an exception.

4.1 Small-to-Large: Initialization Using Transferring Weights with Proportional Uniform Distribution

The aim is to reach the same accuracy as a normal large model using the proposed method with less energy consumption. So, the simulations will be stopped once reach the same training energy and, approximately, time. At first, the small model with two hidden layers with a size of $(128, 200)$ was randomly initialized and trained using the MNIST dataset for only two epochs. Then, the weights

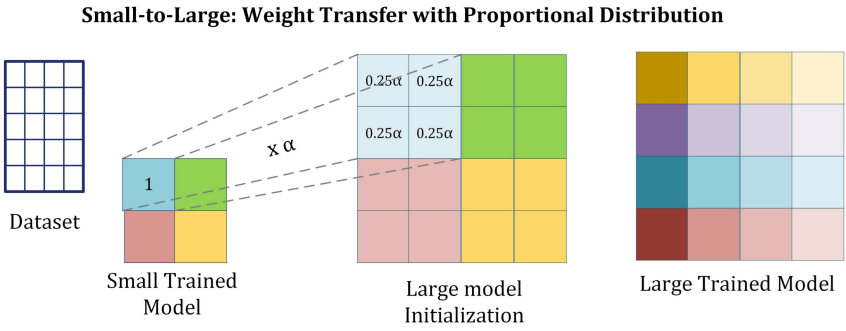


Fig. 6. The general concepts of energy consumption reduction of training MLP model: training a small model and transferring the weights to the larger model using a proportional uniform distribution with the aim of good initialization of large model and reach to the desired accuracy in less time and with less energy consumption.

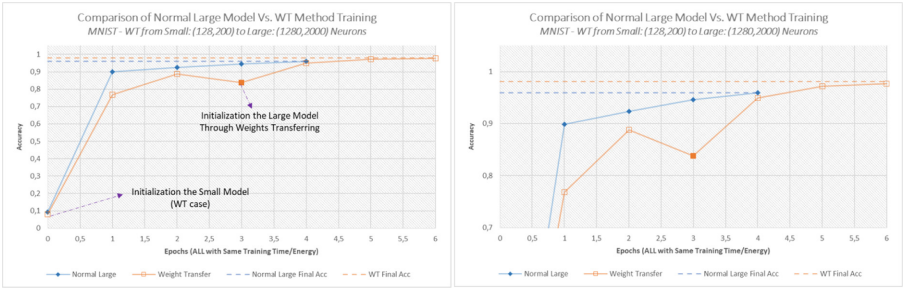


Fig. 7. Comparison of training results using MNIST dataset for normal large model and weights transferring methods with hidden layer sizes (1280,2000). The small model size used to initialize the large model is (128,200). All simulations stopped after the same amount of energy consumption (approximately the same time). The proposed method has better results.

are transferred to the large model using the proposed PUD method. The results have been illustrated in Fig. 7 and Table 2. It is obvious that the weight transfer has been carried out efficiently (accuracy drop from 0.88 to only 0.83) and that the accuracy of the large model at initialization is close to the accuracy of the small-size model at the final training epoch.

The large initialized model in Fig. 7 (orange line) has reached a higher efficiency than the normal large model (blue line) which means we can reach the same accuracy with less energy consumption. The proposed method reached the same accuracy as the normal large model after some iterations followed by the first epoch only with, approximately, 0.6 of energy consumption of training the normal large model (Table 2). It is to be noted that the initialization of the normal large model and proposed method are 0.09 and 0.83, respectively. The total epochs of the normal large model have been 4 and the large model after initialization reached better results only after a few iterations followed by the

first epoch. However, we should consider the energy consumed to train the small model which considerably consumes less energy than the larger model.

It is to be noted that the same idea of weight transferring was tried to transfer weights from a large model to a small model with the reverse way of using PUD with the aim of matrix compression. But the results were not desirable.

Table 2. Comparison of energy consumption and time of training the normal large model and the proposed method including training the small and initialized large models.

