FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Curriculum Learning of Neural Networks
Student:	Gary Fibiger
Supervisor:	Ing. Daniel Vašata, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2020/21

Instructions

Curriculum learning is a training strategy that can significantly improve the generalization abilities of deep neural networks. It is based on the paradigm that humans and animals learn much better when training examples are organized in a meaningful order which illustrates more concepts, and gradually more complex ones.

1) Review and theoretically describe the state of the art approaches to curriculum learning.
2) Use or implement at least two of the reviewed approaches and experimentally compare their performance on a suitable data set. Use existing implementations as much as possible.
3) Propose a direction for further improvement of curriculum learning approaches.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 2, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Curriculum Learning of Neural Networks

Bc. Gary Fibiger

Department of Applied Mathematics

Supervisor: Ing. Daniel Vašata, Ph.D.

January 8, 2020

Acknowledgements

Many thanks to Dr. Vašata for leading the thesis and to my family and loved ones for their support on my academic journey.

Declaration

In Prague on January 8, 2020 . . . .

Citation of this thesis

Fibiger, Gary. *Curriculum Learning of Neural Networks.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020. Also available from: ⟨`https://github.com/fibigerg/curriculum-learning`⟩.

Abstrakt

Umělé neuronové sítě se běžně trénují na náhodně seřazených datech. V mnoha směrech je tento přístup podobný učení živých organismů, to však nebývá náhodné. Lidé používají učební plány, podle kterých se jejich učení řídí. V posledních letech bylo navrženo mnoho přístupů, které mají za cíl vylepšit trénování neuronových sítí učebními plány. Tato práce obsahuje přehled těchto přístupů. Některé přístupy byly implementovány a experimentálně vyhodnoceny. Výsledky ukazují, že úspěšnost různých učebních plánů je závislá na mnoha faktorech.

Klíčová slova strojové učení, neuronové sítě, učební plán, postupné učení

Abstract

Artificial neural networks are usually trained by observing samples from a training set in a random order. This approach is similar to biological organisms, but their learning process is hardly ever random. Human supervised learning utilizes a curriculum that leads the learning process. Many approaches were proposed to introduce a curriculum to artificial neural networks training in recent years. This thesis provides an overview of those approaches. Many of the approaches were implemented and experimentally evaluated. The results show that different approaches are favorable under different circumstances.

Keywords machine learning, neural networks, curriculum learning, self-paced learning

Contents

List of Figures

List of Tables

Introduction

Motivation and Objectives

Artificial intelligence (AI) is a rapidly growing field of computer science. Although a goal of creating conscious artificial general intelligence seems to be far away, we use many applications of the so-called *weak AI* in our everyday lives in the form of machine learning models. Artificial neural networks (ANN) are one of the most favorite of these models. Despite having access to exponentially growing computational power, models are becoming more complex and there is ever-present need for optimization.

This work investigates one of methods that aim to improve and optimize training of ANN. A general overview of the method called *curriculum learning* is presented in the Chapter 1. Then, related work on the topic is analyzed in the Chapter 2. Subsequently, many experiments inspired by related work are reconstructed in the Chapter 3. Finally, obtained results together with proposed directions for further improvement are discussed in the Chapter 4.

Artificial Neural Networks

ANN are a class of machine learning models that were originally inspired by a biological brain. These models consist of connected units called artificial neurons that compute a non-linear function of a weighted sum of its inputs. The units are usually structured in layers as depicted in Fig. A.

Deep learning goes beyond the biological point of view and generally refers to a multi-level composition. [1] In the context of ANN, it means stacking multiple layers on top of each other. The layers that are not directly connected to any input or output are called hidden. This method creates a deep architecture with many connections parametrizable by their weights and additional biases.

Despite being inspired by the nature, ANN are a purely mathematical model that is trained by adjusting connection weights. The adjustment is done by iteratively propagating model's output error (i.e. the difference between a desired and actual outcome of the model) back to its input by a method known as a *backpropagation*. Starting at an output layer, model's error for every hidden layer is calculated at each step, parameters for that layer are adjusted according to the error and the process repeats for an adjacent layer in the direction of an input layer by a so-called *chain rule*. The amount of an adjustment for each parameter is usually calculated by a *gradient descent*, i.e. adjusting by the greatest change of the model's underlying mathematical function. [1]

A training of ANN is traditionally done by providing a sequence of training samples from a training set. The samples are usually grouped to randomly sampled mini-batches for reduced computational complexity. At each iteration an error for a mini-batch is calculated and parameters are updated. This approach replaces an actual gradient by its approximation; therefore, it is known as stochastic gradient descent (SGD). [2]

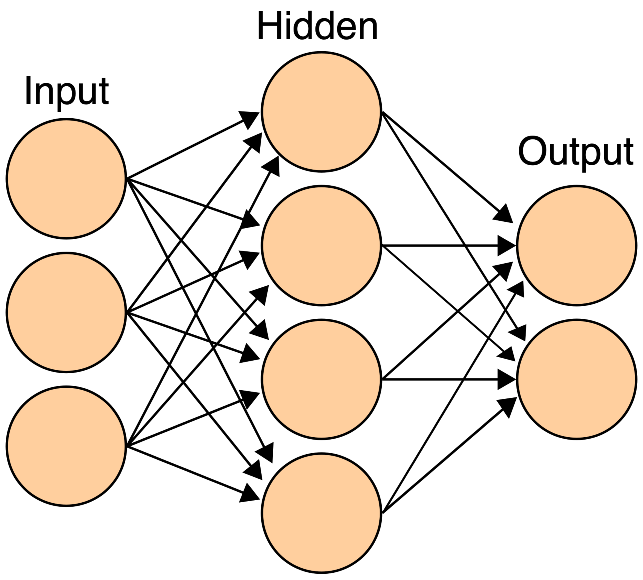


Figure A: A simplified illustration of an ANN structure

Training Improvement

Many methods for improvement of training aspects of ANN were developed. For example, ADAM optimizer [3] replaces SGD for faster learning. One of the main goals of machine learning is to reduce *generalization error*, the ability of a model to correctly predict previously unseen data. Methods such as *data augmentation* [4], *weight regularization* [5] and *dropouts* [6] address this problem.

Curriculum learning is a technique that aims to improve a training performance by studying an order of training samples provided to ANN. Experiments in related work show that usage of a curriculum has a positive effect both on speed of training and quality of results. However, adoption of CL in a real-world usage is still slow.

Chapter 1

Curriculum Learning

Animals learn much better when task are assigned to them in a meaningful order. For example, *shaping* introduced by Skinner [7] is reinforcement of positive approximations to a desired outcome. Each subsequent task becomes a closer approximation for a final task. This idea is routinely exploited in animal training. [8]

Early work connected cognitive science and machine learning by showing that shaping could be beneficial for machine learning algorithms. The basic idea was to start small by learning easier tasks or subtasks and gradually increase difficulty. [9] Similar idea of a guided learning process was proposed in robotics. [10]

Curriculum learning (CL) for ANN training was formalized by Bengio [11]. He showed that a very simple hand-picked multi-stage curriculum can improve *generalization error* and convergence speed of a machine learning model on natural language processing (NLP) and computer vision (CV) tasks. Easy samples were strictly emphasized in his work.

On the other hand, human teaching experiments focused on a learning example selection and presentation were conducted. The results suggest that combination of shaping and a kernel-based coverage model (i.e. sampling from a complement of already learnt examples) achieves the greatest gains. Selecting examples near a decision boundary was confusing for human learners. [12] In further work, it was observed that extreme strategies (e.g. picking a distant example and gradually moving towards a decision boundary) are more common in human learning. This observation is consistent with CL principles. [13]

1.1 Original Definition

Bengio introduced a hypothesis that a well-chosen curriculum strategy can act as a continuation method [14], i.e. it can help to find a better local minima of a non-convex training criterion. His experiments in addition showed, that curriculum strategies appear to operate like a regularizer, their effect is most visible on the test set. [11]

Continuation methods are family of strategies that ensures that local optimization spends most of its time in a well-behaved region of space. The idea behind this approach is to construct new cost functions over the same parameters. Newly constructed cost functions are designed to be increasingly difficult. [1] This approach can also be viewed as optimizing of a smoothed objective and gradually reducing the smoothness.

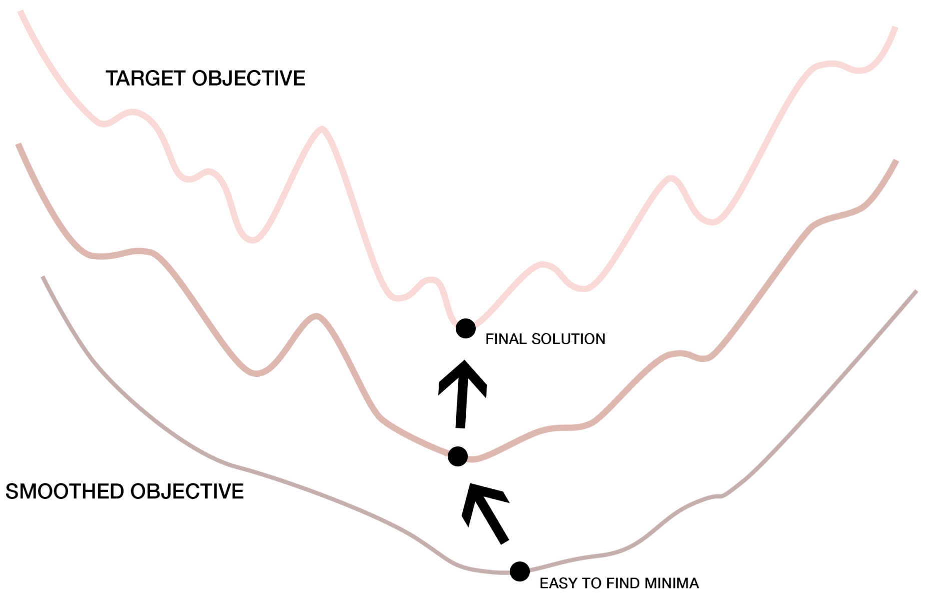


Figure 1.1: Smoothing of an objective function [15]

To apply a continuation method to a training problem we can utilize a sequence of intermediate training sets ordered from a training set with easy to optimize samples to a final set of training samples. A curriculum can be viewed as such a sequence. Each training sample is given a weight on the beginning of a training or during a reweighting. Weights favor easy to learn examples first, gradually introducing more difficult examples by increasing their weight. [11]

1.2 Understanding of Difficulty

It is not always clear how to address the concept of difficulty. A human learner has a different point of view than a machine learning algorithm such as ANN. Therefore, sorting a training set by hand may not reflect the true difficulty and cannot be optimal. Many approaches to this problem were proposed in related work.

Easy examples can simply be characterized as being less “noisy”. An example is considered noisy if it falls on the incorrect side of the decision surface of a Bayes classifier as shown in Bengio [11].

The concept of a *scoring function* was formalized in Hacohen et al. [16]. Two types of scoring functions were defined: (i) *Transfer scoring* function, which uses another model for example weighting. (ii) *Self-thought scoring* function, which uses the learner itself trained on uniformly selected samples. In both cases classifier confidence score for each sample is used for weighting. This approach requires additional preparations and prolongs the whole training process.

Transferring difficulty from another model was proposed in many works. The idea is related to transfer learning, but unlike in transfer learning it is not the instance representation which is being transferred but the weighting of samples. [17][18][19] In other words, another model is presented that predicts sample easiness (i.e. the weight) for the learner.

So far, we discussed hand-picked or fixed curriculums based on prior knowledge. Several ways for curriculum automatization were studied and are closely connected to methods related to CL. Those methods use a feedback from a learner during a training process to change weights of training samples, e.g. *self-paced learning* and *hard example mining.* [20][21]

A data-driven curriculum was proposed in Jiang et al. [17]. This type of approach uses a machine learning model as a teacher that predicts weighs based on features given to him by a learner. The teacher is given a prior knowledge by pre-training and then it is updated multiple times during a training of the learner.

A measurement of difficulty by rate of increase in network complexity was studied in Graves et al. [21]. Many learning progress signals were defined and experimentally studied with variable results. Furthermore, it was shown that guiding a learner through intermediate easier tasks is more likely to yield better results. [22]

1.3 Related Methods

Many authors relate to CL in a broader context of a learning process that utilizes any form of curriculums i.e. weighting of samples. [21][17] Some authors hold strictly to the original CL definition that defines curriculum by prior knowledge, either given to a learner by a human teacher, another machine learning model or the learner itself trained on the same data without use of a curriculum. [19] The term CL in the following text is used in the broader meaning that is in my opinion consistent with Bengio's original work.

Bengio encouraged an active reweighting method during the learning process. At any point during the learning process some examples can be considered "too easy", while other can be considered "too difficult". He stated, that easy examples do not help to improve the model further, while difficult examples are too hard to assess at the given point.

1.3.1 Self-Paced Learning

Self-paced learning (SPL) can be viewed as such a reweighting method. In many real-word applications weighting of samples beforehand might be difficult or computationally intricate. SPL refers to a method where the curriculum i.e. weights of samples are determined by the learner itself. In context of human education system, it would refer to a student that chooses what to learn. [23]

This approach iteratively selects a set of easy examples to train on and updates all sample weights at each iteration by the result of the selected examples. Sample is considered easy if it is easy to predict its true output. Similarly, easy examples are the ones whose correct output can be predicted easily. [23]

SPL determines a scoring function based on a loss with respect to the current hypothesis i.e. current progress of the training, while CL by the original definition scores each sample by its loss with respect to the target hypothesis i.e. previous finished training. [16]

In the original definition of CL, a curriculum is given to the learner beforehand and remains fixed. While in SPL the curriculum is updated at each iteration by a learner progress. CL has advantage of a prior knowledge usage, but it is not flexible to the learning process from optimization perspective. On the other hand, SPL is limited in the usage of prior knowledge. Proposed self-paced curriculum learning (SPCL) method joins the two described approaches and refers to "instructor–student" collaboration. This method uses both prior knowledge and evolving information from the student. [24]

1.3.2 Hard Example Mining

It was shown that emphasizing difficult training examples can speed up learning process. The method was originally called *bootstrapping* and was used for training SVM models with a main idea of selecting examples that give false positive results. [25] The method is now called *hard example mining* in the newer work related to ANN.

The original idea of *hard example mining* for ANN was to hold two training sets, one to train on and one to iteratively select false positive examples from. The automatic method proposed in Shrivastava et al. [26] is similarly to SPL, it is based on the current loss for each data sample. Contrary to SPL, examples with a higher loss are given more weight during a training.

1.3.3 Active Learning

Active learning deals with a problem in which unlabeled data are common but manual labeling is expensive. The key idea of active learning is that a learner can achieve better results when it is trained on data it chooses by itself. An outside oracle (e.g. a manual annotator) is iteratively used to provide labels for unlabeled samples when requested by the learner. [27]

Chapter 2

Related Work

This chapter analyses technical details of many methods that are used to implement *curriculum learning* in related work. The methods are divided into the following sections based on the type of curriculum they use.

Transfer learning methods utilize another pre-trained model as a curriculum with a prior knowledge (Section 2.1).

Automated curriculum methods determine a curriculum during a model's training process (Section 2.2).

Subtask selection methods use a curriculum to select an appropriate task or a label to train on and are mostly studied in the context of reinforcement learning (Section 2.3).

2.1 Transfer Learning

Methods based on transfer learning use another machine learning model to predict sample difficulty before the actual learning process begins. There is an additional computation needed for acquiring an order of samples beforehand. As described in recent Weinshall's work [16] the transfer can be done from a more powerful learner (e.g. big pre-trained model) or the same student model can be pre-train with uniformly selected samples as it is normally done (i.e. *self-taught* method). Once an order of samples is acquired, a student model is trained on gradually more difficult samples.

