On The Power of Curriculum Learning in Training Deep Networks

Guy Hacohen [1 2] **Daphna Weinshall** [1]

Abstract

Training neural networks is traditionally done by providing a sequence of random mini-batches sampled uniformly from the entire training data. In this work, we analyze the effect of curriculum learning, which involves the non-uniform sampling of mini-batches, on the training of deep networks, and specifically CNNs trained for image recognition. To employ curriculum learning, the training algorithm must resolve 2 problems: (i) sort the training examples by difficulty; (ii) compute a series of mini-batches that exhibit an increasing level of difficulty. We address challenge (i) using two methods: transfer learning from some competitive "teacher" network, and bootstrapping. In our empirical evaluation, both methods show similar benefits in terms of increased learning speed and improved final performance on test data. We address challenge (ii) by investigating different pacing functions to guide the sampling. The empirical investigation includes a variety of network architectures, using images from CIFAR-10, CIFAR-100 and subsets of ImageNet. We conclude with a novel theoretical analysis of curriculum learning, where we show how it effectively modifies the optimization landscape. We then define the concept of an ideal curriculum, and show that under mild conditions it does not change the corresponding global minimum of the optimization function.

1. Introduction

In order to teach complex tasks, teachers are often required to create a curriculum. A curriculum imposes some order on the concepts that constitute the final task, an order which typically reflects their complexity. The student is then gradually introduced to these concepts by increasing complexity, in order to allow her to exploit previously learned concepts and thus ease the abstraction of new ones. But the use of a curriculum is not limited to complex tasks. When teaching a binary classification task, for example, teachers tend to present typical examples first, followed by the more ambiguous examples (Avrahami et al., 1997).

In many traditional machine learning paradigms, a target function is estimated by a learner (the "student") using a set of training labeled examples (provided by the "teacher"). The field of curriculum learning (CL), which is motivated by the idea of a curriculum in human learning, attempts at imposing some structure on the training set. Such structure essentially relies on a notion of "easy" and "hard" examples, and utilizes this distinction in order to teach the learner how to generalize easier examples before harder ones. Empirically, the use of CL has been shown to accelerate and improve the learning process (e.g. Selfridge et al., 1985; Bengio et al., 2009) in many machine learning paradigms.

When establishing a curriculum for human students, teachers need to address two challenges: (i) Arrange the material in a way that reflects difficulty or complexity, a knowledge which goes beyond what is available in the training set in most machine learning paradigms. (ii) Attend to the pace by which the material is presented – going over the simple ideas too fast may lead to more confusion than benefit, while moving along too slowly may lead to boredom and unproductive learning (Hunkins & Ornstein, 2016). In this paper, we study how these principles can be beneficial when the learner is a neural network.

In order to address the first challenge, Weinshall et al. (2018) introduced the idea of curriculum learning by transfer. The idea is to sort the training examples based on the performance of a pre-trained network on a larger dataset, fine-tuned to the dataset at hand. This approach was shown to improve both the speed of convergence and final accuracy for convolutional neural networks, while not requiring the manual labeling of training data by difficulty.

In our work, we address both challenges. We begin by decomposing CL into two separate - but closely related - subtasks and their corresponding functions. The first, termed *scoring function*, determines the "difficulty" or "complexity"

[1]School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel [2]Edmond and Lily Safra Center for Brain Sciences, The Hebrew University of Jerusalem, Jerusalem 91904, Israel. Correspondence to: Guy Hacohen <guy.hacohen@mail.huji.ac.il>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

of each example in the data. The *scoring function* makes it possible to sort the training examples by difficulty, and present to the network the easier (and presumably simpler) examples first. Scoring is done based on transfer learning as in Weinshall et al. (2018), or bootstrapping as explained below. The second function, termed *pacing function*, determines the pace by which data is presented to the network. The pace may depend on both the data itself and the learner.

We show that the use of different pacing functions can indirectly affect hyper-parameters of the neural network, reminiscent of increased learning rate. As Weinshall et al. (2018) did not employ parameter tuning in their empirical study, the improvement they report might be explained by the use of inappropriate learning rates. As part of our work, we repeat their experimental paradigm while optimizing the network's hyper-parameters (with and without cross-validation), showing improvement in both the speed of convergence and final accuracy in a more reliable way. We then extend these results and report experiments on larger datasets and architectures, which are more commonly used as benchmarks.

We continue by analyzing several *scoring* and *pacing* functions, investigating their inter-dependency and presenting ways to combine them in order to achieve faster learning and better generalization. The main challenge is, arguably, how to obtain an effective *scoring function* without additional labeling of the data. To this end we investigate two approaches, each providing a different estimator for the target *scoring function*: (i) Knowledge transfer as in (Weinshall et al., 2018), based on transfer learning from networks trained on the large and versatile Imagenet dataset. (ii) Bootstrapping based on self-tutoring - we train the network without curriculum, then use the resulting classifier to rank the training data in order to train the same network again from scratch. Unlike curriculum by transfer, bootstrapping does not require access to any additional resources. Both *scoring functions* maintain the required property that prefers points with a lower loss with respect to the target hypothesis. The aforementioned functions are shown in Section 3 to speed up learning and improve the generalization of CNNs.

We investigate three *pacing functions*. (i) *Fixed exponential pacing* presents the learner initially with a small percentage of the data, increasing the amount exponentially every fixed number of learning iterations. (ii) *Varied exponential pacing* allows the number of iterations in each step to vary as well. (iii) *Single-step pacing* is a simplified version of the first protocol, where mini-batches are initially sampled from the easiest examples (a fixed fraction), and then from the whole data as usual. In our empirical setup, the three functions have comparable performance.

In Section 4 we conclude with a theoretical analysis of the effects of curriculum learning on the objective function of neural networks. We show that curriculum learning modifies the optimization landscape, making it steeper while maintaining the same global minimum of the original problem. This analysis provides a framework by which apparently conflicting heuristics for the dynamic sampling of training points can coexist and be beneficial, including SPL, boosting and hard data mining, as discussed under *previous work*.

Previous work. Imposing a curriculum in order to speed up learning is widely used in the context of human learning and animal training (Skinner, 1958; Pavlov, 2010; Krueger & Dayan, 2009). In many application areas, it is a common practice to introduce concepts in ascending order of difficulty, as judged by either the human teacher or in a problem dependent manner (e.g. Murphy et al., 2008; Zaremba & Sutskever, 2014; Amodei et al., 2016). With the rebirth of deep learning and its emerging role as a powerful learning paradigm in many applications, the use of CL to control the order by which examples are presented to neural networks during training is receiving increased attention (Graves et al., 2016; 2017; Florensa et al., 2017).

In a closely related line of work, a pair of teacher and student networks are trained simultaneously, where mini-batches for the student network are sampled dynamically by the teacher, based on the student's output at each time point. As opposed to our method, here the curriculum is based on the current hypothesis of the student, while achieving improved performance for corrupted (Jiang et al., 2018) or smaller (Fan et al., 2018) datasets. Improvement in generalization over the original dataset has not been shown.

In some machine learning paradigms, which are related to CL but differ from it in an essential manner, mini-batches are likewise sampled dynamically. Specifically, in Self-Paced Learning (SPL- Kumar et al., 2010), boosting (Freund et al., 1996), hard example mining (Shrivastava et al., 2016) and even active learning (Schein & Ungar, 2007), mini-batches are sampled at each time point based on the ranking of the training examples by their difficulty with respect to the **current hypothesis** of the model. Thus they differ from CL, which relies on the ranking of training points by their difficulty with respect to some **target hypothesis**.