MNIST Dataset	Energy RAM (Wh)	Energy CPU (Wh)	Time (s)	Final Accuracy	Accuracy of Weight Transferring
Small Model 2 epochs (128,200)	0.022	0.099	$\tilde{1}2$	0.8883	-
Large Model < 4 epochs Initialized (1280,2000)	0.404	1.582	$\tilde{1}27$	0.9808	0.8883 to 0.8384
Large Model < 2 epochs Initialized (1280,2000)	0.242	0.8914	$\tilde{8}5$	0.9590	0.8883 to 0.8384
Normal Large Model 4 epochs (1280,2000)	0.421	1.669	$\tilde{1}40$	0.9591	-

To observe the effect of different model sizes on the proposed method, both utilized small and large model sizes have been changed and the results have been presented in Fig. 8. In this experiment, the training of the proposed method has been stopped in the 10th epoch but without full iterations. The result of the proposed method is better than the normally trained model. The reason for different initialization accuracies is also different model sizes. In this case, the proposed method had also better results than the normally trained model.

To see the effect of the proposed method in different model structures, one hidden layer was added to the model. The proposed method has slightly better results in comparison to training the normal large model (Fig. 9). As is obvious, the trend of the accuracy of the proposed method (orange line) has a little fluctuation. Changing the learning rate can help to have a better final accuracy. The other point is that the advantage of the proposed method over the normal training of the model reduces in large epochs. Moreover, the small model is still too large which consumes lots of energy. So, reducing the hidden layer sizes can help to reduce the required energy consumption for the initialization of the large model while maintaining the quality of transferred weights.

In all previous cases, the small model had better results than the large model. So, that was the reason that the small model with few epochs reached a good accuracy which has a direct effect on the good initialization of the large model. In this case, as it is obvious in Fig. 10, the small model (blue line) has worse efficiency in comparison to the large model (orange line). This is the reason

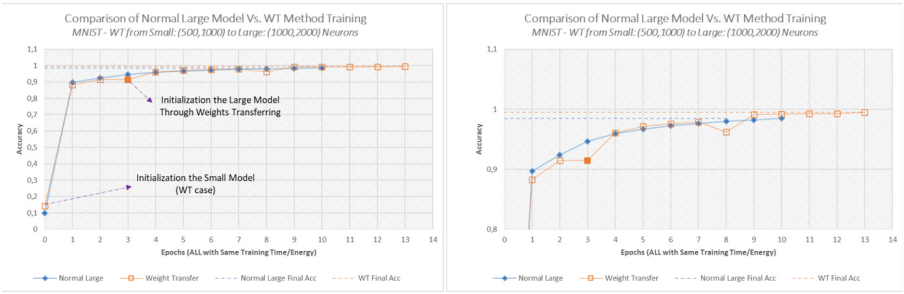


Fig. 8. Comparison of training results using MNIST dataset for normal large model and weights transferring methods with hidden layer sizes (1000,2000). The small model size used to initialize the large model is (500,1000). All simulations stopped after the same amount of energy consumption (approximately the same time). The proposed method has better results.

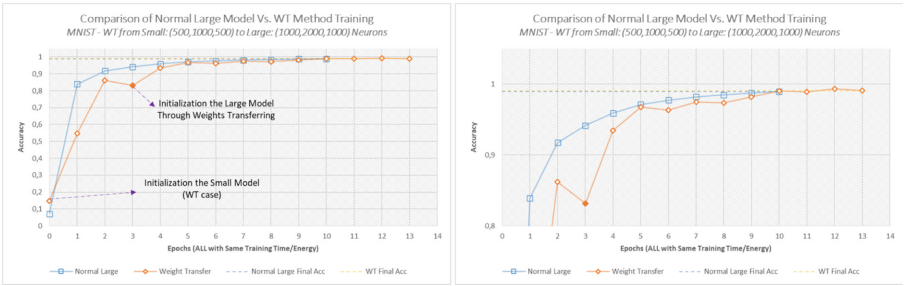


Fig. 9. Comparison of training results using MNIST dataset for normal large model and weights transferring methods with hidden layer sizes (1000,2000,1000). The small model size used to initialize the large model is (500,1000,500). All simulations stopped after the same amount of energy consumption (approximately the same time). The proposed method has slightly better results.

