Cyclical Curriculum Learning

H. Toprak Kesgin and M. Fatih Amasyali

Abstract—Artificial neural networks (ANNs) are inspired by human learning. However, unlike human education, classical ANN does not use a curriculum. Curriculum learning (CL) refers to the process of ANN training in which samples are used in a meaningful order. When using CL, training begins with a subset of the dataset and new samples are added throughout the training, or training begins with the entire dataset and the number of samples used is reduced. With these changes in training dataset size, better results can be obtained with curriculum, anti-curriculum, or random-curriculum methods than the vanilla method. However, a generally efficient CL method for various architectures and datasets is not found. In this article, we propose cyclical CL (CCL), in which the data size used during training changes cyclically rather than simply increasing or decreasing. Instead of using only the vanilla method or only the curriculum method, using both methods cyclically like in CCL provides more successful results. We tested the method on 18 different datasets and 15 architectures in image and text classification tasks and obtained more successful results than no-CL and existing CL methods. We also have shown theoretically that it is less erroneous to apply CL and vanilla cyclically instead of using only CL or only the vanilla method. The code of the cyclical curriculum is available at https://github.com/CyclicalCurriculum/Cyclical-Curriculum.

Index Terms—Curriculum learning (CL), deep learning, optimization.

I. INTRODUCTION

WHILE we learn during our education, we use a specific subject order, namely a curriculum. We begin with basic, simple information and proceed with more challenging, complex issues. Following a curriculum helps humans and animals learn better [1].

Artificial neural networks (ANNs) were inspired by the communication of biological neuron cells. However, unlike human sequential learning, the dataset in neural network training is randomly ordered in the classical method. In ANN training, it has been demonstrated that ordering the data in a specific order rather than randomly can improve the success of ANNs [2], [3]. This method is known as curriculum learning (CL). The ordering here can be thought of as simple to difficult or basic to complex.

ANNs have many hyperparameters for training. The learning rate, the epoch number, and the batch size are some of them. While performing the training of ANNs, the classical method divides the whole training set according to a certain number of batches. It updates the weights of the model with these batches to optimize the determined loss function. One epoch is completed when weights are updated using all data according to the determined batch size. When the weights have been updated for the specified number of epochs, the training is complete. We will call this classical method used as the vanilla method in this article. The vanilla method has shown its success, especially in image classification [4], [5].

In curriculum approaches, the data is sorted according to a specific criterion first. This order can be from easy to difficult, from hard to easy, or random. After the ranking is determined, the dataset is divided according to specific parts. After the splitting process is performed, optimization is applied for the number of epochs chosen for the first part of the data, and the weights of the model are updated. This subset is enlarged after a certain number of epochs, and the training continues over these merged sets. The training is completed when the whole dataset is used and the number of updates is made for the determined epoch number.

The rank can be determined using a variety of methods. An expert can determine the rank. Any machine learning or ANN-generated estimated probability values could be used for ranking. Alternatively, as in the self-taught algorithm, the model itself is trained with the vanilla method, and the loss values produced for the training samples can be used to rank the samples [3]. Without using another model, the properties of the dataset can also be used to rank the samples. For example, object number [6] and shape variability [2] for image datasets, sentence length [7], and word rarity [8] for text datasets.

There are studies that have achieved successful results using curriculum [2], [3], [9], [10], anti-curriculum [8], [11], [12], and random-curriculum [13]. The common point of these studies is to achieve more successful results in training by increasing the size of the training set. The training set size does not have to start small to go up. For example, the training can start with the whole dataset, and after a specific number of epochs, the subsets of the training set can be started to use. As a result, more successful results could be obtained than with vanilla [14]. Randomly ordered samples can perform as well as or better than curricula and anti-curricula, and better results can be obtained as the training dataset dynamically increases or decreases over time, implying that any benefit is entirely due to the dynamic training set size [13].

We compared popular CL methods on numerous datasets and deep learning architectures in the text and image domains, and found that current CL methods do not increase success much, in line with the results of studies [3], [13] recently published in the literature. Almost all the methods in the literature increase the size of the dataset [2], [3], [9]. There are few studies that reduce it [14].

In human learning, the success of spiral curriculum design and spaced repetition has been demonstrated rather than classical sequential curriculum [15], [16]. With a spiral curriculum, themes or topics are reviewed iteratively throughout the course. Our proposed cyclical CL (CCL) algorithm uses techniques similar to spiral curriculum and spaced repetition, and it might provide the following potential benefits of these techniques: a visit can bring new knowledge or skills about the topic or more advanced applications of previously learned topics. Learning new skills and knowledge in subsequent lessons reinforces what is already known and integrates it into previously learned information [15]. The motivations behind the CCL in ANN training are a progression from simple to complicated, reinforcement through continuous exposure, logical sequences, and more customizable curriculums.

We investigate the effects of cyclical training set size rather than simply increasing or decreasing it by combining these two approaches. We propose a cyclical dataset size approach and found that this approach significantly improves the success over existing methods. The CCL is easy to implement and can be integrated with other techniques in a flexible manner. The main contributions of this article are as follows.

