Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Due Date: 21 April 2021
ECSE 552 - Course Project

OXFORD

ECSE 552 - Deep Learning

Exploring Super-Convergence in Analog Hardware Acceleration Kit for In-memory Computing Design

Hung-Yang Chang / Jingxu Hu / Alexander J. Fernandes / Mian Hamza

[1]Department of Electrical and Computer Engineering, McGill University, Montreal, PQ, H3A 0G4, Canada

Abstract

Motivation: To mitigate the Von Neumann's bottleneck and accelerate the training time of Neural network applications, we propose to implement hardware-software co-optimization approach with in-memory computing architectures and super-convergence algorithm.
Results: We demonstrated that the super-convergence algorithm improves the analog neural network performance by mitigating convergence of local minimums producing higher accuracies at lower epochs.
Availability: Code freely available at `https://github.com/HungYangChang/ECSE552`
Contact: {hung-yang.chang, jingxu.hu, alexander.fernandes, mian.hamza}@mail.mcgill.ca

1 Introduction

Deep Neural Networks (DNNs), a popular AI/ML algorithm, has achieved promising results in recent years in many complex tasks such as image/speech recognition and adaptive AI responses in game engines. However, training large DNNs is considered a computationally intensive task that demands extensive use of computational resources for days at a time. One reason being is that the computation and memory endpoints are in different locations on a chip, and data is transferred back and forth between storage and computation units frequently. This phenomenon is known as the so-called **Von Neumann's bottleneck** or *Memory Wall* as shown in Ielmini and Wong (2018) which prevents a system from achieving real-time and energy-efficient computations. Thus, to mitigate this bottleneck and accelerate the training time, we propose to implement a hardware-software co-optimization approach with in-memory computing architectures and super-convergence algorithm as follows:

(1) From a hardware perspective, in-memory computing architectures: In-memory computing architectures, such as analog-based devices proposed in Ambrogio *et al.* (2018), are designed to eliminate this bottleneck. This is because the analog devices contain Resistive Processing Units (RPUs) that can perform multiplications and additions using Ohm's law and Kirchhoff's law respectively. This enables both computation and storage components to be within a single chip. As shown in Fig. 1, computations can be done inside a memory block in single access of ALU. Nevertheless, the circuit non-ideality issues remain a daunting problem when implementing analog devices with an in-memory computing design as shown in Gokmen and Vlasov (2016). Recently, IBM (2020) has proposed **Analog Hardware Acceleration Kit** provides a platform for simulating in-memory computation non-idealities of analog devices in Python. This tool could simulate non-idealities such as the device-to-device variations, cycle-to-cycle variations, and asymmetry for increasing and decreasing updates. Furthermore, due to the analog nature of the RPUs, during each update phase, all calculations experience noise from the environment modeled as a stochastic pulse with a step constant for a selected material. This is modeled as the different types of noises such as Additive White Gaussian Noise Additive White Gaussian Noises (AWGNs) during the weight update.

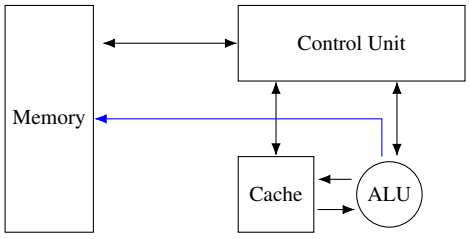


Fig. 1: Concept of in-memory computing architecture in IBM (2020)

(2) From the software perspective, super-convergence algorithm: Smith and Topin (2018), and Smith (2017) have demonstrated that Neural Networks (NNs) could converge to a high accuracy in a few epochs (less than 50) by using large cyclical learning rate policies. Furthermore, when the amount of labeled training data is limited, the super convergence algorithm can provide a greater boost in performance relative to standard training. Therefore, this method demonstrates the possibility that NNs can be trained in an order of magnitude faster than standard methods.

The remainder of this paper is organized as follows. In section 2, we will explain how neural networks are constructed with RPU from the

IBM tool-kit. Then, we will discuss which RPU configurations will be used throughout our work and finally how we will perform the super-convergence algorithm with IBM's Analog Hardware Acceleration Kit. In section 3, simulation results and discussion will be shown comparing the baseline model's performance vs all other models' performances across different data sets. Finally, in section 4, concluding remarks about the results discussed in the previous section will be stated along with future possible avenues of work.