Confusingly, based on the same ephemeral ranking, SPL advocates the use of easier training examples first, while the other approaches prefer to use the harder examples. Still, all approaches show benefit under different empirical settings (Chang et al., 2017; Zhang et al., 2017). This discrepancy is analyzed in Weinshall et al. (2018), where it is shown that while it is beneficial to prefer easier points with respect to the target hypothesis as advocated by CL, it is at the same time beneficial to prefer the more difficult points with respect to the current hypothesis in agreement with hard data mining and boosting, but contrary to SPL. In contrast, our theoretical analysis (Section 4) is consistent with both heuristics being beneficial under different circumstances.

2. Curriculum Learning

Curriculum learning as investigated here deals with the question of *how to use prior knowledge about the difficulty of the training examples*, in order to sample each mini-batch non-uniformly and thus boost the rate of learning and the accuracy of the final classifier. The paradigm of CL is based on the intuition that it helps the learning process when the learner is presented with simple concepts first.

2.1. Notations and Definitions

Let $X = fX_i g_{i=1}^N = f(x_i, y_i) g_{i=1}^N$ denote the data, where $x_i \in R^d$ denotes a single data point and $y_i \in [K]$ its corresponding label. Let $h_\# : R^d \to [K]$ denote the target classifier (or learner), and mini-batch $B \subset X$ denote a subset of X. In the most common training procedure, which is a robust variant of Stochastic Gradient Descent (SGD), $h_\#$ is trained sequentially when given as input a sequence of mini-batches $[B_1, ..., B_M]$ (Shalev-Shwartz & Ben-David, 2014). The common approach – denoted henceforth *vanilla* – samples each mini-batch B_i uniformly from X. Both in the common approach and in our work, the size of each mini-batch remains constant, to be considered as a hyper-parameter defining the learner.

We measure the difficulty of example (x_i, y_i) by its minimal loss with respect to the set of optimal hypotheses under consideration. We define a *scoring function* to be any function $f : X \to R$, and say that example (x_i, y_i) is more difficult than example (x_j, y_j) if $f(x_i, y_i) > f(x_j, y_j)$. Choosing f is the main challenge of CL, as it encodes the prior knowledge of the teacher.

We define a *pacing function* $g_\# : [M] \to [N]$, which may depend on the learner $h_\#$. The *pacing function* is used to determine a sequence of subsets $X'_1, ..., X'_M \subset X$, of size $jX'_i j = g_\#(i)$, from which $fB_i g_{i=1}^M$ are sampled uniformly. In CL the i-th subset X'_i includes the first $g_\#(i)$ elements of the training data when sorted by the *scoring function* f in ascending order. Although the choice of the subset can be encoded in the distribution each B_i is sampled from, adding a *pacing function* simplifies the exposition and analysis.

2.2. Curriculum Learning Method

Together, each *scoring function* f and *pacing function* $g_\#$ define a curriculum. Any learning algorithm which uses the ensuing sequence $[B_i]_{i=1}^M$ is a **curriculum learning algorithm**. We note that in order to avoid bias when picking a subset of the N examples for some N, it is important to keep the sample balanced with the same number of examples from each class as in the training set. Pseudo-code for the CL algorithm is given in Alg. 1.

In order to narrow down the specific effects of using a *scoring function* based on ascending difficulty level, we examine two control conditions. Specifically, we define 2 additional *scoring functions* and corresponding algorithms: (i) The **anti-curriculum algorithm** uses the *scoring function* $f^0 = -f$, where the training examples are sorted in *descending order* of difficulty; thus harder examples are sampled before easier ones. (ii) The **random-curriculum algorithm** (henceforth denoted **random**) uses a *scoring function* where the training examples are randomly scored.

Algorithm 1 Curriculum learning method

Input: *pacing function* $g_\#$, *scoring function* f, data X.
Output: sequence of mini-batches $\left[B'_1, ..., B'_M\right]$.
sort X according to f, in ascending order
$result \leftarrow []$
for all $i = 1, ..., M$ **do**
  $size \leftarrow g_\#(i)$
  $X'_i \leftarrow X[1, ..., size]$
  uniformly sample B'_i from X'
  append B'_i to $result$
end for
return $result$

2.3. Scoring and Pacing Functions

We evaluate two *scoring functions*: (i) *Transfer scoring function*, computed as follows: First, we take the Inception network (Szegedy et al., 2016) pre-trained on the ImageNet dataset (Deng et al., 2009) and run each training image through it, using the activation levels of its penultimate layer as a feature vector (Caruana, 1995). Second, we use these features to train a classifier and use its confidence score as the *scoring function* for each image[1]. (ii) *Self-taught scoring function*, computed as follows: First, we train the network using uniformly sampled mini-batches (the *vanilla* method). Second, we compute this network's confidence score for each image to define a *scoring function*[2].

The *pacing function* can be any function $g_\# : [M] \to [N]$. However, we limit ourselves to monotonically increasing functions so that the likelihood of the easier examples can only decrease. For simplicity, $g_\#$ is further limited to staircase functions. Thus each *pacing function* is defined by the following hyper-parameters, where *step* denotes all the learning iterations during which $g_\#$ remains constant: *step_length* - the number of iterations in each *step*; *increase* - an exponential factor used to increase the size of the data used for sampling mini-batches in each *step*; *start-*

[1] Similar results are obtained when using different confidence scores (e.g, the classifier's margin), different classifiers (e.g, linear SVM), and different teacher networks (e.g, VGG-16 (Simonyan & Zisserman, 2014) or ResNet (He et al., 2016)), see Appendix A.

[2] *Self-taught* can be used repeatedly, see Appendix B.

ing_percent - the fraction of the data in the initial *step*. An illustration of these parameters can be seen in Fig. 1.

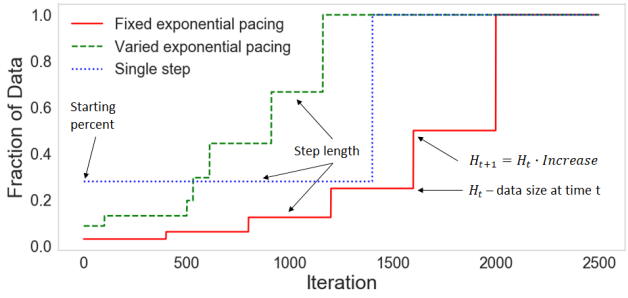


Figure 1. Illustration of the 3 *pacing functions* used below, showing the different hyper-parameters that define each of them (see text). The values of the hyper-parameters used in this illustration were chosen arbitrarily, for illustration only.

