Non-Uniform Mini Batch Sampling for Relaxed Curriculum-based Learning

By

GREGORY PASTERNAK

Faculty of Computer Science and Engineering
THE HEBREW UNIVERSITY OF JERUSALEM

A dissertation submitted to the Hebrew University of Jerusalem as a partial fulfillment of the requirements of the degree of MASTER OF SCIENCE in the Faculty of Computer Science and Engineering.

JULY 2020

Abstract

Curriculum-based training of deep neural networks showed that sampling mini-batches from a distribution other than plain uniform on the data sorted by sample difficulty improves both the learning speed and the final classifier accuracy. Previous works propose different approaches to order the data by difficulty, as well as to define a proper pacing function that is used to restrict the training process to relevant parts of data in each epoch. In this work we explore a family of normal pacing functions, which, instead of explicitly limiting the epochs to a part of data and drawing mini-batches uniformly from that part, defines a sampling probability over the full dataset according to a discrete Gaussian-like distribution, and draws mini-batches from it. However, we show that despite this, the networks trained with this approach perform comparably well, while reducing number of curriculum hyper-parameters to tune in the learning process, and allows the model to occasionally see out-of-curriculum examples, essentially guiding the focus of the model in each epoch, instead of restricting it. This approach introduces a previously irrelevant challenge: differently from classical training epoch definition, where the network is trained on each sample exactly once at every epoch, in our setup some of the training examples might be shown to the network much less than others, or even not be shown at all.

TABLE OF CONTENTS

Page

List of Tables **v**

List of Figures **vii**

1 Introduction **1**

2 Related Work **5**

3 Generalized Curriculum Framework **11**

3.1 Data Scoring 11

3.1.1 Teacher and Separate Classifier 12

3.1.2 Teacher Prediction 12

3.1.3 Offline Self-Taught Scoring 12

3.1.4 Online Self-Pacing 12

3.2 Pacing Schedule 13

3.2.1 Normal Distribution as Pacing Schedule 14

3.2.2 Moving and Extending the Gaussian 14

3.2.3 Catastrophic Forgetting 15

3.2.4 Asymmetrical Gaussian 15

3.3 Hardness estimation 16

4 Experimental Evaluation **19**

4.1 Experimental Settings 19

4.1.1 Datasets 19
4.1.2 Teacher Networks 21
4.1.3 Student Networks Architecture and Training Parameters 21
4.1.4 Optimisation hyper-parameters 22
4.1.5 Projection of continuous distributions onto a dataset 25
4.2 Hyper-parameters tuning 25
4.2.1 Cross-validation 25
4.2.2 Balanced per-class sampling 26
4.3 Results 26

5 Summary and Conclusions **29**

A Methodology, additional details **31**

Bibliography **33**

LIST OF TABLES

TABLE **Page**

4.1 Student network performance on test set. Standard CL uses pacing function from [16]. Static pacing schedule as defined in 4.1.4.4, PacedMean in 4.1.4.5, PacedMean+B in 4.1.4.7, Hardness in 4.1.4.9. * Baseline results are as reported in [16]. ** Based on relative improvement as reported in [16]. . . 28

List of Figures

Figure **Page**

4.1 Transfer learning *scoring function* output for each superclass of CIFAR100, ordered by descending scores. Each subplot represents one of 20 superclasses. *Hardness score* is specified in parentheses. 21

4.2 Non-uniform sampling distributions. X axes denote training example index, ordered according to the curriculum. Whenever the distribution differs between training steps, the corresponding step is annotated. 27

4.3 Comparison of student networks performance on test set for all setups. 28

A.1 Transfer learning *scoring function* output for CIFAR10, ordered by descending scores. Each color represents single class. 32

CHAPTER 1

INTRODUCTION

Classical supervised training of deep convolutional neural networks on a given dataset is usually done by repeatedly presenting mini-batches of labeled examples randomly drawn from uniform distribution, with no assumptions of any structured relation between different samples in the dataset, besides coming from the same domain. In practice, often this is done by randomly shuffling the dataset before each training epoch, and splitting the shuffled sequence to equal-sized chunks. In contrast, curriculum learning [5] proposes a way to imitate human learning of complex tasks, which usually imposes the teacher to define a certain curriculum - a partitioning of the learning material to conceptual sections - and follow it by teaching gradually section after section. Often such curriculum is ordered by increasing level of complexity as perceived by the teacher, based on their experience, common practices, expected performance of the students being taught, and more. This allows using previously taught material as a basis for the following material, in part to use simple, concrete, or intuitive concepts to help explain more complex, ambiguous, or abstract ones. Previous works show that applying curriculum learning to the training of neural models improves both their convergence speed, and the final performance they achieve [5, 43]. Clearly, this introduces a hard challenge: obtaining a high-quality curriculum (*data scoring* challenge).

In the context of deep learning, we define the curriculum as an ordering of training examples according to some *scoring*. In this work we focus on the scores which highly correlate with the complexity of the examples, as perceived by the composer of the curriculum. Different learning approaches define such composers very differently, for example: the teacher network in case of transfer learning [19, 43]; the network in training in case of self-paced learning [25]; additional network trained jointly with student network with the purpose of samples scoring (self-paced curriculum) [23]; another instance of same network fully trained with sole purpose of generating curriculum for itself (self-generated curriculum); or any other arbitrary external oracle. Such scoring can either be static (student-independent, pre-computed before the training process), or dynamic (computed during the training process, often utilizing current state of the student being trained).

In addition to obtaining a scoring for training data, another challenge introduced by curriculum learning is the *data pacing* function, which defines what part of the scored data should be shown to the student at each training step. In its simplest form, a pacing function splits the examples to distinct chunks, starts the training with a subset of the chunks (a learning set), and extends it from time to time, based on the number of training steps passed, current model performance, or other factors.

Previous works in deep curriculum learning, while presenting different approaches to solve both the scoring and the pacing challenges, still sample the mini-batches from a given part of data using uniform sampling distribution over that part. This approach however requires careful tuning of the pacing function and the training hyperparameters in order to prevent overfitting in the very beginning of the training, when current learning set is small, and to fully utilize the granularity of individual examples' scores.

We propose to draw mini-batches from a discrete non-uniform sampling distribution defined over all the samples in training set, ordered by a given static or dynamic curriculum. This approach does not limit the student to a specific part of training data at any time, and allows to include both easier and harder examples from the beginning of training, while still focusing the overall training process according to the curriculum.

Effectively, this approach eliminates a need in explicit pacing function over the data or the curriculum, and instead introduces a *pacing schedule* for the parameters of sampling distribution in use, thus proposing a more general solution for the pacing challenge. We show in empirical studies that for particular family of Gaussian distributions this preserves the learning speedup, while further improving final student performance.

In uniform random mini-batch sampling, the randomization is defined such that each sample from the learning set is shown to the student exactly once per epoch (in essence, the data is shuffled and then split to batches), thus assuring that the learning process utilizes all of the training set evenly. In classical curriculum learning, this assumption isn't true any more, as mini-batches are now drawn from a subset of data. However, as the distribution is still uniform over the learning set, one can account for the ratio between learning set size and full training set size, and adjust the training process accordingly, for example by decreasing the learning rate or decreasing number of batches.