2 Methods

The main goal of this project is to evaluate whether if the super-convergence phenomena can be demonstrated within analog NN to achieve hardware-software co-optimization. The IBM tool setup is shown in section 2.1. The super convergence algorithm is shown in section 2.2. As for the models, we've decided to adopt ResNet proposed in He *et al.* (2015) as the main structure and other models such as LeNet & VGG shown in section 2.3. In this work, we focus only on the training phase of the network since this is the most time-consuming and computationally expensive phase. Furthermore, the experiments would be constructed in Python with both the IBM tool kit and Pytorch frameworks.

2.1 IBM Analog Hardware Acceleration Kit Setup

The IBM's Analog Hardware Acceleration Kit emulates in-memory analog NN. The model implements RPUs, which are analog weights that can represent values and be updated locally during the weight update step. As shown in figure 2, the overall network adheres to Pytorch's sequential structure but the per-layer and optimizers are replaced with analog layers and analog optimizers. Here, RPU is a concept, reflecting the behavior of an actual device, which could be resistive switching in device Ielmini and Wong (2018), phase-change memory (PCM) in Ambrogio *et al.* (2018), or other analog devices. For detailed documentation about the RPU unit please refer to Gokmen and Vlasov (2016).

All the network operations such as forward backward propagation and weight updates could be placed into the GPU with different RPU configurations. For the forward and backward propagation, multiplications and additions could be modeled by Ohm's law and Kirchhoff's law respectively in analog devices. In the weight update step, it used stochastic computations. The update rule for any weight ω_{ij} is shown in Eq. (1).

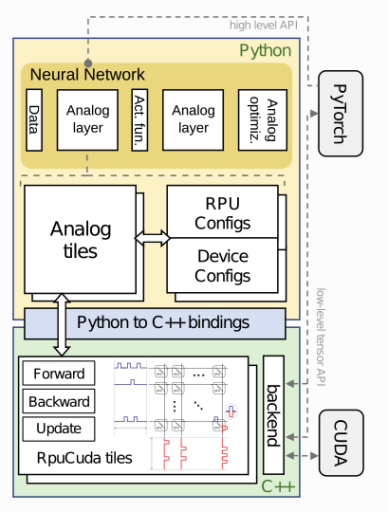


Fig. 2: Architecture of IBM AI hardware kit in IBM (2020)

$$\omega_{ij} \leftarrow \omega_{ij} + \Delta\omega_{min} \sum_{n=1}^{BL} A_i^n \wedge B_j^n \,, \tag{1}$$

where BL is the length of the stochastic bit stream that is used during the update cycle, $\Delta\omega_{min}$ is the minimum update step size, A_i^n and B_j^n are the random variables that are characterized by Bernoulli process with n representing the position in trial sequence.

The absolute values of weights are $|w| \leq 1$. Since the maximum value of each device that can be represented is different, IBM tool provides a parameter *weight omega* to define the value where the max weight will be scaled to. The IBM libraries provide several device configurations. Among them, we apply the following ones in our experiments.

- **Gokmen Vlasov device configuration**
  As defined in Gokmen and Vlasov (2016), this device setting includes the cycle-to-cycle variation device-to-device variation and both are given in relative units. Cycle-to-cycle variation is defined with parameter *dw_min_std* $= 0.3$, meaning 30% spread around the mean. This results in a cycle-to-cycle standard deviation $\sigma_{c-to-c} =$ *dw_min* $\times$ *dw_min_std*, where *dw_min* is mean of the minimal update step sizes across devices and directions. As for device-to-device variation, *dw_min_dtod* $= 0.3$ is defined to implement device-to-device variation of minimal update step. Moreover, each device experiences a different up or down effect due to device-to-device variation. [1] This variation is modeled by parameter *up_and_down_dtod* and set to be 0.01.

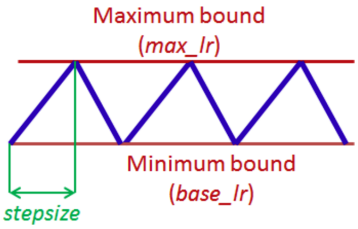
- **Semi-ideal device configuration**
  In this device, it is assumed that all settings are the same as Gokmen & Vlasov's configuration except for some certain differences. Firstly, weight update device-to-device variation is ignored. (i.e. *up_and_down_dtod* $= 0$). In physical circuit terms, this might be implemented as a difference of two resistive elements. Another difference is that *dw_min* it is set to 2×10^{-4}, which provides a more accurate weight update in each step.