Let (2.1) denotes a training set, where $\mathbf{x}_i \in \mathbb{X}^d$ is a single sample, $y_i \in [K]$ is its corresponding label and a mini-batch $\mathbb{B} \subseteq \mathbb{X}$ denote a subset of $\mathbb{X}$. The standard training is done by presenting a learner with a sequence of uniformly sampled mini-batches $[\mathbb{B}_1, \ldots, \mathbb{B}_M]$ which is a variant of SGD. [16]

$$\mathbb{X} = \{X_i\}_{i=1}^{N} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \tag{2.1}$$

2.1.1 Scoring Function

Difficulty of samples $(\mathbf{x}_i, y_i)$ is measured by its loss with respect to a set of optimal hypotheses. *Scoring function* is defined as $f : \mathbb{X} \rightarrow \mathbb{R}$. A sample $(\mathbf{x}_i, y_i)$ is said to be more difficult than a sample $(\mathbf{x}_j, y_j)$ if $f(\mathbf{x}_i, y_i) > f(\mathbf{x}_j, y_j)$. In general, a *scoring function* encodes a prior knowledge and defines a curriculum. In Hacohen et al. [16] the confidence scores of a pre-trained classifier model are used as a *scoring function*.

Two additional control curriculums are defined in [16] for comparison and examination of learning progresses.

Anti-Curriculum uses a *scoring function* $f' = -f$, training samples are sorted in descending order (i.e. harder examples are chosen before easier ones). This corresponds to the principle of *hard example mining*.

Random-Curriculum uses a *scoring function* where training samples are randomly scored.

2.1.2 Source Models

The following models to transfer sample difficulty from were proposed.

Simple Classifier Motivated by a lower time consumption, a less powerful model can be used as a curriculum source. Simple classifiers based on the multinomial logistic regression, support vector machines (SVM) or a multilayer perceptron (MLP) were studied. [18][20]

Pretrained Large Model In [16] a pre-trained Inception network on ImageNet dataset was used to obtain scores for every sample in a training set that was used to train a simpler custom CNN classifier.

Self-Taught For the *self-taught* scoring function a source classifier is the learner itself. It is trained on uniformly sampled mini-batches first (i.e. the *vanilla* method) and its confidence score is used as a *scoring function* for each sample. After obtaining the scores the learning process is restarted. [16]

2.1.3 Pacing Function

As defined in [16] a *pacing function* $g_\vartheta : [M] \rightarrow [N]$ is used to acquire a sequence of subsets $\mathbb{X}_1', \ldots, \mathbb{X}_M' \subseteq \mathbb{X}$ of size $|\mathbb{X}_i'| = g_\vartheta(i)$, from which mini-batches $\{\mathbb{B}_i\}_{i=0}^{M}$ are sampled uniformly. Each $\mathbb{X}_i$ contains the first $g_\vartheta(i)$ training data samples sorted by the *scoring function* in ascending order (i.e. from easiest to hardest). Importantly, the implementation of CL in [16] balances each $\mathbb{X}_i$ by picking the same number of samples from each class in a training dataset $\mathbb{X}$ to avoid bias.

The *pacing function* can be any function $g_\vartheta : [M] \to [N]$, but only monotonically increasing staircase functions were studied so far. The functions were for simplicity limited to staircase functions; a step defines a set of learning iterations during which g_ϑ is constant. Each pacing function is defined by the following hyper-parameters.

- *step_length* – a 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
- *starting_percent* – a fraction of the data in the initial step

The three pacing functions that were used in [16] are illustrated in Figure 3.1 and described below.

Fixed Exponential Pacing has a fixed *step_length* and exponentially increasing data size in each step.

$$g_\vartheta(i) = \min(starting\_percent \cdot increase^{\lfloor \frac{i}{step\_length} \rfloor}) \cdot N \tag{2.2}$$

Varied Exponential Pacing is the same as *fixed exponential pacing* while allowing *step_length* to vary.

Single Step Pacing simplifies a staircase function to a single step function, resulting in fewer hyper-parameters.

$$g_\vartheta(i) = starting\_percent \cdot increase^{\mathbf{1}[i<step\_length]} \cdot N \tag{2.3}$$

Alternatively, the choice of a subset can be done by introducing a custom distribution that is used to sample each $\mathbb{B}_i$ as it was done in Chang et al. [20].

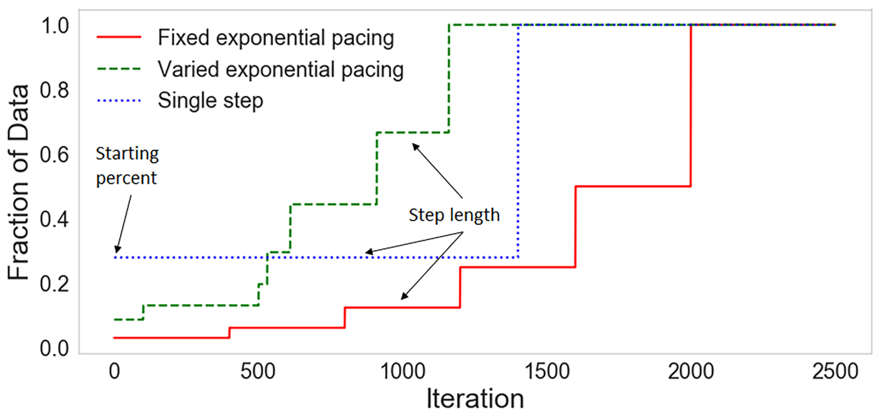


Figure 2.1: Illustration of three pacing functions from [16]

2.1.4 Curriculum Learning Method

The pseudocode for the learning method that was used in [16] is given in Alg. 1.

Algorithm 1: Curriculum learning method

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

2.2 Automated Curriculum

The curriculum method described in Section 2.1 uses a prior knowledge to sort training samples. Many researchers tried to remove the need of spending additional time on pre-training of another model. The following text describes work of [17][20] which are both based on association of weight to each training sample on the fly, therefore, partially or fully eliminating time required for the pre-training.

Weight v_i for the i-th training sample is determined during a learning process of a student model by a curriculum function that acts similarly to the *scoring function* described in section 2.1.1. Overview of different curriculum functions is given in section 2.2.3.

2.2.1 Weighted Loss Function

As defined in Eq. (2.4) a model's loss function is modified to account for different sample weights v_i, where W are the parameters in the model, $loss_i(W)$ is the prediction loss and $\lambda R(W)$ is the regularization term of the model. [20]

$$L = \sum_i v_i \cdot loss_i(W) + \lambda R(W) \tag{2.4}$$

The training samples are therefore given different emphasis for the model's parameters update during SGD.

2.2.2 Mini-Batch Sampling

Alternatively, to the approach described in Section 2.1.3, mini-batches can be sampled on the fly using sample weights. Two baseline sampling methods were defined in Chang et al. [20].

Uniform (SGD-Uni) Given a training dataset $\mathcal{D} = (\mathbf{x}_i, y_i)_i$, for uniform sampling with replacement, the probability of selection each sample is equal $P_s(i|\mathcal{D}) = \frac{1}{|\mathcal{D}|}$.

Scan (SGD-Scan) This method scans through training samples in each epoch and picks samples uniformly without replacement. The sampling probability $P_s(i|S_e, \mathcal{D})$ is $(\frac{1}{|\mathcal{D}|-|S_e|})\mathbf{1}_{i \notin S_e}$, where $\mathbf{1}$ is an indicator function and S_e is the set of samples that were already used in the current epoch.

Different sampling methods examined in [20] that use a *prediction history* are described below. The *prediction history* of classifying the i-th training sample to its correct class $p(y_i|\mathbf{x}_i)$ at each iteration before the current the t-th iteration is stored as H_i^{t-1}. $H = \bigcup_i H_i^{t-1}$, $\bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i)$ is the average probability of classifying the i-th sample into its correct class y_i over all stored $p(y_i|\mathbf{x}_i)$ in H_i^{t-1}, and ϵ_D is a smoothness constant. It is possible to safe memory by storing only the recent history if the training dataset is large. [20]

Sampled by Easiness (SGD-SE) selects easier samples with higher probability as shown in Eq. (2.5), where a hyper-parameter ϵ_D is a smoothness constant that prevents hard samples from never being selected again.

$$P_s(i|H, S_e, \mathcal{D}) \propto \bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i) + \epsilon_D \tag{2.5}$$

Sampled by Difficulty (SGD-SD) oppositely selects harder samples with higher probability.

$$P_s(i|H, S_e, \mathcal{D}) \propto 1 - \bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i) + \epsilon_D \tag{2.6}$$

Sampled by Prediction Variance (SGD-SPV) balances exploration and exploitation by drawing training samples based on their estimated prediction variance plus its confidence interval.

$$P_s(i|H, S_e, \mathcal{D}) \propto \widehat{std}_i^{\text{conf}}(H) + \epsilon_D \tag{2.7}$$

$$\text{where} \quad \widehat{std}_i^{\text{conf}}(H) = \sqrt{\widehat{var}(p_{H_i^{t-1}}(y_i|\mathbf{x}_i)) + \frac{\widehat{var}(p_{H_i^{t-1}}(y_i|\mathbf{x}_i))^2}{|H_i^{t-1}| - 1}}$$

Sampled by Threshold Closeness (SGD-STC) is a simpler and more direct approach to the selection of samples where the probability is close to the decision threshold.

$$P_s(i|H, S_e, \mathcal{D}) \propto \bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i)\left(1 - \bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i)\right) + \epsilon_D \tag{2.8}$$

The training method from Chang et al. [20] is described in Alg. 2. Use of initial burn-in epochs e_b is recommended for learning basic patterns from the data. During the burning epochs training samples are sampled and weighted uniformly from the whole training dataset. This is a hyper-parameter of all automated CL methods and it is usually set to 10–20 % of total training epochs. [17]

2.2.3 Curriculum Functions

Curriculum functions are used to obtain weights for each sample in a mini-batch on the fly. The functions utilize different strategies and have different inputs based on a chosen implementation. Implementations from [17][20] are described in the following subsections. The functions are incorporated directly into the main training loop and they are computed after the loss for each sample is obtained in each iteration. Sample weights calculated by a curriculum function are used to weight an overall loss for the whole mini-batch that is then used to update parameters in SGD.

Algorithm 2: SGD Training with Sample Emphasis

```
Require: Training data $\mathcal{D}$, Batch size $|B|$, Number of class $|C|$, # epochs
  $E$, # burn-in epochs $e_b$
Ensure: NN
  Initialize all weights $W$ in NN
  $H_i \leftarrow \{\frac{1}{|C|}\}$ for all training sample $i$
  $v_i \leftarrow 1$ for all training sample $i$
  $t \leftarrow 1$
  for epoch $e \leftarrow 1...E$ do
    $S_e \leftarrow \emptyset$
    for each iteration do
      if $e > e_b$ then
        Sample $B$ according to $P_s(i|H, S_e, \mathcal{D})$
      else
        Sample $B$ uniformly from $\mathcal{D}$
      end if
      Weight sample $i$ by $v_i$ for all $i$ in $B$
      Update parameters $W$ in NN
      for $i$ in $B$ do
        $H_i \leftarrow H_i \cup \{p_t(y_i|\mathbf{x}_i)\}$
        $S_e \leftarrow S_e \cup \{i\}$
        Update $P_s(i|H, S_e, \mathcal{D})$ and $v_i$.
      end for
      $t \leftarrow t + 1$
    end for
  end for
```

2.2.3.1 Prediction History

Similarly, to the sampling functions described in section 2.2.2 there were corresponding weighting functions defined in Chang et al. [20]. Methods are illustrated by a toy example in Fig. 2.2.

Weighted by Easiness (SGD-WE) sets the weight v_i for each sample according to Eq. (2.9), where ϵ_D is a hyper-parameter smoothness constant and N_D is a normalization constant that makes the average of v_i equal to 1.

$$v_i = \frac{1}{N_D}(\bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i) + \epsilon_D) \tag{2.9}$$

Weighted by Difficulty (SGD-WD) oppositely emphasizes harder samples by giving them higher weights.

$$v_i = \frac{1}{N_D}(1 - \bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i) + \epsilon_D) \tag{2.10}$$

Weighted by Prediction Variance (SGD-WPV) similarly to Eq. (2.7) gives weights to the samples based on balancing exploration and exploitation.

$$v_i = \frac{1}{N_D}(\widehat{std}_i^{\text{conf}}(H) + \epsilon_D) \tag{2.11}$$

Weighted by Threshold Closeness (SGD-WTC) similarly to (2.8), the weighting can be viewed as multiplying SGD-WD and SGD-WE together.

$$v_i = \frac{1}{N_D}\bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i)(1 - \bar{p}_{H_i^{t-1}}(y_i|\mathbf{x}_i)) + \epsilon_D \tag{2.12}$$

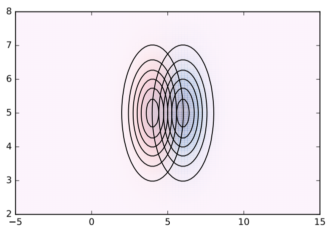
2.2.3.2 Self-Paced Learning Curriculum

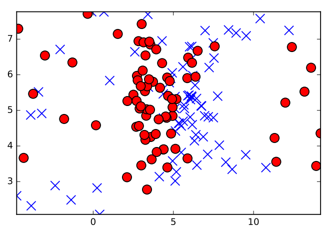
This section describes an approach that was originally described in Kumar et al. [23] and implemented in Jiang et al. [17]. *Self-paced learning* is like SGD-WE that was described above, except it does not provide weights in the continuous space. A sample is either considered for training in the current iteration or it is not.

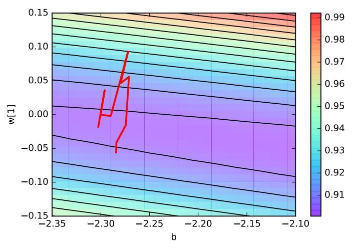
Formally, given a training set $\mathcal{D} = (\mathbf{x}_i, y_i)_i$ of size $n \in \mathbb{N}$, an objective function $g_s(\mathbf{x}_i, \mathbf{w})$ parametrized by $\mathbf{w} \in \mathbb{R}^d$ and $\mathbf{L}(y_i, g_s(\mathbf{x}_i, \mathbf{w}))$ a m-dimensional column vector that denotes a loss over m classes. *Self-paced learning* introduces a latent weight variable $\mathbf{v} \in \mathbb{R}^{n\times m}$, and optimizes the objective described in Eq. (2.13). Where $\|\cdot\|_2$ is the l_2 norm for weight decay and the other model optimizations like augmentation, dropouts etc. are included in g_s. Vector $\mathbf{v}_i \in [0,1]^{m\times 1}$ represents the latent weight variable for the i-th sample in the training set; in scalar form represented as v_i. Function G represents a curriculum that is parametrized by λ.

$$\min_{\mathbf{w}\in\mathbb{R}^d, \mathbf{v}\in[0,1]^{n\times m}} \mathbb{F}(\mathbf{w},\mathbf{v}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{v}_i^T \mathbf{L}(y_i, g_s(\mathbf{x}_i,\mathbf{w})) + G(\mathbf{v};\lambda) + \theta\|\mathbf{w}\|_2^2 \tag{2.13}$$

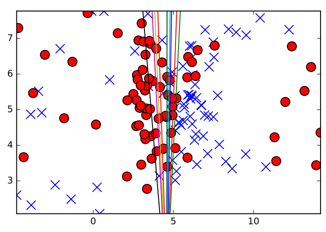
When $\mathbf{w}$ is fixed, $\mathbf{v}^k = \arg\min_{\mathbf{v}} \mathbb{F}(\mathbf{v}^{k-1}, \mathbf{w}^k)$ is computed using the most recently updated $\mathbf{w}^k$ at the epoch k and the optimal $\mathbf{v}$ can be derived by Eq. (2.14). The equation (2.14) explains *self-paced learning* [23]; the i-th sample

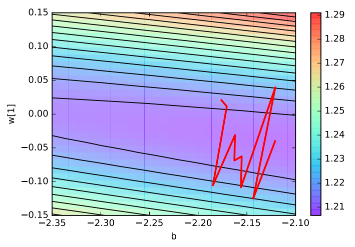

(a) Sampling distribution


(b) Training Samples

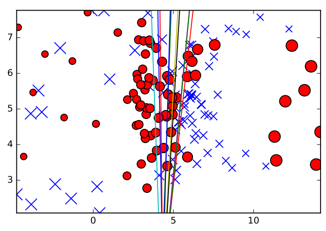


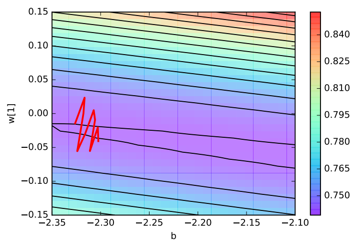
(c) SGD-Scan parameters space


(d) SGD-Scan boundaries



(e) SGD-WD parameters space


(f) SGD-WD sample weights and boundaries



(g) SGD-WPV parameters space

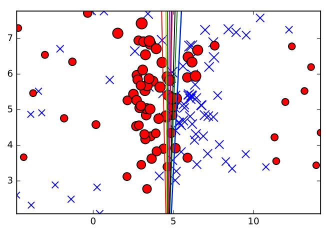
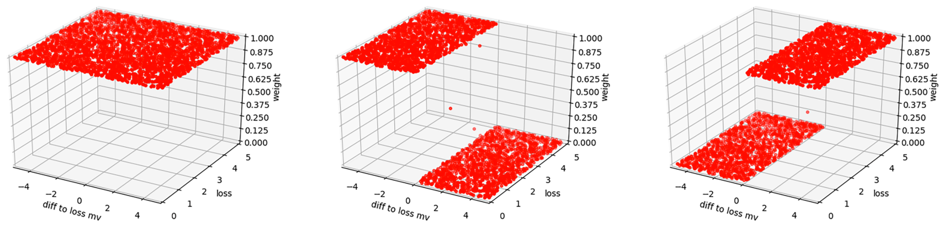

(h) SGD-WPV sample weights and boundaries

Figure 2.2: A toy example from [20] that compares different methods in a two-class logistic regression model. An optimization path is visualized as the red line in (c), (e) and (g). Example shows that SGD-WPV can train more accurate model in noisy conditions.