We evaluate three *pacing functions*: (i) *Fixed exponential pacing* has a fixed *step_length*, and exponentially increasing data size in each *step*. Formally, it is given by:

$$g_{\#}(i) = \min\left(starting\_percent \;\; inc^{b\frac{i}{step\_length}c}, 1\right) \;\; N$$

(ii) *Varied exponential pacing* is the same as (i), while allowing *step_length* to vary. This method adds additional hyper-parameters, but removes the need to re-tune the learning rate parameters (see Appendix A). (iii) *Single step pacing* entails the simplification of the staircase function into a step function, resulting in fewer hyper-parameters. Formally:

$$g_{\#}(i) = starting\_percent^{\mathbf{1}[i<step\_length]} \;\; N$$

3. Empirical Evaluation

Methodology.[3] We define 6 test cases: **Case 1** replicates the experimental design described in (Weinshall et al., 2018), by using the same dataset and network architecture. The dataset is the "small mammals" super-class[4] of CIFAR-100 (Krizhevsky & Hinton, 2009) - a subset of 3000 images from CIFAR-100, divided into 5 classes. The neural network is a moderate size hand-crafted convolutional network, whose architecture details can be found in Appendix C. **Cases 2** and **3** adopt the same architecture while extending the datasets to the entire CIFAR-10 and CIFAR-100 datasets. **Cases 4** and **5** use a well known public-domain VGG-based architecture[5] (Simonyan & Zisserman, 2014; Liu & Deng, 2015), to classify the CIFAR-10 and CIFAR-100 datasets. **Case 6** adopts the same architecture as cases 1-3, trained with a subset of 7 classes of cats (see Appendix C) from the ImageNet dataset.

[3]All the code used in the paper is available at https://github.com/GuyHacohen/curriculum_learning

[4]Other super-classes achieve similar results, see Appendix A.

[5]Code available at https://github.com/geifmany/cifar-vgg.

Hyper-parameter tuning. As in all empirical studies involving deep learning, the results are quite sensitive to the values of the hyper-parameters, hence parameter tuning is required. In practice, in order to reduce the computation time of parameter tuning, in the curriculum conditions, we performed first grid-search on the curriculum hyper-parameters, followed by a second grid-search on the learning rate parameters, thus avoiding the need to tune a large number of parameters at once. In the non-curriculum conditions, a full 1-stage grid-search was performed. In addition, we varied only the first 2 *step_length* instances in the *varied exponential pacing* condition. Accordingly, *fixed exponential pacing*, *varied exponential pacing* and *single step pacing* define 3, 5 and 2 new hyper-parameters respectively, henceforth called the *pacing* hyper-parameters.

With CL, the use of a *pacing function* affects the optimal values of other hyper-parameters, the learning rate in particular. Specifically, the *pacing function* significantly reduces the size of the dataset at the beginning of the learning, which has the concomitant effect of increasing the effective learning rate at that time. As a result, for a fair comparison, when using the *fixed exponential* or the *single step pacing functions*, the learning rate must be tuned separately for every test condition. This tuning is missing in Weinshall et al. (2018), whose results may therefore be tainted by their arbitrary choice of learning rate. Using the *varied exponential pacing function* can overcome the need for learning rate re-tuning, while adding 2 hyper-parameters (see Appendix A).

As traditionally done (e.g Simonyan & Zisserman, 2014; Szegedy et al., 2016; He et al., 2016), we set an initial learning rate and decrease it exponentially every fixed number of iterations. This method gives rise to 3 learning rate hyper-parameters which require tuning: (i) the initial learning rate; (ii) the factor by which the learning rate is decreased; (iii) the length of each step with constant learning rate[6].

Cross-validation. In grid search, parameters are chosen based on performance on the test set. To avoid contamination of the conclusions, all results were cross-validated, wherein the hyper-parameters are chosen based on performance on a validation set before being used on the test set. For more details on the steps we took to ensure a fair comparison when employing a grid search, see Appendix B.

3.1. Curriculum by Transfer

Case 1: A moderate size network is trained to distinguish 5 classes from CIFAR-100, which are members of the same super-class as defined in the original dataset. Results are shown in Fig. 2. Curriculum learning is clearly and significantly beneficial - learning starts faster, and converges to a better solution. We observe that the performance of CL with

[6]Other tuning methods achieve similar results, see Appendix B.

a random *scoring function* is similar to *vanilla*, indicating that the main reason for the improvement achieved by CL is due to its beneficial *transfer scoring function*. In fact, although tuned separately, the learning rate hyper-parameters for both the *random* and the *curriculum* test conditions are very similar, confirming that the improved performance is due to the use of an effective *transfer scoring function*.

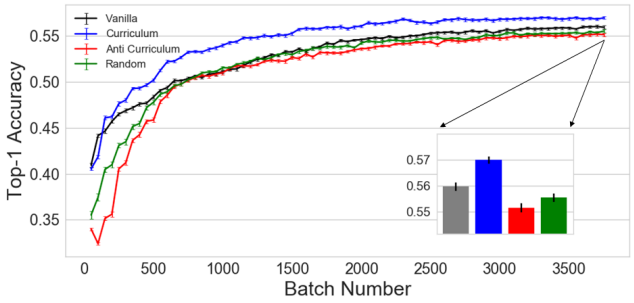


Figure 2. Results in **case 1**, with Inception-based *transfer scoring function* and *fixed exponential pacing function*. Inset: bars indicating the average final accuracy in each condition over the last few iterations. Error bars indicate the STE (STandard Error of the mean) after 50 repetitions. The curriculum method (in blue) reaches higher accuracy faster, and converges to a better solution.

To check the robustness of these results, we repeated the same empirical evaluation using different super-classes of CIFAR-100, with similar results (see Appendix A). Interestingly, we note that the observed advantage of CL is more significant when the task is more difficult (i.e. lower *vanilla* test accuracy). The reason may be that in easier problems there is a sufficient number of easy examples in each mini-batch even without CL. Although the results reported here are based on transfer from the Inception network, we are able to obtain the same results using scoring functions based on transfer learning from other large networks, including VGG-16 and ResNet, as shown in Appendix A.

Cases 2 and 3: Similar empirical evaluation as in **case 1**, using the same moderate size network to classify two benchmark datasets. The results for CIFAR-10 are shown in Fig. 4b and for CIFAR-100 in Fig. 3. Like before, test accuracy with curriculum learning increases faster and reaches better final performance in both cases, as compared to the *vanilla* test condition. The beneficial effect of CL is larger when classifying the more challenging CIFAR-100 dataset.

Cases 4 and 5: Similar empirical evaluation as in **case 1**, using a competitive public-domain architecture. Specifically, we use the Inception-based *transfer scoring function* to train a VGG-based network (Liu & Deng, 2015) to classify the CIFAR-10 and CIFAR-100 datasets. Differently from the previous cases, here we use the *varied exponential pacing function* with a slightly reduced learning rate, as it

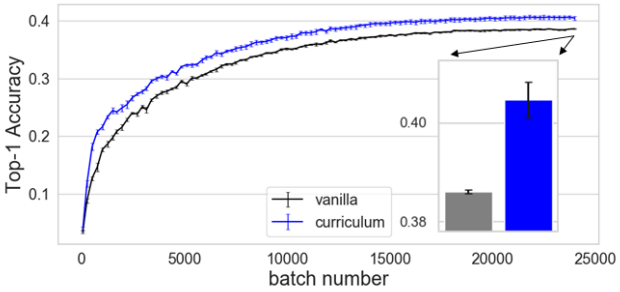
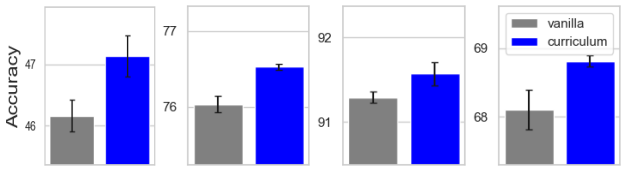


Figure 3. Results in **case 3**, CIFAR-100 dataset, with Inception-based *transfer scoring function* and *fixed exponential pacing function*. Inset: zoom-in on the final iterations, for better visualization. Error bars show STE after 5 repetitions.