1) Introducing CCL, a novel training algorithm that changes the size of the data during training cyclically rather than as it increases or decreases.
2) The CCL algorithm was tested in 18 different datasets and 15 architectures in image and text classification tasks, and its performance was compared with no-CL and existing CL methods.
3) We have theoretically demonstrated that applying CL and vanilla cyclically is less erroneous than using only CL or only vanilla.

II. LITERATURE REVIEW

Previous studies have demonstrated that utilizing a meaningful order of samples in training ANNs results in significant improvements [2]. Self-paced learning (SPL) [17] is a training paradigm that adaptively orders training samples during training. SPL has been demonstrated to be effective in multiple datasets. Other studies [18] also applied SPL successfully with deep curriculum reinforcement learning (DCRL) variants that adaptively select samples from replay memory based on their complexity, which improves the efficiency and effectiveness of deep reinforcement learning models.

An alternative approach to CL is the teacher-student approach, where the teacher model selects subtasks for the student model to train on [19]. It is also possible that use the teacher model as the model itself [3]. This approach has been shown to produce similar improvements as traditional CL methods. Another method for determining dynamic difficulty is dynamic instance hardness (DIH) [14], which considers not only the final loss values of the samples but also how the loss value changes over time. It starts with all training samples, and after a certain number of epochs, reduces the training set size and selects samples with high DIH values.

The cyclical learning rate [20] is a proposed method for one of the critical hyperparameters of ANNs, the learning rate. Instead of gradually decreasing the learning rate, it has been shown that changing the learning rate cyclically between reasonable values produces more successful results in the test set for accuracy. The study provides practical methods for determining the reasonable learning rate range and demonstrates the effectiveness of the approach using various networks.

The CL, anti-CL, and rand-CL methods were compared and analyzed in terms of their effectiveness in different scenarios [13]. The results showed that the rand-CL method performed as well as or better than the other methods and that the improvement was attributed to the change in the size of the dataset used in the training. The experiments also showed that CL methods were more successful in situations with limited time and noisy data.

A comprehensive analysis of CL is provided in the curriculum learning survey [21], which includes discussions on the motivations, definitions, theories, and applications of CL. The researchers in the survey categorized CL into four categories: SPL, transfer teacher, RL teacher, and other automatic CL. The CCL can be considered as a type of transfer teacher, as it uses a teacher model with the same structure as the student model.

III. PROPOSED METHOD

In this section, the cyclical dataset sizes, how the samples are selected for training, how the scores are determined, and the training algorithm are explained.

A. Cyclical Training Dataset Sizes

The vanilla method uses the entire training set in one epoch. The CCL, on the other hand, only uses a subset of the training dataset in one epoch. The size of these subsets is determined by parameters. The samples to be chosen for the subset are determined by certain scores.

Hyperparameters of the cyclical training datasets sizes algorithm are initial percent (ip), final percent (fp), α, and epoch count (T). The initial percentage indicates the percentage of data with which the cycle will start, and the final percentage indicates the point at which it will increase. α determines how fast or slow the changes in dataset size between 0 and 1. The algorithm for the cycling approach that determines the percentage of subsets is given in Algorithm 1.

Fig. 1 shows how the size of the training dataset changes throughout the training. For cyclical dataset sizes, initial percent (ip) = 0.25 final percent (fp) = 1.0 α = 0.5 was used.

B. Selecting Samples

Since a different subset of the training set will be used for each epoch of CCL, the number of samples should be selected as much as the specified subset size for model training. For this selection process, various algorithms can be used: 1) samples for the subset can be randomly selected at each step; 2) all samples are sorted according to a specific criterion and the top n are selected; and 3) probability values are calculated for all samples and these probability values are used to select samples.
When selecting subsets of the dataset, we used the sample losses as probability values. We examine approaches to picking easy samples with higher probability, selecting difficult ones with higher probability, and randomizing them. We observed that using easy samples for the subset is more successful than other methods. And we also found that probabilistic selection works better than the greedy selection of the top \( n \) samples. In the experiments, we used the third method, probability-based sampling.

\[ p_i \sim r(i) \] where \( r(i) \) is probability scores. Training data indices are sampled with a probability proportional to the inverse loss values. Algorithm 1 determines the percentage of indices to be selected per epoch. Sampling is done without replacement. In Algorithm 2 line 14 in select_by_scores method, \( D \) dataset indexes are sampled with according to \( r \) probabilities. \( S[i] \) percent of sample indexes are selected without replacement.