(a) No curriculum (b) Self-paced learning (c) Hard example mining

Figure 2.3: Curriculum functions from [17]

with a loss smaller than λ is considered as "easy" and it is selected for the training i.e. $v_i^* = 1$, otherwise it is considered as "hard" and it is not selected i.e. $v_i^* = 0$. The hyperparameter λ controls a pace of the learning.

$$v_i^* = \mathbf{1}(\ell_i \leq \lambda), \forall i \in [1, n], \tag{2.14}$$

2.2.3.3 Hard Example Mining Curriculum

In the examined work of Jiang et al. [17], the *hard example mining* was exactly opposite to the *self-paced learning*; therefore, the harder samples are selected for training.

In [17] both *self-paced learning* and *hard example mining* are implemented with a use of a model's overall loss moving average. In a *self-paced learning* implementation, a sample is considered for the training if its actual loss is smaller than the moving average; for *hard example mining* the opposite is true as depicted in 2.3.

2.2.3.4 Data Driven Curriculum

The curriculums in the previous sections are defined by a mathematical function. The functions had to be assembled beforehand by a human teacher. Therefore, they are called *predefined curriculums*. The novel method described in [17] removes this fixed dependency and defines a curriculum that is learnt from a training data, therefore, it is called *data-driven curriculum*. In [17] the *data-driven curriculum* is represented in a form of a machine learning model called *MentorNet* and the trained model is called *StudentNet*.

The goal of this approach is to teach a *MentorNet* model weights of each sample. Features passed to a *MentorNet* model are defined by the actual state of a student model at each training iteration. The features for each mini-batch are current sample losses, sample labels and training epoch progress

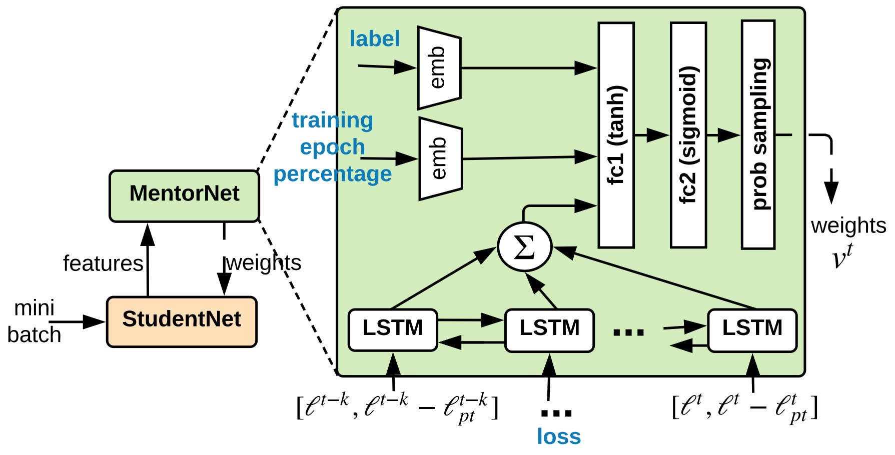


Figure 2.4: *MentorNet* architecture [17]

percentage, output of a *MentorNet* model are weights for each mini-batch sample (Fig. 2.4).

MentorNet is updated together with a learning model *StudentNet* using the *SPADE* algorithm (Alg. 3). The training of *MentorNet* is done by obtaining Θ^* in Eq. (2.15) where $\mathbf{z}_i = \phi(\mathbf{x}_i, y_i, \mathbf{w})$ are input features for the i-th training sample. A *data-driven* curriculum can be obtained by using SGD on a *MentorNet* model with another dataset and therefore acquiring a prior knowledge; then it can be updated multiple times during a *StudentNet* training. [17]

$$g_m(\mathbf{z}_i; \Theta^*) = \arg \min_{v_i \in [0,1]} \mathbb{F}(\mathbf{w}, \mathbf{v}), \forall i \in [1, n] \tag{2.15}$$

Algorithm 3: SPADE for minimizing objective with a curriculum ((2.13))

```
Input   : Dataset $\mathcal{D}$, a predefined $G$ or a learnt $g_m(\cdot;\Theta)$
Output: The model parameter $\mathbf{w}$ of StudentNet.
Initialize $\mathbf{w}^0, \mathbf{v}^0$, $t = 0$;
while Not Converged do
    Fetch a mini-batch $\Xi_t$ uniformly at random;
    For every $(\mathbf{x}_i, y_i)$ in $\Xi_t$ compute $\phi(\mathbf{x}_i, y_i, \mathbf{w}^t)$;
    if update curriculum then
        $\Theta \leftarrow \Theta^*$, where $\Theta^*$ is learnt by training a MentorNet
    if $G$ is used then
        $\mathbf{v}_\Xi^t \leftarrow \mathbf{v}_\Xi^{t-1} - \alpha_t \nabla_\mathbf{v} \mathbb{F}(\mathbf{w}^{t-1}, \mathbf{v}^{t-1})|_{\Xi_t}$
    else $\mathbf{v}_\Xi^t \leftarrow g_m(\phi(\Xi_t, \mathbf{w}^{t-1}); \Theta)$ ;
    $\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} - \alpha_t \nabla_\mathbf{w} \mathbb{F}(\mathbf{w}^{t-1}, \mathbf{v}^t)|_{\Xi_t}$;
    $t \leftarrow t + 1$
return $\mathbf{w}^t$
```

2.3 Subtask Selection

There was another approach to *curriculum learning* proposed and implemented in recent work of Graves et al. [21]. Instead of weighting training samples, it weights tasks that should be learnt. Each dataset label is considered as a separate task, e.g. learning to recognize a digit 9 should be considered as a single task. However, Graves used artificially generated datasets to evaluate the experiments in his work. The tasks in the datasets were design to have hierarchically increasing difficulty. The usage of a curriculum in this setup makes perfect sense. A similar approach was used in Pentina et al. [28].

In [21][29] an adversarial multi-armed bandit is used to select tasks. A training problem has N tasks to solve as a N-armed bandit. At each iteration the bandit plays one of the tasks and gets a reward. The reward is computed by one of many proposed *progress signals*. The progress signals are computed from an observed difference in a loss for current training samples (i.e. *loss-driven* progress signals) or as a change in the student's network complexity (i.e. *complexity-driven* progress signals).

Like in Graves et al. [21], automated *curriculum learning* was used in context of deep reinforcement learning (RL). [30] A RL agent is guiding itself through a set of hierarchically connected subtasks that it should solve. The hardest tasks presented to the agent are composition of easier atomic tasks. Reward shaping for RL was also studied in Justesen et al. [31].

2.4 Natural Language Processing

Curriculum learning was successfully used for natural language processing (NLP) applications. In most cases CL helps to deal with a huge dataset by selecting a subset of samples to train on. [32][33]

Chapter 3

Experiments

Based on work of [16][17][20] a unified framework for an evaluation of *curriculum learning* was developed. The code was reused from the previous implementations and rewritten into modern Tensorflow 2 [34] library. The implementation details of the framework and its parts are described in the Section 3.1.

Many *curriculum learning* approaches described in Chapter 2 were implemented and experimentally evaluated. The results for the *self-taught* curriculum are provided in Section 3.2.1 and the results for the automated curriculum methods are provided in Section 3.2.2. Furthermore, inspired by Jiang et al. [17], experiment with noisy and corrupted data were examined in 3.2.3.

3.1 Implementation Details

The experiments were implemented with the following tools.

Python 3 is an interpreted, high-level, general-purpose programming language that is widely used for machine learning purposes.

Tensorflow 2 is an open-source library for computing and servers as a back-end for machine-learning applications. [34]

Keras is an open-source neural-network library. It is designed for fast experiments with deep neural networks. [35]

The implemented framework consists of the following interchangeable parts. Each part is represented by a Python function or a class with potential arguments. It is easy to implement new methods using this implementation approach.

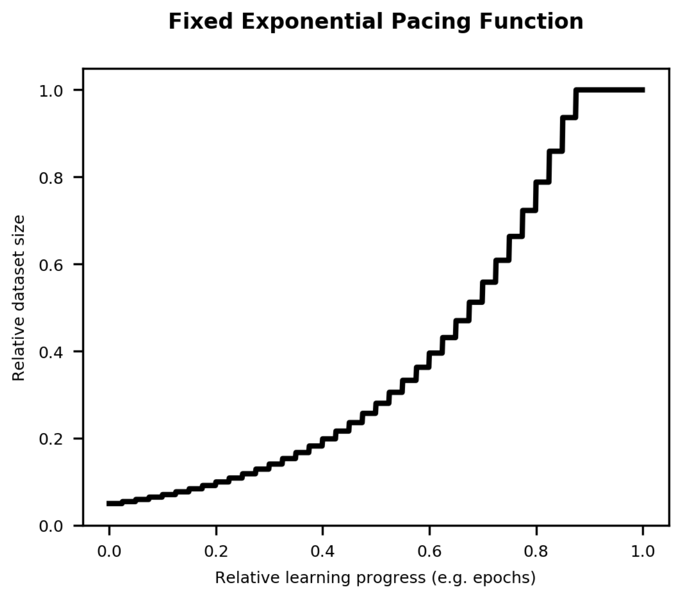


Figure 3.1: The pacing function that was used for the training.

The training algorithm of the implemented framework is described in Section 3.1.1.

Datasets represent loaders for different data that are used for training, they are implemented as a function. List of the examined datasets is provided below and more details about the datasets are provided in Section 3.1.2.

- *MNIST* dataset [36]
- *CIFAR10* dataset [37]
- *SVHN* dataset [38]
- *MNIST_ERASED99* noisy dataset (see Section 3.1.2.2)
- *MNIST_CORRUPTED40* corrupted dataset (see Section 3.1.2.3)

Models represent a student model implemented as a Python function that returns a Keras model. The following models were implemented, details can be found in Section 3.1.3.

- *Normal model* – a CNN model implementation from [16]

- *Simple model* – a simplified CNN model implementation like the *normal model* with fewer convolution layers

Data samplers are responsible for sample selection into the current mini-batch at each training iteration. Data samplers are implemented as a Python class with an optional initialization for acquiring a prior knowledge and a recalculation at the end of each training epoch. The following data samplers were implemented and tested.

- *Automated data sampler* – A sampler that selects samples for training by their current progress indicators (e.g. the predicted history) on the fly as described in Section 2.2.2 and implemented in Chang et al. [20]. The calculation is done by a *sampling function* passed to the sampler class as an argument. The number of mini-batches selected for each epoch remains constant.
- *Pre-trained curriculum model sampler* – Implementation of a sampler that pre-trains a given model passed as an argument (see Section 2.1). A *self-taught* method was implemented; a sampler model and a student are the same model (Section 2.1.2). The order of samples selected for training is given by a *scoring function* passed as an argument. The number of mini-batches selected for each epoch is given by a *pacing function* passed as an argument.

Sampling functions are used for selecting samples into mini-batches on the fly in the *automated data sampler.* The following functions were implemented.

- *Sampled by Easiness (SGD-SE)* emphasizes easy samples using the prediction history during selection given Eq. (2.5)
- *Sampled by Difficulty (SGD-SD)* emphasizes difficult samples using the prediction history during selection given Eq.(2.6)
- *Sampled by Prediction Variance (SGD-SPV)* balances exploration and exploitation given Eq. (2.7)

Scoring functions are used in a *pre-trained curriculum model sampler* for ordering of training samples (see Section 2.1.1). The following functions were implemented.

- *Curriculum* selects easy samples first based on model's confidence intervals
- *Anti-Curriculum* is opposite to the *curriculum* scoring
- *Random-Curriculum* selects random samples

Pacing functions are used for determining a size of training set in a *pre-trained curriculum model sampler* (see Section 2.1.3). The *fixed exponential pacing* function (Eq. (2.2)) was implemented and used in the experiments. The progress of the function is shown in Fig 3.1.

Mentors are used for obtaining mini-batch samples' weights in the automated curriculum methods (see Section 2.2). A mentor is implemented as a Python class with an optional initialization and a method for obtaining the weights based on mini-batch descriptors (i.e. learning progress, sample losses, sample labels) as was done in Jiang et al. [17]. Only a *predefined curriculum mentor* was implemented as described in Section 2.2.3.4.

Curriculum functions are used as a parameter function in a *predefined curriculum mentor* for computing mini-batch sample weights on the fly for an automated curriculum as it was described in Section 2.2. The following functions were implemented and evaluated.

- *Baseline curriculum* weighs all samples uniformly
- *Self-paced learning* uses the implementation of SPL that was described in Section 2.2.3.2
- *Hard example mining* was implemented as opposite to the SPL method similarly to [17]
- *Weighted by Easiness (SGD-WE)* uses the sample prediction history and it is computed by Eq. (2.9)
- *Weighted by Difficulty (SGD-WD)* uses the sample prediction history and it is computed by Eq. (2.10)
- *Weighted by Prediction Variance (SGD-WPV)* uses the sample prediction history and it is computed by Eq. (2.12)

3.1.1 Training Algorithm

The whole algorithm that was implemented is a combination of Alg. 1, Alg. 2 and Alg. 3; the implementation is described in Alg. 4. The SGD training step was unwound by a Tensorflow's `GradientTape`; the unwinding made possible for a more precise acquisition of mini-batch descriptors that are passed to a Mentor for sample weights evaluation.