that the initialization of a large model has less accuracy in comparison to the previous cases. However, the final accuracy of the proposed method (gray line) is slightly better than the large model and much better than the small model. In this case, both small and large models are small and consume less amount of energy. So, training the smaller model does not have a prominent advantage from the viewpoint of energy consumption (Fig. 11 left oval).

The proposed method can be applied to any library. In this case, the method is applied to the Keras library [27] and the results are illustrated in Fig. 12. The energy consumption of model training using Keras is less than the code written by the author which makes it possible to train the small model with higher epochs. However, the effect of using the method has similar results in both codes.

To see the effectiveness of the proposed method on different datasets, the CIFAR-10 dataset has been selected to train the model with many more epochs.

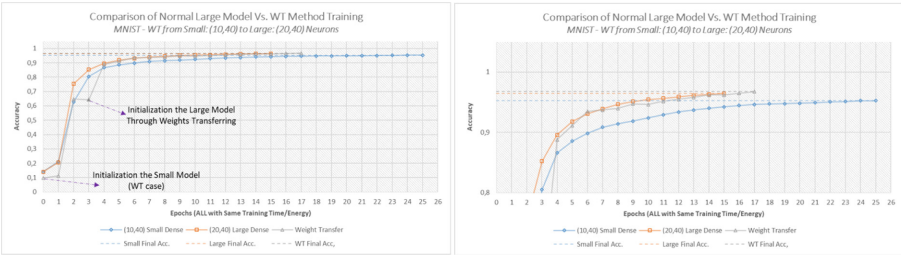


Fig. 10. Comparison of training results using MNIST dataset for normal large model, normal small model, and weights transferring methods with hidden layer sizes (20,40). The small model size used to initialize the large model is (10,40) and it has worse efficiency in comparison to the large model. All simulations stopped after the same amount of energy consumption (approximately the same time). The proposed method has better results.

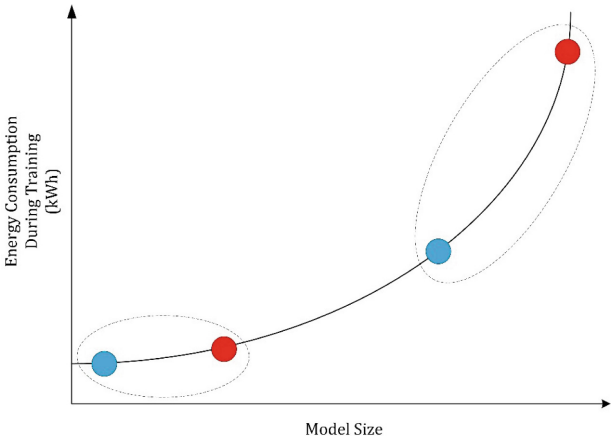


Fig. 11. The difference in energy consumption of small (blue circle) and larger (red circle) models when their sizes are both small (left oval). By increasing the model size, the difference in energy consumption of small and large models will be high due to the energy consumption increase of both models exponentially (right oval). (Color figure online)

The efficiency of the proposed method is obvious in Fig. 13. The proposed method (orange lines) has better results than the normal large model (blue line) during the whole training period. It should be noted that the small model trained over 100 epochs to be able to initialize the large model efficiently. It can be implied that the proposed method has better results when dealing with large datasets and models. The main reason has roots in increasing the energy consumption of model training exponentially. So, through training the small model, which consumes much less energy, with the aim of initialization of the large model, we can save lots of energy.