### C. Determining Scores

In order to make a probabilistic selection, all data must have a score value. We obtained these scores by using the model itself. First, the model is trained for a certain number of epochs. Then the model makes predictions for the training set. From these estimates, the loss value is calculated for each sample. To obtain probability values from these loss values, the following operations are applied. Since the small loss value represents an easy sample, the calculated loss values are reversed according to multiplication. All scores are then divided by the total number of scores for normalization. Formally, the score (probability) values for each sample can be calculated as

\[ k_i = 1/(l(y_i, M_1(X_i, W))) \] and \( r_i = (k_i / \sum k_i) \) where \( l \) is the loss function, \( M_1 \) is the trained model, \( W \) is the final model parameters for \( M_1 \), \( X \) and \( y \) denote the features of the training set, the labels, respectively, and \( r \) is score values for each sample. The samples are then selected with the probabilistic selection algorithm (see Section III-B).

### Algorithm 1 Get Dataset Sizes

**Require:**
1. \( T \): epoch count for training.
2. \( ip \): initial percentage of dataset size.
3. \( fp \): final percentage of dataset size.
4. \( \alpha \): speed of cycle.
5. \( \beta \): training dataset, consists of \( X \): Input and \( y \): Output.
6. \( M_1 \): the model by which the scores will be obtained.
7. \( M_2 \): the model to train.
8. \( T \): epoch count for training.
9. \( ip \): initial percentage of dataset size.
10. \( fp \): end percentage of dataset size.
11. \( \alpha \): speed of cycle.

**procedure** GET_DATASET_SIZES(ip, fp, \( \alpha \), \( T \))

1. \( S \leftarrow [ ] \) (initialize empty list)
2. \( n \leftarrow ip \)
3. \( S.append(n) \)
4. for \( i \in \{1, \ldots, (T - 1)\} \) do
5. \( n \leftarrow \min ((n * (1/\alpha)), fp) \)
6. \( n \leftarrow \max ((n * (\alpha)), ip) \)
7. \( S.append(n) \)
8. end for
9. return \( S \)

### Algorithm 2 Training

**Require:**
1. \( D \): training dataset, consists of \( X \): Input and \( y \): Output.
2. \( M_1 \): the model by which the scores will be obtained.
3. \( M_2 \): the model to train.
4. \( T \): epoch count for training.
5. \( ip \): initial percentage of dataset size.
6. \( fp \): end percentage of dataset size.
7. \( \alpha \): speed of cycle.

**procedure** TRAIN

1. \( S \leftarrow GET_SIZES(fp, ip, \alpha, T) \) (Section III-A)
2. \( r \leftarrow GET_SCORES(M_1, D) \) (Section III-C)
3. \( M_2 \leftarrow INITIALIZE_MODEL \)
4. for \( i \in \{1, \ldots, T\} \) do
5. \( n \leftarrow SELECT_BY_SCORES(D, S[i], r) \) (Section III-B)
6. \( UPDATE_MODEL(M_2, D(n)) \)
7. end for
8. end procedure

### D. Training

After the training set sizes and scores are determined, the training is performed according to Algorithm 2. The get_sizes procedure is as described in Section III-A. It returns a list \( S \) such as \([0.25, 0.5, 1.0, 0.5, 0.25, 0.5, 1.0] \). When \( S[i] = 1 \), since selection is done without replacement, the entire dataset is used in step \( i \). This means applying the vanilla method in step \( i \). When \( S[i] < 1 \), a smaller part of the dataset is used and since the selection is made according to the losses of samples, this is to apply the CL algorithm in step \( i \). Therefore, CCL applies vanilla and CL cyclically.

The CCL algorithm can be easily converted to other CL algorithms. For example, if the \( S \) produced by the get_sizes function consists of only 1 s (\( S[i] = 1 \)) and the get_scores...
function produces equal scores for all samples, it becomes a vanilla method. If the array produced by the get_sizes function is an incremental array and equal scores are used for all instances, the method becomes random CL. In CCL, on the other hand, the S array produced by the get_sizes function consists of cyclically changing values and the scores of all samples are determined by a certain model with the get_scores function. In line 12, the model to be trained can be initialized with random weights or with a pretrained model, depending on the task.

### IV. Experiments

We ran our experiments with various datasets in order to perform various tasks over different networks. Experiments can be divided into two main groups, image classification and text classification. The datasets used for image classification are cifar-10, cifar-100, fmnist, and stl-10 [22], [23], [24]. Image datasets are modeled using a basic CNN network [25].

For natural language processing experiments, dataset, and model pairs were obtained from various sources on the internet [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39]. The datasets used are 20news, sarcasm, reuters, hotel, sweet, ctweet, qa, food, toxic, reddit, squad, imb, ner [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50]. Task and model diversity were prioritized when selecting datasets.

#### A. Training Process

We first trained the model until no improvement until two consecutive epochs in the validation set (early stopping) for the required number of epochs. We generated the scores in this epoch number using the model itself. While conducting the experiments, we trained the models with three times the number of epochs from which we obtained the scores. We used the hyperparameters in the sources from which the dataset model pairs were taken. We determined our evaluation criteria as accuracy and tested the models in linear intervals during training. We determined the success criteria as the maximum accuracy achieved during these tests. We started the models from the same starting point for a fair evaluation and repeated the experiments five times with different initial weights.