Algorithm 4: The training algorithm of the implemented framework

```
Input: Sampler, StudentModel, Mentor, LossFunction, iterations.
Output: Trained StudentModel.
$iteration \leftarrow 0$;
while $iteration < iterations$ do
    $batch \leftarrow Sampler.getBatch()$;
    $labels \leftarrow batch.labels$;
    $predictions \leftarrow StudentModel.evaluate(batch)$;
    $losses \leftarrow LossFunction(predictions, labels)$;
    $weights \leftarrow Mentor.getWeights(iteration, losses, predictions, ...)$;
    $weightedLoss \leftarrow mean(weights \cdot losses)$;
    $gradients \leftarrow$ get gradients with respect to $weightedLoss$;
    gradient descent step on StudentModel parameters with $gradients$;
    $iteration \leftarrow iteration + 1$;
end while
return $StudentModel$;
```

3.1.2 Datasets

3.1.2.1 MNIST

MNIST is a widely used dataset that contains small images of handwritten single digits between 0 and 9. The images are monochromatic; each image has 28×28 pixels. The training set for this dataset contains $60,000$ samples and the testing set has $10,000$ samples. Random samples from the dataset are illustrated in Fig. 3.2.

3.1.2.2 MNIST_ERASED99

A custom dataset was derived from the MNIST dataset by randomly applying black (e.g. zero value) rectangles to 99 % of the dataset samples. The rectangles have a random length and width equal to 25–75 % of the whole image. The dataset was generated only once, and it was used for all methods in the related experiments.

3.1.2.3 MNIST_CORRUPTED40

Labels of 40 % of the samples from the MNIST dataset were changed to uniformly distributed labels. Therefore, simulating heavily corrupted training data. The dataset was generated only once.

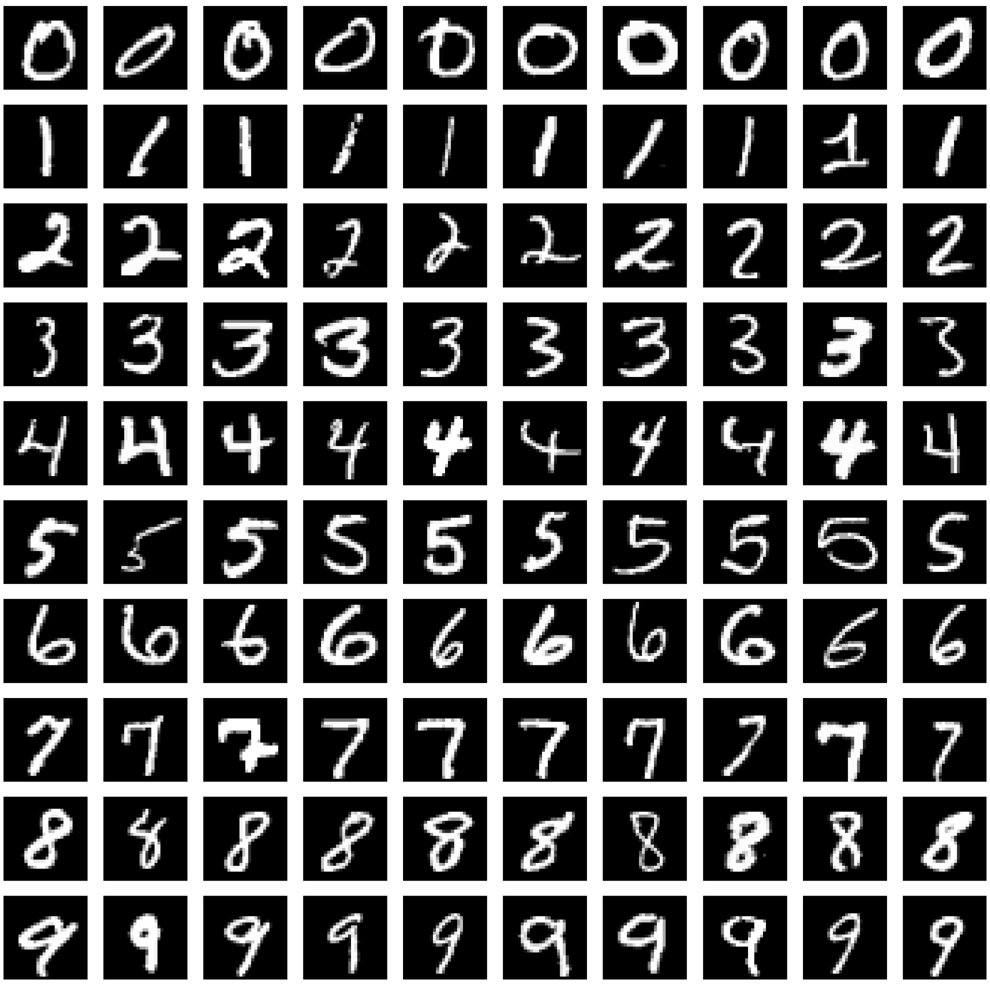


Figure 3.2: MNIST dataset [36]

3.1.2.4 CIFAR10

CIFAR10 is a computer-vision dataset that contains 10 classes of everyday objects. The dataset consists of a 60,000 training sample subset of a larger dataset; it has 6,000 images per class. The training set contains 50,000 samples and the testing set has 10,000 samples. Images has 3 color channels and 32×32 pixels. Sample images are illustrated in 3.3.

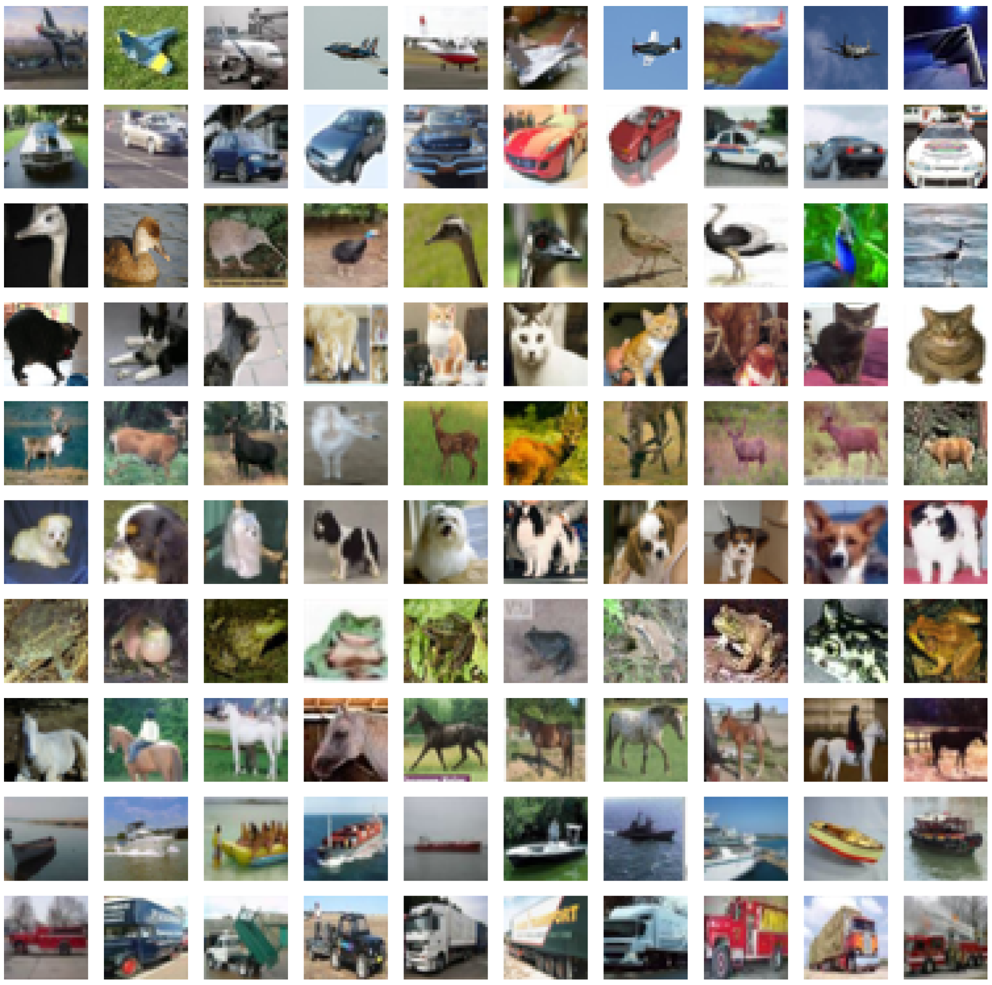


Figure 3.3: CIFAR10 dataset [37]

3.1.2.5 SVHN

The Street View House Numbers (SVHN) is a real-world image dataset for machine learning. The dataset contains small cropped and centered images of single house numbers. [38] There are 73257 training samples and 26, 032 testing samples. Each sample has 32×32 RGB pixels. The SVHN is more challenging than MNIST, because the images are more distorted with different shoot angles, colors and fonts as illustrated in Fig. 3.4.

Figure 3.4: SVHN dataset [38]

3.1.3 Student Models

Two models were evaluated in the experiments. Both models are convolutional neural networks with a widely used architecture inspired by VGG [39]. The first model is taken from the work of [16] for direct comparison and it is called a *normal model*. The second model is its simplified version with fewer convolution layers and therefore less parameters; it is called a *simple model*. The models' architectures can be seen in Fig 3.5, categorical cross-entropy from the Tenserflow library was used for the loss function.

Both models utilize dropouts as described in [6] and weight regularization [5]. Data augmentation was implemented in the evaluation framework and can be used with *curriculum learning* in general. It is not a part of the results provided in this work because of possible bias it might presents to the results.

An SGD optimizer implementation from Keras library was used for training. The best learning rate was determined by preliminary experiments on the baseline learning method (without CL) and it was fixed during the whole learning process for all experiments. It should be noted that learning rate tuning in combination with CL is very challenging as described in Hacohen et al. [16].

3.1.4 Experiment Evaluation

Experiments were computed on Nvidia GPUs GTX 1080 Ti with 11GB memory and Titan V100 with 32 GB memory.

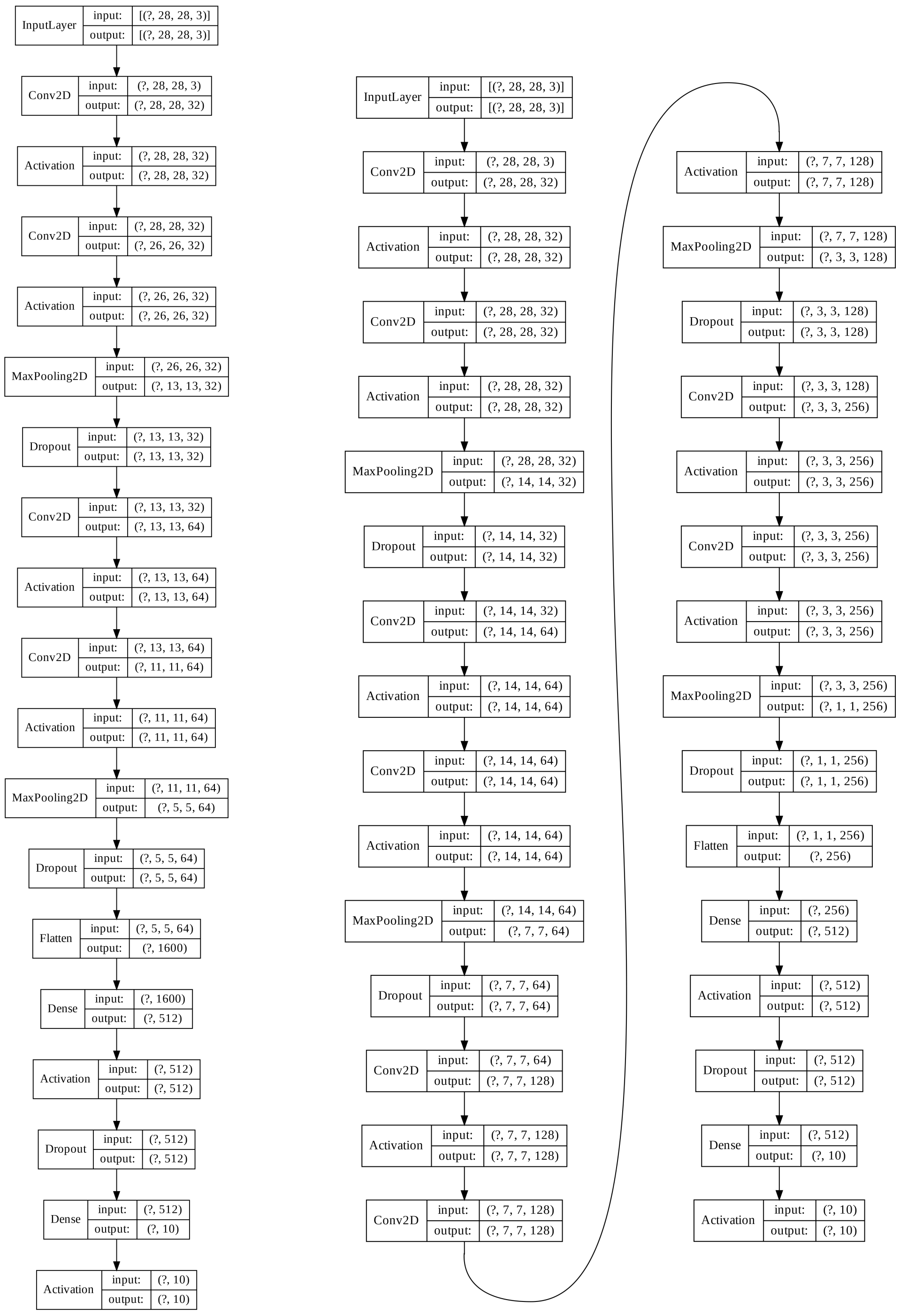
Training time is compared by a number of iterations, i.e. the number of processed mini-batched by the learner in SGD. It is assumed that a single SGD iteration takes a constant time on a given machine. The implemented *curriculum learning* methods do not introduce a huge overhead over training via `keras.fit` function. Although, `keras.fit` might be better optimized in the Tensorflow/Keras code, there were not any deviation in the training times or quality of the results observed.

The number of training epochs is fixed, because the CL implementation uses a learning progress for sample weight computation (300 epochs were used for most of the experiments). *Burn-in* epochs were set to 10 %.

The measurements for every method were run 5 times and the mean values across those measurements are provided in the results. The tables contain mean accuracies measured at the last epoch on train and test sets. The results are provided as a mean ± a standard error of the mean ($mean \pm SE$). Furthermore, the mean of the best observed value for each measurement on a test set is also provided.

The lines in the provided plots are for better readability and clearness smoothed by `scipy.ndimage.gaussian_filter1d` with a sigma parameter equal to 2.

The baseline method in the experiments denotes a training without any form of *curriculum learning*. Samples are picked uniformly from the training set and they are all provided with a uniform weight of 1. This method corresponds to the *vanilla* method that was used in Hacohen et al. [16].