Our method, however, implies that mini-batches are drawn from all of the training set non-uniformly, hence we cannot be assured that each sample is seen equal amount of times by the network, and in the extreme case there might be samples that are not seen at all. However, when choosing the right distribution (e.g. in means of ratio between the most improbable sample to the most probable), and/or performing enough training steps, this is unlikely to happen, and, as shown in our experiments, does not affect the overall performance. Nevertheless, as the regular definition of epoch (single cycle over all of training data) becomes irrelevant, we measure the length of training procedure with the number of training steps, each of which utilizes single (not necessarily distinct) sampling distribution, used throughout the training process, where number of batches sampled with each distribution is constant and is usually close to that of regular epoch over the same training set, given other training parameters stay the same (batch size, architecture, etc).

CHAPTER 2

RELATED WORK

Classification of curriculum learning methods [38] proposes to differentiate them by one or more of the main machine learning components the curriculum is applied to - the *data*, the *model*, the *task*, and the *performance measure*.

Data-level and task-level curriculum are the most extensively studied methods, as they seem a natural part of learning process from the human point of perspective, as described in Chapter 1. Data-level curriculum is described by applying partial selection, ordering, or other mutation of the dataset prior or while training, often utilizing feedback from the model being trained. Task-level curriculum refers to training a model on a sequence of tasks of increasing difficulty, for example by first learning to distinguish between the most dissimilar classes of images, then adding classes somewhat similar to already learned ones, and so on. This may seem very similar to data-level curriculum, however it differs fundamentally by altering the learning objective rather then the dataset. Important to note that these types of curriculum often require external measure of difficulty, which may not always be available for a given dataset or class of tasks. For instance, manual scoring of the dataset, besides being extremely time consuming to perform, also requires understanding, or even guessing, what will be easy or hard for an artificial model to grasp, as it was shown previously that it may or may not learn from

the same clues scientists think it will [13, 14]. On the other hand, some intuitive rules do apply, for example, [5] used the complexity of geometry shapes to order them prior training a classification model; in object recognition task [18] describes multiple image parameters as indicative of image hardness, e.g. background-foreground ratio, number of distinct objects, presence of occlusion; while in text tasks the length of input sequence may be strong indicator of its hardness [39].

The model-level curriculum proposes to mutate the model while training, for example by starting with lower-capacity model in early training steps, and increasing its capacity/expressive power progressively. An example for this approach is Curriculum Dropout [29], where the probability parameter of dropout [40] starts low, and increases monotonically as the training progresses. [21] proposes to grow the capacity of generative adversarial network progressively by adding more layers throughout the training. [37] gradually deblurs convolutional activation maps of the network, which implies reduced learning capacity at the beginning of training, and getting more expressive as the training goes on.

The performance-measure-level curriculum is described by direct utilization of learning metrics used in the training process to dynamically alter the order of samples. Most notable example might be Self-Paced Learning (SPL), for instance in [25] the authors propose to use prediction likelihood as such a metric, while in [26] image *objectness* is used (probability that an object of a given class exists in an image window).

As the methods presented in this work fall under data-level and task-level curriculum, the rest of this chapter provides more extensive overview of previous research in these types of curriculum application.

As presented in Chapter 1, the *data scoring* challenge may be addressed in two fundamental ways from the model's point of view - statically and dynamically.

Static model-agnostic curriculum is computed and applied onto the dataset before the training, does not require model feedback and hence does not introduce computational overhead at training time. However, as pointed out earlier, it does require an external hardness metric. Some examples of such metrics for image classification task include:

number of objects of the same class in an image, number of different objects presented in the image, mean area covered by objects, number of truncated or occluded objects; a regression model specifically trained to estimate image difficulty scores [18]; probability scores of an SVM classification model trained on high-quality features transferred from another deep model [43]; a teacher network *prediction depth* of the training set, which was shown to highly correlate with *learning difficulty* (number of training steps after which the model's prediction for an input example has converged) [4].

Curriculum learning by transfer learning introduced in [43] proposes to use a large, expressive network, pre-trained on large-scale in-domain dataset (here and below: "teacher network"), estimate its confidence score on each of the training samples, and sort the samples in accordance before presenting to the smaller network in training (here and below: "student network"), with the idea that a good teacher which learned on similar tasks would be very confident on easy examples, and might be less so on more confusing ones. This approach was shown to improve both the convergence rate and the final performance of convolutional neural networks on image classification tasks.

The *pacing schedule* in this case does not depend on specific dataset, and may or may not utilize student feedback while training to adapt itself; examples of purely independent schedules include the family of ladder-like functions (when the learning set is fixed for a number of training steps, and whole chunks are added to it in next training parts in constant or varied, relatively slow, pace) [16]; linear and logarithmic functions [15], square and higher-order roots [31] (when single or a few examples are frequently added to the learning set). These schedules usually don't assume or compute any structured relation between examples in the dataset besides their order in the curriculum, however they might hint on some insights (for example, the logarithmic function implies that the student network should be able to rapidly pass over easy examples and focus more on the hard ones).

In the **dynamic** setting, the curriculum is not built explicitly for a given dataset regardless of the student model, but instead is queried iteratively using the model itself at each step of training.

Self-paced learning [25, 27], for instance, selects samples for which the likelihood of student prediction of the correct class is high at current step. Note that this is not the same as selecting samples for which the student has high confidence, as confident answer is not necessarily the correct one.

Importance sampling seeks to approximate true complexity for training examples. In the domain of convex problems, it has been extensively studied and multiple adaptive algorithms were proposed to optimize model convergence rate [6, 32]. Later works explored gradient estimates of stochastic gradient descent and showed that the optimal sampling distribution is proportional to the per-sample gradient norm [46], a technique which requires extensive additional computation, and hence is very hard to apply directly to large-scale datasets [2]. However, it is very common for deep embedding learning to sample hard examples due to the available relational inter-sample structure in the embedding space [34, 35].

LILAC (Learning with Incremental Labels and Adaptive Smoothing) [12] proposes to introduce ground truth labels incrementally, starting with examples of a single class, while all other labels are marked with a pseudo-label; later, it utilizes label smoothing to compensate overconfidence that was possibly acquired by the student while learning on low number of classes.

Most related to our approach, [33] and [28] utilize history of losses for previously seen examples to create the mini-batch sampling distribution. Specifically, [33] samples mini-batches proportionally to per-sample loss, using various smoothing techniques to handle the loss fluctuations that are inevitable to happen while training; [28] introduces a large number of additional hyper-parameters that control when to score the examples based on their loss, and how to compute a sampling distribution from the obtained scoring.

Additional sample selection methods include a distribution specifically designed to maximize the diversity of losses in a mini-batch [44]; training additional network instead of defining a sampling distribution to score or select samples for a student network [10, 19, 23]; multiple approaches to stochastic variance-reduced gradient [3, 9, 20]; [22]

derives theoretic upper bound for the gradient norm to reduce variance of the gradient estimates in the stochastic gradient descent step and focus the model on the samples which more significantly change its parameters.

Our approach employs both the prior knowledge about the dataset in question (the static curriculum), and the dynamic non-uniform sampling distribution in order to approximate importance sampling flow without introducing severe additional computational or hyper-parameter tuning overhead.

CHAPTER 3

GENERALIZED CURRICULUM FRAMEWORK

We next describe the generic modular framework for curriculum learning, and introduce the proposed family of normal pacing functions, as well as additional data-agnostic and data-aware pacing functions.