#### B. Summary of Results

Table I displays the average accuracy across test sets. The accuracy of the named entity recognition (NER) has been calculated without the “O” tag because the vast majority of tags (85%) are the “O” tag among 17 tags. The Squad dataset is for a question-answering task rather than a classification task. Therefore, the predicted and ground truth answer’s exact match percentage used as success criteria.

In these experiments, we used the same initial weights for methods. We repeated the experiments five times with different initial weights. The average of these five values is shown in Table I. For comparison of methods, we used the student’s t-test with \( p = 0.05 \). We compare all the methods with the vanilla method. Statistically better ones are shown in green, and worse ones are shown in red. The CCL outperforms vanilla in ten datasets out of eighteen. In one single dataset, its performance is lower. We used the same cycle rate \( ip = 0.25, fp = 1, \alpha = 0.5 \) for all datasets. However, it can be possible to achieve more successful results with optimization of cycling hyperparameters.

Since a subset of the dataset is used in each epoch, different methods are used to select samples for this subset. The CCL selects samples using score values as probability values. These score values are composed of the loss values from a model trained with different initial weights on the training set. Obtained loss values are reversed (lower loss and higher probability) and normalized as a probability value. Cyclical anti-curriculum means the opposite. Cyclical random-curriculum refers to random selection according to subset size. The curriculum is not the probabilistic selection of subsets, but dataset growth using fixed rank according to scores. For example, let the number of samples in the whole dataset be \( 3n \). The easiest \( n \) samples are used for a certain number of epochs. After that, the easiest \( 2n \) samples are used for a given epoch and finally, the entire dataset is used. With these subsets, it is also important that keeping the sample

### Table I

|       | Vanilla | CCL  | Anti-CCL | Rand-CCL | CL   | Anti-CL | Rand-CL | SPL | Anti-SPL |
|-------|---------|------|----------|----------|------|---------|---------|-----|----------|
| cifar10 | 72.09   | 75.51 | 69.83    | 70.01    | 72.98 | 66.46   | 69.26   | 72.34 | 72.09    |
| cifar100 | 38.77   | 42.45 | 35.40    | 36.54    | 37.05 | 31.70   | 33.16   | 39.11 | 38.13    |
| fmnist | 90.74   | 91.32 | 90.66    | 90.64    | 91.10 | 89.83   | 90.23   | 90.95 | 90.96    |
| stl_10 | 59.53   | 56.94 | 56.78    | 55.58    | 59.06 | 58.49   | 57.90   | 59.61 | 59.42    |
| 20_news | 71.65   | 72.43 | 69.34    | 69.61    | 72.36 | 68.80   | 71.05   | 71.88 | 71.23    |
| sarcasm | 82.23   | 82.65 | 81.33    | 81.36    | 81.74 | 82.10   | 82.18   | 81.42 | 81.77    |
| reuters | 79.82   | 80.30 | 79.68    | 79.68    | 80.63 | 79.26   | 79.46   | 79.73 | 79.74    |
| hotel   | 60.51   | 60.42 | 59.97    | 59.77    | 60.67 | 59.35   | 60.22   | 59.07 | 58.91    |
| sweet   | 78.02   | 78.03 | 77.82    | 78.03    | 78.03 | 77.96   | 77.96   | 77.79 | 76.39    |
| ctweet  | 85.88   | 85.73 | 85.71    | 85.49    | 85.46 | 83.80   | 84.49   | 85.65 | 85.69    |
| qa      | 78.44   | 78.28 | 78.52    | 78.42    | 78.07 | 78.39   | 78.21   | 77.51 | 77.97    |
| food    | 94.04   | 94.20 | 93.92    | 93.99    | 94.18 | 93.81   | 93.96   | 94.03 | 93.95    |
| sof     | 88.13   | 88.31 | 87.93    | 88.32    | 88.16 | 87.10   | 87.22   | 88.16 | 87.41    |
| toxic   | 92.75   | 93.31 | 92.71    | 92.76    | 93.35 | 92.74   | 93.29   | 92.44 | 92.96    |
| reddit  | 72.82   | 72.81 | 72.81    | 72.78    | 72.64 | 72.80   | 72.77   | 72.56 | 72.41    |
| squad   | 78.11   | 78.26 | 78.18    | 77.86    | 78.66 | 77.70   | 78.66   | 78.72 | 78.65    |
| imb     | 92.11   | 92.24 | 91.27    | 92.14    | 91.16 | 91.65   | 92.10   | 92.34 | 92.24    |
| ner     | 73.40   | 74.25 | 72.96    | 72.93    | 72.77 | 72.82   | 73.38   | 73.84 | 72.37    |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
balanced with the same number of examples from each class as in the training set. The comparison of these methods during the training of the cifar-10 dataset is given in Fig. 2. The method that selected more easy samples proportionally to their probabilistic values, CCL, yielded the best results from these methods.