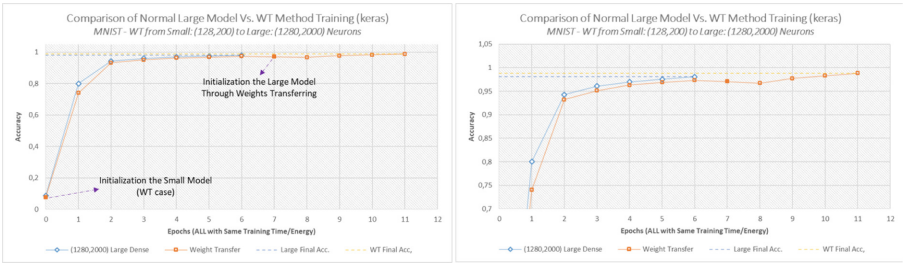


Fig. 12. Comparison of training results using MNIST dataset for normal large model and weights transferring methods with hidden layer sizes (1280,2000) using Keras library. The small model size used to initialize the large model is (128,200). All simulations stopped after the same amount of energy consumption (approximately the same time). The proposed method has better results.

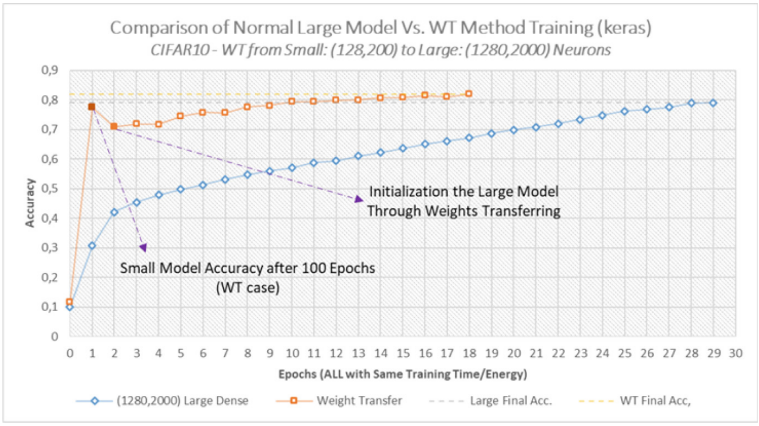


Fig. 13. Comparison of training results using MNIST dataset for normal large model and weights transferring methods with hidden layer sizes (1280,2000) using Keras library. The small model size used to initialize the large model is (128,200). All simulations stopped after the same amount of energy consumption (approximately the same time). The proposed method has better results.

5 Conclusion and Future Work

In this work, training the MLP model with FC layers has been investigated from the viewpoint of energy consumption. The exponential energy consumption increasing of training the DL models was the main incentive to propose the method. Based on that, the small model with low energy consumption is trained, and using the proposed PUD method its weights will be transferred to the larger model with the aim of better initialization. So, the larger model will need fewer epochs, and consequently, less energy consumption, to reach the same accuracy as the larger model. The method has been tested in various

cases. It was observed that the proposed method has better efficiency in dealing with larger models and datasets. Moreover, the efficiency of the method will decrease in the cases where the small model has too low efficiency in comparison to the larger model. The results of applying the model on the Keras package show the efficiency of applying the method on other packages and libraries. One of the challenges in comparing the energy consumption of different methods is having a fair comparison. This includes the same base code of the model, the same hardware, using the same backend and accelerator, having the same data, model structure, random parameters, etc. So, comparing the results of different methods of the same group will be easier than comparing the results of various methods. This can be a good incentive for future work to have a fair comparison among all well-known available methods. Moreover, in some cases, the convergence of the proposed method accuracy during training had fluctuation which can be mitigated by modifying the hyper-parameters like learning rate. Finding a method to define optimally the hyper-parameters or a better optimization method can improve the method's efficiency.

Acknowledgments. This project has received funding from the European Union's Important Project of Common European Interest - Next Generation Cloud Infrastructure and Services (IPCEI CIS), in collaboration with the Netherlands Enterprise Agency (RVO).

