Advances in Artificial Intelligence and Machine Learning; Perspective 1 (1) 958-976 Received 5-2-2023; Accepted 23-3-2023; Published 30-3-2023

General Cyclical Training of Neural Networks

Leslie N. Smith leslie.smith@nrl.navy.mil
U.S. Naval Research Laboratory,
Naval Center for Applied Research in AI, Code 5514
4555 Overlook Ave., SW.,
Washington, D.C. 20375

Corresponding Author: Leslie N. Smith

Copyright © 2023 Leslie N. Smith. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

This position paper describes the principle of "General Cyclical Training" in machine learning, where training starts and ends with "easy training" and the "hard training" happens during the middle epochs. We propose several manifestations for training neural networks, including algorithmic examples (via hyper-parameters and loss functions), data-based examples, and model-based examples. Specifically, we introduce several novel techniques: cyclical weight decay, cyclical batch size, cyclical focal loss, cyclical softmax temperature, cyclical data augmentation, cyclical gradient clipping, and cyclical semi-supervised learning. In addition, we demonstrate that cyclical weight decay, cyclical softmax temperature, and cyclical gradient clipping (as three examples of this principle) are beneficial in the test accuracy performance of a trained model. Furthermore, we discuss model-based examples (such as pretraining and knowledge distillation) from the perspective of general cyclical training and recommend some changes to the typical training methodology. In summary, this paper defines the general cyclical training concept and discusses several specific ways in which this concept can be applied to training neural networks. In the spirit of reproducibility, the code used in our experiments is available at `https://github.com/lnsmith54/CFL`.

Keywords: Deep learning, Training neural Networks, Curriculum learning, Cyclical hyper-parameters

1. INTRODUCTION

Deep neural networks lie at the heart of many of the artificial intelligence applications that are ubiquitous in our society. Over the past several years, methods for training these networks have become more automatic [1–5], but still remain more an art than a science. This paper introduces the high-level concept of general cyclical training as another step in making it easier to optimally train neural networks. We argue that many of the settings that are held constant throughout training need not be and training improves when they are not constant.

We define general cyclical training as any collection of settings where the training starts and ends with "easy training" and the "hard training" happens during the middle epochs. In other words,

Citation: Leslie N. Smith. General Cyclical Training of Neural Networks Advances in Artificial Intelligence and Machine Learning. 2022;3(1):57.

it can be considered as a combination of curriculum learning [6], in the early epochs with fine-tuning toward the end of training, plus training over the full problem space for greater generalization happening during the middle epochs. General cyclical training is analogous to curriculum learning in the sense that numerous specific techniques can embody its principles. The difference is that curriculum learning resembles learning in childhood, while general cyclical training resembles a human lifetime of learning.

It has been shown that many important aspects of neural network learning take place within the very earliest iterations or epochs of training [7, 8]. It is best to construct neural network training such that the network's weight updates during the earliest epochs are relatively easy and of the highest quality for the task. This first part of a network's training could use a curriculum learning approach. As the training proceeds, one increases the learning to span the full problem space and the hard work of learning to generalize is achieved during the middle epochs. The final epochs of the training should fine-tune the model on the desired data or tasks, because this is when the network learns the more complex patterns [9], from the most relevant training samples.

Based on the above intuition, this paper proposes that cyclical approaches for training can be generalized to all aspects of neural network training. In addition to learning rates [10], cyclical training can extend to other hyper-parameters, such as weight decay and batch size. In addition to hyper-parameters, this approach can be extended to loss functions, data-based methods, and model-based methods. For example, data augmentation methods can be cyclical by using no augmentation or weak augmentations early in the training cycle, then adding complex augmentations as the training proceeds, and eliminating augmentations in the later part of training. In addition, general cyclical training answers the question, "Which samples should be learned first?" In most scenarios, one should learn the easy, most relevant samples first and the hard ones during the middle epochs.

Adaptive hyper-parameters during training have become common. Cyclical learning rates [10], one cycle learning rates [11], and cosine annealing with warm restarts [12], have been accepted by the deep learning community and incorporated in PyTorch. General cyclical training provides an intuitive understanding for the value of a one cycle training regime. Furthermore, this idea of allowing a hyper-parameter value to change during training has been extended to other hyper-parameters, such as weight decay [13–16], and batch sizes [17].

In summary, the concept of general cyclical training is to start training in a simpler fashion during the early epochs, to train within the entire problem space and challenging conditions during the middle epochs, and to finish with fine-tuning on the most confident samples. General cyclical training includes adapting any and all factors that impact the network's training (i.e., hyper-parameters, data, loss functions).

2. GENERAL CYCLICAL TRAINING

When training in machine learning, and especially with neural networks, there are several settings that the practitioners must set that impact the final performance of the model. There are decisions regarding the hyper-parameters and the training samples that impact not only the ease of the training, but also the final generalization performance.

More formally, let us define the set of training settings $P = \{(P_j) : j \in (0, 1, ..., n)\}$, where P_0 can be a hyper-parameter value, a subset of the data, or a set of any combination that is easiest for the network to learn, while P_n represents the set of training environmental conditions that is difficult for the network to learn. In practice, one chooses a single P_j as a trade-off that provides the best performance. A cyclical approach improves on this trade-off by using a range of settings: at the beginning of the training, use a smaller j to jump-start the learning, shift gradually to P_j for a larger j during the middle of the training, and followed that with settings for which there is a decrease of j to the end of training.

The general structure of cyclical training can proceed with any schedule over some range of j for $P = \{(P_j) : j \in (0, 1, ..., n)\}$, but for the sake of simplicity, we limit our comments to a linear schedule from P_i to P_m according to

$$\xi_k P_i + (1 - \xi_k) P_m \tag{1}$$

where ξ_k is defined over the training for a number of epochs e_n as:

$$\xi_k = \begin{cases} 1 - f_c \frac{e_k}{e_n} & \text{if } f_c \times e_k \leq e_n \\ \left(f_c \frac{e_k}{e_n} - 1\right) / (f_c - 1) & \text{otherwise} \end{cases} \tag{2}$$

where e_k corresponds to the current training epoch number. Here, we introduce a cyclical factor f_c that generalizes the shape of the cycle. If $f_c = 1$, Equation 1 goes from the P_i at the beginning of the training to P_m at the end (see FIGURE 1). If $f_c = 2$, the shape for ξ_k resembles an upside down equilateral triangle (i.e., going from P_i to P_m in the first half of training and from P_m to P_i in the second half). If $f_c = 4$, P_m, is reached at a quarter of the way through the training and then linearly decreases from P_m to P_i for the remaining epochs. Although we don't discuss any other schedule, one can imitate the more complex learning rate schedules, such as stochastic gradient descent with warm restarts (SGDR) [12], or a polynomial schedule.