We compared CCL and DIH guided curriculum learning (DIHCL) on cifar-10 with DIHCL on the CNN architecture that we used in the experiments. Compared to vanilla, DIHCL had an average success rate of 72.21%. There were no significant differences in the success rates between DIHCL and vanilla. Using the default settings of [51] the architecture, we conducted a second comparison. Vanilla, DIHCL, and CCL reached a maximum accuracy of 93.80, 93.73, and 94.20, respectively. DIHCL contains many hyperparameters. As the authors demonstrated in their study, DIHCL can perform better when these hyperparameters are optimized for the dataset and the architecture. In the case of large architectures, it is not practical to optimize these hyperparameters. CCL, on the other hand, has fewer hyperparameters and with the default hyperparameters used in all experiments, it can achieve significant success, as we have shown in the experiments.

C. Investigating Training Process

In the experiments, we obtained better results by using the CCL compared to the vanilla method. It can be observed in Fig. 2 that CCL achieves a much better minimum point than the vanilla method and other curriculum methods. In trials of cyclic curriculum variants, we observed that using fixed scores at the end of each epoch gives better results than using dynamic scores as in the SPL method. We have seen that the change of training dataset size with time gives better results than using fixed scores at the end of each epoch gives better results than using dynamic scores as in the SPL method. We have seen that the change of training dataset size with time gives better results than using fixed scores at the end of each epoch gives better results than using dynamic scores as in the SPL method. We have seen that the change of training dataset size with time gives better results than using fixed scores at the end of each epoch gives better results than using dynamic scores as in the SPL method. We have seen that the change of training dataset size with time gives better results than using fixed scores at the end of each epoch gives better results than using dynamic scores as in the SPL method. We have seen that the change of training dataset size with time gives better results than using fixed scores at the end of each epoch gives better results than using dynamic scores as in the SPL method.

This optimization process can be performed with the gradient descent (GD) algorithm $w_{i+1} = w_i - \gamma \nabla_w F(w_i)$ where $w$ is all parameters of the model (weights of deep learning model) $\nabla_w$ is the derivative of the loss function with $w$ parameters and $\gamma$ is the learning rate. However, the GD algorithm is not suitable for large-scale problems. Instead, in practice, optimization is performed using the stochastic GD (SGD) algorithm with the equation $w_{i+1} = w_i - \gamma \nabla_w f_i(w_i)$. The difference between the SGD algorithm from the GD algorithm is that it updates the weights at each step, not according to all samples in the training set, but according to a small number of randomly selected samples at each step. Since SGD does not use all the training examples in each step, it makes some errors compared to GD. The error of updating with sample $i$ at time $t$ can be shown with $\text{error}(w_t) = | \nabla_w f_i(w_t) - \nabla_w F(w_t) |$. If the $i$th sample is selected from a uniform distribution, it represents SGD and is calculated where $N$ is the number of samples and $r_i$ is $i$th sample’s probability of being selected in the following equation:

$$r_i = \frac{1}{N}. \quad (2)$$

If samples are selected according to (3), we name it as exponential distributed stochastic GD (ESG)

$$r_i = \frac{\lambda \exp(-\lambda f_i)}{\sum_{n=1}^{N} \lambda \exp(-\lambda f_n)}. \quad (3)$$
Selecting \( r_i \) using ESG makes an error with respect to GD where all examples are used instead of the \( i \)th example. The mean squared error made can be described by the following equation:

\[
E[\text{error}(w_t)] = E[(\nabla_w f_i(w_t) - \nabla_w F(w_t))^2].
\] (4)

We can write the expected value of the error for SGD with (5) and ESG (6) using variance bias decomposition. For \( A = \nabla_w f_i(w_t) \) and \( B = \nabla_w F(w_t) \),

\[
E_{\text{sgd}}[\text{error}(w_t)] = (E_{\text{sgd}}[A] - B)^2 + E_{\text{sgd}}[A^2] - E_{\text{sgd}}[A] \quad (5)
\]

\[
E_{\text{esg}}[\text{error}(w_t)] = (E_{\text{esg}}[A] - B)^2 + E_{\text{esg}}[A^2] - E_{\text{esg}}[A]. \quad (6)
\]

**Theorem 1:** If the loss values come from a normal distribution (ND) \( N(\mu, \sigma^2) \) at time \( t \), the expected error of SGD will be lower than the ESG error

\[
f_i(w_t) \sim N(\mu, \sigma^2) \implies E_{\text{sgd}}[\text{error}(w_t)] < E_{\text{esg}}[\text{error}(w_t)].
\]

**Proof:** For \( A = \nabla_w f_i(w_t) \) and \( B = \nabla_w F(w_t) \), the expected values of SGD and ESG errors are given in (5) and (6) and can be calculated as follows.

Since \( f_i(w_t) \sim N(\mu, \sigma^2) \), \( \text{E}_{\text{sgd}}[A] = \mu \).

From the definition of \( F(w_t) \), \( B = \mu \).