(a) ImageNet Cats **(b)** Cifar10 **(c)** Cifar10 VGG **(d)** Cifar100 VGG

Figure 4. Curriculum by transfer learning. Bars indicate the average final accuracy, and error bars indicate the STE. We performed 25 repetitions in (a), 5 in (b) and 3 in (c,d). (a) Cats subset of ImageNet. (b) CIFAR-10, trained on a small network. (c, d) CIFAR-10 and CIFAR-100 respectively, trained on the VGG network.

has the fewest hyper-parameters to tune since learning rate parameters do not need to be re-tuned, an important factor when training large networks. Results for CIFAR-10 are shown in Fig. 4c and for CIFAR-100 in Fig. 4d; in both cases, no data augmentation has been used. The results show the same qualitative results as in the previous cases; CL gives a smaller benefit, but the benefit is still significant.

Case 6: Similar to **case 1**, using the same moderate size network to distinguish 7 classes of cats from the ImageNet dataset (see Appendix C for details). The results are shown in Fig. 4a. Again, the test accuracy in the curriculum test condition increases faster and achieves better final performance with curriculum, as compared to the *vanilla* test condition.

3.2. Curriculum by Bootstrapping

The *self-taught scoring function* is based on the loss of training points with respect to the final hypothesis of a trained network - the same network architecture pre-trained without a curriculum. Using this scoring function, training is re-started from scratch. Thus defined, *curriculum by bootstrapping* may seem closely related to the idea of *Self-Paced Learning* (SPL), an iterative procedure where higher weights are given to training examples that have lower cost with re-

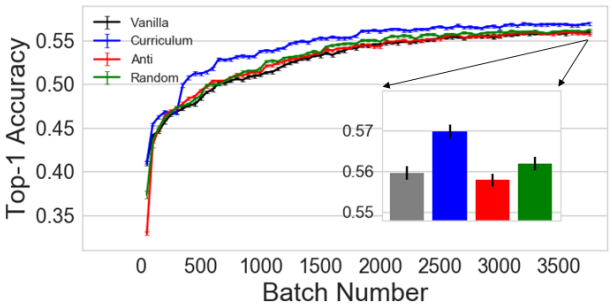


Figure 5. Results in **case 1**, with Inception-based *transfer scoring function* and *Single step pacing function*. Inset: bars indicating the average final accuracy in each condition over the last few iterations. Error bars indicate the STE after 50 repetitions.

spect to the current hypothesis. There is, however, a very important difference between the methods: SPL determines the *scoring function* based on the loss with respect to the current hypothesis (or network), while *bootstrapping CL* scores each point by its loss with respect to the target hypothesis. SPL, it appears, has not yet been introduced to deep neural networks in a way that benefits accuracy.

To compare the *self-taught scoring function* and the *self-paced scoring function*, we investigate their effect on CL in the context of test **case 1**. Final accuracy results are shown in Fig. 7 (the entire learning curves are depicted in Appendix C, Fig. 12). As expected, we see that *bootstrapping CL* improves test accuracy throughout the entire learning session. On the other hand, CL training using the *self-paced scoring function* decreases the test accuracy throughout. This decrease is more prominent at the beginning of the learning, where most of the beneficial effects of the curriculum are observed, suggesting that the *self-paced scoring function* can significantly delay learning.

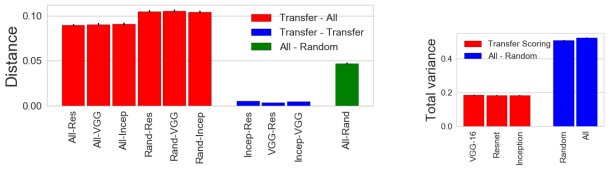
3.3. Alternative Pacing Functions

Single step pacing. Curriculum learning can be costly, and it affects the entire learning protocol via the *pacing function*. At the same time, we observed empirically that the main effect of the procedure seems to have taken place at the beginning of training. This may be due, in part, to the fact that the proposed *scoring function* f is based on transfer from another network trained on a different dataset, which only approximates the unknown *ideal scoring function*. Possibly, since the *scoring function* is based on one local minimum in a complex optimization landscape which contains many local minima, the score given by f is more reliable for low scoring (easy) examples than high scoring (difficult) examples, which may be in the vicinity of a different local minimum.

Once again we evaluate **case 1**, using the *transfer scoring function* and the *single step pacing function*. We see improvement in test accuracy in the curriculum test condition, which resembles the improvement achieved using the *exponential pacing*. Results are shown in Fig. 5. It is important to note that this *pacing function* ignores most of the prior knowledge provided by the *scoring function*, as it only uses a small percent of the easiest examples, and yet it achieves competitive results. Seemingly, in our empirical setup, most of the power of CL lies at the beginning of training.

3.4. Analysis of Scoring Function

In order to analyze the effects of transfer based *scoring functions*, we turn to analyze the gradients of the network's weights w.r.t the empirical loss. We evaluate the gradients using a pre-trained *vanilla* network in the context of **case 1**. First, for each method and each *scoring function*, we collect the subset of points used to sample the first mini-batch according to the *pacing function* $g_{\#}(1)$[7]. For comparison, we also consider the set of all training points, which are used to compute the exact gradient of the empirical loss in batch learning using GD. We then compute the corresponding set of gradients for the training points in each of these subsets of training points, treating each layer's parameters as a single vector, and subsequently estimate the gradients' mean and total variance[8]. We use these measurements to evaluate the coherence of the gradients in the first mini-batch of each *scoring function*. The Euclidean distance between the mean gradient in the different conditions is used to estimate the similarity between the different *scoring functions*, based on the average preferred gradient. We compare the set of gradients defined by using three *transfer scoring functions*, which differ in the teacher network used for scoring the points: 'VGG-16', 'ResNet', and 'Inception'. We include in the comparison the gradients of the *random scoring function* denoted 'Random', and the gradients of the whole batch of training data denoted 'All'. Results are shown in Fig. 6.



(a) Distances between mean gradients. (b) Total variance.

Figure 6. (a) Distance between the mean gradient direction of preferred examples under different *scoring functions*. Each bar corresponds to a pair of mean gradients in two different conditions, see text. (b) The total variance of each set of gradients.

We see in Fig. 6a - blue bars - that the average gradient

[7] In this experiment $g_{\vartheta}(1)$ is set such that it corresponds to 10% of the data or 250 examples. This number was set arbitrarily, with similar qualitative results obtained for other choices.

[8] Total variance denotes the trace of the covariance matrix.

vectors, computed based on the 3 *transfer scoring functions*, are quite similar to each other. This suggests that they are pointing towards nearby local minima in parameters space. We also see - green bar - that the average gradient vector computed using a random subset of examples resembles the exact empirical gradient computed using all the training data. This suggests that a random subset provides a reasonable estimate of the true empirical gradient. The picture changes completely when we compute - red bars - the distance between the average gradient corresponding to one of the 3 transfer *scoring functions*, and the average random gradient or the empirical gradient. The large distances suggest that CL by transfer stirs the weights towards different local minima in parameter space as compared to *vanilla* training.

We see in Fig. 6b that the total variance for the 3 transfer *scoring functions* is much smaller than the total variance of some random subset of the whole training set. This intuitive result demonstrates the difference between training with easier examples and training with random examples, and may – at least partially – explain the need for a different learning rate when training with easier examples.