Furthermore, we use P_i instead of P_0 and P_m instead of P_n where $i \geq 0$ and $m \leq n$ to incorporate the flexibility not to include in the training regime settings that are too easy or too difficult.

3. ALGORITHMIC EXAMPLES

Algorithmic examples primarily include the use of dynamic hyper-parameters and various loss functions, which are discussed in this Section.

3.1 Hyper-Parameters and Regularization

The use of a decaying learning rate schedule (where the learning rate value is reduced during the training) is standard practice for network training but the use of adaptive hyper-parameters during training has become more common relatively recently. For example, the use of cyclical learning rates [10–12], has become widely accepted by the deep learning community. Unlike the decaying learning rate schedule, cyclical learning rates start with a small value of the learning rate, which enables the network's weights to move toward a good direction in the loss landscape [18], and increases the learning rate in the early epochs. In addition, learning rate warmup [19], is essentially

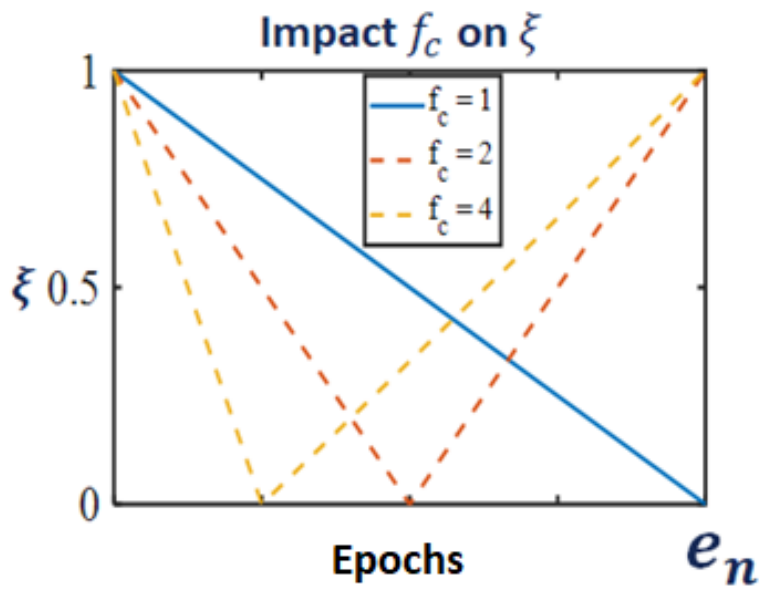

Figure 1: **Effect of f_c on the cyclical schedule:** This figure shows the impact of different values for f_c on the cyclical schedule.

equivalent to cyclical learning rates, although the warm up period is often restricted to a few of the early epochs (i.e., the same as setting f_c in Equation 2 to a large value).

This idea of allowing a hyper-parameter value to change during training (in replacement for a learning rate schedule) has been extended to other hyper-parameters, such as weight decay [13–16], and batch sizes [17]. Previous work [20], has demonstrated empirically a relationship between the optimal hyper-parameters of learning rate (LR), weight decay (WD), batch size (BS), and momentum (m) as

$$\frac{LR \times WD}{BS \times (1-m)} \simeq 10^{-6}. \tag{3}$$

Equation 3 and the success of cyclical learning rates implies that a cyclical approach also might work for weight decay, batch size, and momentum as well. That is, it might be easier for the network to learn when weight decay or momentum starts smaller or if batch size starts larger, followed by a larger weight decay/momentum or smaller batch size in the middle epochs. While [17], proposed increasing batch size instead of decreasing the learning rate, we propose taking this a step further to a cyclical batch size.

In addition, one can combine small cyclical changes in all four hyper-parameters (i.e., LR, WD, BS, and momentum) because it reduces the amount of hyper-parameter tuning required — so long as the optimal values of the hyper-parameters are within the range of the cyclical values, good generalization results can be obtained. That is, we found that being close counts in training networks. Finally, we mention that in our experiments, we found that much of the benefits of cyclical training may be achieved with even one cyclical hyper-parameter.

Table 1: **Cyclical Weight Decay:** Top-1 test classification accuracies comparing cyclical weight decay (CWD) to constant weight decay for CIFAR-10, 4K CIFAR-10 (i.e., only 4000 training samples), CIFAR-100, and ImageNet. In all of these experiments, CWD improved on the network's performance as compared to training with a constant weight decay.

Data set	Accuracy	Accuracy	Accuracy	Accuracy
CIFAR-10	97.33± 0.07	97.36± 0.02	97.45± 0.11	**97.47± 0.06**
WD range	5×10^{-4}	$4 \times 10^{-4} - 6 \times 10^{-4}$	$2 \times 10^{-4} - 8 \times 10^{-4}$	$10^{-4} - 10^{-3}$
4K CIFAR-10	86.68± 0.34	86.95± 0.21	87.35± 0.03	**87.55± 0.23**
WD range	5×10^{-4}	$4 \times 10^{-4} - 6 \times 10^{-4}$	$2 \times 10^{-4} - 8 \times 10^{-4}$	$10^{-4} - 10^{-3}$
CIFAR-100	83.82± 0.26	84.27± 0.31	84.44± 0.19	**84.36± 0.12**
WD range	2×10^{-4}	$10^{-4} - 3 \times 10^{-4}$	$5 \times 10^{-5} - 5 \times 10^{-4}$	$5 \times 10^{-5} - 7 \times 10^{-4}$
ImageNet	80.27± 0.01	**80.50± 0.05**	80.49± 0.01	80.41± 0.11
WD range	2×10^{-5}	$10^{-5} - 3 \times 10^{-5}$	$5 \times 10^{-6} - 5 \times 10^{-5}$	$5 \times 10^{-6} - 7 \times 10^{-5}$

As an example of general cyclical training with hyper-parameters and for regularization, we here propose and test cyclical weight decay (CWD). In CWD, the value for weight decay varies over the course of the training by:

$$WD = \xi_k WD_{min} + (1 - \xi_k) WD_{max}, \tag{4}$$

where ξ_k follows Equation 2 and WD_{min} and WD_{max} are user-defined hyper-parameters that specify the range for weight decay.