From the definition of variance, \( E_{\text{esg}}[A^2] - E_{\text{esg}}[A]^2 = \sigma^2 \)

\[
E_{\text{esg}}[\text{error}(w_t)] = \sigma^2. \quad (7)
\]

\[
E_{\text{esg}}[A] \text{ and } E_{\text{esg}}[A^2] \text{ can be calculated, respectively,}
\]

\[
E_{\text{esg}}[A] = \frac{E[\lambda f_i(w_t) \exp(-\lambda f_i(w_t))]}{E[\exp(-\lambda f_i(w_t))]}
= \frac{\lambda \exp\left(\frac{\lambda(\sigma^2-2\mu)}{2}\right)(\mu-\lambda \sigma^2)}{\lambda \exp\left(\frac{\lambda(\sigma^2-2\mu)}{2}\right)}
= \mu - \lambda \sigma^2. \quad (8)
\]

\[
E_{\text{esg}}[A^2] = \frac{E[\lambda f_i^2(w_t) \exp(-\lambda f_i(w_t))]}{E[\exp(-\lambda f_i(w_t))]}
= \frac{\lambda^2 \sigma^4 - 2\lambda \mu \sigma^2 + \mu^2 + \sigma^2}{\lambda \exp\left(\frac{\lambda(\sigma^2-2\mu)}{2}\right)}
\]

\[
E_{\text{esg}}[A^2] - E_{\text{esg}}[A]^2 = \lambda^2 \sigma^4 - 2\lambda \mu \sigma^2 + \mu^2 + \sigma^2
- \left(\mu - \lambda \sigma^2\right)^2
= \sigma^2. \quad (9)
\]

Therefore,

\[
E_{\text{esg}}[\text{error}(w_t)] = \left(\mu - \lambda \sigma^2 - \mu\right)^2 + \sigma^2
= \lambda^2 \sigma^4 + \sigma^2. \quad (10)
\]

We calculated the expected value of the SGD error as \( E_{\text{sgd}}[\text{Error}(w_t)] = \sigma^2 \) and the expected value of the ESG error as \( E_{\text{esg}}[\text{error}(w_t)] = \lambda^2 \sigma^4 + \sigma^2 \). Therefore, we found that the expected value of the SGD error is lower than the expected value of the ESG error

\[
\sigma^2 < \lambda^2 \sigma^4 + \sigma^2 \implies E_{\text{sgd}}[\text{Error}(w_t)] < E_{\text{esg}}[\text{error}(w_t)].
\]

**Theorem 2:** If the loss values come from a half ND (HND) at time \( t \) and \( \sigma \lambda < \pi \) the expected value of the ESG error will be lower than the expected value of the SGD error

\[
f_i(w_t) \sim \text{HalfNorm}(\mu, \sigma^2) \land \sigma \lambda < \pi \implies E_{\text{esg}}[\text{error}(w_t)] < E_{\text{sgd}}[\text{error}(w_t)].
\]

**Proof:** From (5), we can calculate the expected error of SGD as follows.

Since \( f_i(w_t) \sim \text{HalfNorm}(\mu, \sigma^2) \)

\[
E_{\text{sgd}}[A] = \mu + \sigma \sqrt{\frac{2}{\pi}} \quad (12)
\]

\[
B = \mu + \sigma \sqrt{\frac{2}{\pi}} \quad (13)
\]

\[
E_{\text{sgd}}[A^2] - E_{\text{sgd}}[A] = \sigma^2 \left(1 - \frac{2}{\pi}\right)
\]

\[
E_{\text{esg}}[\text{Error}(w_t)] = 0^2 + \sigma^2 \left(1 - \frac{2}{\pi}\right)
= \sigma^2 \left(1 - \frac{2}{\pi}\right). \quad (14)
\]

From (6), we can calculate the expected error of ESG as follows:

\[
E_{\text{esg}}[A] = \frac{E[\lambda \nabla_w f_i(w_t) \exp(-\lambda \nabla_w f_i(w_t))]}{E[\exp(-\lambda \nabla_w f_i(w_t)))]}
\]

\[
= \frac{\lambda \exp(\frac{\lambda \sigma^2 - 2\mu}{2}) - \lambda \mu \exp(\frac{\lambda \sigma^2 - 2\mu}{2})}{\sigma \sqrt{\exp(\lambda \sigma^2 - 2\mu)}}
\]

\[
= \mu - \sigma \lambda + \sigma \sqrt{2} \exp(\frac{-2\lambda \mu}{2}) \quad (15)
\]