3.5. Summary of Results

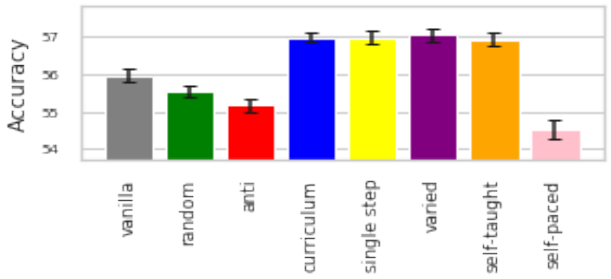


Figure 7. Results in **case 1**, bars showing final accuracy in percent for all test conditions. Error bars indicate STE after 50 repetitions.

Fig. 7 summarizes the main results presented in this section, including: curriculum with an Inception-based *scoring function* for (i) *fixed* exponential pacing (denoted *curriculum*), (ii) *single step* pacing, and (iii) *varied* exponential pacing. It also shows curriculum with fixed exponential pacing for (iv) *self-taught* scoring, and (v) *self-paced* scoring. In addition, we plot the control conditions of *vanilla*, *anti*-curriculum, and *random*. The bars depict the final accuracy in each condition. All the curriculum conditions seem to improve the learning accuracy throughout the entire learning session while converging to similar performance, excluding the self-paced *scoring function* which impairs learning. While different conditions seem to improve the final accuracy in a similar way, the results of *curriculum by transfer* are easier to obtain, and are more robust (see Appendix C for details).

4. Theoretical Analysis

Let H denote a set of hypotheses $h_\#$ defined by the vector of hyper-parameters ϑ. Let $L_\#(X_i)$ denote the loss of hypothesis $h_\#$ when given example X_i. In order to compute the best hypothesis $h_{\#^*}$ from the data, one commonly uses the Empirical Risk Minimization (ERM) framework where[9]

$$L(\vartheta) = \hat{\mathbb{E}}[L_\#] = \frac{1}{N}\sum_{i=1}^{N} L_\#(X_i)$$
$$\vartheta^* = \arg\min_{\#} L(\vartheta) \tag{1}$$

$L(\vartheta)$ denotes the empirical loss given the observed data, thus defining the Risk of choosing hypothesis $h_\#$. (1) can be rewritten as follows:

$$\vartheta^* = \arg\min_{\#} \sum_{i=1}^{N} L_\#(X_i) = \arg\max_{\#} \exp\Big(-\sum_{i=1}^{N} L_\#(X_i)\Big)$$
$$= \arg\max_{\#} \prod_{i=1}^{N} e^{-L_\#(X_i)} \triangleq \arg\max_{\#} \prod_{i=1}^{N} \alpha P(\vartheta | X_i)$$

Thus ERM can be interpreted as Maximum Likelihood (ML) estimation with probability defined by the loss function as $P(\vartheta|X) \propto e^{-L_\#(X)}$.

The choice of loss $L_\#(X)$, and the choice of the estimation framework used to select some optimal hypothesis $h_{\#^*}$, are somewhat arbitrary. In a similar manner we may choose to maximize the average *Utility* $U_\#(X) = e^{-L_\#(X)}$ of the observed data, which is defined as follows

$$U(\vartheta) = \hat{\mathbb{E}}[U_\#] = \frac{1}{N}\sum_{i=1}^{N} U_\#(X_i) \triangleq \frac{1}{N}\sum_{i=1}^{N} e^{-L_\#(X_i)}$$
$$\vartheta^* = \arg\max_{\#} U(\vartheta) \tag{2}$$

The ERM formulation defined in (1) is different from the empirical utility maximization formulation defined in (2). Both formulations can be similarly justified from first principles.

The *scoring function* in curriculum learning effectively provides a Bayesian prior for data sampling. This can be formalized as follows:

$$U_p(\vartheta) = \hat{\mathbb{E}}_{\boldsymbol{p}}[U_\#] = \sum_{i=1}^{N} U_\#(X_i)p(X_i) = \sum_{i=1}^{N} e^{-L_\#(X_i)}p_i$$
$$\vartheta^* = \arg\max_{\#} U_p(\vartheta) \tag{3}$$

Above $p_i = p(X_i)$ denotes the induced prior probability, which is determined by the *scoring function* and *pacing function* of the curriculum algorithm. Thus $p(X_i)$ will always

[9] $\hat{A}$, for any operator A, denotes the empirical estimate of A.

be a non-increasing function of the difficulty level of X_i. In our algorithm, $p(X_i) = \frac{1}{M}$ for M training points whose difficulty score is below a certain threshold, and $p(X_i) = 0$ otherwise. The threshold is determined by the *pacing function* which drives a monotonic increase in the number of points M, thus changing the optimization function in a corresponding manner.

From (3), $U_p(\vartheta)$ is a function of ϑ which is determined by the correlation between two random variables, $U_\#(X)$ and $p(X)$. We rewrite (3) as follows

$$\begin{aligned} U_p(\vartheta) &= \sum_{i=1}^{N}(U_\#(X_i) \quad \hat{\mathbb{E}}[U_\#])(p_i \quad \hat{\mathbb{E}}[p]) + N\hat{\mathbb{E}}[U_\#]\hat{\mathbb{E}}[p] \\ &= \hat{\mathrm{Cov}}[U_\#, p] + N\hat{\mathbb{E}}[U_\#]\hat{\mathbb{E}}[p] = U(\vartheta) + \hat{\mathrm{Cov}}[U_\#, p] \end{aligned} \tag{4}$$

This proves the following result:

Proposition 1 *The difference between the expected utility when computed with and without prior p is the covariance between the two random variables $U_\#(X)$ and $p(X)$.*

Curriculum learning changes the landscape of the optimization function over the hyper-parameters ϑ from $U(\vartheta)$ to $U_p(\vartheta)$. Intuitively, (4) suggests that if the induced prior probability p, which defines a random variable over the input space $p(X)$, is positively correlated with the optimal utility $U_\#(X)$, and more so than with any other $U_\#(X)$, then the gradients in the direction of the optimal parameter $\vartheta^\#$ in the new optimization landscape may be overall steeper.

More precisely, assume that $\vartheta^\#$ maximizes the covariance between $p_\#(X)$ and the utility $U_\#(X)$, namely

$$\arg\max_{\#} U(\vartheta) = \arg\max_{\#} \hat{\mathrm{Cov}}[U_\#, p] = \vartheta^\# \tag{5}$$

Proposition 2 *For any curriculum satisfying (5):*

1. $\vartheta^\# = \arg\max_\# U(\vartheta) = \arg\max_\# U_p(\vartheta)$
2. $U_p(\vartheta^*) \quad U_p(\vartheta) \quad U(\vartheta^*) \quad U(\vartheta) \quad \forall\vartheta$

Proof can be found in Appendix C. We conclude that when assumption (5) holds, the modified optimization landscape induced by curriculum learning has the same global optimum $\vartheta^\#$ as the original problem. In addition, the modified optimization function in the parameter space ϑ has the property that the global maximum at $\vartheta^\#$ is more pronounced.