InputLayer input: [(?, 28, 28, 3)] output: [(?, 28, 28, 3)]
Conv2D input: (?, 28, 28, 3) output: (?, 28, 28, 32)
Activation input: (?, 28, 28, 32) output: (?, 28, 28, 32)
Conv2D input: (?, 28, 28, 32) output: (?, 26, 26, 32)
Activation input: (?, 26, 26, 32) output: (?, 26, 26, 32)
MaxPooling2D input: (?, 26, 26, 32) output: (?, 13, 13, 32)
Dropout input: (?, 13, 13, 32) output: (?, 13, 13, 32)
Conv2D input: (?, 13, 13, 32) output: (?, 13, 13, 64)
Activation input: (?, 13, 13, 64) output: (?, 13, 13, 64)
Conv2D input: (?, 13, 13, 64) output: (?, 11, 11, 64)
Activation input: (?, 11, 11, 64) output: (?, 11, 11, 64)
MaxPooling2D input: (?, 11, 11, 64) output: (?, 5, 5, 64)
Dropout input: (?, 5, 5, 64) output: (?, 5, 5, 64)
Flatten input: (?, 5, 5, 64) output: (?, 1600)
Dense input: (?, 1600) output: (?, 512)
Activation input: (?, 512) output: (?, 512)
Dropout input: (?, 512) output: (?, 512)
Dense input: (?, 512) output: (?, 10)
Activation input: (?, 10) output: (?, 10)
InputLayer input: [(?, 28, 28, 3)] output: [(?, 28, 28, 3)]
Conv2D input: (?, 28, 28, 3) output: (?, 28, 28, 32)
Activation input: (?, 28, 28, 32) output: (?, 28, 28, 32)
Conv2D input: (?, 28, 28, 32) output: (?, 28, 28, 32)
Activation input: (?, 28, 28, 32) output: (?, 28, 28, 32)
MaxPooling2D input: (?, 28, 28, 32) output: (?, 14, 14, 32)
Dropout input: (?, 14, 14, 32) output: (?, 14, 14, 32)
Conv2D input: (?, 14, 14, 32) output: (?, 14, 14, 64)
Activation input: (?, 14, 14, 64) output: (?, 14, 14, 64)
Conv2D input: (?, 14, 14, 64) output: (?, 14, 14, 64)
Activation input: (?, 14, 14, 64) output: (?, 14, 14, 64)
MaxPooling2D input: (?, 14, 14, 64) output: (?, 7, 7, 64)
Dropout input: (?, 7, 7, 64) output: (?, 7, 7, 64)
Conv2D input: (?, 7, 7, 64) output: (?, 7, 7, 128)
Activation input: (?, 7, 7, 128) output: (?, 7, 7, 128)
Conv2D input: (?, 7, 7, 128) output: (?, 7, 7, 128)
Activation input: (?, 7, 7, 128) output: (?, 7, 7, 128)
MaxPooling2D input: (?, 7, 7, 128) output: (?, 3, 3, 128)
Dropout input: (?, 3, 3, 128) output: (?, 3, 3, 128)
Conv2D input: (?, 3, 3, 128) output: (?, 3, 3, 256)
Activation input: (?, 3, 3, 256) output: (?, 3, 3, 256)
Conv2D input: (?, 3, 3, 256) output: (?, 3, 3, 256)
Activation input: (?, 3, 3, 256) output: (?, 3, 3, 256)
MaxPooling2D input: (?, 3, 3, 256) output: (?, 1, 1, 256)
Dropout input: (?, 1, 1, 256) output: (?, 1, 1, 256)
Flatten input: (?, 1, 1, 256) output: (?, 256)
Dense input: (?, 256) output: (?, 512)
Activation input: (?, 512) output: (?, 512)
Dropout input: (?, 512) output: (?, 512)
Dense input: (?, 512) output: (?, 10)
Activation input: (?, 10) output: (?, 10)

(a) Simple model (b) Normal model

Figure 3.5: A visualization of the CNN models that were used for the experiments.

3.2 Results

3.2.1 Self-Taught CL

The first experiment run the *simple model* on the MNIST dataset. Results for *self-taught* curriculum methods are plotted in Figure 3.6, all the results can be found in Table 3.1. The examined methods have different training paths and results. The steps of the pacing function are visible in the training accuracies. There is a huge dip in the training accuracy of the *self-taught* method (green line) caused by the introduction of the most distorted training samples to the model. The *self-taught* method struggles to reach competitive test accuracies, this might be due to the late introduction of harder samples during the training. On the other hand, the *anti-curriculum* method has low accuracies on the training set, but it can extract information from harder samples and its test accuracy is better than baseline during the whole training. The random curriculum has similar result to the baseline once it sees all the training samples at the end of the training. The illustration of easy and hard samples identified by the pre-training is depicted in Figure 3.7

The second experiment used the *normal model* on the MNIST dataset, results are in Figure 3.8 and Table 3.2. The random method has again similar results to the baseline. The *self-taught* method (green line) again yields worse results than other methods. In contrast to the *simple model*, the *anti-curriculum* method does not have a better accuracy on the test set than the baseline.

The third experiment was conducted on the CIFAR10 dataset using the *normal model*, results are in Figure 3.9 and Table 3.3. We can again see the dip in a *self-taught* method train accuracy at the end of the training. The random and *self-taught* methods have lower accuracies on the test data than the baseline. The *anti-curriculum* method has even lower test accuracy.

The fourth experiment examined the SVHN dataset using the *normal model*, results are in Figure 3.10 and Table 3.4. The random and *self-taught* methods have test accuracies close to the baseline, the *anti-curriculum* has worse accuracy than the other methods. The samples sorted by the pre-training are illustrated in Figure 3.11.

3.2.2 Automated CL

The fifth experiment examined the automated curriculum methods on the MNIST dataset with the *simple model.* The results can be found in Figure 3.12 and Table 3.1. The *self-paced* method struggles to train well. This happens because the sample weighting in this method is not continuous and at some point it converges to train only on easy samples. The hyper-parameters for this implementation of *self-paced learning* from Jiang et al. [17] were obtained

Table 3.1: Results on the MNIST dataset using the *simple model*. The results are presented in percentages as $mean \pm SE$, see Section 3.1.4 for more information.

Method	Train accuracy	Test accuracy	Test best result
Baseline	99.676 ± 0.00787	99.424 ± 0.02182	99.498 ± 0.01985
Self-paced	97.608 ± 0.03046	98.790 ± 0.01483	99.052 ± 0.00735
Hard example	99.672 ± 0.01337	99.422 ± 0.01393	99.486 ± 0.02821
SGD-WE	99.501 ± 0.01574	99.435 ± 0.02592	99.495 ± 0.01765
SGD-WD	99.832 ± 0.00773	99.482 ± 0.01908	99.528 ± 0.00969
SGD-WPV	99.770 ± 0.03018	99.467 ± 0.02728	$\mathbf{99.537} \pm 0.01797$
Self-taught	98.737 ± 0.01150	99.213 ± 0.01931	99.233 ± 0.04333
Anti	$\mathbf{99.929} \pm 0.00276$	$\mathbf{99.482} \pm 0.01250$	99.533 ± 0.02057
Random	99.511 ± 0.02459	99.373 ± 0.01453	99.430 ± 0.01155

Table 3.2: Results on the MNIST dataset using the *normal model*.

Method	Train accuracy	Test accuracy	Test best result
Baseline	99.850 ± 0.00677	99.568 ± 0.01960	99.634 ± 0.01166
Self-paced	98.807 ± 0.01692	99.318 ± 0.02782	99.398 ± 0.01855
Hard example	99.854 ± 0.00596	99.548 ± 0.01393	99.618 ± 0.00800
SGD-WE	99.775 ± 0.00710	$\mathbf{99.596} \pm 0.01208$	$\mathbf{99.640} \pm 0.01225$
SGD-WD	99.926 ± 0.00238	99.594 ± 0.01077	99.634 ± 0.00980
SGD-WPV	99.920 ± 0.00551	99.560 ± 0.01095	99.624 ± 0.00510
Self-taught	99.086 ± 0.00080	99.463 ± 0.01548	99.483 ± 0.00750
Anti	$\mathbf{99.971} \pm 0.00493$	99.565 ± 0.01848	99.613 ± 0.01181
Random	99.726 ± 0.01208	99.577 ± 0.02333	99.607 ± 0.00667

Table 3.3: Results on the CIFAR10 dataset using the *normal model*.

Method	Train accuracy	Test accuracy	Test best result
Baseline	95.940 ± 0.03805	85.604 ± 0.11818	85.874 ± 0.06485
Self-paced	76.659 ± 0.21056	77.046 ± 0.21951	78.276 ± 0.15128
Hard example	91.988 ± 0.09756	84.434 ± 0.13269	84.714 ± 0.15766
SGD-WE	92.711 ± 0.10738	85.290 ± 0.09965	85.484 ± 0.06185
SGD-WD	$\mathbf{97.064} \pm 0.02439$	85.426 ± 0.14137	85.656 ± 0.15911
SGD-WPV	95.885 ± 0.06499	$\mathbf{85.706} \pm 0.09469$	$\mathbf{86.006} \pm 0.10003$
Self-taught	85.327 ± 0.09849	83.143 ± 0.17169	83.323 ± 0.15301
Anti	94.068 ± 0.11847	79.237 ± 0.25360	79.827 ± 0.10039
Random	92.640 ± 0.08442	83.250 ± 0.15961	83.360 ± 0.12373

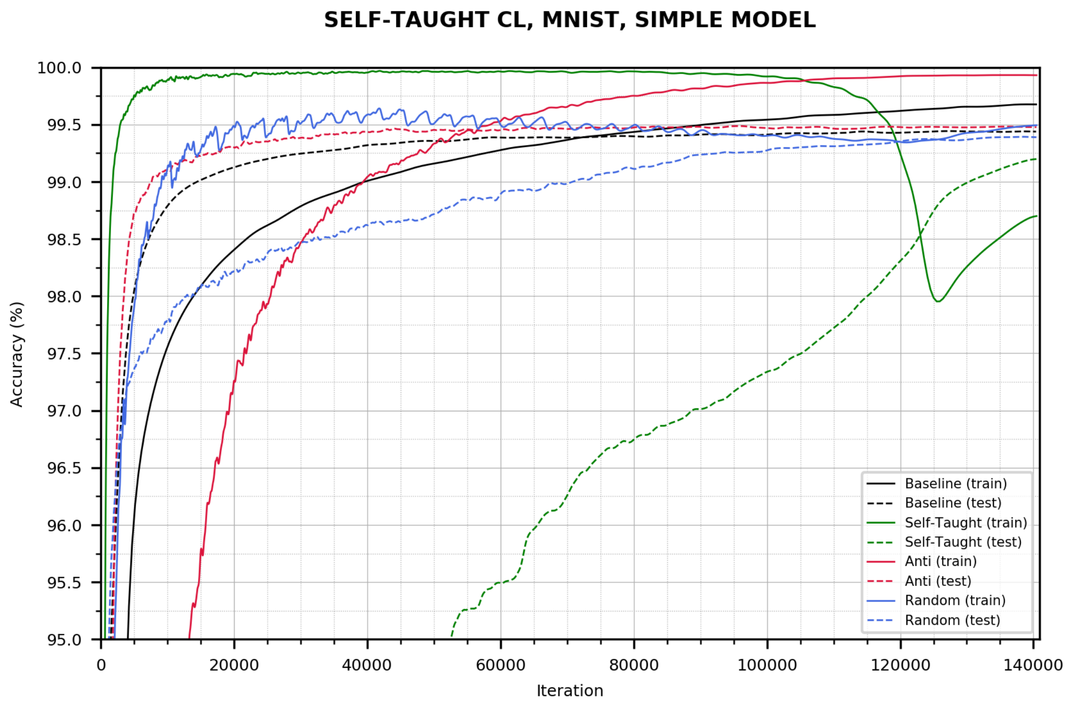
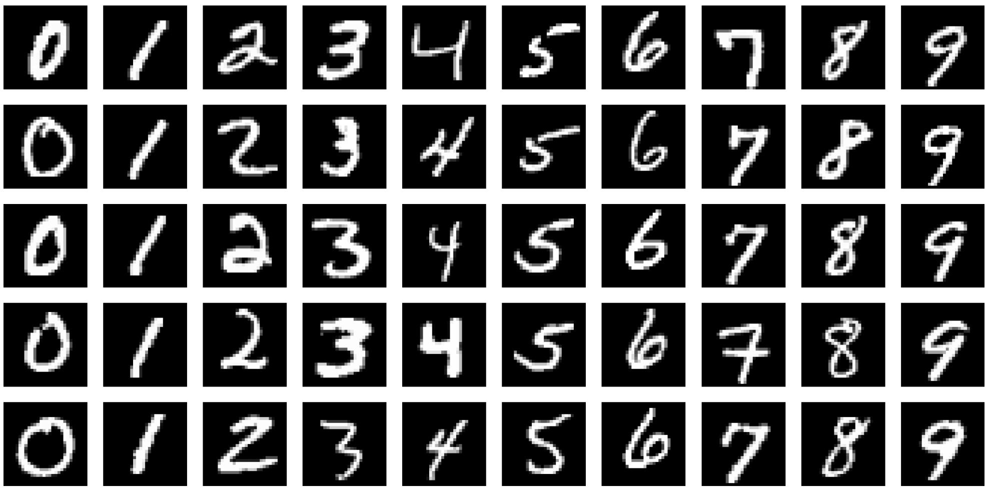


Figure 3.6: Results of the *self-taught* curriculum methods on the MNIST dataset using the *simple model*.

by a grid search and the method with the best results is presented in the plots. The SGD-SPV and SGD-WD methods scored much higher accuracies than the baseline method for this experiment.

The sixth experiment used the MNIST dataset and the *normal model*, results are in Figure 3.13 and Table 3.2. In this experiment the SGD-WE method scores the best results, although, it is comparable to the baseline. Emphasizing harder samples in this experiment does not help to improve the test accuracies, but test accuracies rise slightly faster.

The seventh experiment used the CIFAR10 dataset and the *normal model*, results are in Figure 3.14 and Table 3.3. In this experiment only the SGD-SPV method has slightly better results than the baseline. The other methods have much worse test accuracies. Again, the accuracies of the methods that emphasizes harder samples rise faster.


(a) Samples scored with the highest weight.

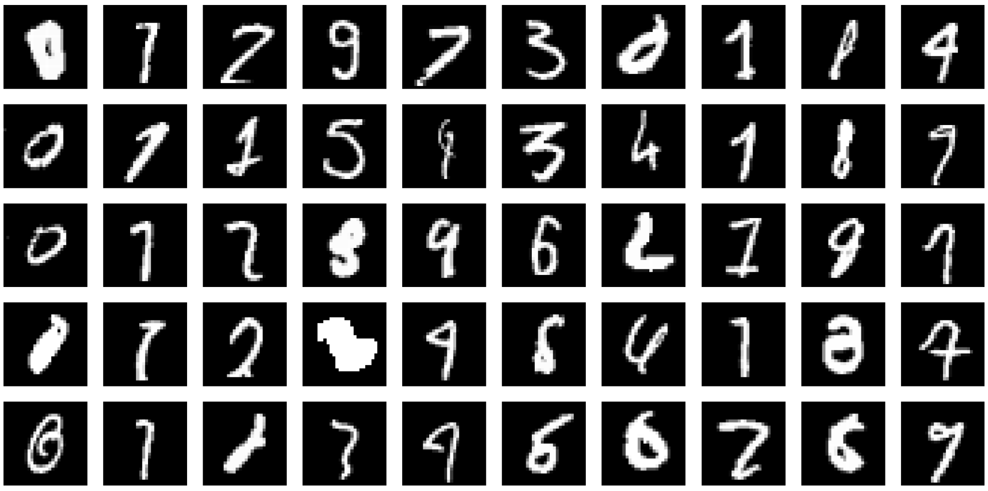

(b) Samples scored with the lowest weight.

Figure 3.7: Selected samples for the MNIST dataset and the *self-taught* method. Each column corresponds to one label. There are corrupted labels for a few digits in the unmodified dataset (e.g. the fourth column corresponds to digit 3).