3.1 Data Scoring

A *scoring function* defines a difficulty measure for each example, namely given a training set $X = \{X_1, ..., X_T\}$

$$s : \mathbf{X} \longrightarrow \mathbf{R}^+ \tag{3.1}$$

A curriculum is then defined as an ordering of X by increasing value of $s(X_t)$.

Note that the *scoring function* is applied on all of the training set, without distinction by data classes. This by itself may reveal a scorer-depended structure in the dataset, e.g. some classes may consistently appear easier then other (see AppendixA for examples).

In this work, we focus primarily on scoring functions acquired by transfer learning, as described in [16]. We employ two variants of this approach: using teacher as a

feature extractor in pair with another classifier, properly defined per-sample probability estimates; and using teacher predictions right away.

3.1.1 Teacher and Separate Classifier

Given the teacher network, we first use it as a powerful feature extractor, then train linear SVM classifier on feature vectors of the training examples obtained from the teacher, and use its probability estimates for the correct class as the output of *scoring function*, as defined in [45].

3.1.2 Teacher Prediction

In this case, we feed all the training set into the teacher network in single forward pass, and use normalized activation values of last layer (before classification) for the correct class as the output of the *scoring function*.

In additional experiments, we also use two flavours of self-taught scoring:

3.1.3 Offline Self-Taught Scoring

Here, we remove the need of pre-trained teacher network, and instead first train the student in purely vanilla method (using uniform sampling over all of the train samples), and then use this instance as a weak teacher and proceed as previously. This approach may be repeated multiple times, gradually generating better weak teachers.

3.1.4 Online Self-Pacing

In this approach, similarly to SPL [25], we obtain the scoring at each training step by querying the student for loss values of each training sample at its current state and use them as the scores for the following step. Note however that in the very beginning of the training process, as the student is initialized with random weights, it lacks any training, therefore such scores essentially produce a random permutation of the dataset. In the

few following training steps, the student begins to grasp some knowledge of the data, and so the scores become more and more meaningful.

3.2 Pacing Schedule

In standard curriculum learning (CL), a *pacing function* defines a part of data from which the mini-batches are drawn. Specifically, given a function $p : e \longrightarrow [0,1]$ (fraction), the mini-batches for epoch e will be drawn from $[0, p(e)]$ part of the training set ordered by arbitrary scoring function.

However, as described in Chapter 1, this definition is obsolete in our approach, hence we instead define a sampling distribution pacing schedule over all the training set:

$$p_D : e \longrightarrow D_e$$

$$s.t.$$

$$\forall t \in [1, |X|] \quad D_e : t \longrightarrow [0,1]$$

$$\forall e : \sum_{t=1}^{|X|} D_e(X_t) = 1.$$

where D_e is an arbitrary discrete probability distribution for training step e, and $D_e(t)$ is the probability of a specific sample t in the training set.

This approach provides smooth control on which part of the curriculum we want to focus the student on at each stage of training, while not defining any explicit bounds and still allowing to see samples with out-of-focus difficulty.

Note: Here we define D_e as a discrete probability distribution over the training dataset. However, for simplicity of notation and implementation, we will from now on use continuous distributions such as normal, and denote its mean and standard deviation as a fraction of training dataset, e.g. if a mean of D_e equals 0.5, then the mean of corresponding discrete distribution used to select indices of the dataset will be $0.5 \cdot |X|$. In the next chapter we will explain how the discretization is done to transform any continuous distribution over $\mathbb{R}$ to a bounded discrete distribution over $[1, |X|] \subseteq \mathbb{N}$.

3.2.1 Normal Distribution as Pacing Schedule

We perform experiments with specific family of such pacing schedules, where all the distributions are Gaussian, or are combined of different Gaussian and possibly uniform parts. In its simplest form, such a schedule would comply $D_e \sim \mathbf{N}(\mu, \sigma^2)$, and we tune the values of μ and σ in addition to student network hyper-parameters, replacing the usual CL hyper-parameters. We explore two approaches for this normal pacing schedules: static schedule, when the sampling distribution is fixed over all training steps, emitting $\forall i, j: \; p_D(i) = p_D(j)$; and dynamic schedule, where, inspired by CL intuition, we slowly increase μ, moving the Gaussian over the ordered training set, essentially focusing the student on data of different difficulty at each training step.

3.2.2 Moving and Extending the Gaussian

Following standard CL, the described movement should only progress from easier to harder part of the data. However, we show that in some cases it may be beneficial first to pass over all the data from easy to hard, and then go back and focus more on the intermediate part. This complies with our intuition that, despite the curriculum ordering and original desire to progressively introduce more and more complex concepts, it is more efficient to practice moderately complex examples more after being introduced to all kinds of difficulty, while allowing all kinds of difficulty to be seen. Extreme cases of focusing on samples of specific difficulty are: trying to generalize from the very easy examples, which can let the student overfit on specific task, domain, or dataset; or trying to extract meaningful insights from on the very hard samples, which can only confuse the student without proper basic knowledge and won't contribute to overall learning. Moreover, the examples defined as "hard" may be mislabeled, or out-of-distribution, thus can even harm the training process by introducing noisy concepts.

Similar behaviour can also be achieved by increasing σ with static Gaussian after it finished its pass over the different parts of data. This makes the effective sampling distribution closer to uniform, and lets the student to refine its knowledge about all

levels of complexity after they were gradually introduced and it already holds substantial knowledge. This matches the behaviour shown in [16], where *pacing function* produced a relatively large number of training steps with all the training samples in the learning set, after quickly growing it in the beginning of training process. To further confirm this, we compare the pure Gaussian *pacing schedules* with schedules where after progressively moving the Gaussian mean over the dataset, we add learning steps with explicit uniform distribution over the whole training set.

3.2.3 Catastrophic Forgetting

Following our proposed definition of dynamic pacing scheduling, we note that, in contrast to standard CL, where the sampling distribution always includes the easiest examples in a mini-batch with the same relative probability as the harder ones, our normal distribution tends to rapidly reduce the easier examples' sampling probability over time, as the Gaussian progresses through the training process. This leads to so-called catastrophic forgetting [11], a well-known phenomenon in deep learning, where the model performs poorly on the examples it was shown in the distant past. This may lead the student to de-emphasize the concepts learnt from those easier examples, tune itself to the harder work, and perform poorly on same or other easier samples in the future. We qualitatively test this hypothesis by explicitly cutting off the easy tail of the Gaussian completely, or quite the opposite - replacing the easy tail with uniform part.

3.2.4 Asymmetrical Gaussian

We define an asymmetrical sampling distribution as follows:

$$P_{AD_e}(x_i) = \begin{cases} \frac{P_{D_e^L}(x_i)}{\sum_j P_{D_e^L}(x_j) + P_{D_e^R}(x_j)} & i \leq \mu \\ \frac{P_{D_e^R}(x_i)}{\sum_j P_{D_e^L}(x_j) + P_{D_e^R}(x_j)} & i > \mu \end{cases}$$

In case of asymmetrical Gaussian distribution this yields

$$D_e^L \sim \mathbf{N}(\mu, (\sigma^L)^2)$$

and

$$D_e^R \sim \mathbf{N}(\mu, (\sigma^R)^2)$$

At the price of one additional hyper-parameter, this allows better sampling focus control of the training process by emphasizing or de-emphasizing easier or harder part of the data more than the other, relative to current mean.