TABLE 1 compares the test accuracies for cyclical weight decay (CWD) to training with optimized hyper-parameters (with a constant weight decay) and learning rate warmstart and cosine annealing [12]. For each dataset in this Table there are two rows: the first row presents the mean test accuracy and the standard deviation over four runs (for ImageNet, this is the mean and standard deviation over two runs), and the second row provides the range of weight decay used in the training. The second column in the Table provides the results of training with a constant weight decay, and the subsequent columns, show the results of training with an increasing range for weight decay. In our experiments, we found that the performance was relatively insensitive to the value of f_c.

The results in TABLE 1 show that even though the constant hyper-parameters were optimized, there is a benefit to training over a range of weight decay values. For CIFAR-10, using cyclical weight decay improves the network performance relative to using a constant value of 5×10^{-3}, and the range from 10^{-4} to 10^{-3} has the best performance but using the range from 2×10^{-4} to 8×10^{-3} is within the precision of our experiments. Although the gain is small, there is no computational cost to using a range of weight decay and using a range reduces the amount of hyper-parameter tuning required.

The second row of TABLE 1 shows the results when training on only a fraction of the CIFAR-10 training set. Here we used the first 4,000 samples in the CIFAR-10 training dataset. Using cyclical weight decay improves the network performance relative to using a constant value of 5×10^{-3}, and the range from 10^{-4} to 10^{-3} has the best performance. It is noteworthy that CWD provides a more substantial benefit when the amount of training data is limited. In addition, the third row of TABLE 1 shows results for CIFAR-100 and the range from 10^{-4} to 8×10^{-4} has the best performance.

The fourth row of TABLE 1 shows the results of our experiments with CWD on ImageNet. For ImageNet, the optimal weight decay is 2×10^{-5}. Using cyclical weight decay improves the network performance relative to using a constant value and the gain appears to be stable over the small ranges we used in our experiments. FIGURE 2 compares the test accuracy curves during training for on ImageNet with a constant weight decay versus a range (test accuracy from single ImageNet runs are plotted). While the two curves are similar, the test accuracy rises slightly faster for the cyclical weight decay experiment, which illustrates that the smaller values for weight decay in the earlier epochs allow for slightly faster training than is provided by a constant weight decay. In addition, note from TABLE 1 that the final test accuracy for training with cyclical weight decay is higher than for using a constant weight decay.

Cyclical weight decay introduces a new hyper-parameter, f_c, in addition to the user-defined WD_{min} and WD_{max}. FIGURE 3 shows the test accuracies from training with a range of values for f_c in CWD on ImageNet with a TResNet_m architecture [21]. The Figure compares the accuracies when using two different weight decay ranges and the shapes of these curves imply different optimal values for f_c. Actually, these results are mostly within the precision of our experiments, implying that the performance is relatively insensitive to the value of f_c. In TABLE 1, we used a value of $f_c = 4$ for training with CWD.

Implementation of cyclical weight decay is straightforward and is described in the Appendix. Furthermore, PyTorch code is provided at `https://github.com/lnsmith54/CFL` to aid in the reproducibility of our experiments.

3.2 Loss Functions

In this Section we describe how the cyclical approaches can be applied to loss functions. A recent paper proposes a novel cyclical focal loss (CFL) [22], which emphasizes the confident samples early and late but focuses on the misclassified examples in the middle of training. This is accomplished by introducing a new loss term to the focal loss [23], term that weights confident samples more heavily than the cross-entropy softmax does and is used in the beginning and the end of the training. In addition, the focal loss term is used during the middle epochs because it more heavily weights hard, less confident samples. The result is a loss function that is superior to either the cross-entropy softmax or focal loss across balanced, imbalanced, or long-tailed datasets.

Another example of a cyclical loss function is possible by making the softmax temperature [24], dynamic — letting the temperature vary over the course of the training. We call this method cyclical softmax temperature (CST).

Softmax with temperature [24], can be expressed as:

$$Softmax(x_i) = \frac{exp(z_i/T)}{\sum_{j=1}^{N} exp(z_j/T)} \tag{5}$$

where z_i is the model's predictions or logits and T is the softmax temperature. Typically, the temperature is 1 in softmax. If the temperature is less than 1, the softmax predictions become more confident. This is also referred to as "hard" softmax probabilities. If T is greater than 1, then the

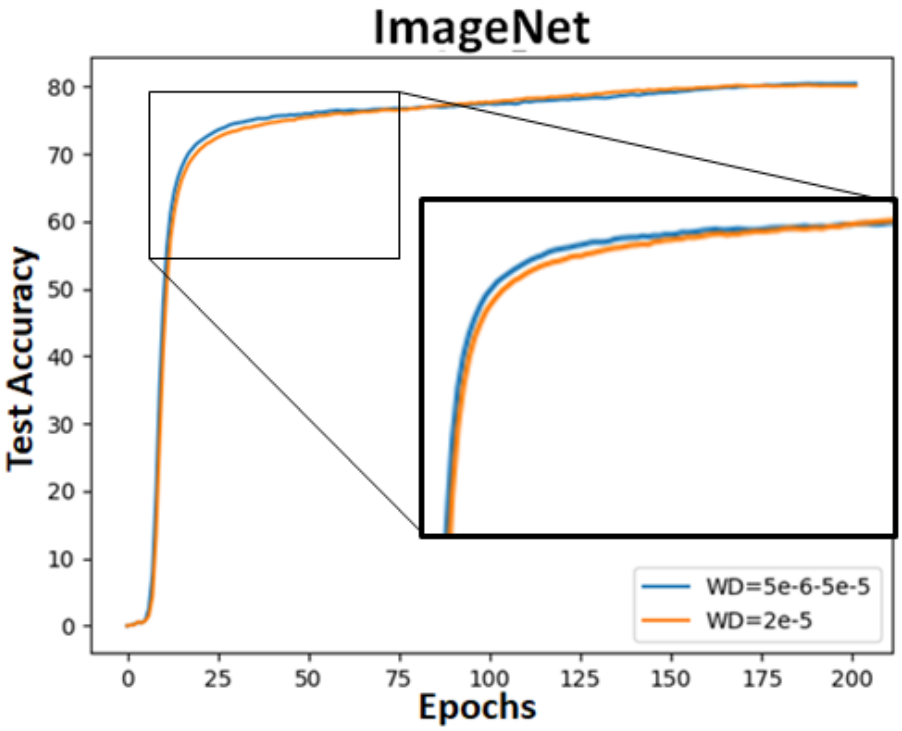

Figure 2: **CWD on ImageNet:** This figure shows the ImageNet test accuracy curve during training for cyclical weight decay versus a constant weight decay. The difference between the curves is minor, but shows cyclical weight decay allows a slightly faster rise in test accuracy.