We call the last part of this expression as \( C \) for ease of representation. \( C = (\sigma \sqrt{2})^{-1} \exp((-2\sigma^2/\lambda^2)) / (\text{erfc}(\lambda \sqrt{2}/\sigma^2)) \)

\[
E_{\text{esg}}[A^2] = \frac{E[\lambda \nabla_w f_i^2(w_t) \exp(-\lambda \nabla_w f_i(w_t))]}{E[\exp(-\lambda \nabla_w f_i(w_t)))]}
= \sigma^4 \lambda^2 - 2\sigma^2 \lambda \mu + \mu^2 + \sigma^2 - C(\sigma^2 \lambda - 2\mu) \quad (16)
\]

\[
E_{\text{esg}}[A^2] - E_{\text{esg}}[A] = \sigma^4 \lambda^2 - 2\sigma^2 \lambda \mu + \mu^2 + \sigma^2
- C(\sigma^2 \lambda - 2\mu - (\mu - \sigma^2 \lambda + C)^2)
\]

\[
= \sigma^2 + C \sigma^2 \lambda - C^2 \quad (17)
\]

\[
E_{\text{esg}}[A] - B = \mu + \sigma \sqrt{\frac{2}{\pi}} - (\mu - \sigma^2 \lambda + C)
\]

\[
= \sigma \sqrt{\frac{2}{\pi}} + \sigma^2 \lambda - C \quad (18)
\]

\[
E_{\text{esg}}[\text{Error}(w_t)] = \left(\sigma \sqrt{\frac{2}{\pi}} + \sigma^2 \lambda - C\right)^2 + \sigma^2 + C \sigma^2 \lambda - C^2
= (\sigma^2 \lambda - C) \left(\sigma^2 \lambda + 2\sigma \frac{2}{\pi} + \sigma^2 \left(\frac{2}{\pi} + 1\right). \quad (19)
\]
Since the $C$ term contains the erfc function, its analytic equivalent cannot be written. For this reason, we evaluate these expressions numerically. We plotted $E_{\text{esg}}[\text{Error}(w_t)] - E_{\text{sgd}}[\text{Error}(w_t)]$ according to $\sigma$ and $\lambda$ in Fig. 4. Fig. 4 shows the region where ESG is lower than SGD. The dark-blue and green area shows that the expected value of ESG error is lower than the expected value of SGD error. The lower base of the green area shows the line where $\lambda \sigma = \pi$. As seen in Fig. 4, $\sigma \lambda < \pi \Rightarrow E_{\text{esg}}[\text{Error}(w_t)] < E_{\text{sgd}}[\text{Error}(w_t)]$.

Observation 1: In deep learning models, the losses of the samples come from the ND before the training starts. As the training continues, the losses of the samples change to the HND.

Observation 2: During training, if the distributions of the training sample’s losses are HND and the model is optimized with ESG, the training sample’s losses change HND to ND. See Fig. 5.

Fig. 5 shows the variation of the losses of the samples during the training of the deep learning model. In the first graph, while the training has not started, the losses are distributed according to the ND. In this case, after SGD is applied to the losses, the second graph is formed. In the second graph, the losses turn into an HND. If SGD is continued to be applied at this stage, the losses will remain in the HND. However, at this stage, if ESG, which makes less error than SGD, is applied to the losses distributed with the HND, the loss values of the samples will again resemble the ND. In the CCL algorithm, this cyclical process continues throughout training.

**Theorem 3:** The expected value of the error of applying SGD and ESG cyclically (CCL algorithm) is smaller than SGD

$$\text{MSE(CCL)} < \text{MSE(SGD)}.$$  \hfill (20)

**Proof:** From Observation-1, the losses come from the ND before training starts. From Observation-2, if the losses come from the ND, when SGD is applied, the losses are distributed according to the HND, and when the losses are distributed according to the HND, when ESG is applied, the losses turn into the ND. According to Theorem 1, if the losses come from an ND, the expected value of the error of applying SGD is lower than the ESG. According to Theorem 2, if the losses come from a HND, the expected value of the error of applying ESG is lower than SGD.

For $S(t) = 1$ CCL uses SGD when losses are normally distributed, and for $S(t) < 1$ CCL uses ESG when losses are half normally distributed. Because of this CCL makes fewer errors than SGD

$$\text{MSE(CCL)} < \text{MSE(SGD)}.$$  \hfill (21)

**Theorem 4:** If $r_t = 1/(f_i \sum (1/f_i))$ is defined for ESG, the expected value of the CCL error is lower than the SGD.

**Proof:** When we solve (22) $x = f_i$ for $\lambda$, we get $\lambda = (\ln(x))/x$. This function’s maximum value is $1/e$ for $x = e^\lambda$, where $\lambda < \pi$.

$$\exp(-\lambda \cdot x) = \frac{1}{x}.$$  \hfill (22)

From Theorem 2 ESG makes fewer errors than SGD where $\lambda \sigma < \pi$. When we replace $\lambda$ with its maximum value, we get

$$\sigma < \pi e.$$  \hfill (23)

For values of $\sigma < \pi e$, using $(1/x)$ instead of $\exp(-\lambda \cdot x)$, the expected value of CCL error is lower than SGD.

The variance of the losses of all datasets used in the experiments is lower than the $\pi e$ constant.

The variances of the losses obtained by the loss functions used in deep-learning architectures comply with (23). For this reason, it is also appropriate to use $(1/x)$ instead of $\exp(-\lambda \cdot x)$ in the CCL.