2.2 Super Convergence Algorithm

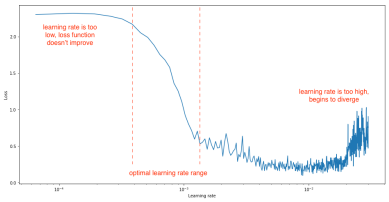
One way to accelerate the training phase on most architectures is to simply use a larger learning rate. For instance, Smith and Topin (2018) proposed the super-convergence algorithm approach that allows certain networks to achieve a learning rate as high as an order of magnitude larger than its usual starting learning rates. Thus, this dramatically reduces training time by a significant amount of epochs. However, exceptionally large learning rates can prevent networks from converging. Therefore, the super-convergence algorithm refers to the cyclical learning rate (CLR) policy proposed in Smith (2017) where it ensures a one-cycle CLR policy to eliminate this issue. In CLR policy, each CLR cycle consists of two steps; the lr increases and then the lr decreases. This happens between the provided minimum and maximum learning rate values as seen in figure 3a. The one-cycle policy is just a slight modification of the CLR policy. The intuition behind one-cycle CLR policy is that the learning rate initially starts small to allow convergence in the correct direction in the latent space. As the network traverses the flat valley, the learning rate is large allowing for swift progress through the valley. Then, in the final stages of training, when the network

[1] Only device-to-device variation is considered here. Although the device may also experience asymmetric non-linearities during the weight update phase, this effect is ignored in this setting. Therefore, it is assumed that one up and one down weight update has the same effect and perfectly symmetrical.

needs to converge to a local minimum the learning rate is then once again reduced to a small value.



(a) Cyclical learning rate policy Smith (2017)



(b) Learning rate range test Smith and Topin (2018)

Fig. 3: Super convergence explanation for learning rate.

Furthermore, one issue pertains to knowing how to exactly choose the global learning rates for neural networks. Super-convergence introduces the concept of the learning rate range (LR Range) test to determine optimal learning rate ranges for CLR. LR test starts training with a very small learning rate and increases it linearly over time. Once the learning rate becomes too large the network will fail to converge and the performance will drop as well. Thus, there is a maximum lr value just before such a decline where the accuracy was the largest. This value is proposed as a maximum learning rate to be used in CLR according to figure 3b shown from Howard (2021). As for the minimum learning rate, it is recommended to use a value that is 3–4 times less than the maximum learning rate used. LR test provides the advantage of eliminating the need to perform a vast amount of experiments to find the best values with no additional computation.

The other challenge of super-convergence is to find a reasonable amount of regularization to be used with a CLR. It is known that the definition of regularization is any modification made to a learning algorithm that is intended to reduce its generalization error. According to Smith (2017), it shows an increasing training loss and decreasing test loss while the learning rate increases from approximately 0.2 to 2.0 in ResNet architecture with CIFAR-10 dataset. This implies that regularization is happening while training with these large learning rates. Therefore, it is essential to modify or remove some regularization methods to balance the regularization effect. The optimal hyper-parameter setting is found to use small weight decay (e.g. $1 \times 10-4$) and cyclical momentum. For example, a large cyclical learning rate may be paired with a smaller momentum and vice versa. For a more in-depth explanation in this topic, we refer viewers to Smith (2017).

2.3 Model

2.3.1 ResNet

Deep Residual networks are a type of deep convolution network that use residual learning building blocks. These blocks offer skip connections between layers He *et al.* (2015). Shown in Fig. 4 is how each architecture is constructed from ResNet-18 to Resnet-152.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
	56×56	3×3 max pool, stride 2				
conv2_x		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig. 4: Architecture for Resnet of each layer size obtained from table 1 in He *et al.* (2015)

2.3.2 LeNet

The LeNet architecture is given in Table 1. Each Conv2d parameters are kernel size=5x5, stride=1 and zero padding. Each MaxPool2d parameters are kernel size=2x2, stride=2 and zero padding. These parameters were taken directly from Y. LeCun and Haffner. (1998). The final 3 layers of LeNet are fully connected layers with a decreasing number of features, with the last layer containing 10 output features: the amount of classes in the dataset. The datasets used to test this model were *MNIST*. It represents simple image classification difficulties. Thus, a 7 layer model would generalize well to *MNIST* and avoid the risk of over-fitting if deeper CNN architectures were used, such as VGG-16 from Simonyan and Zisserman (2015).

