Supplementary Material

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).

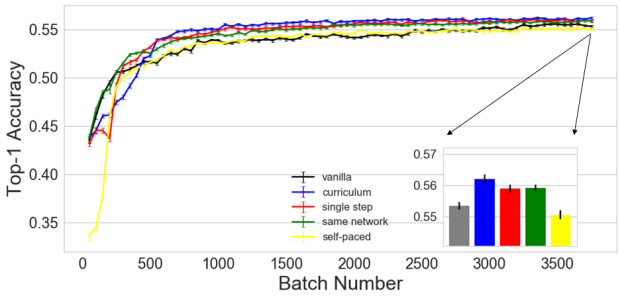
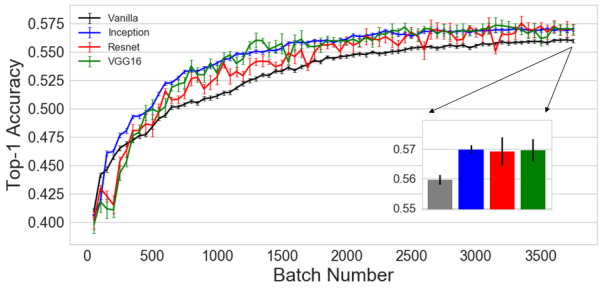


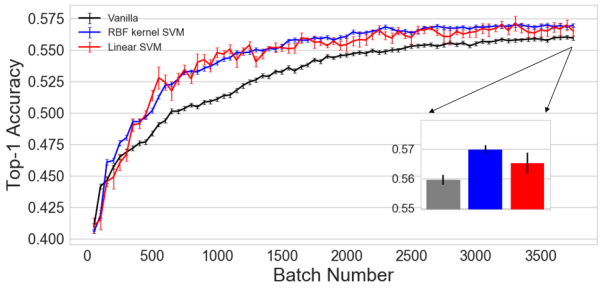
Figure 8. Results under the same conditions as in Fig. 7, using instead the "aquatic mammals" CIFAR-100 super-class. Error bars show STE after 50 iterations.

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:

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

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

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*:

$$\#step = \lceil -\log_{increase}(starting\_percent) \rceil$$

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.

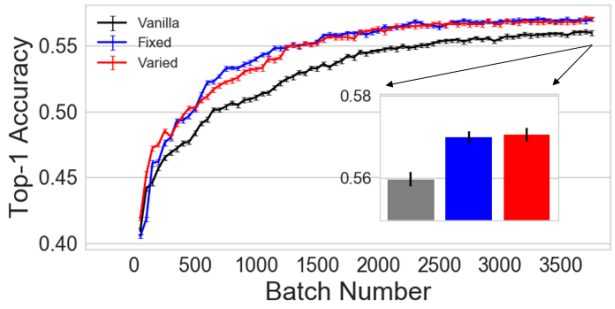


Figure 10. Comparing *fixed exponential pacing* to *varied exponential pacing* in **case 1**, with Inception-based *transfer scoring function*. Inset: bars indicating the average final accuracy in each condition over the last few iterations. Error bars indicate the STE after 50 repetitions.

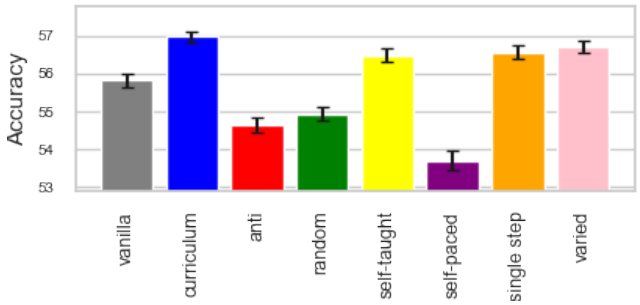


Figure 11. Results in **case 1**, when using the AUC as the grid-search optimization criteria. Bars showing final accuracy in percent for all test conditions. Error bars indicate STE after 50 repetitions.

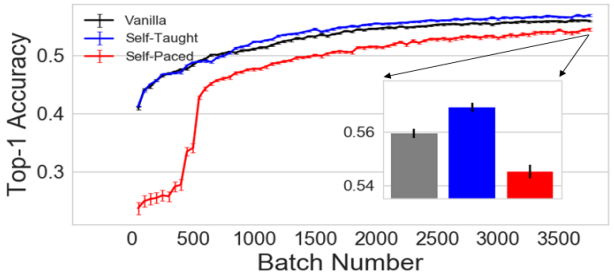


Figure 12. Self-taught learning vs. self-paced learning. Results are in **case 1** with the Inception-based *transfer scoring function*. Inset: bars indicating the average final accuracy in each condition, over the last few iterations. Error bars indicate the STE after 50 repetitions

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 Suppl 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.

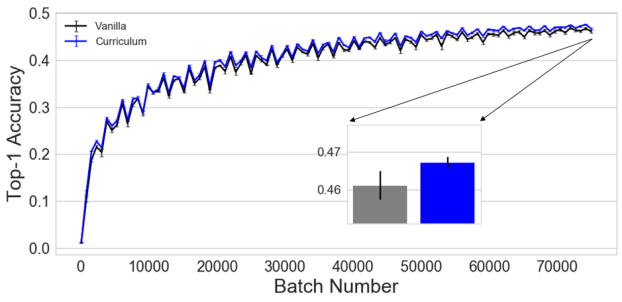


Figure 13. Results under conditions similar to test **case 3** as shown in 3, using cyclic scheduling for the learning rate as proposed by Smith (2017).

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: $0.1 \sim 0.01$; (ii) learning rate exponential decrease $2 \sim 1.1$; (iii) learning rate *step size* $200 \sim 800$; (iv) *step size* $20 \sim 400$, for both varied and fixed; (v) *increase* $1.1 \sim 3$; (vi) *starting percent* $4\% \sim 15\%$ (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: $0.2 \sim 0.05$; (ii) learning rate exponential decrease $2 \sim 1.1$; (iii) learning rate *step size* $200 \sim 800$; (iv) *step size* $100 \sim 2000$, for both varied and fixed; (v) *increase* $1.1 \sim 3$; (vi) *starting percent* $0.4\% \sim 15\%$. 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* $50 \sim 2500$, for both varied and fixed; (ii) *increase* $1.1 \sim 2$; (iii) *starting percent* $2\% \sim 20\%$. For **case 6**: (i) initial learning rate: $0.2 \sim 0.01$; (ii) learning rate exponential decrease $3 \sim 1.05$; (iii) learning rate *step size* $300 \sim 5000$; (iv) *step size* $50 \sim 400$; (v) *increase* 1.9; (vi) *starting percent* $2\% \sim 15\%$.

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:

$$\begin{aligned}\mathcal{U}_p(\tilde{\vartheta}) - \mathcal{U}_p(\vartheta) &= \mathcal{U}_p(\tilde{\vartheta}) - \mathcal{U}(\vartheta) - \hat{\mathrm{Cov}}[U_\vartheta, p] \\ &\geq \mathcal{U}_p(\tilde{\vartheta}) - \mathcal{U}(\vartheta) - \hat{\mathrm{Cov}}[U_{\tilde{\vartheta}}, p] = \mathcal{U}(\tilde{\vartheta}) - \mathcal{U}(\vartheta)\end{aligned}$$

■