Table 2: **Cyclical Softmax Temperature:** CIFAR-10, CIFAR-100 and ImageNet top-1 test classification accuracies comparing cyclical softmax temperature (CST) to softmax with a temperature = 1. For all three datasets, CST with a temperature from 0.5 to 2 improved on the network's performance as compared to training with T = 1. $f_c = 1$ were used in these experiments.

Data set	$T = 1$	$T = 0.75 - 1.5$	$T = 0.5 - 2.0$	$T = 0.33 - 3.0$
CIFAR-10	97.33± 0.07	97.32± 0.07	**97.43±0.06**	97.28± 0.12
4K CIFAR-10	86.68± 0.34	87.14±0.21	87.09±0.35	**87.16± 0.54**
CIFAR-100	83.82± 0.26	83.91± 0.13	**83.98± 0.25**	83.19± 0.21
ImageNet	80.27±0.01	80.42±0.12	**80.50±0.04**	80.12±0.13

predictions are less confident (also called "soft" probabilities). Softer probabilities provide more information from the network as to which classes seem similar to the target class.

The intuition for cyclical softmax temperature is that in the first epochs, more confident predictions will help start moving the network's weights in the right direction. However, in the middle and

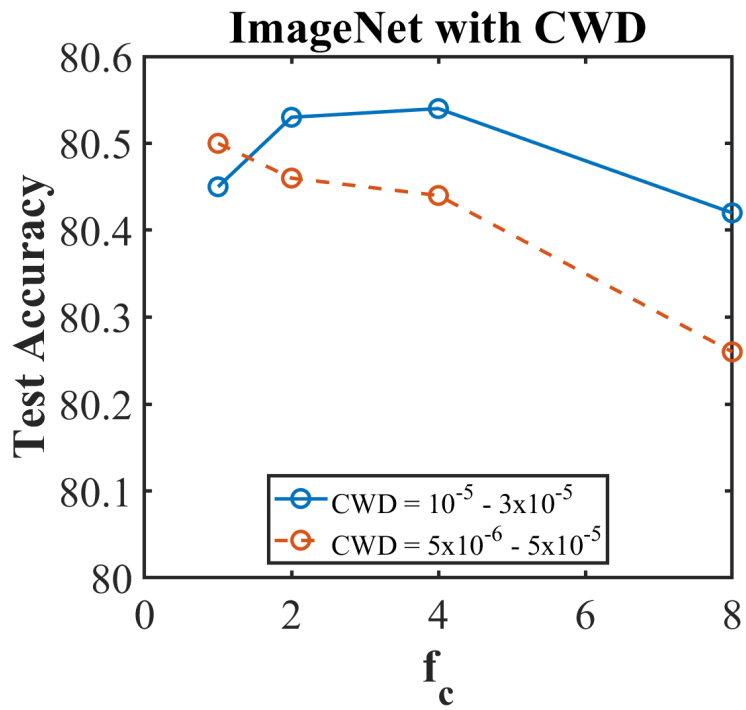

Figure 3: **Sensitivity of f_c:** This figure shows the ImageNet test accuracy for a range of values for f_c when trained with cyclical weight decay. The accuracy changes are small over this range of f_c values and $f_c = 2$ or 4 are generally good values.

later epochs, softer probabilities are more appropriate for the remaining training samples. In addition, softer probabilities reduce the confidence miscalibration of the predictions [25], so that the network's training is based on losses that better match reality.

Specifically, we propose a cyclical softmax temperature (CST) where the value for the temperature varies over the course of the training by replacing T in Equation 5 with T_c, which we define as:

$$T_c = \xi_k T_{min} + (1 - \xi_k) T_{max} \tag{6}$$

where ξ_k follows Equation 2, and T_{min} and T_{max} are user-defined hyper-parameters that specify the range for the softmax temperature.

TABLE 2 provides the top-1 test accuracies for CIFAR-10, 4K CIFAR-10, CIFAR-100, and ImageNet when training with cyclical softmax temperatures. Our experiments include using a constant $T = 1$ in softmax (column 2), and ranges of $T = 0.75$ to 1.5 (column 3), $T = 0.5$ to 2 (column 4), and $T = 0.33$ to 3 (column 5). In addition, we found that the best performance was obtained with a value of $f_c = 1$, which is what we used in our experimental results shown in TABLE 2. In all four datasets, using a range of temperatures we obtained a modest improvement over the constant $T = 1$ results. We obtained our best results for a temperature range of $T = 0.5 - 2$, and the performance usually suffered when the range was increased to $T = 0.33 - 3$. While the improvements are small, there is little to no cost to use cyclical softmax temperatures in the range from $T = 0.5 - 2$.

The implementation of cyclical softmax temperature is straightforward and PyTorch code is available at `https://github.com/lnsmith54/CFL` to aid reproducibility.

4. DATA-BASED EXAMPLES

Data is essential for training neural networks; the characteristics of the data greatly impact the training. There is a recent focus in the deep learning community on training data and data augmentation: Andrew Ng launched a campaign for data-centric machine learning and held his first data-centric AI competition[1].

The data-based examples of general cyclical training discussed here primarily include varying the use of data augmentation methods and varying which data are used during training. In addition, we discuss in this Section the proposed technique of varying the amount of unlabeled data included during training for semi-supervised learning.

4.1 Data Augmentation

Data augmentation is used widely and includes numerous techniques for improving the generalization ability of deep networks by transforming the training data in ways that do not change the associated label. A substantial amount of work has gone into the topic of data augmentation and finding automated ways to find the optimal amount and types of data augmentation for training a network on a specific dataset [26–31]. However, the previous methods assume that the amount of data augmentation used is constant from the start to the end of training.