Table 1. LeNet

Layer Size	**NNs Type**	**Activation Function**
1, 16	Conv2d	TanH
16, 16	MaxPool2d	TanH
16, 32	Conv2d	TanH
(32, 64) -> (64, 128)	MaxPool2d, Flatten(32,64), Linear	TanH
128, 10	Linear	Softmax

2.3.3 VGG-8

The VGG-8 architecture model is given in Table 2. Each Conv2d parameters are kernel size=(3,3), stride=(1,1) and padding=(1,1). Each BatchNorm2d parameters are eps=1e-05, momentum=0.1. Each MaxPool2d parameters are kernel size=2, stride=2, padding=0 and dilation=1. These parameters were taken directly from Simonyan and Zisserman (2015). In their paper, the authors didn't specify a specific 8 layer architecture thus we took inspiration from their layer specifications and created our own model. The number of hidden units per layer from Simonyan and Zisserman (2015) were adapted as well. The data sets used to test this model were *SVHN* and *Cifar10*. They represent intermediate to hard image classification difficulties. Thus, an 8 layer model would generalize well to these data sets and avoid the risk of over-fitting if deeper VGG architectures were to be used from Simonyan and Zisserman (2015).

2.4 Data sets

To verify the super convergence phenomena within the IBM tool-kit, several experiments were performed on the chosen data sets below. These data sets incorporate different image classification difficulties ranging between "easy" (MNIST) and "hard" (CIFAR-10).

Table 2. VGG-8

Layer Size	NNs Type	Activation Function
3, 128	Conv2d	ReLU
128, 128	Conv2d, BatchNorm2d	ReLU
128, 256	MaxPool2d, Conv2d	ReLU
256, 256	Conv2d, BatchNorm2d	ReLU
256, 512	MaxPool2d, Conv2d	ReLU
512, 512	Conv2d, BatchNorm2d	ReLU
(512, 8192) -> (8192, 1024)	MaxPool2d, Flatten(512,8192), Linear	ReLU
1024, 10	Linear	LogSoftmax

2.4.1 SVHN

One major real-world image classification task involves recognizing numbers found in natural scenes. For instance, street numbers and residential civic numbers. This data set comprises of more than $600,000$ labeled images of different colors, orientations, and sizes of residential number plates. Each image is a 3-channel RGB 32x32 resolution PNG photo. The data set involves a 10 category classification where digit "0" has label 10 and digits "1" through "9" are assigned labels 1 through 9 respectively. SVHN represents an intermediate image classification difficulty according to Yuval Netzer and Ng (2011).

2.4.2 CIFAR-10

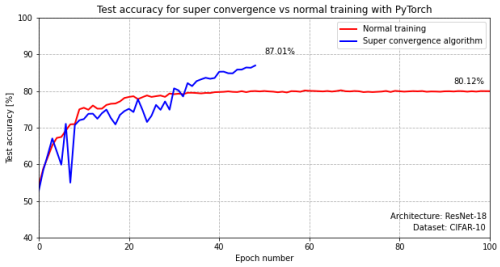
Another major image classification task involves discerning real-world objects from one another found in natural scenes. For example, an image of a cat would be very different from an image of a truck. This data set comprises of $60,000$ labeled 3-channel RGB photos of real-world objects. It also involves a 10 category classification task where the labels range from 0 to 9. Each label is associated with a specific category, for instance, label 0 = "airplane". CIFAR-10 represents a difficult image classification task according to Krizhevsky and Hinton (2009) where the classification does not remain in a single object domain like MNIST and SVHN.

2.4.3 MNIST

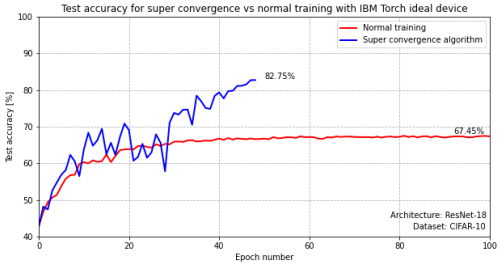
The MNIST dataset comprises of handwritten digits ranging from "0" through "9". The dataset is organized into $60,000$ training and $10,000$ test images. All the images are labeled. Each image is a gray-scale single channel 28x28 resolution PNG photo. The dataset involves a 10 category classification where each class represents an integer label ranging from 0 to 9. This data set involves the easiest classification task according to Y. LeCun and Haffner. (1998).