Note that setting left-side σ^L to its extremes is the relaxed variant of [non-]forgetting setup: when it approaches zero, it is equivalent to explicit exclusion of easy examples from all following mini-batches, while at infinity it represents uniform non-forgetting.

In contrast, setting right-side σ^R to its extremes is the relaxed variant of the opposite: when σ^R approaches zero, it is equivalent to not seeing hard examples at all until the Gaussian mean moves further (in par with standard CL ideas), while at infinity it will force the student to focus much more on the hard data from the beginning (while still being able to see some easy examples, although with less relative probability than the hard ones), hence contradicting the CL intuition.

For simplicity of notation, given two distributions $D_e^L \sim \mathbf{N}(\mu, (\sigma^L)^2)$ and $D_e^R \sim \mathbf{N}(\mu, (\sigma^R)^2)$ we denote their respective combined asymmetric distribution at combination point p as

$$AD_e \sim \mathbf{N}(\mu, (\sigma^L)^2) \bigcup^p \mathbf{N}(\mu, (\sigma^R)^2)$$

The resulting function is then normalized such that its sum over all samples equals 1 and it will be a well-defined probability density function.

3.3 Hardness estimation

Additionally we employ importance sampling ideas to try and estimate a sampling distribution which will best reflect the complexity distribution of a dataset according to some given scoring function. Intuitively, as the scoring function $s : \mathbf{X} \longrightarrow \mathbb{R}^+$ of any non-synthetic dataset (e.g., natural images) almost surely does not behave in any "nice" way, (e.g. an easily fit parametric distribution such as Gaussian) we should not assume anything about the structure of optimal sampling distribution, and instead should utilize the scores themselves in order to build more tailored distribution:

$$P_{HD_e}(x_i) = \frac{s(x_i)}{\sum_j s(x_j)}$$

CHAPTER 4

EXPERIMENTAL EVALUATION

Here we empirically evaluate the benefits of our proposed approach, comparing the performance of different pacing schedules as described in Chapter 3 on object recognition tasks for multiple well-known datasets.

4.1 Experimental Settings

4.1.1 Datasets

For the evaluation we used 3 datasets: CIFAR10, CIFAR100 [24], CIFAR100-Small-Mammals (superclass 16 of CIFAR100, following the setup in [16]).

CIFAR100 superclass scores In addition to CIFAR100-Small-Mammals, we also evaluated our methods on other CIFAR100 superclasses. In order to better understand the differences between them, we drew intuition from their respective scores. Figure 4.1 shows the scores of each superclass obtained by transfer learning. For a given *scoring function*, we denote the superclass *hardness score* as the area under curve of the scores of all training examples in it, in descending order (higher is easier). We further notice that our methods are more beneficial for student model's final performance when trained on

superclasses with lower hardness scores. For example, superclass 16 (small mammals) which is extensively used in this and previous works, is approximately 0.622 (average difficulty). In addition, we note that superclasses 3 (food containers), 4 (fruit and vegetables), 5 (household electrical devices), 10 (large natural outdoor scenes), 19 (vehicles 2) have much higher *hardness score*, and we confirm empirically that our method performs on par with uniform mini-batch sampling (vanilla), while both achieve significantly higher accuracy than e.g. superclass 16. Conversely, superclass 14 (people) has significantly lower *hardness score*, and the performance gap between vanilla and our method is higher, in favour of curriculum-based method, with overall reduction for both methods. Figure 4.1 shows ordered scores of all superclasses, as well as their respective *hardness score*.

In addition, Figure 4.1 shows that some classes are inherently easier (have higher scores) then others. For example, in superclass 17 (trees), most of the examples from "palm" class appear with much higher scores then other classes. The same phenomenon appears e.g. in superclass 3 (flowers), where "sunflower" class also has higher scores then other classes; in superclass 16 (small mammals), where this is true for "hamster" class; in superclass 14 (people), where "baby" class has significantly higher scores. This may suggest biases in data. For example, sunflowers are much larger than other flowers, and have more regular circular shape; palms grow in more tropical biomes and hence usually appear in "summer" surroundings (sun, green leaves, blue sky, etc.), whereas maples, oaks, pines and willows grow in more temperate biomes and may appear with both "summer" and "winter" (clouds, snow, yellow leaves or branches without leaves, etc.) surroundings).

Similar figure for CIFAR10 dataset can be found in Appendix A.

Overall *hardness score* for full CIFAR100 is 54.26%, whereas for CIFAR10 it is 77.89%.

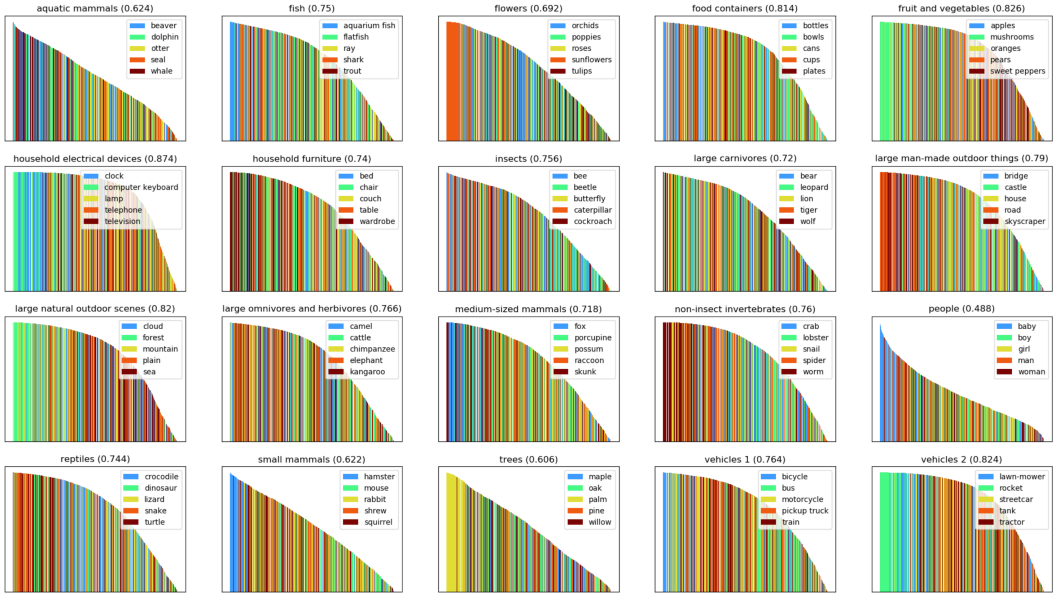

Figure 4.1: Transfer learning *scoring function* output for each superclass of CIFAR100, ordered by descending scores. Each subplot represents one of 20 superclasses. *Hardness score* is specified in parentheses.

4.1.2 Teacher Networks

We use two competitive networks as teachers - Inception-v3 [41] and ResNet-50 [17]. For the case of CIFAR10 and CIFAR100-based datasets, we use features extracted with teacher network as an input to SVM classifier from [30] with default parameters, and use its probability estimates as the scoring function.