Our experimental studies have shown that in the later stages of training, the HND transforms into a beta distribution. As the number of epochs increases, since the loss distribution is skewed toward zero, the beta distribution fits better. The left-skewness of beta distribution increases as the training continues. The theorem we explained above also applies to the beta distribution. The distributions do not change from ND to HND, but from left-skewed beta distribution (LSBD) to high LSBD (HLSBD). Similarly, in the theorems described above, ESG has less MSE in the HLSBD, while SGD has less MSE in the LSBD. When ESG is used, LSBD turns into HLSBD. When only SGD is used during the entire training, the beta distribution does not become less left-skewed but gradually becomes more left-skewed throughout the training.

As a result, the three theorems described above also apply when loss distributions turn into beta distributions in the later stages of the training.

**VI. COMPLEXITY ANALYSIS**

In this section, we perform a complexity analysis of the CCL algorithm and compare it to the vanilla method. Assuming the total number of samples $n$ and the total number of epochs $T_1$, the complexity of the vanilla method can be defined as follows

$$C_{\text{vanilla}} = n T_1.$$  

CCL consists of two stages. Determining the curriculum ($C_{\text{pre}}$) and training ($C_{\text{train}}$). Therefore, the complexity of CCL

$$C_{\text{CCL}} = C_{\text{pre}} + C_{\text{train}}.$$
is $C_{\text{CCL}} = C_{\text{pre}} + C_{\text{train}}$. In CCL training, after the curriculum is determined, samples are selected before each epoch (Algorithm 2 line 14). Since this computation is very minor compared to the model update, it is neglected in the complexity analysis. $C_{\text{pre}}$ has the same training procedure as the vanilla. They only differ in epoch count. In the experiments, we trained the vanilla with one-third of the number of epochs. In this case, $C_{\text{pre}}$ is defined as $C_{\text{pre}} = nT_2$ in where $T_2 = T_1/3$.

The complexity $C_{\text{train}}$ depends on the number of CCL train epochs and $S$ representing the dataset sizes defined in Section III-A. Thus, $C_{\text{train}}$ can be defined as $C_{\text{train}} = n \sum_{i=1}^{T_3} S_i$. The CCL training uses a subset of the dataset set at each epoch, as shown in Algorithm 2. The number of samples in the subset at each epoch depends on $ip$, $fp$ and $\alpha$ given in Algorithm 1. In the experiments, we chose $T_3$ as the value for CCL training to update the models the same number of times as vanilla $T_3 = \lceil T_1/E[S] \rceil$ where $E[S]$ is expected value of $S$. Therefore, considering the hyperparameters used in the experiments, $C_{\text{train}}$ and $C_{\text{vanilla}}$ have the same complexity, which is $C_{\text{train}} = C_{\text{vanilla}}$ and $C_{\text{pre}} = C_{\text{vanilla}}/3$. As a result, the complexity of CCL used in the experiments relative to vanilla is as in the following equation:

$$C_{\text{CCL}} = \frac{4}{3} C_{\text{vanilla}}. \quad (24)$$

It can be seen that the difference between the complexity of vanilla and CCL depends mainly on the complexity of the $C_{\text{pre}}$. In our experiments, we found that $T_2 = T_1/3$ was sufficient for a significant increase in success. Increasing $T_2$ further did not significantly change the results. This leads us to the conclusion that CCL can increase success without increasing the complexity too much compared to vanilla.

VII. CONCLUSION

In this study, we were inspired by prior curriculum methods that involve starting with a small training dataset that gradually grows or starting with the entire dataset and decreasing it gradually. We proposed an algorithm that cyclically changes the size of the training dataset. The algorithm includes parameters for the speed of the cycle change and the initial and final percentages. To select the samples for the subset, we used a probabilistic ratio based on predetermined scores from the model. We then trained ANNs using these values.

We examined CCL’s effectiveness on four image classifications and 14 text classification datasets. In the test sets, in ten out of eighteen datasets, we obtained statistically better accuracy values than the vanilla method. The same cycling hyperparameters were found to be effective across different types of datasets and architectures. Furthermore, even better results can be achieved by fine-tuning the hyperparameters to the specific dataset/architecture.

A theoretical explanation of the improved performance of CCL is provided. Some studies reported that CL methods could not outperform the vanilla method, and some reported that more successful results were obtained with the CL method. As shown in the theoretical analysis, under some conditions, the vanilla method makes fewer mistakes, while under some conditions, the curriculum methods make fewer mistakes. This situation provides an explanation for the literature. The CCL, on the other hand, makes fewer errors than the existing curriculum and vanilla methods because it uses the SGD algorithm when SGD makes fewer errors and the ESG algorithm when ESG makes fewer errors. The CCL is found to be more successful than both SGD and other CL methods because it uses the appropriate algorithm for the current situation.

REFERENCES

[1] K. A. Krueger and P. Dayan, “Flexible shaping: How learning in small steps helps,” Cognition, vol. 110, no. 3, pp. 380–394, Mar. 2009.
[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in Proc. 26th Annu. Int. Conf. Mach. Learn., 2009, pp. 41–48.
[3] G. Hacohen and D. Weinshall, “On the power of curriculum learning in training deep networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 2535–2544.