We define an *ideal curriculum* to be the prior corresponding to the optimal hypothesis (or one of them, if not unique):

$$p_i = \frac{e^{\quad L_{\vartheta^\#}(X_i)}}{C}, \quad C = \sum_{i=1}^{N} e^{\quad L_{\vartheta^\#}(X_i)}$$

From[10] (4):

$$U_p(\vartheta) = U(\vartheta) + \frac{1}{C}\mathrm{Cov}[U_\#, U_\#]$$

The utility at the optimal point $\vartheta^\#$ in parameter space is:

$$U_p(\vartheta^\#) = U(\vartheta^\#) + \frac{1}{C}\mathrm{Cov}[U_\#, U_\#] = U(\vartheta^\#) + \frac{1}{C}\mathrm{Var}[U_\#] \tag{6}$$

In any other point

$$\begin{aligned} U_p(\vartheta) &= U(\vartheta) + \frac{1}{C}\mathrm{Cov}[U_\#, U_\#] \\ &\quad U(\vartheta^\#) + \frac{1}{C}\sqrt{\mathrm{Var}[U_\#]\mathrm{Var}[U_\#]} \end{aligned} \tag{7}$$

Note that if $\mathrm{Var}[U_\#] = b \ \forall\vartheta$ for some constant b, then assumption (5) immediately follows from (7):

$$U_p(\vartheta) \quad U(\vartheta^\#) + \frac{1}{C}\sqrt{b^2} = U_p(\vartheta^\#) \quad =) \quad \vartheta^\# = \arg\max_{\#} U_p(\vartheta)$$

Therefore

Corollary 1 *When using the ideal curriculum, Proposition 2 holds if the variance of the utility function is roughly constant in the relevant range of plausible parameter values.*

From (6) and (7) we can also conclude the following

Proposition 3 *When using the ideal curriculum*

$$U_p(\vartheta^*) \quad U_p(\vartheta) \quad U(\vartheta^*) \quad U(\vartheta) \quad \forall\vartheta : \mathrm{Cov}[U_\#, U_\#] \quad \mathrm{Var}[U_\#]$$

This implies that the optimization landscape is modified to amplify the difference between the optimal parameters vector and all other parameter values whose covariance with the optimal solution (the covariance is measured between the induced prior vectors) is small, and specifically smaller than the variance of the optimum. In particular, this includes all parameters vectors which are uncorrelated (or negatively correlated) with the selected optimal parameter vector.

Discussion: Training a network based on its current hypothesis ϑ_t can be done in one of 2 ways: (i) using a prior which is monotonically increasing with the current utility, as suggested by self-paced learning; (ii) using a prior monotonically decreasing with the current utility, as suggested by hard data mining or boosting. Our analysis suggests that as long as the curriculum is positively correlated with the optimal utility, it can improve the learning; hence both strategies can be effective in different settings. It may even be possible to find a curriculum which is directly correlated with the optimal utility, and that outperforms both methods.

[10] Henceforth we will assume that $N \to \infty$, so that the estimation symbol $\hat{}$ can be omitted.

Appendix

A. Additional Empirical Results

CL with other CIFAR-100 super-classes. In Section 3 we present results when learning to discriminate the "small mammals" super-class of CIFAR-100. Similar results can be obtained for other super-classes of CIFAR-100. Each super-class contains 3000 images, divided into 5 related classes of CIFAR-100. Each class contains 600 images divided into 500 train images and 100 test images. Specifically, we tested our method on the super-classes of "people", "insects" and "aquatic mammals" and found that CL trained on these different super-classes shows the same qualitative results. We note once again that CL is more effective in the harder tasks, namely, the super-classes containing classes that are harder to discriminate (measured by lower *vanilla* accuracy). As an example, Fig. 8 shows results using the "aquatic mammals" super-class, which greatly resembles the results we've seen when discriminating the "small mammals" super-class (cf. Fig.7).

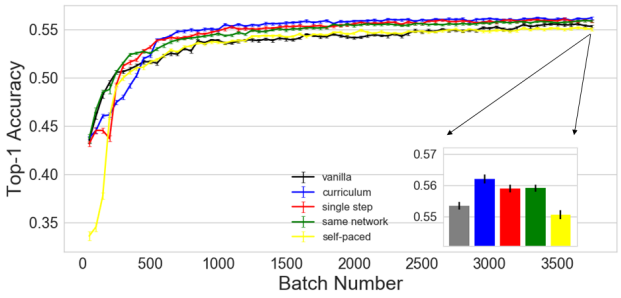
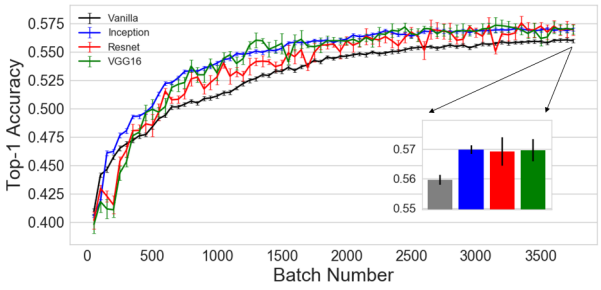


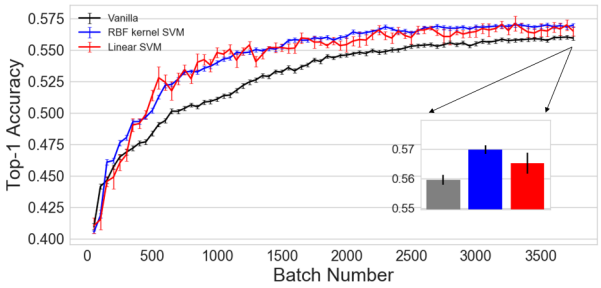
Figure 8. Results under the same conditions as in Fig. 7, using instead the "aquatic mammals" CIFAR-100 super-class. Error bars show STE after 50 iterations.

Transfer based scoring function. In the experiments described in Section 3, when using the *transfer scoring function* defined in Section 2.3, we use the pre-trained Inception network available from https://github.com/Hvass-Labs/TensorFlow-Tutorials. We normalized the data similarly to the normalization done for the neural network, resized it to 299 299, and ran it through the Inception network. We then used the penultimate layer's activations as features for each training image, resulting in 2048 features per image. Using these features, we trained a Radial Basis Kernel (RBF) SVM (Scholkopf et al., 1997) and used its confidence score to determine the difficulty of each image. The confidence score of the SVM was provided by *sklearn.svm.libsvm.predict_proba* from Python's Sklearn library and is based on cross-validation.

Choosing Inception as the teacher and RBF SVM as the classifier was a reasonable arbitrary choice – the same qualitative results are obtained when using other large networks trained on ImageNet as teachers, and other classifiers to establish a confidence score. Specifically, we repeated the experiments with a *transfer scoring function* based on the pre-trained VGG-16 and ResNet networks, which are also trained on Imagenet. The curriculum method using the *transfer scoring function* and *fixed exponential pacing function* are shown in Fig. 9a, demonstrating the same qualitative results. Similarly, we used a linear SVM instead of the RBF kernel SVM with similar results, as shown in Fig. 9b. We note that the STE error bars are relatively large for the control conditions described above because we only repeated these conditions 5 times each, instead of 50 as in the main experiments.



(a) Three competitive networks trained on Imagenet.



(b) Two different classifiers.

Figure 9. Results in **case 1**. Comparing different variants of the *transfer scoring function*. The inset bars show the final accuracy of the learning curves. The error bars shows STE after 50 repetitions for the *vanilla* and Inception conditions with *RBF kernel SVM*, and 5 repetitions for the *ResNet*, *VGG-16* and the *Linear SVM* conditions. (a) Comparing different teacher networks. (b) Comparing different classifiers for the hardness score.