4.1.3 Student Networks Architecture and Training Parameters

We now describe the neural networks used in the following experiments for the respective datasets, including the hyper-parameters used during training. All student networks were trained with Exponential-Linear-Unit activations (ELU) [8] for non-linear activation. The networks were augmented with an L_2 regularizer with weight decay factor of $5 \cdot 10^{-3}$.

For CIFAR10 and CIFAR100-based datasets we use a moderate-size network from [43], as well as a larger public-domain VGG-based architecture [1, 36] (see Appendix A for more details). For ImageNet-based datasets we only use the VGG-based architecture.

We train all the student networks using the SGD optimizer without momentum, and use two learning rate schedules, as described in the rest of this chapter under relevant experiments.

4.1.4 Optimisation hyper-parameters

Here we describe the hyper-parameters of baseline vanilla, standard CL, and different non-uniform distribution setups. General (not tuned, but rather pre-selected for a given dataset) parameters are: a desired number of training epochs E (selected a few epochs after model convergence, and fixed between different setups for easier comparison), batch size b; number of batches to sample using each distribution (in Vanilla: number of batches per epoch $d = \lceil \frac{|X|}{b} \rceil$); minimal learning rate lr_{min}.

4.1.4.1 Vanilla

As usually done ([7], [8], [9]), we start with an initial learning rate and then decrease it every fixed number of steps. We experimented with two learning rate decay methods - naïve exponential decay and inverse time decay [7]. Both methods require three hyper-parameters to tune: initial learning rate lr_i, learning rate decay factor lrf, learning rate decay step lrs. In case of exponential decay, the learning rate at epoch e is

$$lr(e) = \max(\frac{lr_i}{lrf^{\lfloor \frac{e}{lrs} \rfloor}}, lr_{min})$$

Whereas for inverse time decay, the learning rate at epoch e is

$$lr(e) = \max(\frac{lr_i}{1 + lrf \cdot \frac{e}{E}}, lr_{min})$$

4.1.4.2 Standard Curriculum

In addition to all the Vanilla hyper-parameters, there are *pacing step* b and *pacing speed* s: standard pacing schedule for some epoch $1 \leq e \leq E$ is then defined as follows:

$$p_{CL}(e;b,s) = \begin{cases} start + b \cdot \frac{e}{E} & e \leq \lceil s \cdot E \rceil \\ 1 & e > \lceil s \cdot E \rceil \end{cases}$$

Note: as shown in [16], b may be constant, may depend on e, or use other heuristics to define pacing schedule that is better suited for a given dataset.

4.1.4.3 General Normal Distribution

The most general Gaussian sampling distribution uses all the hyper-parameters from Vanilla, and adds μ_{start}, μ_{end}, σ^L_{start}, σ^L_{end}, σ^R_{start}, σ^R_{end}, s_μ and s_σ, s.t. for each sampling step e:

$$\mu = \begin{cases} \mu_{start} + (\mu_{end} - \mu_{start}) \cdot \frac{e}{E} & e \leq \lceil s_\mu \cdot E \rceil \\ \mu_{end} & e > \lceil s_\mu \cdot E \rceil \end{cases}$$

$$\sigma^L = \begin{cases} \sigma^L_{start} + (\sigma^L_{end} - \sigma^L_{start}) \cdot \frac{e}{E} & e \leq \lceil s_\sigma \cdot E \rceil \\ \sigma^L_{end} & e > \lceil s_\sigma \cdot E \rceil \end{cases}$$

$$\sigma^R = \begin{cases} \sigma^R_{start} + (\sigma^R_{end} - \sigma^R_{start}) \cdot \frac{e}{E} & e \leq \lceil s_\sigma \cdot E \rceil \\ \sigma^R_{end} & e > \lceil s_\sigma \cdot E \rceil \end{cases}$$

$$D_e \sim \mathbf{N}(\mu,(\sigma^L)^2) \overset{\mu}{\bigcup} \mathbf{N}(\mu,(\sigma^R)^2)$$

Note: in this work we only consider linear change in μ and σ.
The following distributions in 4.1.4.4 - 4.1.4.6 are defined as special cases of this notation.

4.1.4.4 Static Normal Distribution

$\mu_{start} = \mu_{end}$, $\sigma^L_{start} = \sigma^L_{end} = \sigma^R_{start} = \sigma^R_{end}$

4.1.4.5 Moving Normal Distribution

$\mu_{start} < \mu_{end}$, $\sigma^L_{start} = \sigma^L_{end} = \sigma^R_{start} = \sigma^R_{end} = \sigma$

4.1.4.6 Expanding Normal Distribution

$\mu_{start} = \mu_{end}$, $\sigma^L_{start} = \sigma^L_{end} = \sigma^R_{start} = \sigma^R_{end}$

4.1.4.7 Moving with Post-Pace Backout

$\mu_{start} < \mu_{backout} < \mu_{end}$, $\sigma^L_{start} = \sigma^L_{end} = \sigma^R_{start} = \sigma^R_{end} = \sigma$

The training is split to two phases: first, in the pacing phase, Gaussian average μ increases from μ_{start} to μ_{end}; then, in the post-pace phase, it decreases from μ_{end} to $\mu_{backout}$, possibly in different pace. Number of backout training steps does not exceed number of paced training steps.

4.1.4.8 Moving with Post-Pace Uniform

$\mu_{start} < \mu_{end}$, $\sigma^L_{start} = \sigma^L_{end} = \sigma^R_{start} = \sigma^R_{end} = \sigma$

After the pacing reaches μ_{end}, additional training steps are performed with uniform distribution over all the training set. The number of uniform training steps does not exceed 20% of number of paced training steps.

4.1.4.9 Hardness Split Distribution

Note that the distribution HD_e as defined in 3.3 doesn't have any external parameters, hence significantly simplifies the hyper-space tuning while training. However, this results in a monotonically non-increasing distribution, which naturally emphasizes easier examples over the harder ones throughout all of the training process. To prevent this, we introduce the "split" hyper-parameter $k \in [0, |\mathbf{X_t}|]$:

$$HSD_e(x_i; k) = \begin{cases} 1 - HD_e(x_i) & i \le k \\ HD_e(x_i) & i > k \end{cases}$$

Note: the distribution notation used in 4.1.4.4-4.1.4.9 can also be used in the Vanilla and Standard Curriculum setups, when in Vanilla it is simply $D_e \sim \mathbf{U}(0,1)$ for every e,

and in Standard Curriculum it is:

$$D_e^{CL}(start, s) \sim \begin{cases} \mathbf{U}(0, start + b \cdot \frac{e}{E}) & e \leq \lceil s \cdot E \rceil \\ \mathbf{U}(0,1) & e > \lceil s \cdot E \rceil \end{cases}$$

4.1.5 Projection of continuous distributions onto a dataset

In order to be able to sample examples from the dataset into the mini-batch, we need to convert a continuous distribution D_e defined with its probability density function $PDF_{D_e}(x)\ \forall x \in \mathbb{R}$ to a discrete distribution S_e, defined with its probability mass function $PMF_{S_e}(X_i)\ \forall i \in [1, |X|]$:

$$PMF_{S_e}(X_i) = \frac{PDF_{D_e}(i)}{\sum_{j=1}^{|X|} PDF_{D_e}(j)}$$

Note this does not depend on any specific dataset, but rather only on its size $|X|$.