Here, we argue that a better technique is to start and end without any data augmentation (or only weak augmentation) and to use an increasing and then decreasing strength of augmentation in the middle epochs. This follows naturally from the premise of this paper: to ease and encourage learning in the earliest iterations or epochs, then gradually to increase the span of the problem space of the network's learning as the training progresses. This implies starting with no or only very weak augmentation (i.e., image flips) and gradually incorporating stronger augmentation as the training progresses. In addition, as one approaches the end of the training, it is intuitive to eliminate the strong data augmentation and to fine-tune in the final epochs on the original training data to encourage learning of more prototypical patterns [9].

One important question that needs to be explored is how to assign data augmentation transformations on a scale from weak to strong. Intuitively, it is reasonable to measure a transformation by the degree that the transformation modifies the original image's statistics. Some have assigned image flip and shift transformation as weak augmentation, while strong augmentation includes a number of the image transformations included in the Python Image Library [27], mixup [31], and cutout [29].

While it is natural at this point in a research paper to test our hypothesis on cyclical data augmentation (CDA), as a higher level position paper that covers several specific manifestations of the general

[1] More information is available at `https://https-deeplearning-ai.github.io/data-centric-comp/`

cyclical training concept, we leave the investigation of a metric for the strength of data augmentation (from the perspective of the trained network) and the testing of CDA for future work.

4.2 Data

Not every training example contributes to a network's learning in the same way. Training data examples can differ from each other in quality, how representative they are of their class, and uniqueness. A training sample might have distracting backgrounds, corruptions, or a poor foreground/background ratio. For these reasons and more, training samples have been divided into categories of easy/medium/hard [32, 33]. This has led to the research question: Which training samples should be learned first and then in what order?

Kumar, et al. [34], propose that the order is determined by how easy the samples are to learn. The authors propose a self-paced learning algorithm where training starts with only easy samples and the number of samples increases with each epoch until all the training data is used. They define easy as "a set of samples is easy if it admits a good fit in the model space". The authors demonstrated on a SVM that their learning algorithm outperformed training on all of the training data.

This algorithm is an example of data-based cyclical training, which corresponds to setting $f_c = 1$ in Equation 2. We argue that this self-paced learning algorithm would outperform training on all the training data for deep networks based on the same logic in this earlier work [34]. In addition, Kumar, et al. provides evidence that a data-based cyclical training approach likely would be at least comparable to training on a constant training on all the training data.

One of the challenges of data-based cyclical training is to automatically measure how easy or hard each training sample is to learn while training the network. Such a metric should be with respect to the learner at its current competency level.

This is an open question but we suggest that each sample's loss can be used as one such measure. Since it is desirable in the early epochs to reduce the contributions to the loss of harder examples with high loss, this implies that a cyclical gradient clipping (CGC) method could improve a network's training. Specifically, the clipping threshold would be relatively low at the start of training, would increase to a large value in the middle epochs, and gradually would decrease to a low threshold at the end of the training epochs. A small variation on this method is to set to zero the contribution to the loss of any sample exceeding this cyclical threshold rather than clipping the loss, which would be equivalent to eliminating the hard samples from the training. Therefore, this variation to cyclical gradient clipping is one way to implement cyclical data training.

TABLE 3 shows results for cyclical gradient clipping that follows the formula:

$$C = \xi_k GC_{min} + (1 - \xi_k) GC_{max} \tag{7}$$

where C is the gradient clipping threshold, ξ_k follows Equation 2, and GC_{min} and GC_{max} are user-defined hyper-parameters that specify the range of values to use for clipping. The first column of this Table gives the dataset name, the second gives the accuracy without clipping, and the third is the performance for cyclical gradient clipping, which provides evidence supporting our hypothesis in this Section (i.e., to start with easy training samples, use hard ones in the middle epochs, and end with easy training samples). In this example, we used a clipping mode of value and set $GC_{min} = 4$

and $GC_{max} = 10$. In these experiments $f_c = 2$ was used. These results show a small improvement with cyclical clipping. We did not test the variation discussed above where the contribution of samples exceeding the clipping threshold is set to zero, which would provide additional evidence to the benefits of cyclical training with data.

Finally, we note that some researchers suggest that better results can be obtained by training with hard samples first [33]. These authors demonstrate this for imbalanced datasets, where they define "hard" as the same as rare examples. This is fundamentally different from how Kumar, et al. or we define "easy/hard". In the case of training a highly imbalanced dataset, it is reasonable to include rare examples from the beginning. The way one defines easy or hard is crucial when stating this question.

Table 3: **Cyclical gradient clipping:** Comparison of the test classification accuracies for CIFAR-10, CIFAR-100, and ImageNet. The first column gives the dataset name, the second gives the accuracy without clipping, and the third gives the performance for cyclical gradient clipping with a clipping minimum value of 4 (mode is value) and a maximum value of 10 ($f_c = 2$). These results show an improvement with cyclical gradient clipping.

Dataset	No clipping	Clip = 4 to 10
CIFAR-10	97.33± 0.07	97.48± 0.03
CIFAR-100	83.82± 0.26	84.11± 0.05
ImageNet	80.27± 0.01	80.38± 0.02

4.3 Semi-supervised Learning

Semi-supervised learning is a hybrid between supervised and unsupervised learning, which combines the benefits of both [35]. As with supervised learning, semi-supervised learning defines a task (i.e., classification) from the labeled data, but typically, it requires many fewer labeled samples than supervised learning by leveraging feature learning from unlabeled data to avoid overfitting the limited labeled samples.

Based on the general cyclical concept, this paper proposes a cyclical semi-supervised learning approach where one starts training the network with only the small labeled dataset and increasingly includes learning with unlabeled samples in the first part of training. In the second part of the training, the learning with unlabeled samples is decreased gradually so that the training ends with training only on the labeled target data (i.e., fine-tuning on the labeled data, possibly with weak data augmentation). We leave the testing of this technique to future work.

5. MODEL-BASED EXAMPLES

The principle of general cyclical training can extend to the network's architecture. Although it is currently unusual for the architecture to change while training, there exists work on growing a network during training [36–38]. However, it is more practical to consider the the general cyclical training

principle in the scenarios involving more than one network, such as with transfer learning/pre-training and with teacher-student/knowledge distillation [24, 39].

This Section discusses how cyclical training concepts imply procedural changes to training in two important domains: pre-training a network and knowledge distillation. We leave the testing of the techniques discussed in this Section to future work.