References

1. Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. J. Mach. Learn. Res. **23**, 1–39 (2022)
2. Desislavov, R., Martínez-Plumed, F., Hernández-Orallo, J.: Compute and energy consumption trends in deep learning inference, arXiv [cs.LG], 12-Sep-2021
3. Yang, T.-J. ., Chen, Y.-H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning, arXiv [cs.CV], 15-Nov-2016
4. Yang, T.-J., Chen, Y.-H., Emer, J., Sze, V.: A method to estimate the energy consumption of deep neural networks," 2017 51st Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2017, pp. 1916–1920. https://doi.org/10.1109/ACSSC.2017.8335698.
5. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017)
6. Dai, B., Zhu, C., Wipf, D.: Compressing neural networks using the variational information bottleneck. In: International Conference on Machine Learning (2018)
7. Courbariaux, M., Bengio, Y., David, J.P.: BinaryConnect: training deep neural networks with binary weights during propagations. In: Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2, Montreal, Canada, 2015, pp. 3123–3131 (2015)
8. Velez, D., Santa, S., Patino, G.: MLP neural network based on PCA and K-means clustering for PM2.5 forecasting. In: Computing, Internet of Things and Data Analytics, 2024, pp. 202–212 (2024)
9. Spurlock, K., Elgazzar, H.: A genetic mixed-integer optimization of neural network hyper-parameters. J. Supercomput. **78**(12), 14680–14702 (2022)

10. Courty, B., et al., 'mlco2/codecarbon: v2.4.1'. Zenodo (May 2024)
11. He, Y., Xiao, L.: Structured pruning for deep convolutional neural networks: a survey. IEEE Trans. Pattern Anal. Mach. Intell. **46**(5), 2900–2919 (2024). https://doi.org/10.1109/TPAMI.2023.3334614
12. Mocanu, D.C., et al.: Sparse Training Theory for Scalable and Efficient Agents, arXiv [cs.AI], 02-Mar-2021
13. Koanantakool, P., et al.: Communication-avoiding parallel sparse-dense matrix-matrix multiplication. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2016)
14. Wheatman, B., Madhyastha, M., Burns, R.: Masked matrix multiplication for emergent sparsity. ArXiv:abs/2402.14118 (2024)
15. Wesselink, W., Grooten, B., Xiao, Q., de Campos, C., Pechenizkiy, M.: Nerva: a truly sparse implementation of neural networks, arXiv [cs.LG]. (2024)
16. El-Hassani, F.Z., Amri, M., Joudar, N.-E., Haddouch, K.: A new optimization model for mlp hyperparameter tuning: modeling and resolution by real-coded genetic algorithm. Neural Process. Lett. **56**(2), 105 (2024)
17. Dietterich, T.G.: Ensemble methods in machine learning. In: multiple classifier systems, 2000, pp. 1–15 (2000)
18. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS 2014 Deep Learning Workshop, arXiv [stat.ML] (2015)
19. Wang, Y., et al.: 'Revisiting the transferability of supervised pretraining: an MLP perspective. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) **2022**, pp. 9173–9183 (2022)
20. Gale, T., Zaharia, M., Young, C., Elsen, E.: Sparse GPU kernels for deep learning. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2020)
21. Nvidia, 'CuSPARSE'. https://docs.nvidia.com/cuda/cusparse/. Accessed 12 Jan 2025
22. Virtanen, P., et al.: SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat. Methods **17**(3), 261–272 (2020)
23. Abadi, M., et al.:'TensorFlow: a system for large-scale machine learning'. In: 12th $USENIXSymposiumonOperatingSystemsDesignandImplementation(OSDI$ 16), pp. 265–283 (2016)
24. Gu, X., Liu, L., Yu, H., Li, J., Chen, C., Han, J.: On the Transformer growth for progressive BERT training, arXiv [cs.CL], 23-Oct-2020
25. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images, University of Toronto (05 2012)
26. Deng, L.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Process. Mag. **29**(6), 141–142 (2012)
27. Chollet, F., et al.:'Keras' (2015). https://github.com/fchollet/keras