4.2 Hyper-parameters tuning

As in all empirical studies of deep networks, the final model performance is sensitive to the values of relevant hyper-parameters. However, as the Vanilla hyper-parameters are seemingly unrelated to the ones of sampling distributions, we perform coarse grid search first for the distribution parameters only, and then on the Vanilla parameters, significantly reducing the search hyperspace.

In the Vanilla setup, only the latter search is performed. However, this results in a grid search of lower dimension, and makes the comparison unfair for the vanilla method. To overcome this, we expand and refine the ranges of vanilla parameters search, such that the overall number of parameter combinations will be approximately equal in all setups (see Appendix A for more details).

4.2.1 Cross-validation

In grid search, parameters are selected based on performance on the test set. To avoid biased conclusions, we hence select the hyper-parameters based on average performance

on the validation sets of 5-fold cross-validation, retrain from scratch with selected parameters and record performance on the test set.

Additionally, for each proposed method, we perform 5 repetitious training procedures from scratch, and report average of their performance.

4.2.2 Balanced per-class sampling

In supervised learning, class imbalance (significant difference number of samples that belong to distinct classes) in the training set may play crucial role in the model's performance. Usually it is solved using various data manipulations, most popular of which are random under-sampling (removing examples from the larger classes) and random over-sampling (duplicating examples in smaller classes) [42]. These techniques, among others, yield a balanced dataset, where the number of examples in each class is equal.

In vanilla uniform mini-batch sampling, the balanced dataset combined with uniform sampling distribution creates mini-batches which are balanced on average throughout a training epoch. However, whenever the dataset becomes ordered according to the *scoring function*, as described briefly in Chapter 3.1, samples that belong to different classes are no longer uniformly distributed, and hence non-uniform sampling might produce highly-imbalanced mini-batches.

To refute this, we employ per-class sampling distributions, which are similar to the main sampling distribution, but are scaled according to each class separately. The examples sampled from each class are then combined into single mini-batch, reassuring it is balanced. Formally, for normal distribution for each class c with number of samples $|C|$, $D_{e,c} \sim \mathbf{N}(\mu \cdot \frac{|C|}{|X|}, \sigma^2 \cdot \frac{|C|}{|X|})$, where μ and σ^2 are the parameters of D_e. If D_e is combined of two different parts, each part is scaled separately in accordance.

4.3 Results

Figure 4.2 shows examples of baseline and proposed non-uniform sampling distributions.

Top-left: single uniform distribution used throughout the training process.

Top-right: pacing phase of standard curriculum schedule. It can be seen that the training starts with sampling mini-batches only from about 18% of the training set for the first epoch, then progresses to larger parts, until it reaches uniform distribution over all the training set after 10 epochs.

Bottom-left: static Gaussian distribution. As in Vanilla, in this setting the distribution is fixed throughout the training process.

Bottom-right: Paced Mean setting - training process starts with a Gaussian distribution centered at *start* hyper-parameter, and progresses towards *end* hyper-parameter in the pacing stage of the training, after which it stays fixed there until training ends. In this example the pacing phase takes all of the predefined training epochs, and there is no post-pace phase. Note that the maximal sampling probability differs depending on the Gaussian mean, as each distribution is normalized separately, according to its parameters, in range $[1, |X|]$.

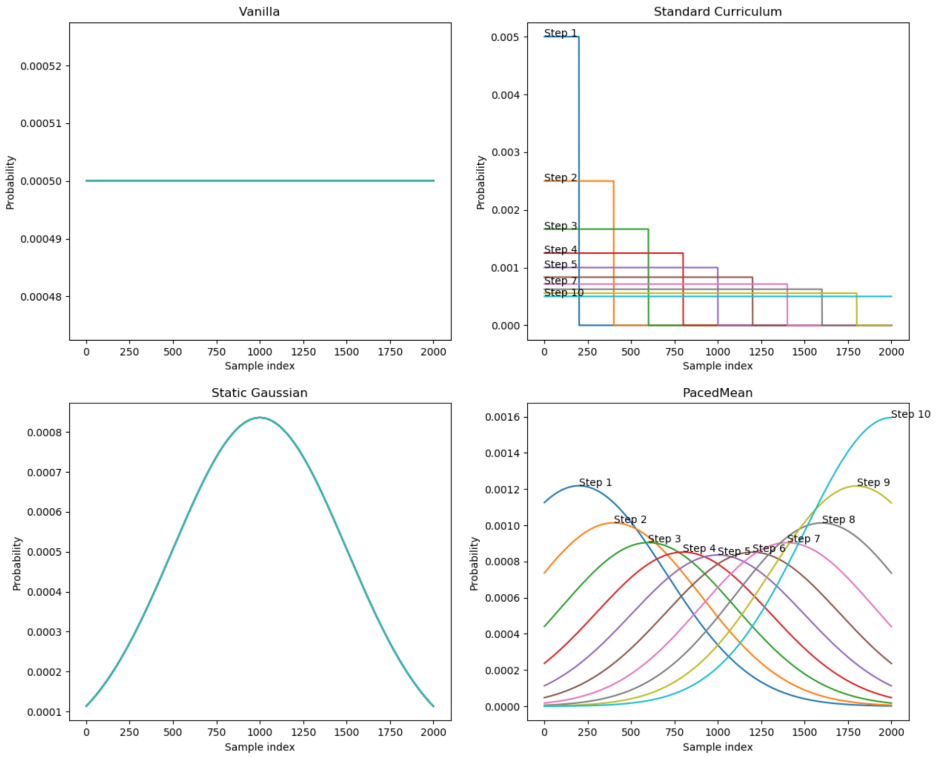

Figure 4.2: Non-uniform sampling distributions. X axes denote training example index, ordered according to the curriculum. Whenever the distribution differs between training steps, the corresponding step is annotated.

Final performance of student networks on test set trained with proposed methods, as well as the baseline vanilla and standard curriculum methods, can be seen in Table 4.1. Comparison of their performance over multiple repetitions can be seen in Figure 4.3. Error bars represent standard error (STE) over 25 repetitions for SmallMammals, and 5 repetitions for CIFAR10 and CIFAR100.

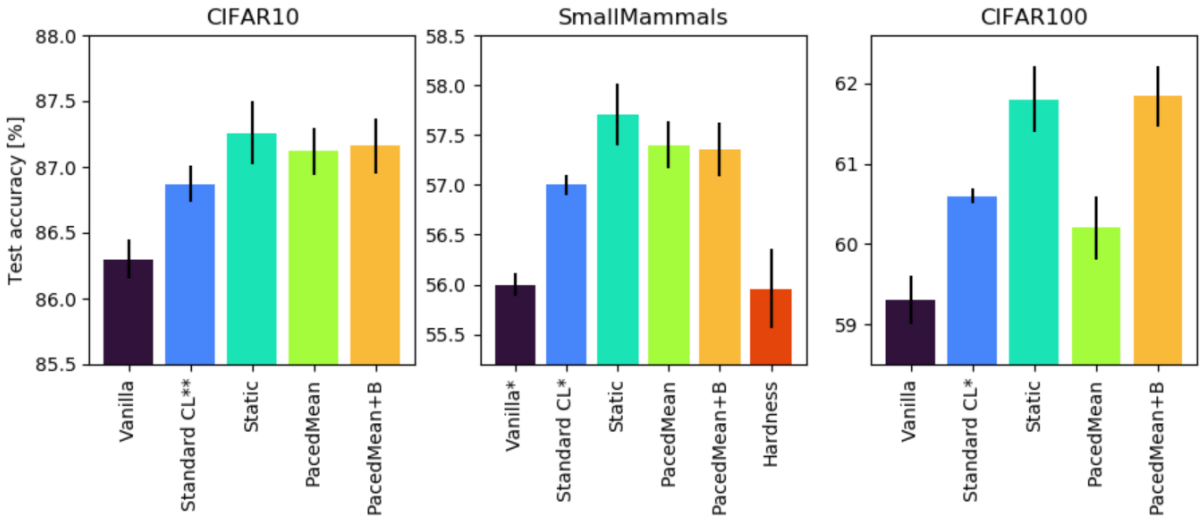

Figure 4.3: Comparison of student networks performance on test set for all setups.