5.1 Transfer Learning and Pre-training

A common technique used in situations involving a limited number of labeled training data is to pre-train the network's weights on a large, labeled source dataset that has similar characteristics to the target dataset or to use unsupervised pre-training when there's a large number of unlabeled samples. Pre-training is similar to transfer learning when a network's weights were trained on another dataset and are used to initialize the weights and then the model is fine-tuned on the target dataset. In both cases, the goal is to maximize the performance on the target data, rather than maximizing the source data performance.

General cyclical training provides guidance on techniques for pre-training. It is intuitive that the pre-training stage represents the first part of the cycle, where the training should start easy (i.e., HP, data augmentation, and loss), and fully span the problem space as the training proceeds (i.e., to be inclusive of the target data's features). This is equivalent to using $f_c = 1$ in Equation 2 when pre-training. Fine-tuning on the target dataset is equivalent to the second half of the cycle, which implies a large value for f_c. Therefore, it is wise to consider the two training steps (i.e., pre-training and fine-tuning) as part of a single training cycle.

For example, if one is training for the purpose of transfer learning, one should start training in an easy manner (to encourage the weights to move in an optimal direction) and end with hard training. Specifically, one can use the techniques described in this paper with $f_c = 1$, such as cyclical softmax temperature, cyclical data augmentation, and cyclical weight decay. This should prepare the learned features better than the standard training methodology and there is no computational cost in doing this. Then one can employ a fine-tuning methodology with the target data, such as using a small learning rate and minimizing the use of strong data augmentation.

5.2 Knowledge Distillation

Knowledge distillation was described by Hinton, et al. [24], using two networks for model compression: the teacher and student networks. Usually, the student network is smaller than the teacher network and the goal is network compression, in which the smaller student network achieves similar performance as the teacher.

In student-teacher model compression, it is desirable that the student networks should learn as much as possible from the teacher [39], with the goal of maximizing the mutual information between the teacher and student. For this reason, the student loss function often contains not simply a cross-entropy softmax for the labels, but also loss terms for the student to match the teacher's features (the features from the teacher's hidden layers are also called "dark information"). This includes deep

supervision, in which features at several layers are matched and not just the final hidden layer's features [40, 41].

Along these lines, it is helpful when training the student to diversify the training characteristics. This can include using cyclical data augmentation so the student is exposed to strong augmentation during the middle epochs (in addition to weak augmentation for the early and final epochs). It also can include using cyclical softmax temperature, where the higher temperatures expose the student to what the teacher considers as other close classes in addition to training the student to classify the target class.

We note that there exists work on teacher-student models in which the teacher chooses what the student should learn [42, 43]. The teacher, in these cases, is guided by curriculum learning in the choice of data or task for the student to learn; these techniques also can be modified to be cyclical.

6. DISCUSSION

This position paper introduces the concept of general cyclical training, which we define as any collection of techniques where training starts and ends with "easy training" and the "hard training" happens during the middle epochs. In other words, we can consider a combination of curriculum learning in the early epochs, training on the full expanse of the problem space in the middle epochs, and fine-tuning at the end as general cyclical training. In addition, general cyclical training is analogous to curriculum learning in the sense that numerous specific techniques can embody its principles but it is different in that curriculum learning resembles learning in childhood, while general cyclical training resembles a lifetime of learning.

This paper specified several novel techniques that follow this principle: cyclical weight decay, cyclical batchsize, cyclical softmax temperature, cyclical data augmentation, cyclical gradient clipping, and cyclical semi-supervised learning. Each of these techniques is unique, but they are all manifestations of the same concept. We provided empirical evidence for some of these techniques by showing that they can improve the test performance of trained models, which is also evidence of the validity of the general cyclical training concept.

Although several novel methods are described in this paper, these are only a few of the many potential cyclical techniques that are possible. Curriculum learning can be considered a component of cyclical training and at the time of this writing, there are well over 3,000 citations to Bengio, et al. [6]: how many of these can be converted to a cyclical technique?

Although cyclical methods introduce new hyper-parameters, one of the benefits of cyclical training methods is a potential overall reduction of the amount of hyper-parameter tuning that is required. In this paper, we have demonstrated that the cliche of "close" applies in neural network training (in addition to the game of horseshoes and with hand grenades). That is, if the optimal training hyper-parameters fall within the cyclical ranges, even if not in the center, the trained network's performance is generally optimal. When using ranges rather than specific values, a grid search will require fewer tests.

The intended purposes of this paper is to propose the general cyclical training hypothesis, for the reader to find general cyclical training enlightening, and that it illustrates relationships between previously separate methods. We hope that it encourages the creation of additional novel techniques based on its principles.

7. ACKNOWLEDGEMENTS

We thank the US Office of Naval Research for their support of this research. The views, opinions and/or findings expressed are those of the author and do not reflect the official policy or position of the US Navy, Department of Defense or the US Government.

References

[1] Awad N, Mallik N, Hutter F. Dehb: Evolutionary Hyperband for Scalable, Robust and Efficient Hyperparameter Optimization. 2021. arXiv preprint: https://arxiv.org/pdf/2105.09821.pdf

[2] Bischl B, Binder M, Lang M, Pielok T, Richter J, Coors S, Thomas J, Ullmann T, Becker M, Boulesteix AL, Deng D. Hyperparameter Optimization: Foundations, Algorithms, Best Practices, and Open Challenges. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2021:e1484.

[3] Eggensperger K, Müller P, Mallik N, Feurer M, Sass R, et al. Hpobench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for Hpo. 2021. arXiv preprint: https://arxiv.org/pdf/2109.06716.pdf

[4] Dong X, Kedziora DJ, Musial K, Gabrys B. Automated Deep Learning: Neural Architecture Search Is Not the End. 2021. arXiv preprint: https://arxiv.org/pdf/2112.09245.pdf

[5] Morales-Hernández A, Van Nieuwenhuyse I, Rojas Gonzalez S. A Survey on Multi-Objective Hyperparameter Optimization Algorithms for Machine Learning. Artificial Intelligence Review. 2022:1-51.

[6] Bengio Y, Louradour J, Collobert R, Weston J. Curriculum Learning. In: Proceedings of the 26th annual international conference on machine learning. 2009:41-48.