3 Results and Discussion

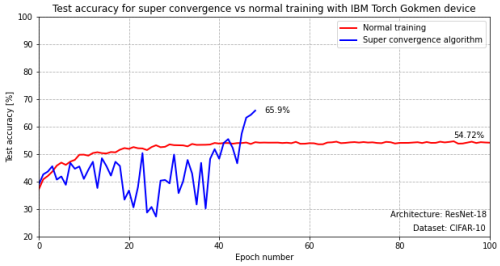
Table 3 shows the results for testing super convergence (rows with CLR) compared with normal training (rows with step) trained using the normal device, semi-ideal device, and Gokmen Vlasov device configurations. The table shows four configurations of dataset trained on each neural network architecture using the given parameters. The accuracies represent the average and standard deviation for the given epoch. Across all datasets and architectures, super-convergence produced higher test accuracies compared to normal training for all device configuration comparisons. This shows strong evidence that super convergence benefits convolutional neural networks in image classification problems and will improve results for semi-ideal and Gokmen Vlasov device configurations. The largest improvement in terms of test accuracies made with super-convergence was with CIFAR-10 ResNet-18. Using the semi-ideal device, the accuracy improved by 16% when comparing CLR of 50 epochs to the regular stepLR



(a) Test accuracy with PyTorch



(b) Test accuracy with IBM Torch ideal device



(c) Test accuracy with IBM Torch Gokmen device

Fig. 5: Test accuracy for super convergence vs normal training with different devices

of 200 epochs. A possible reason for this is that the skip networks converge to a local minimum more frequently when noise is introduced to the device. However, super-convergence can counter local minimum convergences by varying the learning rate between the minimum and maximum bounds. This allows for the optimizer to bypass local minimum convergences.

Improvements can be seen in the other datasets and architectures from super convergence. For SVHN and MNIST datasets, despite being easier classification problems, accuracies are lower with semi-ideal and Gokmen Vlasov device configurations and are improved after using super convergence. VGG-8 had higher accuracies than ResNet-18 for semi-ideal and Gokmen Vlasov devices, however, it took longer to train. Fig. 5 and 6 show how the use of CLR prevents the test accuracy and loss to converge. It bypasses the converged value around the 40-50th epoch for normal, semi-ideal, and Gokmen Vlasov device configurations.

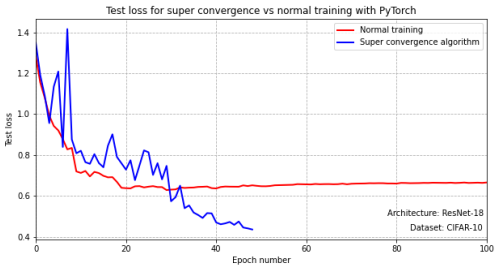
4 Conclusion

In this study, we compared super convergence cyclic learning rate experiments to normal learning rate per step training on 4 separate data sets and 3 different models. Across each of the 4 independent experiments, we

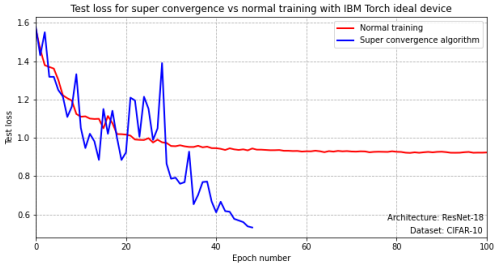
Dataset	Architecture	PL/Setting	CM/SS	WD	Batch Size	Epoch	Accuracy N (%)	Accuracy SI (%)	Accuracy GV (%)
Cifar-10	ResNet-18	step/0.01/0.5/10	0.9	10^{-4}	512	100	80.01 ± 0.10	66.74 ± 0.52	54.32 ± 0.37
Cifar-10	ResNet-18	step/0.01/0.5/10	0.9	10^{-4}	512	200	79.96 ± 0.09	66.75 ± 0.38	54.44 ± 0.55
Cifar-10	ResNet-18	CLR/0.1-0.5/12	0.95-0.85/12	10^{-4}	512	25	83.88 ± 0.36	79.92 ± 0.27	58.83 ± 1.69
Cifar-10	ResNet-18	CLR/0.1-0.5/23	0.95-0.85/23	10^{-4}	512	50	86.59 ± 0.31	82.98 ± 0.16	63.95 ± 0.88
Cifar-10	VGG-8	step/0.01/0.1/10	0.9	10^{-4}	128	50	87.47 ± 0.12	84.20 ± 0.46	75.02 ± 0.68
Cifar-10	VGG-8	CLR/0.01-0.15/12	0.95-0.85/12	10^{-4}	128	25	88.88 ± 0.18	86.16 ± 0.50	79.92 ± 0.99
SVHN	VGG-8	step/0.01/0.1/10	0.9	5×10^{-4}	128	50	95.01 ± 0.01	93.89 ± 0.01	88.53 ± 0.28
SVHN	VGG-8	CLR/0.01-0.15/12	0.95-0.85/12	5×10^{-4}	128	25	95.71 ± 0.04	93.04 ± 0.78	91.74 ± 0.08
MNIST	LeNet	step/0.01/0.94/2	0.9	5×10^{-4}	512	50	84.44 ± 3.38	79.78 ± 0.02	82.99 ± 3.38
MNIST	LeNet	step/0.01/0.94/2	0.9	5×10^{-4}	512	85	92.97 ± 0.34	92.02 ± 0.27	91.92 ± 0.29
MNIST	LeNet	CLR/0.01-0.1/5	0.95-0.85/5	5×10^{-4}	512	12	98.59 ± 0.03	89.89 ± 0.38	90.07 ± 1.11
MNIST	LeNet	CLR/0.01-0.1/12	0.95-0.85/12	5×10^{-4}	512	25	98.89 ± 0.06	93.74 ± 0.36	94.25 ± 0.14