Pacing	CIFAR10	SmallMammals	CIFAR100
Vanilla	86.3%	56.0%*	59.3%
Standard CL	86.87%**	57.0%*	60.6%*
Static	**87.26%**	**57.7%**	61.8%
PacedMean	87.12%	57.4%	60.2%
PacedMean+B	87.16%	57.36%	**61.84%**
Hardness	-	55.96%	-

Table 4.1: Student network performance on test set. Standard CL uses pacing function from [16]. Static pacing schedule as defined in 4.1.4.4, PacedMean in 4.1.4.5, PacedMean+B in 4.1.4.7, Hardness in 4.1.4.9.
* Baseline results are as reported in [16].
** Based on relative improvement as reported in [16].

CHAPTER 5

SUMMARY AND CONCLUSIONS

In this work we explored a new approach to training deep neural networks with curriculum-based schedule, while utilizing prior information about training data. We combined data prior knowledge utilisation from curriculum learning methods with non-uniform mini-batch sampling used in importance mining.

We proposed a relaxed solution to the pacing challenge present in all curriculum learning works, essentially smoothing the training process sample-wise and reducing number of optimization hyper-parameters.

We built up on the intuition that a family of Gaussian sampling distributions is well-suited for curriculum-ordered data; implemented a generic learning framework for training arbitrary models with non-uniform batch sampling. Using the framework, we performed empirical evaluation which shows that our methods surpass the vanilla approach, and perform comparably well or better to previous works in curriculum learning, while introducing less hyper-parameters, and keeping previously established convergence speedup.

We proposed several types of sampling distribution schedules on top of Gaussian distribution. First, we showed that a schedule as simple as a single-parameter Gaussian already performs better than previous methods. We reasoned that the intuition for this

is that samples of average hardness hold enough diverse information to learn from, while not being extreme cases of their representative labels. Then, following general curriculum learning intuition, we showed that slowly advancing average sample difficulty throughout the training process further improves the model performance. Finally, we confirmed that, as in previous works, curriculum paced schedule affects the training mostly in the beginning of training, and an average-difficulty-focused distribution may be used after such short pacing phase to further simplify the training optimization.

In additional experiments, we found that self-controlled curriculum may perform on par with externally provided curriculum, however it requires additional computation for the student model to reach comparable performance.

The field of deep curriculum learning still stays relatively under-explored, and we hope that our work will inspire further research and discovery of new ideas about better utilization of the data properties available at training time, to reduce training costs and simplify machine learning methods integration at any scale. Such ideas may include evaluation of individual data samples' effect on the training process using global student-agnostic or student-specific data properties, dynamically adjusting sampling distribution parameters, or ever dynamically building distributions as the training progresses.

APPENDIX A

METHODOLOGY, ADDITIONAL DETAILS

Architecture details The moderate-size neural network we used for CIFAR10, CIFAR100 and its superclasses, 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 other 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 is set to 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, without momentum, using cross-entropy loss.

Grid-search hyper-parameters When using grid search, identical ranges of values are used for the Static, PacedMean, PacedMean+Backout, and PacedMean+Uniform test conditions. Since Vanilla and Standard CL contain fewer parameters to tune – as it has no pacing parameters – we used a finer and broader search range, such that overall number of parameter combinations is approximately equal between all cases. The range of parameters was similar between different scoring functions and pacing functions and was determined by the architecture and dataset.

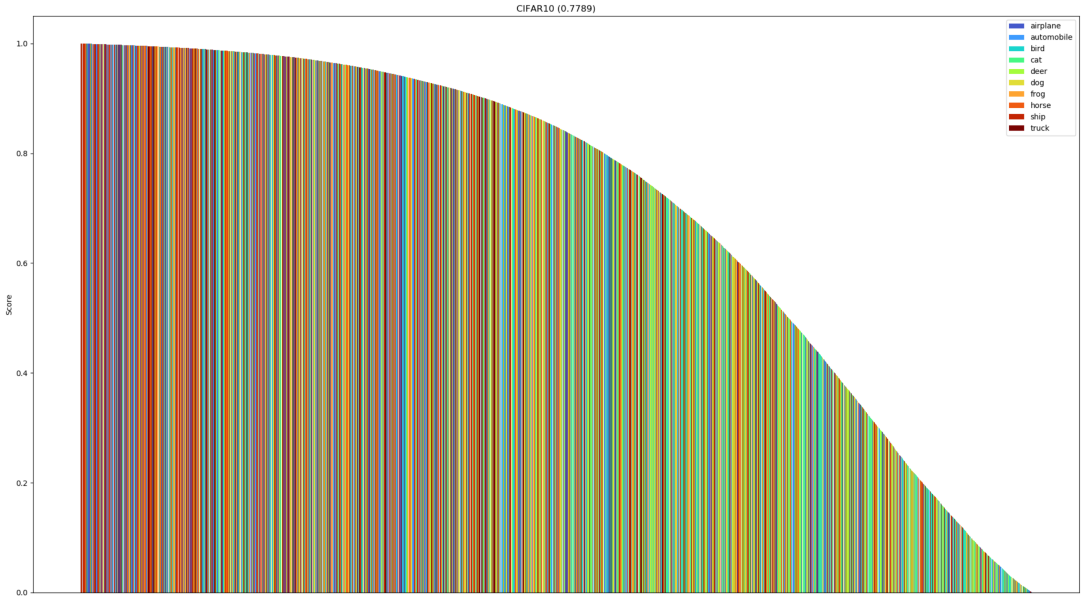

Figure A.1: Transfer learning *scoring function* output for CIFAR10, ordered by descending scores. Each color represents single class.

BIBLIOGRAPHY

[1] *cifar-vgg*.
https://github.com/geifmany/cifar-vgg.

[2] G. ALAIN, A. LAMB, C. SANKAR, A. COURVILLE, AND Y. BENGIO, *Variance reduction in sgd by distributed importance sampling*, (2015).

[3] Z. ALLEN-ZHU, *Katyusha: the first direct acceleration of stochastic gradient methods*, Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing - STOC 2017, (2017).

[4] R. J. N. BALDOCK, H. MAENNEL, AND B. NEYSHABUR, *Deep learning through the lens of example difficulty*, 2021.

[5] Y. BENGIO, J. LOURADOUR, R. COLLOBERT, AND J. WESTON, *Curriculum learning*, in Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, New York, NY, USA, 2009, Association for Computing Machinery, p. 41–48.