[7] Jonathan Frankle, David J Schwab, and Ari S Morcos. The Early Phase of Neural Network Training. 2020. arXiv preprint: https://arxiv.org/pdf/2002.10365.pdf

[8] Golatkar A, Achille A, Soatto S. Time Matters in Regularizing Deep Networks: Weight Decay and Data Augmentation Affect Early Learning Dynamics, Matter Little Near Convergence. 2019. arXiv preprint: https://arxiv.org/pdf/1905.13277.pdf

[9] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. How Does Learning Rate Decay Help Modern Neural Networks? 2019. arXiv preprint: https://arxiv.org/pdf/1908.01878.pdf

[10] Smith LN. Cyclical Learning Rates for Training Neural Networks. In: IEEE winter conference on applications of computer vision (WACV). IEEE Publications. 2017:464-472.

[11] Smith LN, Topin N. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. In: Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications. International Society for Optics and Photonics. 2019;11006:1100612.

[12] Loshchilov I, Hutter F. Sgdr: Stochastic Gradient Descent With Warm Restarts.2016. arXiv preprint: https://arxiv.org/pdf/1608.03983.pdf

[13] Bjorck J, Weinberger K, Gomes C. Understanding Decoupled and Early Weight Decay. 2020. arXiv preprint: https://www.cs.cornell.edu/gomes/pdf/2020_bjorck_arxiv_decay.pdf

[14] Aitor Lewkowycz and Guy Gur-Ari. On the Training Dynamics of Deep Networks With l_2 Regularization. 2020 arXiv preprint: https://arxiv.org/pdf/2006.08643.pdf

[15] Nakamura K, Hong BW. Adaptive Weight Decay for Deep Neural Networks. IEEE Access. 2019;7:118857-118865.

[16] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three Mechanisms of Weight Decay Regularization.2018. arXiv preprint: https://arxiv.org/pdf/1810.12281.pdf

[17] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't Decay the Learning Rate, Increase the Batch Size.2017 arXiv preprint: https://arxiv.org/pdf/1711.00489.pdf

[18] Li H, Xu Z, Taylor G, Studer C, Goldstein T. Visualizing the Loss Landscape Of Neural Nets. 2017. arXiv preprint: https://arxiv.org/pdf/1712.09913.pdf

[19] Goyal P, Dollár P, Girshick R, Noordhuis P, Wesolowski L, et al. Accurate, Large Minibatch Sgd: Training Imagenet in 1 Hour.2017. arXiv preprint: https://arxiv.org/pdf/1706.02677.pdf

[20] https://docs.google.com/presentation/d/124xUU9g7e4IqCTpSGjIHjo_2wN-KkAoq/edit#slide=id.p1

[21] Ridnik T, Lawen H, Noy A, Ben Baruch E, Sharir G, et al. Tresnet: High Performance Gpu-Dedicated Architecture. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision; 2021: 1400-1409.

[22] Leslie N Smith. Cyclical Focal Loss.2022. arXiv preprint: https://arxiv.org/pdf/2202.08978.pdf

[23] Lin TY, Goyal P, Girshick R, He K, Dollár P. Focal Loss for Dense Object Detection. In: Proceedings of the IEEE international conference on computer vision. 2017:2980-2988.

[24] Hinton G, Vinyals O, Dean J. Distilling the Knowledge in a Neural Network.2015. arXiv preprint: https://arxiv.org/pdf/1503.02531.pdf

[25] Guo C, Pleiss G, Sun Y, Weinberger KQ. On Calibration of Modern Neural Networks. In: International Conference on Machine Learning; PMLR. 2017: 1321-1330.

[26] Bayer M, Kaufhold MA , Reuter C. A Survey on Data Augmentation for Text Classification. 2021. arXiv preprint: https://arxiv.org/ftp/arxiv/papers/2107/2107.03158.pdf

[27] Cubuk ED , Zoph B, Mane D, Vasudevan V, Quoc V Le. Autoaugment: Learning Augmentation Policies From Data. 2018. arXiv preprint: https://arxiv.org/pdf/1805.09501.pdf

[28] Cubuk ED, Zoph B, Mane D, Vasudevan V, Quoc V Le. Randaugment: Practical Automated Data Augmentation With a Reduced Search Space. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. 2020:702-703.

[29] Terrance DeVries, Graham W Taylor. Improved Regularization of Convolutional Neural Networks With Cutout. 2017. arXiv preprint: https://arxiv.org/pdf/1708.04552.pdf

[30] Shorten C, Khoshgoftaar TM. A Survey on Image Data Augmentation for Deep Learning. J Big Data. 2019;6:1-48.

[31] Zhang H, Cisse M, Dauphin YN, Lopez-Paz D. Mixup: Beyond Empirical Risk Minimization.2017. arXiv preprint: https://arxiv.org/pdf/1710.09412.pdf

[32] Hendrycks D, Dietterich T. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations.2019. arXiv preprint: https://arxiv.org/pdf/1903.12261.pdf

[33] Zhou X, Wu O. Which Samples Should Be Learned First: Easy or Hard? 2021. arXiv preprint:https://arxiv.org/pdf/2110.05481.pdf

[34] Pawan KM, Packer B, Koller D. Self-Paced Learning for Latent Variable Models. In: Nips. 2010;1:2.

[35] Smith LN, Conovaloff A. Building One-Shot Semi-supervised (Boss) Learning Up to Fully Supervised Performance. 2020. arXiv preprint: https://arxiv.org/pdf/2006.09363.pdf

[36] Chen T, Goodfellow I, Shlens J. Net2net: Accelerating Learning via Knowledge Transfer.2015. arXiv preprint:https://arxiv.org/pdf/1511.05641.pdf

[37] Smith LN, Hand EM, Doster T. Gradual Dropin of Layers to Train Very Deep Neural Networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016: 4763-: 4771.

[38] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training Very Deep Networks. 2015. arXiv preprint: https://arxiv.org/pdf/1507.06228.pdf

[39] Wang L, Yoon KJ. Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks. IEEE Trans Pattern Anal Mach Intell. 2022;44:3048-3068.

[40] Romero A, Ballas N, Kahou SE, Chassang A, Gatta C, Bengio Y. Fitnets: Hints for Thin Deep Nets. 2014. arXiv preprint: https://arxiv.org/pdf/1412.6550.pdf

[41] Zhang L, Jiebo S, Gao A, Chen J, Bao C, et al. Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation. In: Proceedings of the IEEE/CVF international conference on computer vision; 2019:3713-3722.

[42] Jiang L, Zhou Z, Leung T, Li-Jia Li, Fei-Fei L. Mentornet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. In: International Conference on Machine Learning. PMLR. 2018:2304-2313.