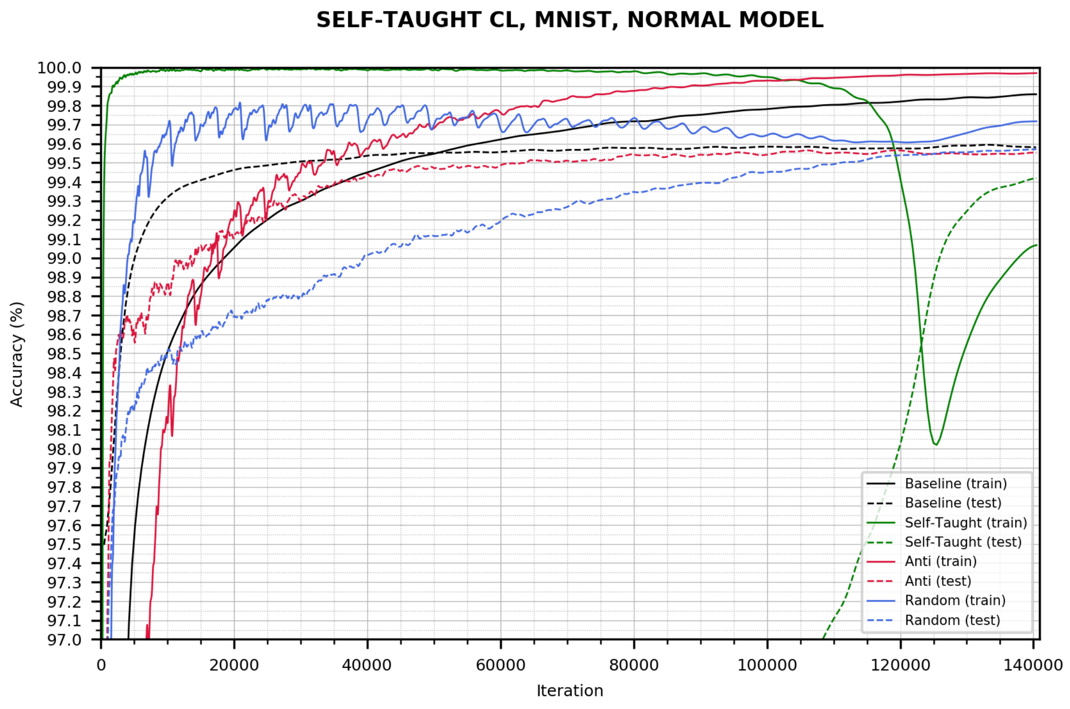


Figure 3.8: The results of the *self-taught* curriculum methods on the MNIST dataset using the *normal model*.

The eight experiment was conducted on the SVHN dataset and the *normal model*, results are in Figure 3.15 and Table 3.4. In this experiment only the SGD-SPV method has slightly better results than the baseline. The other methods have much worse test accuracies. Again, the accuracies of the methods that emphasizes harder samples rise faster.

3.2.3 Noisy and Corrupted Data

Like in the work of [17][20] the experiments on corrupted datasets were conducted. The datasets are described in Section 3.1.2.

The ninth experiment run on the MNIST_CORRUPTED40 dataset and the *simple model* using the *self-taught* curriculum methods. The results can be found in Figure 3.16 and Table 3.5. Training on easier samples (green line) yields the same results as the baseline. The random and *anti-curriculum*

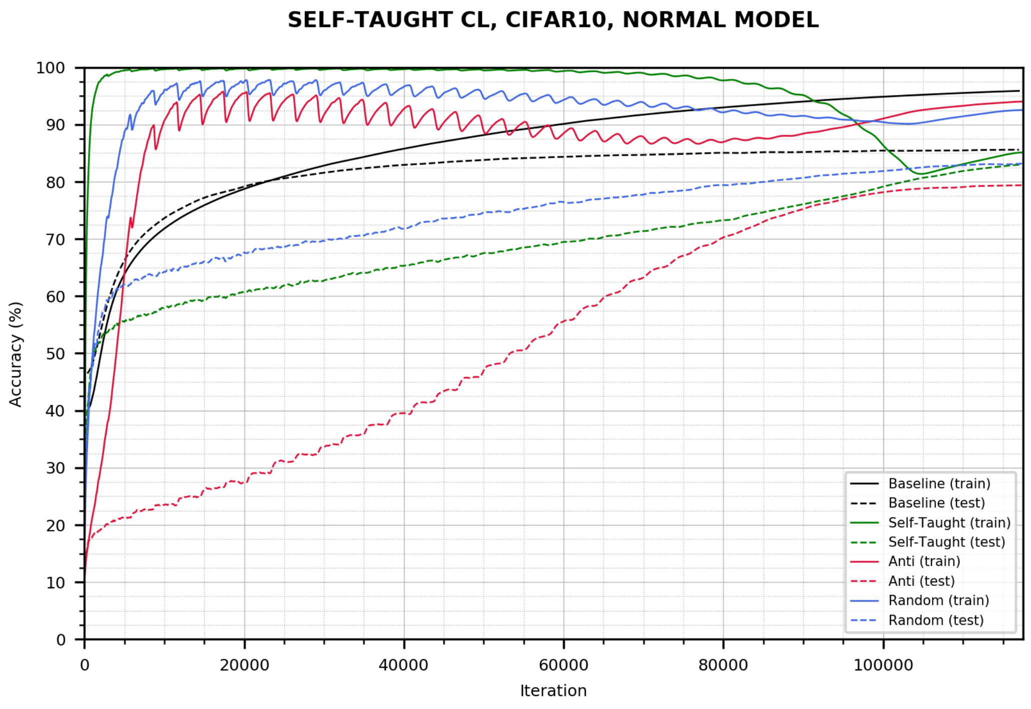


Figure 3.9: The results of the *self-taught* curriculum methods on the CIFAR10 dataset using the *normal model*.

methods have much worse results.

The tenth experiment on the MNIST_CORRUPTED40 dataset and the *simple model* using the *automated* curriculum methods. The results can be found in Figure 3.17 and Table 3.5. The SGD-WE method (weighted by easiness) yielded the best results for the MNIST_CORRUPTED40 dataset, the SGD-SPV has almost similar performance in this experiment.

The eleventh experiment on the MNIST_ERASED99 dataset and the *simple model* using the *self-taught* curriculum methods. The results can be found in Figure 3.18 and Table 3.6. All the *self-taught* methods are strictly worse than the baseline in this experiment.

The twelfth experiment on the MNIST_ERASED99 dataset and the *simple model* using the *automated* curriculum methods. The results can be found in Figure 3.19 and Table 3.6. Emphasizing harder samples gives better result in

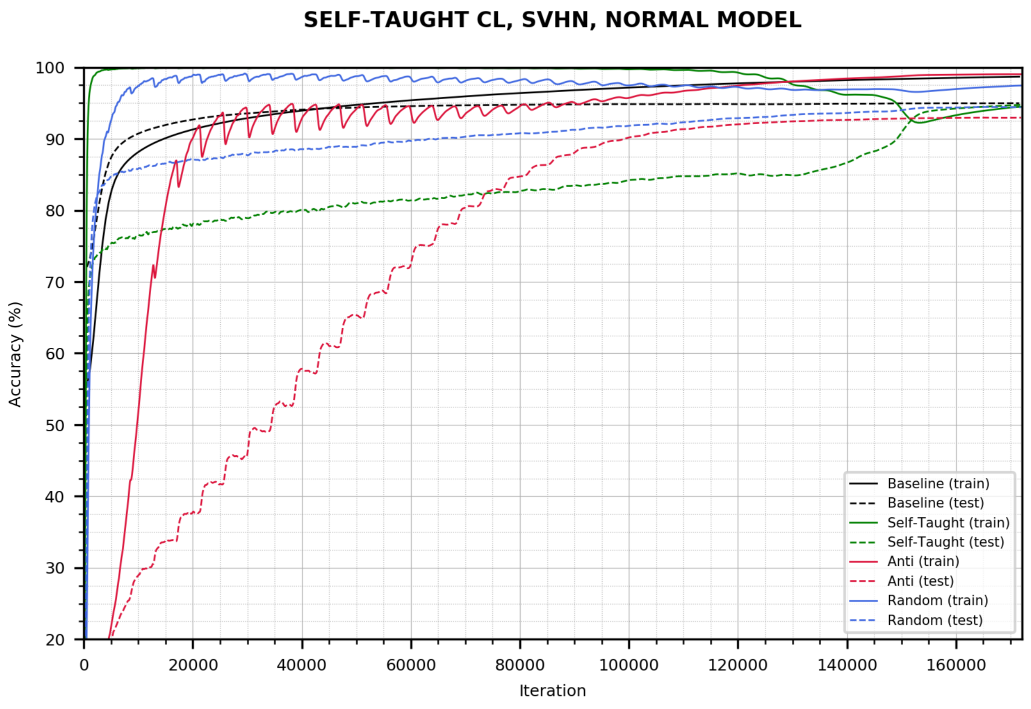


Figure 3.10: The results of the *self-taught* curriculum methods on the SVHN dataset using the *normal model*.

this experiment. The SGD-WPV method has the best test accuracies for the MNIST_ERASED99 dataset. These results are similar to the results on the clean MNIST dataset. This might be due to the fact, that there is still 1 % of clean samples which is enough for the network to successfully train on.

Table 3.4: Results on the SVHN dataset using the *normal model*. The results are presented in percentages as $mean \pm SE$, see Section 3.1.4 for more information.

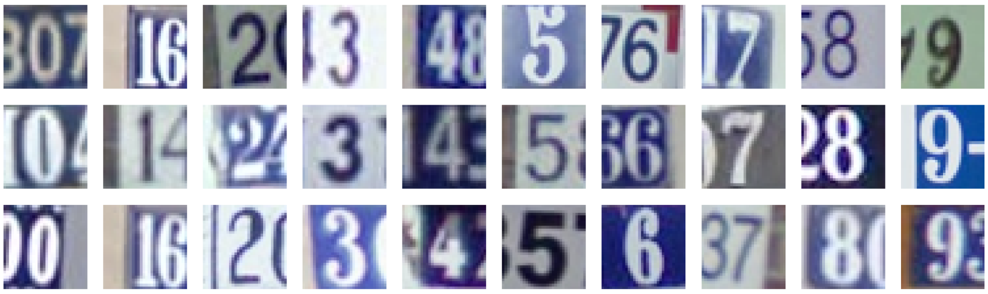
Method	Train accuracy	Test accuracy	Test best result
Baseline	98.711 ± 0.01491	94.935 ± 0.03190	95.108 ± 0.04072
Self-paced	89.129 ± 0.05245	91.124 ± 0.06166	92.480 ± 0.06338
Hard example	98.649 ± 0.02151	95.001 ± 0.04456	95.127 ± 0.03779
SGD-WE	97.301 ± 0.02004	**95.202** ± 0.02648	95.263 ± 0.02215
SGD-WD	**99.331** ± 0.01373	95.068 ± 0.05430	95.158 ± 0.02945
SGD-WPV	98.899 ± 0.02367	95.109 ± 0.07434	**95.267** ± 0.02384
Self-taught	94.596 ± 0.03660	94.779 ± 0.06014	94.851 ± 0.04393
Anti	99.008 ± 0.02862	92.995 ± 0.08985	93.092 ± 0.10519
Random	97.481 ± 0.03234	94.418 ± 0.03633	94.564 ± 0.09567

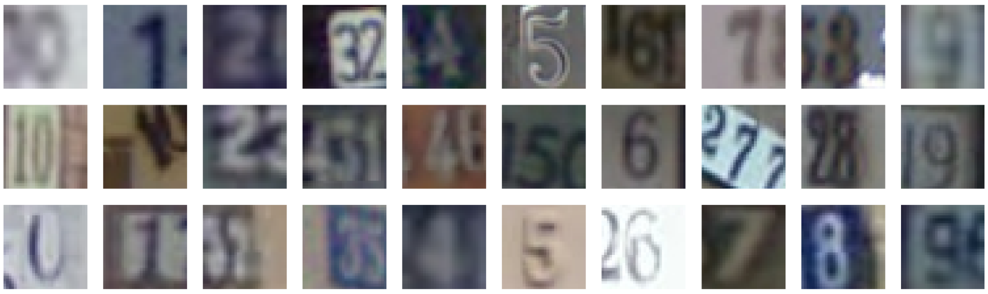
Table 3.5: Results on the MNIST_CORRUPTED40 dataset using the *simple model*.

Method	Train accuracy	Test accuracy	Test best result
Baseline	63.367 ± 0.02060	99.010 ± 0.04159	99.166 ± 0.01860
Self-paced	64.035 ± 0.02903	98.448 ± 0.03397	98.982 ± 0.03121
Hard example	10.167 ± 0.07673	10.290 ± 0.01393	97.950 ± 0.03847
SGD-WE	63.385 ± 0.01457	**99.222** ± 0.01463	**99.244** ± 0.01166
SGD-WD	63.150 ± 0.01762	98.426 ± 0.04202	99.152 ± 0.01393
SGD-WPV	63.493 ± 0.00875	99.144 ± 0.01939	99.228 ± 0.01114
Self-taught	63.114 ± 0.01150	99.000 ± 0.04726	99.036 ± 0.04333
Anti	**67.892** ± 0.13442	92.030 ± 0.28378	92.530 ± 0.12288
Random	67.141 ± 0.10637	94.563 ± 0.12347	95.956 ± 0.12347

Table 3.6: Results on MNIST_ERASED99 dataset using the *simple model*.

Method	Train accuracy	Test accuracy	Test best result
Baseline	92.526 ± 0.06761	98.903 ± 0.02404	98.990 ± 0.01732
Self-paced	80.642 ± 0.06752	96.606 ± 0.01667	97.410 ± 0.01527
Hard example	91.410 ± 0.04796	98.883 ± 0.03283	98.937 ± 0.00333
SGD-WE	90.765 ± 0.02896	98.697 ± 0.07126	98.753 ± 0.06227
SGD-WD	94.097 ± 0.02624	98.893 ± 0.06064	99.017 ± 0.00882
SGD-WPV	92.822 ± 0.03386	**98.943** ± 0.01202	**99.050** ± 0.01528
Self-taught	87.861 ± 0.02246	98.550 ± 0.02887	98.650 ± 0.04041
Anti	**94.133** ± 0.06765	98.067 ± 0.02604	98.193 ± 0.01764
Random	91.580 ± 0.04304	98.527 ± 0.06766	98.617 ± 0.03844


(a) Samples scored with the highest weight


(b) Samples scored with average weights

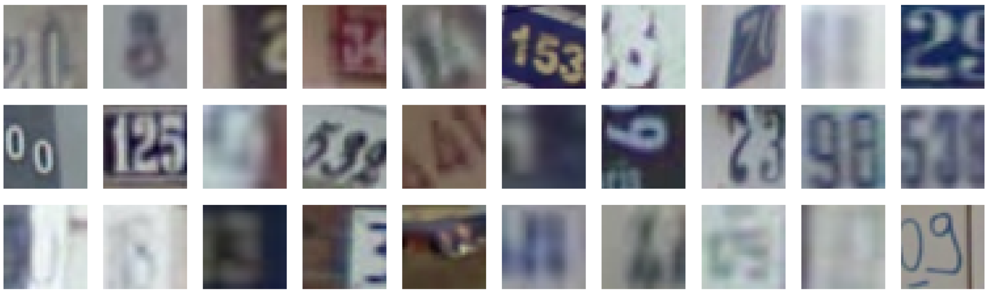

(c) Samples scored with the lowest weight

Figure 3.11: Selected samples for the SVHN dataset and the *self-taught* method. Each column corresponds to one label. There are distorted samples in the dataset.

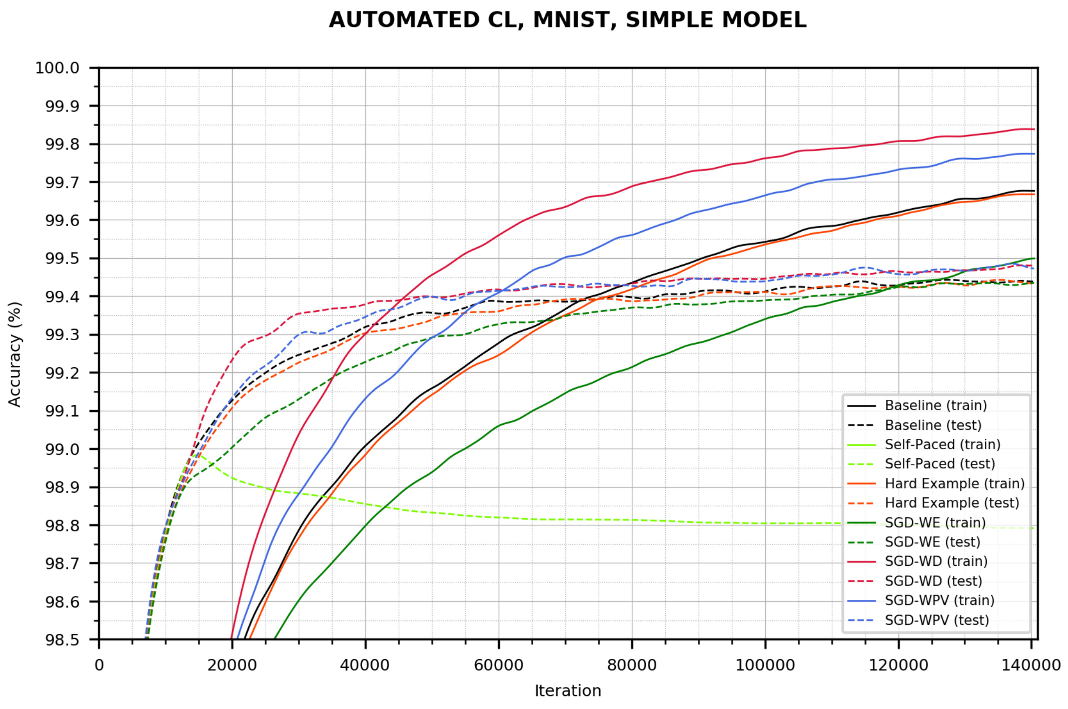


Figure 3.12: The results of the automated curriculum methods on the MNIST dataset using the *simple model*.