Varied exponential pacing. We define *Varied exponential pacing* similarly to *fixed exponential pacing*, only allowing to change the *step_length* for each step. Theoretically, this method results in additional hyper-parameters equal to the number of performed steps. In practice, to avoid an unfeasible need to tune too many hyper-parameters, we vary only the first two *step_length* instances and fix the rest. This is reasonable as most of the power of the curriculum lies in the first few *steps*. Formally, *Varied exponential pacing* is

given by:

$$g_{\#}(i) = \min\left(starting\_percent \quad increase^{z(i)}, 1\right) \quad N$$

$$z(i) = \sum_{k=1}^{\#steps} \mathbf{1}_{[i > step\_length_k]}$$

where *starting_percent* and *increase* are the same as *fixed exponential pacing*, while *step_length* may vary in each step. The total number of *steps* can be calculated from *starting_percent* and *increase*:

$$\#step = d \quad \log_{increase}(starting\_percent)e$$

This *pacing function* allows us to run a CL procedure without the need for further tuning of learning rate. The additional parameters added by this method control directly the number of epochs the network trains on each dataset size. If tuned correctly, this allows the pacing function to mitigate most of the indirect effect on the learning rate, as it can choose fewer epochs for data sizes which has a large effective learning rate.

Once again we evaluate **case 1**, fixing the learning rate parameters to be the same as in the *vanilla* test condition, while tuning the remaining hyper-parameters as described in Section 2.3 using a grid search with cross-validation. We see improvement in the accuracy throughout the entire learning session, although smaller than the one observed with *fixed exponential pacing*. However, decreasing the learning rate of the *vanilla* by a small fraction and then tuning the curriculum parameters achieves results which are very similar to the *fixed exponential pacing*, suggesting that this method can almost completely nullify the indirect manipulation of the learning rate in the *fixed exponential pacing* function. These results are shown in Fig. 10.

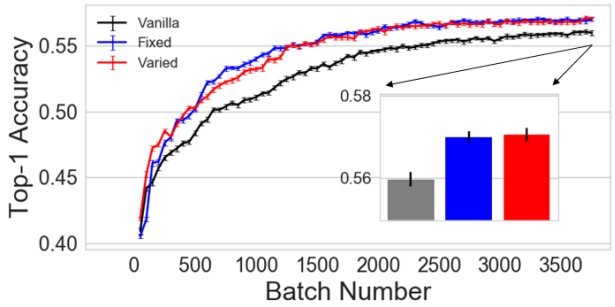


Figure 10. Comparing *fixed exponential pacing* to *varied exponential pacing* in **case 1**, with Inception-based *transfer scoring function*. Inset: bars indicating the average final accuracy in each condition over the last few iterations. Error bars indicate the STE after 50 repetitions.

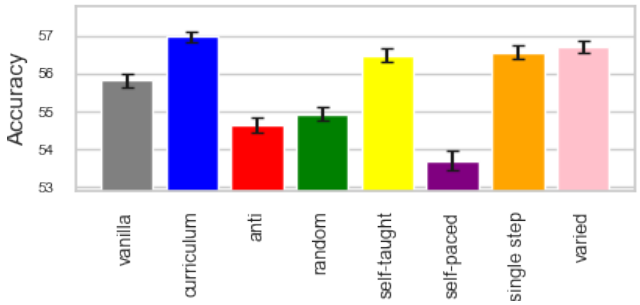


Figure 11. Results in **case 1**, when using the AUC as the grid-search optimization criteria. Bars showing final accuracy in percent for all test conditions. Error bars indicate STE after 50 repetitions.

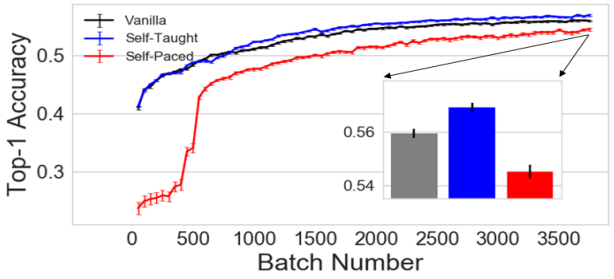


Figure 12. Self-taught learning vs. self-paced learning. Results are in **case 1** with the Inception-based *transfer scoring function*. Inset: bars indicating the average final accuracy in each condition, over the last few iterations. Error bars indicate the STE after 50 repetitions

B. Extended Discussion

Self-taught bootstrapping In principle, the *self-taught scoring function* can be used repeatedly to boost the performance of the network indefinitely: after training the network using a curriculum, we can use its confidence score to define a new *scoring function* and retrain the network from scratch. However, *scoring functions* created by repeating this procedure tend to accumulate errors: once an example is misclassified as being easy, this example will be shown more often in subsequent iterations, making it more likely to be considered easy. In practice, we did not observe any benefit to repeated bootstrapping, and even observed an impairment after a large number of repetitions.

Fair comparison in parameter tuning

When using the moderate size hand-crafted network (**cases 1, 2, 3** and **6**), learning rate tuning is done for the *vanilla* case as well. In these cases, for the *curriculum*, *anti-curriculum* and *random* test conditions, we perform a coarse grid search for the *pacing* hyper-parameters as well as the learning rate hyper-parameters, with an identical range of values for all conditions. For the *vanilla* condition, there are no *pacing*

hyper-parameters. Therefore, we expand and refine the range of learning rate hyper-parameters in the grid search, such that the total number of parameter combinations for each condition is approximately the same.

When using a public domain competitive network (**case 4**), the published learning rate scheduling is used. Therefore we employ the *varied exponential pacing function* without additional learning rate tuning and perform a coarse grid search on the *pacing* hyper-parameters. To ensure a fair comparison, we repeat the experiment with the *vanilla* condition the same number of times as in the total number of experiments done during grid search, choosing the best results. The exact range of values that are used for each parameter is given below in Appendix C. All prototypical results were confirmed with cross-validation, showing similar qualitative behavior as when using the coarse grid search.

Learning Rate Tuning

To control for the possibility that the results we report are an artifact of the way the learning rate is being scheduled, which is indeed the method in common use, we test other learning rate scheduling methods, and specifically the method proposed by Smith (2017) which dynamically changes the learning rate, increasing and decreasing it periodically in a cyclic manner. We have implemented and tested this method using **cases 2** and **3**. The final results of both the *vanilla* and *curriculum* conditions have improved, suggesting that this method is superior to the naïve exponential decrease with grid search. Still, the main qualitative advantage of the CL algorithm holds now as well - CL improves the training accuracy during all stages of learning. As before, the improvement is more significant when the training dataset is harder. Results for **case 3** (CIFAR-100) are shown in Fig. 13.

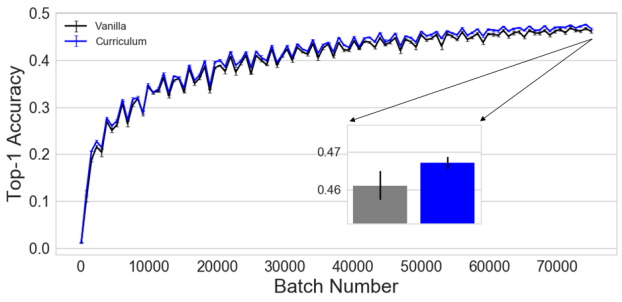


Figure 13. Results under conditions similar to test **case 3** as shown in 3, using cyclic scheduling for the learning rate as proposed by Smith (2017).

C. Methodology, additional details

Exponential Pacing Throughout this work, we use *pacing functions* that increase the data size each *step* exponentially. This is done in line with the customary change of learning rate in an exponential manner.