[43] Portelas R, Colas C, Hofmann K, Oudeyer PY. Teacher Algorithms for Curriculum Learning of Deep RL in Continuously Parameterized Environments. In: Conference on Robot Learning. PMLR. 2020:835-853.

[44] https://github.com/rwightman/pytorch-image-models

Appendix A. Software and Implementation

We used PyTorch Image Models (timm) [44], as a framework in our experiments on CIFAR and ImageNet. This framework provides the models and downloads the data used in our experiments. The original code is available at `https://github.com/rwightman/pytorch-image-models`. The file `train.py` was modified by inserting additional several new input parameters via calls to `add_argument` and adding a few lines of code for cyclical weight decay (CWD), cyclical softmax temperature (CST), and cyclical gradient clipping (CGC).

Specifically, implementing CWD and CGC involves including the following in the training loop:

```
if args.wd_min > 0 or args.clip_min > 0:
        if args.cyclical_factor*epoch < num_epochs:
                eta = 1.0 - args.cyclical_factor *epoch/(num_epochs-1)
        elif args.cyclical_factor == 1.0:
                eta = 0
        else:
                eta = (args.cyclical_factor*epoch/(num_epochs-1) - 1.0) /
                        (args.cyclical_factor - 1.0)

if args.wd_min > 0:
        optimizer.param_groups[0]['weight_decay'] =
                (1 - eta)*args.wd_max + eta*args.wd_min
elif args.clip_min > 0:
        args.clip_grad = (1 - eta)*args.clip_max + eta*args.clip_min
```

Similarly, implementing CST can be performed as follows:

```
if args.T_min > 0:
        if args.cyclical_factor*epoch < args.epochs:
                eta = 1.0 - args.cyclical_factor *epoch/(args.epochs-1)
        elif args.cyclical_factor == 1.0:
                eta = 0
        else:
                eta = (args.cyclical_factor*epoch/(args.epochs-1) - 1.0) /
                        (args.cyclical_factor - 1.0)
        Temperature = (1 - eta)*args.T_max + eta*args.T_min
        output = torch.div(output, Temperature)
```

The full revised `train.py` is available as part of our code-base at `https://github.com/lnsmith54/CFL`.

Table 4: The hyper-parameters used for experiments whose results are presented in the main paper.

Table	Dataset	Model	Batch size	LR	WD	f_c
Table 1	CIFAR-10	TResNet_m	384	0.15	5×10^{-4}	2, 2, 2
Table 1	CIFAR-100	TResNet_m	64	0.2	2×10^{-4}	2, 2, 2
Table 1	ImageNet	TResNet_m	192	0.6	2×10^{-5}	1,4,4
Table 2	CIFAR-10	TResNet_m	384	0.15	5×10^{-4}	1, 1, 1
Table 2	CIFAR-100	TResNet_m	64	0.2	2×10^{-4}	1, 1, 1
Table 2	ImageNet	TResNet_m	192	0.6	2×10^{-5}	1, 1, 1
Table 3	CIFAR-10	TResNet_m	384	0.15	5×10^{-4}	2
Table 3	CIFAR-100	TResNet_m	64	0.2	2×10^{-4}	2
Table 3	ImageNet	TResNet_m	192	0.6	2×10^{-5}	2

Appendix B. Command Lines and Hyper-parameters

In the spirit of easy replication, it is important to know the values of the hyper-parameters used. TABLE 4 specifies the batch sizes, learning rates, and weight decay values used for the results in TABLE 1, TABLE 2 and TABLE 3 in the main body of the paper.

Here, we present the command line for submitting an experiment on cyclical softmax temperature (CST) with Imagenet:

```
./distributed_train.sh 4 data/imagenet -b=192 --lr=0.6 --warmup-lr 0.02
--warmup-epochs 3 --T_min 0.5 --T_max 2 --weight-decay 2e-5 --cooldown-epochs 1
--model-ema --checkpoint-hist 4 --workers 8 --aa=rand-m9-mstd0.5-inc1 -j=16 --amp
--model=tresnet_m --epochs=200 --mixup=0.2 --sched='cosine' --reprob=0.4
--remode=pixel  --cyclical_factor 1
```

Default values for hyper-parameters not specified in this command line were used (i.e., see the software at `https://github.com/lnsmith54/CFL` or the original code at `https://github.com/rwightman/pytorch-image-models` for default values of the hyper-parameters). The new hyper-parameters for CST (i.e., T_min and T_max) are specified on the command line. For cyclical weight decay (CWD), the command line was modified by replacing:

```
--T_min 0.5 --T_max 2 --weight-decay 2e-5
--cyclical_factor 1
```

with:

```
--wd_min 1e-5 --wd_max 8e-5
--cyclical_factor 2
```

For CIFAR-10, the following command line was used for CST:

```
CUDA_VISIBLE_DEVICES=0 python train.py  data/cifar10 --dataset torch/cifar10
-b 384 --model tresnet_m --checkpoint-hist 4 --sched cosine --epochs 200 --lr 0.5
--warmup-lr 0.01 --warmup-epochs 3 --cooldown-epochs 1 --weight-decay 5e-4
--T_min 0.5 --T_max 2 --amp --remode pixel --reprob 0.6 --aug-splits 3
--aa rand-m9-mstd0.5-inc1 --resplit --split-bn  --dist-bn reduce  --cyclical_factor 1
```

For CIFAR-100, the following command line was used for CST:

```
CUDA_VISIBLE_DEVICES=0 python train.py  data/cifar100 --dataset torch/cifar100
-b 64 --model tresnet_m --checkpoint-hist 4 --sched cosine --epochs 200 --lr 0.2
--warmup-lr 0.01 --warmup-epochs 3 --cooldown-epochs 1 --weight-decay 2e-4
--T_min 0.5 --T_max 2 --amp --remode pixel --reprob 0.6 --aug-splits 3
--aa rand-m9-mstd0.5-inc1 --resplit --split-bn  --dist-bn reduce  --cyclical_factor 1
```

As with ImageNet, for cyclical weight decay (CWD), each of the above command lines were modified by replacing:

```
--T_min 0.5 --T_max 2 --weight-decay 2e-5
--cyclical_factor 1
```

with:

```
--wd_min 1e-5 --wd_max 8e-5
--cyclical_factor 2
```

The default values for hyper-parameters are available in the software.