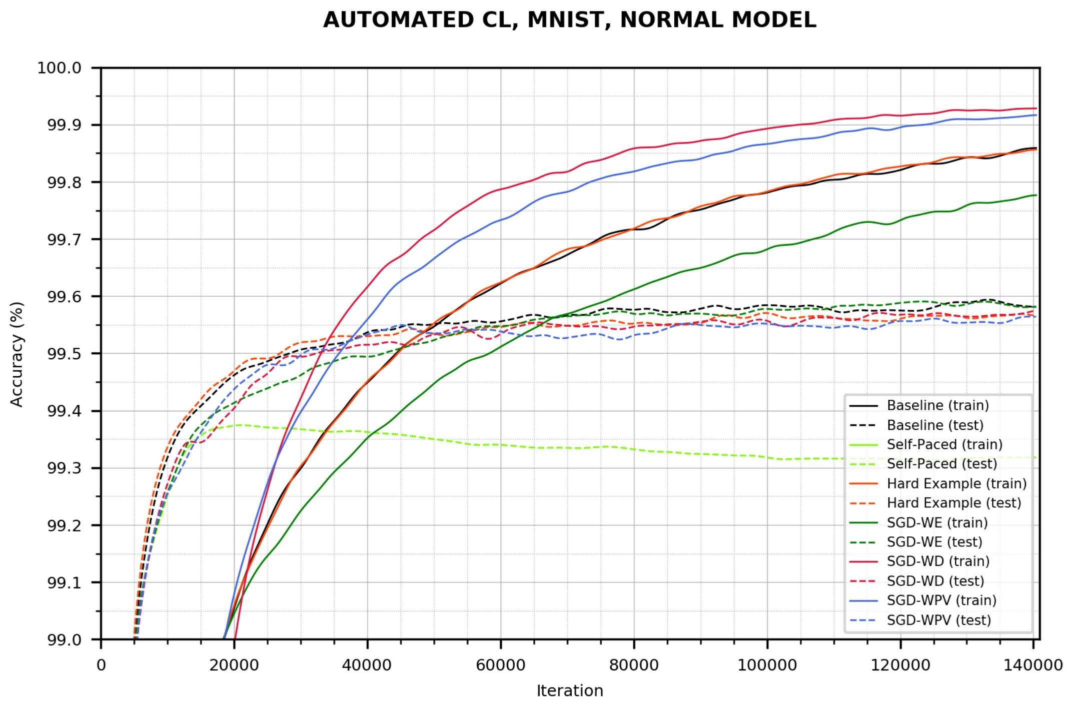


Figure 3.13: The results of the automated curriculum methods on the MNIST dataset using the *normal model*.

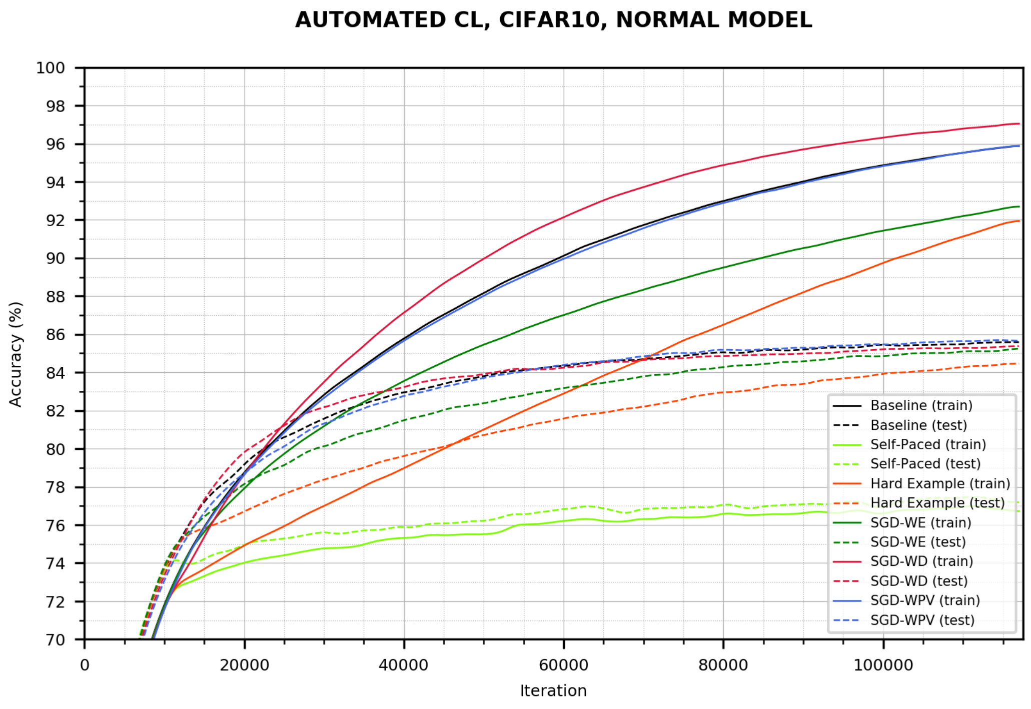


Figure 3.14: The results of the automated curriculum methods on the CIFAR10 dataset using the *normal model*.

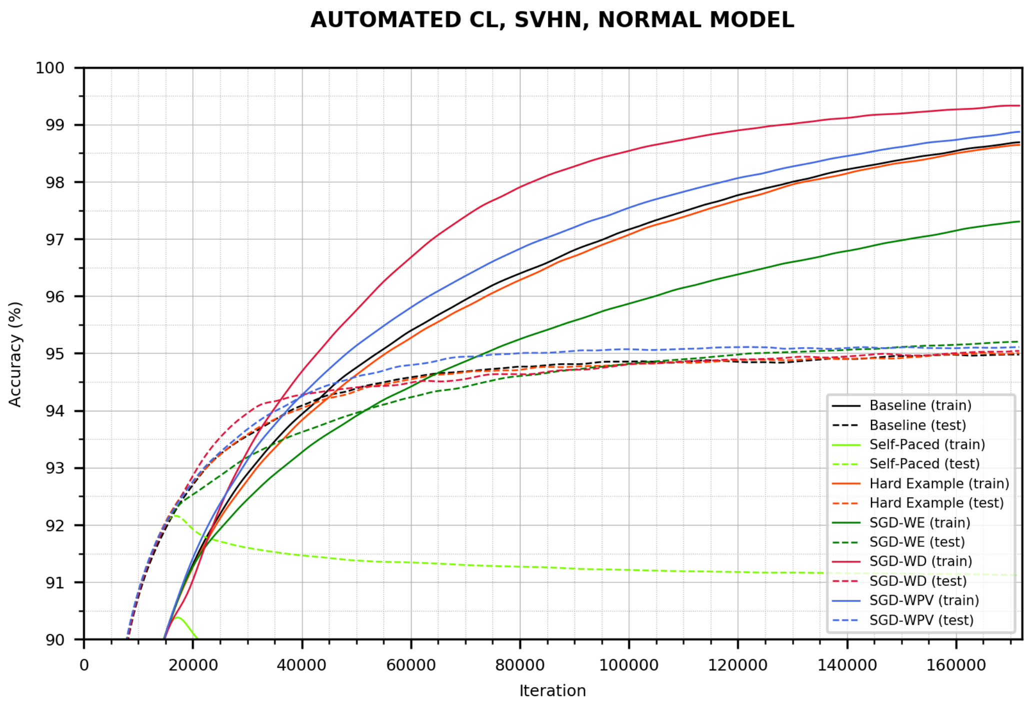


Figure 3.15: The results of the automated curriculum methods on the SVHN dataset using the *normal model*.

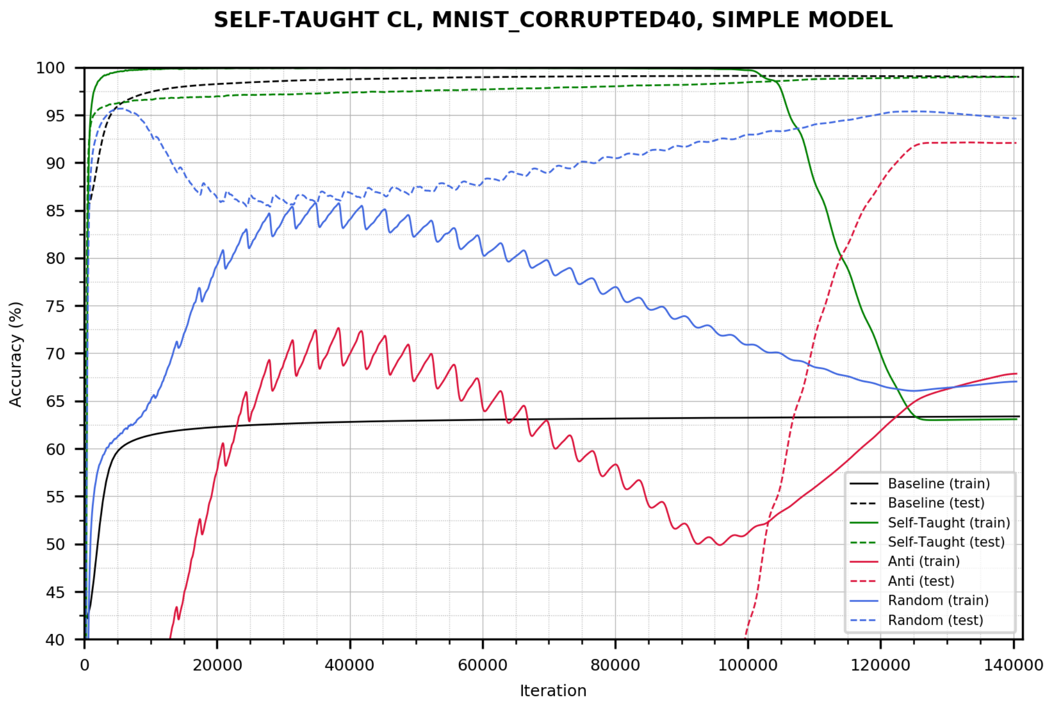


Figure 3.16: The results of *self-taught* curriculum methods on MNIST_CORRUPTED40 dataset using the *simple model*.

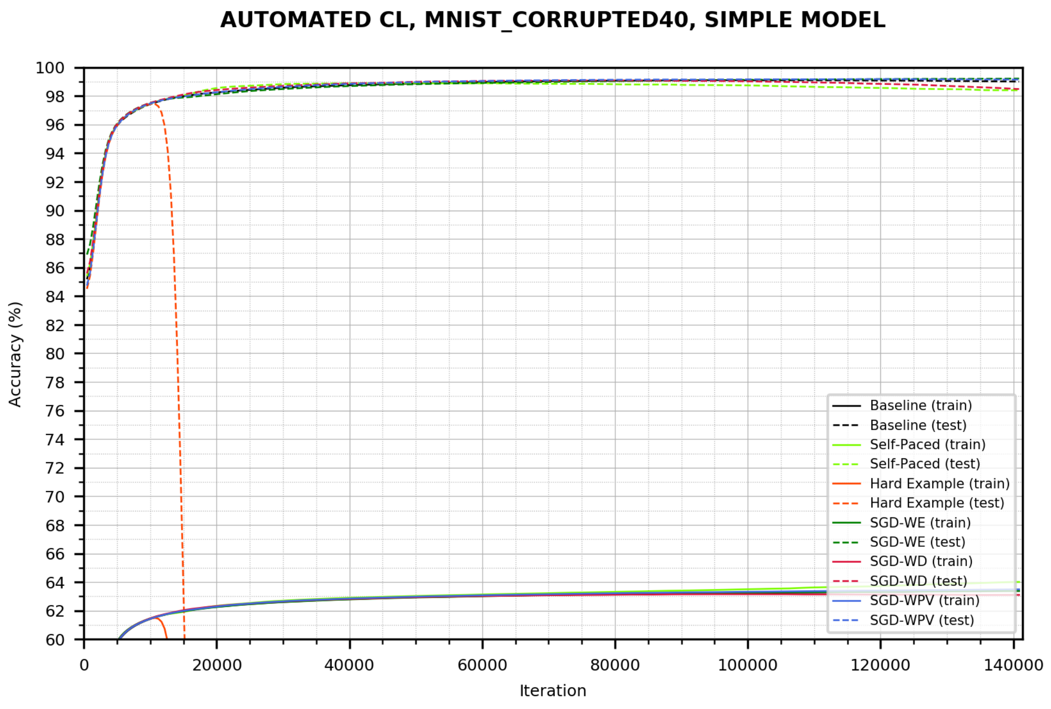


Figure 3.17: The results of the automated curriculum methods on MNIST_CORRUPTED40 dataset using the *simple model*.

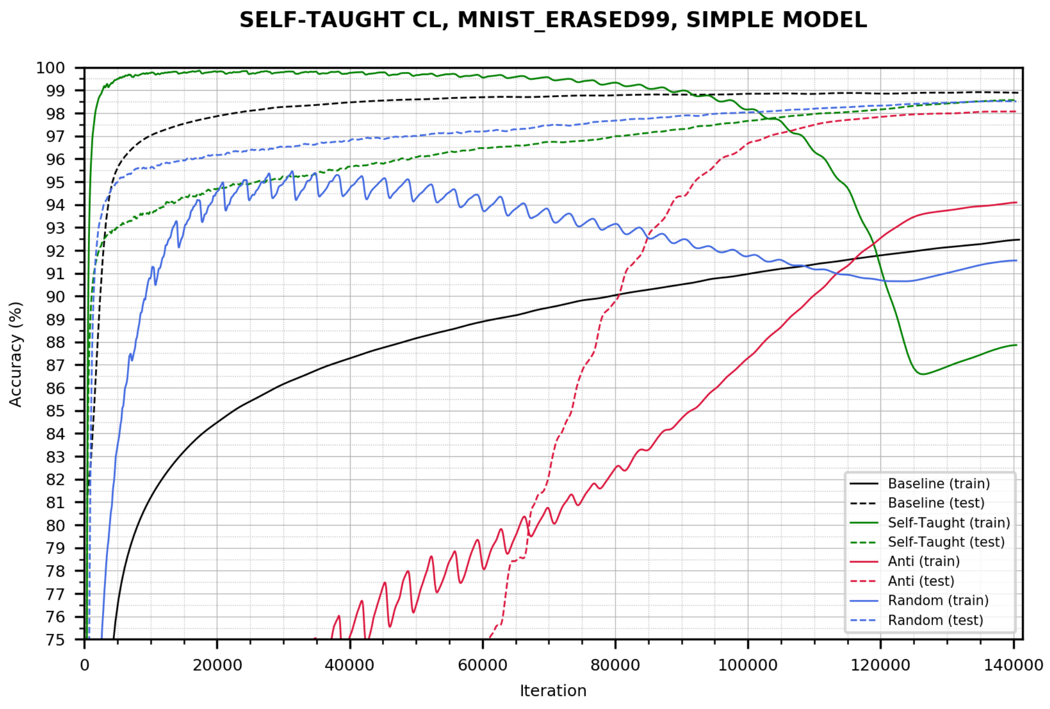


Figure 3.18: The results of *self-taught* curriculum methods on the MNIST_ERASED99 dataset using the *simple model*.