Table 3. Final accuracy and standard deviation for the specified datasets and architectures used for these experiments. Each experiment is run three times to generate the mean and standard deviation of each accuracy. For the given cyclic learning rate (CLR) the minimum and maximum learning rate is given across the step size length. PL is learning rate policy. For step LR, the order of setting is starting learning rate, gamma, and step size (SS) in epochs. As for CLR, the order of setting is base learning rate to maximum learning rate and step size in epochs. WD is weight decay and CM is cyclic momentum. For each of the models with the given parameters, three accuracies are generated from seperate experiments conducted for normal device configuration (N), semi-ideal device configuration (SI) and Gokmen Vlasov device configuration (GV).

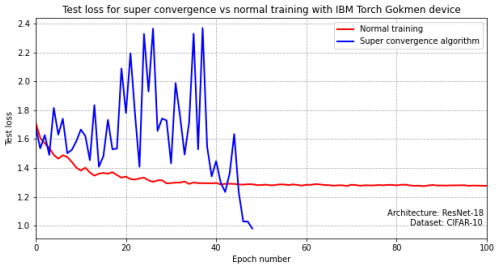
observed that super convergence improved the accuracy for normal, semi-ideal, and Gokmen Vlasov device configurations. The largest noticeable improvement was for the semi-ideal device for CIFAR-10 ResNet-18. It improved the test accuracy by 16% (67% to 83%) after the 50th epoch. This study shows that even though when noise was introduced to the weight update steps in the IBM tool kit, the super convergence trained neural networks was still able to avoid non-ideal local minimums. Utilizing the super convergence method along with removing the Von Neumann bottleneck with IBM analog hardware will improve the speed and accuracy of deep learning training for future studies. This study was limited to using google colab GPU devices. Future studies should involve using more powerful hardware and should be conducted on other deep learning applications (ex: sentiment analysis).



(a) Test loss with PyTorch



(b) Test loss with IBM Torch ideal device



(c) Test loss with IBM Torch Gokmen device

Fig. 6: Test loss for super convergence vs normal training with different devices

References

Ambrogio, S. *et al.* (2018). Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, **558**.

Gokmen, T. and Vlasov, Y. (2016). Acceleration of deep neural network training with resistive cross-point devices: Design considerations. *Frontiers in Neuroscience*, **10**, 333.

He, K. *et al.* (2015). Deep residual learning for image recognition.

Howard, J. (2021). Fast ai course. https://course.fast.ai/.

IBM (2020). Ibm analog hardware acceleration kit. https://github.com/IBM/aihwkit.

Ielmini, D. and Wong, H.-S. P. (2018). In-memory computing with resistive switching devices. *Nature Electronics*, **1**(6), 333–343.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.

Smith, L. N. (2017). Cyclical learning rates for training neural networks.

Smith, L. N. and Topin, N. (2018). Super-convergence: Very fast training of neural networks using large learning rates.

Y. LeCun, L. Bottou, Y. B. and Haffner., P. (1998). Gradient-based learning applied to document recognition.

Yuval Netzer, Tao Wang, A. C. A. B. B. W. and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.