Architecture Details The moderate-size neural network we used for **cases 1,2,3,6**, is a convolutional neural network, containing 8 convolutional layers with 32, 32, 64, 64, 128, 128, 256, 256 filters respectively. The first 6 layers have filters of size 3 3, and the last 2 layers have filters of size 2 2. Every second layer there is a 2 2 max-pooling layer and a 0.25 dropout layer. After the convolutional layers, the units are flattened, and there is a fully-connected layer with 512 units followed by 0.5 dropout layer. The batch size was 100. The output layer is a fully connected layer with output units matching the number of classes in the dataset, followed by a softmax layer. We trained the network using the SGD optimizer, with cross-entropy loss. All the code will be published upon acceptance.

Grid-search hyper-parameters When using grid search, identical ranges of values are used for the *curriculum, anti-curriculum* and *random* test conditions. Since *vanilla* contains fewer parameters to tune – as it has no *pacing* parameters – we used a finer and broader search range. The range of parameters was similar between different *scoring functions* and *pacing functions* and was determined by the architecture and dataset. The range of parameters for **case 1**: (i) initial learning rate: 0.1 0.01; (ii) learning rate exponential decrease 2 1.1; (iii) learning rate *step size* 200 800; (iv) *step size* 20 400, for both varied and fixed; (v) *increase* 1.1 3; (vi) *starting percent* 4% 15% (note that 4% is in the size of a single mini-batch). For **cases 2, 3** the ranges is wider since the dataset is larger: (i) initial learning rate: 0.2 0.05; (ii) learning rate exponential decrease 2 1.1; (iii) learning rate *step size* 200 800; (iv) *step size* 100 2000, for both varied and fixed; (v) *increase* 1.1 3; (vi) *starting percent* 0.4% 15%. For **cases 4, 5**, the learning rate parameters are left as publicly determined, while the initial learning rate has been decreased by 10% from 0.1 to 0.09. The *pacing* parameter ranges are: (i) *step size* 50 2500, for both varied and fixed; (ii) *increase* 1.1 2; (iii) *starting percent* 2% 20%. For **case 6**: (i) initial learning rate: 0.2 0.01; (ii) learning rate exponential decrease 3 1.05; (iii) learning rate *step size* 300 5000; (iv) *step size* 50 400; (v) *increase* 1.9; (vi) *starting percent* 2% 15%.

ImageNet Dataset Details In **case 6**, we used a subset of the ImageNet dataset ILSVRC 2012. We used 7 classes of cats, which obtained by picking all the hyponyms of the cat synset that appeared in the dataset. The 7 cat classes were:

'Egyptian cat', 'Persian cat', 'cougar, puma, catamount, mountain lion, painter, panther, Felis concolor', 'tiger cat', 'Siamese cat, Siamese', 'tabby, tabby cat', 'lynx, catamount'. All images were resized to size 56 56 for faster performance. All classes contained 1300 train images and 50 test images. The dataset mean was normalized to 0 mean and STD 1 for each channel separately.

Robustness Of Results The learning curves are shown in Fig. 7 were obtained by searching for the parameters that maximize the final accuracy. This procedure only takes into account a few data points, which makes it less robust. In Fig. 11 we plot the bars of the final accuracy of the learning curves obtained by searching for the parameters that maximize the Area Under the Learning Curve. AUC is positively correlated with high final performance while being more robust. Comparing the different conditions using this maximization criterion gives similar qualitative results - the performance in all the curriculum conditions is still significantly higher than the control conditions. However, now the curriculum based on the Inception-based *scoring function* with *fixed exponential pacing* achieves performance that is significantly higher than the other curriculum methods, in evidence that it is more robust.

Theoretical Section Proof for proposition 2:

Proof Claim 1 follows directly from (5), while for claim 2:

$$U_p(\vec{\vartheta}) \quad U_p(\vartheta) = U_p(\vec{\vartheta}) \quad U(\vartheta) \quad \widehat{\mathrm{Cov}}[U_{\#}, p]$$
$$U_p(\vec{\vartheta}) \quad U(\vartheta) \quad \widehat{\mathrm{Cov}}[U_{\#}, p] = U(\vec{\vartheta}) \quad U(\vartheta)$$

■

Acknowledgements

This work was supported in part by a grant from the Israel Science Foundation (ISF), MAFAT Center for Deep Learning, and the Gatsby Charitable Foundations.

References

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pp. 173–182, 2016.

Avrahami, J., Kareev, Y., Bogot, Y., Caspi, R., Dunaevsky, S., and Lerner, S. Teaching by examples: Implications for the process of category acquisition. *The Quarterly Journal of Experimental Psychology: Section A*, 50(3): 586–606, 1997.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48. ACM, 2009.

Caruana, R. Learning many related tasks at the same time with backpropagation. In *Advances in neural information processing systems*, pp. 657–664, 1995.

Chang, H.-S., Learned-Miller, E., and McCallum, A. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, pp. 1002–1012, 2017.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

Fan, Y., Tian, F., Qin, T., Li, X.-Y., and Liu, T.-Y. Learning to teach. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJewuJWCZ.

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning*, pp. 482–495, 2017.

Freund, Y., Schapire, R. E., et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pp. 148–156. Citeseer, 1996.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1311–1320. JMLR. org, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hunkins, F. P. and Ornstein, A. C. *Curriculum: Foundations, principles, and issues*. Pearson Education, 2016.

Jiang, L., Zhou, Z., Leung, T., Li, L.-J., and Fei-Fei, L. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*, pp. 2309–2318, 2018.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Krueger, K. A. and Dayan, P. Flexible shaping: How learning in small steps helps. *Cognition*, 110(3):380–394, 2009.

Kumar, M. P., Packer, B., and Koller, D. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pp. 1189–1197, 2010.

Liu, S. and Deng, W. Very deep convolutional neural network based image classification using small training sample size. In *Pattern Recognition (ACPR), 2015 3rd IAPR Asian Conference on*, pp. 730–734. IEEE, 2015.

Murphy, R. R., Tadokoro, S., Nardi, D., Jacoff, A., Fiorini, P., Choset, H., and Erkmen, A. M. Search and rescue robotics. In *Springer handbook of robotics*, pp. 1151–1173. Springer, 2008.

Pavlov, P. I. Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex. *Annals of neurosciences*, 17(3):136, 2010.

Schein, A. I. and Ungar, L. H. Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3):235–265, 2007.

Scholkopf, B., Sung, K.-K., Burges, C. J., Girosi, F., Niyogi, P., Poggio, T., and Vapnik, V. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE transactions on Signal Processing*, 45 (11):2758–2765, 1997.

Selfridge, O. G., Sutton, R. S., and Barto, A. G. Training and tracking in robotics. In *IJCAI*, pp. 670–672, 1985.

Shalev-Shwartz, S. and Ben-David, S. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Shrivastava, A., Gupta, A., and Girshick, R. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 761–769, 2016.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Skinner, B. F. Reinforcement today. *American Psychologist*, 13(3):94, 1958.

Smith, L. N. Cyclical learning rates for training neural networks. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pp. 464–472. IEEE, 2017.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Weinshall, D., Cohen, G., and Amir, D. Curriculum learning by transfer learning: Theory and experiments with deep networks. In *International Conference on Machine Learning (ICML)*, volume 36, 2018.

Zaremba, W. and Sutskever, I. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

Zhang, D., Yang, L., Meng, D., Xu, D., and Han, J. Spftn: A self-paced fine-tuning network for segmenting objects in weakly labelled videos. In *IEEE CVPR*, pp. 4429–4437, 2017.