[6] A. BORDES, S. ERTEKIN, J. WESTON, AND L. BOTTOU, *Fast kernel classifiers with online and active learning*, J. Mach. Learn. Res., 6 (2005), p. 1579–1619.

[7] F. CHOLLET ET AL., *Keras*.
https://keras.io/api/optimizers/learning_rate_schedules/inverse_time_decay/, 2015.

[8] D.-A. CLEVERT, T. UNTERTHINER, AND S. HOCHREITER, *Fast and accurate deep network learning by exponential linear units (elus)*, (2015).

[9] A. DEFAZIO, F. BACH, AND S. LACOSTE-JULIEN, *Saga: A fast incremental gradient method with support for non-strongly convex composite objectives*, (2014).

[10] Y. FAN, F. TIAN, T. QIN, J. BIAN, AND T.-Y. LIU, *Learning what data to learn*, (2017).

[11] R. M. FRENCH, *Catastrophic forgetting in connectionist networks*, Trends in Cognitive Sciences, 3 (1999), pp. 128–135.

[12] M. R. GANESH AND J. J. CORSO, *Rethinking curriculum learning with incremental labels and adaptive compensation*, (2020).

[13] R. GEIRHOS, J.-H. JACOBSEN, C. MICHAELIS, R. ZEMEL, W. BRENDEL, M. BETHGE, AND F. A. WICHMANN, *Shortcut learning in deep neural networks*, (2020).

[14] R. GEIRHOS, P. RUBISCH, C. MICHAELIS, M. BETHGE, F. A. WICHMANN, AND W. BRENDEL, *Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness*, (2018).

[15] J. GUO, X. TAN, L. XU, T. QIN, E. CHEN, AND T.-Y. LIU, *Fine-tuning by curriculum learning for non-autoregressive neural machine translation*, (2019).

[16] G. HACOHEN AND D. WEINSHALL, *On the power of curriculum learning in training deep networks*, (2019).

[17] K. HE, X. ZHANG, S. REN, AND J. SUN, *Deep residual learning for image recognition*, (2015).

[18] R. T. IONESCU, B. ALEXE, M. LEORDEANU, M. POPESCU, D. P. PAPADOPOULOS, AND V. FERRARI, *How hard can it be? estimating the difficulty of visual search in an image*, 2017.

[19] L. JIANG, Z. ZHOU, T. LEUNG, L. LI, AND L. FEI-FEI, *Mentornet: Regularizing very deep neural networks on corrupted labels*, CoRR, abs/1712.05055 (2017).

[20] R. JOHNSON AND T. ZHANG, *Accelerating stochastic gradient descent using predictive variance reduction*, in Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1, NIPS'13, Red Hook, NY, USA, 2013, Curran Associates Inc., p. 315–323.

[21] T. KARRAS, T. AILA, S. LAINE, AND J. LEHTINEN, *Progressive growing of gans for improved quality, stability, and variation*, 2018.

[22] A. KATHAROPOULOS AND F. FLEURET, *Not all samples are created equal: Deep learning with importance sampling*, (2018).

[23] T.-H. KIM AND J. CHOI, *Screenernet: Learning self-paced curriculum for deep neural networks*, 2018.

[24] A. KRIZHEVSKY, *Learning multiple layers of features from tiny images*, University of Toronto, (2012).

[25] M. P. KUMAR, B. PACKER, AND D. KOLLER, *Self-paced learning for latent variable models*, in Advances in Neural Information Processing Systems 23, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds., Curran Associates, Inc., 2010, pp. 1189–1197.

[26] Y. J. LEE AND K. GRAUMAN, *Learning the easy things first: Self-paced visual category discovery*, in CVPR 2011, 2011, pp. 1721–1728.

[27] H. LI AND M. GONG, *Self-paced convolutional neural networks*, in Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 2110–2116.

[28] I. LOSHCHILOV AND F. HUTTER, *Online batch selection for faster training of neural networks*, (2015).

[29] P. MORERIO, J. CAVAZZA, R. VOLPI, R. VIDAL, AND V. MURINO, *Curriculum dropout*, 2017.

[30] F. PEDREGOSA, G. VAROQUAUX, A. GRAMFORT, V. MICHEL, B. THIRION, O. GRISEL, M. BLONDEL, P. PRETTENHOFER, R. WEISS, V. DUBOURG, J. VANDERPLAS, A. PASSOS, D. COURNAPEAU, M. BRUCHER, M. PERROT, AND E. DUCHESNAY, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[31] G. PENHA AND C. HAUFF, *Curriculum learning strategies for IR: an empirical study on conversation response ranking*, CoRR, abs/1912.08555 (2019).

[32] P. RICHTÁRIK AND M. TAKÁČ, *On optimal probabilities in stochastic coordinate descent methods*, Optimization Letters, 10 (2015), p. 1233–1243.

[33] T. SCHAUL, J. QUAN, I. ANTONOGLOU, AND D. SILVER, *Prioritized experience replay*, (2015).

[34] F. SCHROFF, D. KALENICHENKO, AND J. PHILBIN, *Facenet: A unified embedding for face recognition and clustering*, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2015).

[35] E. SIMO-SERRA, E. TRULLS, L. FERRAZ, I. KOKKINOS, P. FUA, AND F. MORENO-NOGUER, *Discriminative learning of deep convolutional feature point descriptors*, in 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 118–126.

[36] K. SIMONYAN AND A. ZISSERMAN, *Very deep convolutional networks for large-scale image recognition*, 2015.

[37] S. SINHA, A. GARG, AND H. LAROCHELLE, *Curriculum by smoothing*, 2021.

[38] P. SOVIANY, R. T. IONESCU, P. ROTA, AND N. SEBE, *Curriculum learning: A survey*, 2021.

[39] V. I. SPITKOVSKY, H. ALSHAWI, AND D. JURAFSKY, *From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing*, in Human Language

Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Los Angeles, California, June 2010, Association for Computational Linguistics, pp. 751–759.

[40] N. SRIVASTAVA, G. HINTON, A. KRIZHEVSKY, I. SUTSKEVER, AND R. SALAKHUTDINOV, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research, 15 (2014), pp. 1929–1958.

[41] C. SZEGEDY, V. VANHOUCKE, S. IOFFE, J. SHLENS, AND Z. WOJNA, *Rethinking the inception architecture for computer vision*, (2015).

[42] J. VAN HULSE, T. M. KHOSHGOFTAAR, AND A. NAPOLITANO, *Experimental perspectives on learning from imbalanced data*, in Proceedings of the 24th International Conference on Machine Learning, ICML '07, New York, NY, USA, 2007, Association for Computing Machinery, p. 935–942.

[43] D. WEINSHALL, G. COHEN, AND D. AMIR, *Curriculum learning by transfer learning: Theory and experiments with deep networks*, (2018).

[44] C.-Y. WU, R. MANMATHA, A. J. SMOLA, AND P. KRÄHENBÜHL, *Sampling matters in deep embedding learning*, (2017).

[45] T.-F. WU, C.-J. LIN, AND R. C. WENG, *Probability estimates for multi-class classification by pairwise coupling*, J. Mach. Learn. Res., 5 (2004), p. 975–1005.

[46] P. ZHAO AND T. ZHANG, *Stochastic optimization with importance sampling*, (2014).