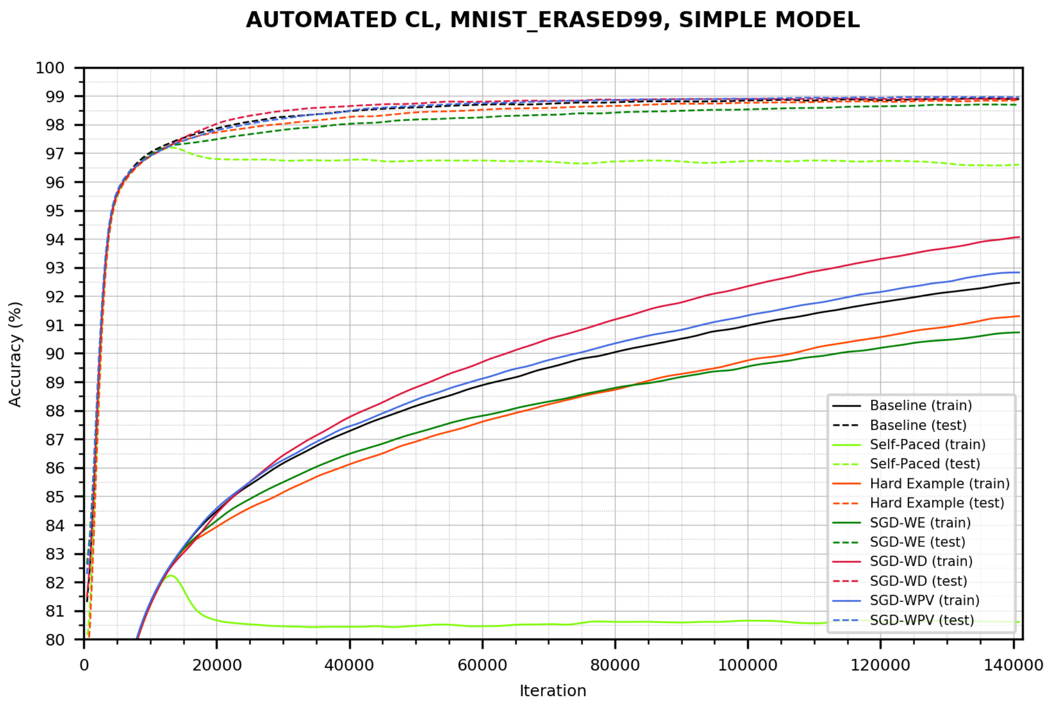


Figure 3.19: The results of the automated curriculum methods on MNIST_ERASED99 dataset using the *simple model*.

Chapter 4

Discussion

First, it should be noted that uniform weighting (i.e. the baseline method) is a very strong benchmark. It was possible to reproduce some results of related work in the experiments. Although, same experiments with a slightly different setup produced completely different outcomes. Results of *curriculum learning* approaches vary among the reviewed related work. The experiments in this work confirms this fact as there were dramatical differences among the tested methods.

It is clear from the experiments that success of a *curriculum learning* method is dependent on the dataset distribution and student's capabilities. These dependencies should be further investigated in future work to fully understand *curriculum learning* principles. The following rules are derived from the results of the experiments in this work.

- Emphasizing harder samples speeds up training. All the experiments show swift rise in test accuracies in the beginning of the training while using the SGD-WD method.
- Emphasizing harder samples works better for less capable students. This behavior was observed in the fifth and the twelfth experiment using the simpler model.
- Emphasizing easy samples works better on corrupted datasets. This was observed in the experiments on the corrupted datasets.

Learning rate tuning is a challenging task in *curriculum learning* as was noted in Hacohen et al. [16]. Our experiments were limited to a fixed rate that was determined by preliminary experiments. This approach might not be ideal and further investigation of learning rate setting while using CL might be beneficial. There were no differences in the observed accuracies while using the Adam optimizer in the preliminary experiments.

Both automated and *self-taught* methods for *curriculum learning* provided similar results given the same experiment setup. The overall better automated methods' results in the experiments might be caused by the usage of universal pacing function and a fixed learning rate for the *self-taught* methods. Additional time required for the hyper-parameter tuning and the need for a pre-training are a big disadvantage of the *self-taught* methods described in Hacohen et al. [16] and a disadvantage of all transfer-learning based approaches in general. Automated methods are much easier to use, especially, the methods based on the prediction history are easy to implement and delivered decent results in the experiments.

The SGD-WPV method that balances exploration and exploitation worked as good or better than the baseline in almost all experiments. I suggest examining this bandit-like approach in future work since it might lead to a general automated *curriculum learning* method. Moreover, it is consistent with findings in human teaching experiments where an unexplored space was divided by a kernel-based approach and exploration was conducted on top of this divided space in Basu et al. [12].

Bandit-like approaches are also common in related *curriculum learning* work addressing deep reinforcement learning problems. [21] In my opinion, *curriculum learning* used in this context might be much more beneficial and significant since it was shown that there are unsolvable tasks without a use of a curriculum in this field. [30]

Curriculum learning was studied on datasets with corrupted labels. [17][20] The experiments in this work confirm that emphasizing easy samples leads to better generalization on corrupted datasets. On the other hand, as was shown in Szegedy et al. [40], deep artificial neural networks have nonintuitive characteristics. The experiments on the distorted dataset MNIST_ERASED99 did not bring any reasonably interpretable results considering that the emphasizing of easy samples unexpectedly produced worse results than the opposite.

As I have already proposed, examining the training dataset properties and learning a curriculum from the observations might be crucial for future work. The curriculum should be correlated with the dataset and the student itself. The similar conclusion was given in Hacohen et al. [16]. Here, the *data-driven* approach introduced in Jiang et al. [17] might be useful.

Regarding the technical details, *curriculum learning* does not bring a huge overhead over a classical neural network training. The overhead and prolongation of training time are mostly caused by a lack of code optimization. Especially, the mini-batch sampling based on the current student progress is a big slowdown because the mini-batches cannot be prepared beforehand.

As it is not possible to pinpoint the best general *curriculum learning* method, for now, it is encouraged to find the best working curriculum for a particular task by the trial and error method. It would be nice to implement basic automated *curriculum learning* methods for emphasizing samples by difficulty in the Keras library so that everyone can easily experiment to find the best working curriculum for a given problem.

Conclusion

This work investigated one of the techniques that aim to increase accuracy of training deep artificial neural networks. Detailed description of *curriculum learning* was provided. Different approaches to introduce a curriculum to the training process of ANN models were analyzed. Subsequently, technical details of *curriculum learning* implementations in related work were provided.

A unified framework for *curriculum learning* evaluation was implemented in this work according to the description provided in Section 3.1. The transfer learning-based method was implemented in the *self-taught* variant, many automated *curriculum learning* methods based on weighting of training samples on the fly were implemented.

The conducted experiments partially confirmed findings of related work. On the other hand, some of the extended experiments provided contrary results. It was observed that a curriculum is highly dependent on a given task. The most consistent results were observed while using the automated method that balances exploration and exploitation.

The automated *curriculum learning* methods were recommended for wide use since they are easy to use and were shown to boost accuracies in the provided experiments.

Bibliography

[1] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[2] Bengio, Y. Practical Recommendations for Gradient-Based Training of Deep Architectures. In *Neural Networks: Tricks of the Trade - Second Edition*, 2012, pp. 437–478, doi:10.1007/978-3-642-35289-8_26. Available from: `https://doi.org/10.1007/978-3-642-35289-8_26`

[3] Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. Available from: `http://arxiv.org/abs/1412.6980`

[4] Krizhevsky, A.; Sutskever, I.; et al. ImageNet classification with deep convolutional neural networks. *Commun. ACM*, volume 60, no. 6, 2017: pp. 84–90, doi:10.1145/3065386. Available from: `http://doi.acm.org/10.1145/3065386`

[5] Krogh, A.; Hertz, J. A. A Simple Weight Decay Can Improve Generalization. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, 1991, pp. 950–957. Available from: `http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization`

[6] Srivastava, N.; Hinton, G. E.; et al. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, volume 15, no. 1, 2014: pp. 1929–1958. Available from: `http://dl.acm.org/citation.cfm?id=2670313`

[7] Skinner, B. F. Reinforcement today. *American Psychologist*, volume 13, 1958: pp. 94–99.

[8] Peterson, G. B. A day of great illumination: B. F. Skinner's discovery of shaping. *Journal of the Experimental Analysis of Behavior*, volume 82, 2004: p. 317–328.

[9] Elman, J. L. Learning and development in neural networks: The importance of starting small. *Cognition*, volume 48, 1993: pp. 71–99. Available from: `https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.2820`

[10] Sanger, T. D. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE Trans. Robotics and Automation*, volume 10, no. 3, 1994: pp. 323–333, doi:10.1109/70.294207. Available from: `https://doi.org/10.1109/70.294207`

[11] Bengio, Y.; Louradour, J.; et al. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, 2009, pp. 41–48, doi: 10.1145/1553374.1553380. Available from: `https://doi.org/10.1145/1553374.1553380`

[12] Basu, S.; Christensen, J. Teaching Classification Boundaries to Humans. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*, 2013. Available from: `http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6359`

[13] Khan, F.; Zhu, X. J.; et al. How Do Humans Teach: On Curriculum Learning and Teaching Dimension. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, 2011, pp. 1449–1457. Available from: `http://papers.nips.cc/paper/4466-how-do-humans-teach-on-curriculum-learning-and-teaching-dimension`

[14] Allgower, E. L.; Georg, K. *Introduction to Numerical Continuation Methods.* Society for Industrial and Applied Mathematics, 2003, doi:10.1137/1.9780898719154. Available from: `https://epubs.siam.org/doi/abs/10.1137/1.9780898719154`

[15] Bengio, Y. Evolving Culture vs Local Minima. *Studies in Computational Intelligence*, volume 557, 03 2012, doi:10.1007/978-3-642-55337-0_3. Available from: `https://www.researchgate.net/publication/221700451_Evolving_Culture_vs_Local_Minima`

[16] Hacohen, G.; Weinshall, D. On The Power of Curriculum Learning in Training Deep Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long*

Beach, California, USA, 2019, pp. 2535–2544. Available from: http://proceedings.mlr.press/v97/hacohen19a.html

[17] Jiang, L.; Zhou, Z.; et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 2309–2318. Available from: http://proceedings.mlr.press/v80/jiang18c.html

[18] Palamuttam, R.; Mohammad, H. H. CS 229 Final Project: Automated Curriculum Learning. 2017.

[19] Weinshall, D.; Cohen, G.; et al. Curriculum Learning by Transfer Learning: Theory and Experiments with Deep Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 5235–5243. Available from: http://proceedings.mlr.press/v80/weinshall18a.html

[20] Chang, H.; Learned-Miller, E. G.; et al. Active Bias: Training More Accurate Neural Networks by Emphasizing High Variance Samples. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 1002–1012. Available from: http://papers.nips.cc/paper/6701-active-bias-training-more-accurate-neural-networks-by-emphasizing-high-variance-samples

[21] Graves, A.; Bellemare, M. G.; et al. Automated Curriculum Learning for Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 1311–1320. Available from: http://proceedings.mlr.press/v70/graves17a.html

[22] Gülçehre, Ç.; Bengio, Y. Knowledge Matters: Importance of Prior Information for Optimization. *J. Mach. Learn. Res.*, volume 17, 2016: pp. 8:1–8:32. Available from: http://jmlr.org/papers/v17/gulchere16a.html

[23] Kumar, M. P.; Packer, B.; et al. Self-Paced Learning for Latent Variable Models. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, 2010, pp. 1189–1197. Available from: http://papers.nips.cc/paper/3923-self-paced-learning-for-latent-variable-models

[24] Jiang, L.; Meng, D.; et al. Self-Paced Curriculum Learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 2015, pp. 2694–2700. Available from: `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9750`

[25] Felzenszwalb, P. F.; Girshick, R. B.; et al. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 32, no. 9, 2010: pp. 1627–1645, doi: 10.1109/TPAMI.2009.167. Available from: `https://doi.org/10.1109/TPAMI.2009.167`

[26] Shrivastava, A.; Gupta, A.; et al. Training Region-Based Object Detectors with Online Hard Example Mining. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 761–769, doi:10.1109/CVPR.2016.89. Available from: `https://doi.org/10.1109/CVPR.2016.89`

[27] Settles, B. *Active Learning Literature Survey.* 1648, 2009. Available from: `http://active-learning.net/`

[28] Pentina, A.; Sharmanska, V.; et al. Curriculum learning of multiple tasks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 5492–5500, doi:10.1109/CVPR.2015.7299188. Available from: `https://doi.org/10.1109/CVPR.2015.7299188`

[29] Collier, M.; Beel, J. An Empirical Comparison of Syllabuses for Curriculum Learning. In *Proceedings for the 26th AIAI Irish Conference on Artificial Intelligence and Cognitive Science Trinity College Dublin, Dublin, Ireland, December 6-7th, 2018*, 2018, pp. 150–161. Available from: `http://ceur-ws.org/Vol-2259/aics_15.pdf`

[30] Behbahani, F. *Automated Curriculum Learning for Reinforcement Learning.* 2018, (accessed December 7, 2029). Available from: `https://rlcurriculum.github.io/`

[31] Justesen, N.; Risi, S. Automated Curriculum Learning by Rewarding Temporally Rare Events. In *2018 IEEE Conference on Computational Intelligence and Games, CIG 2018, Maastricht, The Netherlands, August 14-17, 2018*, 2018, pp. 1–8, doi:10.1109/CIG.2018.8490448. Available from: `https://doi.org/10.1109/CIG.2018.8490448`

[32] Collobert, R.; Weston, J.; et al. Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res.*, volume 12, 2011: pp. 2493–2537. Available from: `http://dl.acm.org/citation.cfm?id=2078186`

[33] Mikolov, T.; Deoras, A.; et al. Strategies for training large scale neural network language models. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding, ASRU 2011, Waikoloa, HI, USA, December 11-15, 2011*, 2011, pp. 196–201, doi:10.1109/ASRU.2011.6163930. Available from: `https://doi.org/10.1109/ASRU.2011.6163930`

[34] Abadi, M.; Agarwal, A.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org. Available from: `https://www.tensorflow.org/`

[35] Chollet, F.; et al. Keras. `https://github.com/keras-team/keras`, 2015.

[36] LeCun, Y.; Cortes, C. MNIST handwritten digit database. 2010. Available from: `http://yann.lecun.com/exdb/mnist/`

[37] Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, 05 2012. Available from: `https://www.cs.toronto.edu/~kriz/cifar.html`

[38] Netzer, Y.; Wang, T.; et al. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS*, 01 2011. Available from: `http://ufldl.stanford.edu/housenumbers/`

[39] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015.

[40] Szegedy, C.; Zaremba, W.; et al. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. Available from: `http://arxiv.org/abs/1312.6199`

APPENDIX A

Acronyms

AI Artifical Intelligence

ANN Artifical Neural Network

CL Curriculum Learning

CNN Convolutional Neural Network

CV Computer Vision

MLP Multilayer Perceptron

NLP Natural Language Processing

RL Reinforcement Learning

SE Standard Error

SPCL Self-Paced Curriculum Learning

SPL Self-Paced Learning

SVM Support Vector Machines

Appendix B

Contents of Enclosed CD

```
src ......................... source codes of the implemented framework
commands................... example commands for running the training
text ......................................... the thesis text directory
  thesis.pdf .......................... the thesis text in PDF format
  thesis.tex .............................. source code of the thesis
  images .................................... images used in the text
```