CURRICULUM LEARNING

Otilia Stretcu

JUNE 2021

CMU-ML-21-105

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

THESIS COMMITTEE

Tom Mitchell, Co-Chair
Barnabás Póczos, Co-Chair
Ruslan Salakhutdinov
Rich Caruana

Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

Copyright © Otilia Stretcu

This research was supported by: the Air Force Research Laboratory award numbers FA865013C7360, FA87501720130, FA95501710218 and FA95502010118; the National Institutes of Health award numbers R01HD075328 and U01NS098969; the National Science Foundation award numbers IIS1247658 and IIS1563887; the United States Army award number W911NF1020022; a fellowship from the University of Pittsburgh Medical Center; gifts from Google and Verizon; and research grants from JP Morgan Chase Bank and Lockheed Martin.

Keywords: curriculum learning, machine learning, deep learning, multi-task learning, semi-supervised learning, self-supervised learning, function composition, classification.

Curriculum Learning

Committee:
1. Tom Mitchell, Co-Chair
2. Barnabás Póczos, Co-Chair
3. Ruslan Salakhutdinov
4. Rich Caruana

Date of submission: May, 2021
Date of defense: June 8th, 2021

Please cite this document as:
Otilia Stretcu "Curriculum Learning."
Carnegie Mellon University PhD Thesis. 2021.
URN: CMU-ML-21-105
URL: http://reports-archive.adm.cs.cmu.edu/anon/ml2020/CMU-ML-21-105.pdf

This document is provided by:
Machine Learning Department Reports,
SCS Technical Report Collection,
Carnegie Mellon University.
http://reports-archive.adm.cs.cmu.edu
reports@cs.cmu.edu

© 2021 by Otilia Stretcu. This thesis is made available under a Creative Commons "Attribution — No Derivatives 4.0" license: https://creativecommons.org/licenses/by-nd/4.0/deed.en.

Dedicated to my parents who inspired me to see the beauty in science and always supported me in pursuing my passions, to my little brother who once told me to aim for the stars, and to my loving grandparents who taught me that hard work and dedication can get you far.

ABSTRACT

Artificial Intelligence (AI) researchers often disagree about the best strategy to train a machine learning system, but there is one belief that is generally agreed upon: humans are still much better learners than machines. Unlike AI systems, humans do not learn challenging new tasks (e.g., solving differential equations) from scratch, by looking at independent and identically distributed examples. Instead, humans often follow sequences of steps that allow them to incrementally build up the necessary skills for performing these new tasks. *Curriculum Learning* (CL) is a line of work that tries to incorporate this human approach to learning into machine learning, with the hope that machines trained in this manner can *learn faster* and *perform better*. However, biological brains are different than silicon brains, and are not trained by using gradient descent, which has become the norm in machine learning. So, can we expect human learning strategies to work for computers, too? Evidence from various studies in the past two decades suggests that CL can indeed benefit machine learning in some cases, while in others it may in fact hinder performance (Elman, 1993; Rohde and Plaut, 2003; Bengio et al., 2009; Bojar et al., 2017b). In this thesis we aim to discover the problem settings in which different forms of CL are beneficial, and the types of benefits they provide. We posit the following statement:

THESIS STATEMENT: *AI systems that learn like humans, starting with easy problems and gradually tackling more and more difficult ones, have the potential to reach* ***better local optima*** *and/or* ***converge faster****. Furthermore, the learning benefits gained using a curriculum depend on the choice of* ***curriculum****, the* ***size and type of data****, and the* ***model architecture****.*

In this work, we provide evidence for this statement, as well as investigate what types of data and models can benefit from CL. We start by introducing a definition of CL and identifying three broad categories of CL methods. We further provide a literature review of the main CL approaches in the past three decades. Moreover, we propose new CL methods and apply them to a variety of models and problem settings, from teaching an LSTM to solve basic arithmetic problems, to neural machine translation using Transformers, image classification using convolutional neural networks, and compositional multitask learning problems. Through these experiments, we observed that CL can be very beneficial in certain settings (e.g., on sequential data such as sentences) if well-designed, but it can also harm the efficiency of learning if performed poorly (e.g., if the curriculum spends too much time on easy problems). Finally, we conduct analyses to understand *why* CL leads to the observed effects.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisors, Barnabás Póczos and Tom Mitchell, for their invaluable advice and guidance throughout my PhD years. Throughout this journey, they have always encouraged me to explore new problems and work on topics that I was personally interested in, while at the same time always reminding me to sleep and rest more. From Barnabás, I learned how to be methodical and how to ask insightful technical questions that get to the heart of a problem. Whenever my methods had a weakness, Barnabás would sure discover it. On the other hand, Tom has always encouraged me to question the question that I am trying to answer, and to find problems that are worth pursuing. All in all, I feel extremely lucky and grateful for having had the chance to be co-advised by two brilliant researchers and wonderful people. Moreover, I am very grateful to my thesis committee members, Ruslan Salakhutdinov and Rich Caruana, for their feedback, advice, time, and very importantly, for their support and excitement about the ideas in my thesis. Their insightful questions made this thesis better!

I would also like to thank my professors and advisors from my undergraduate and master's studies who first introduced me to research and guided me in the decisions that lead to this PhD: Emilia Petrișor, Marius Minea, Marius Leordeanu, and Pietro Liò. I would not be here without their help and advice, for which I will be forever grateful. Moreover, I would like to thank Matthias Seeger and Emtiyaz Khan for giving me the opportunity to work on my very first machine learning research project during a summer internship at EPFL, Lausanne. This experience led me to consider pursuing a PhD in machine learning. I would also like to thank Rodica Crețescu, my very first informatics teacher, who taught me to program and pushed me to exceed my limits in the informatics olympiads. These skills, and more importantly, my passion for computer science, have been crucial in my PhD journey. I am also very grateful to have had the opportunity to test my research skills in practice during my summer internships at Google, working with Chen Wu, Krishnamurthy Viswanathan, and Dana Alon. It was through these wonderful experiences that I decided what I wanted to do next: using the knowledge acquired in my PhD to solve real-world problems that can improve people's lives.

Moreover, this PhD would not have been possible without my many collaborators and fellow PhD students, who shared ideas, provided valuable feedback and inspired a lot of my work: Anthony Platanios, Mariya Toneva, Anna Chrabaszcz, George Stoica, Han Zhao, Abulhair Saparov, Avinava

Dubey, Maruan Al-Shedivat, Chris Dann, Andreea Bobu, Sebastian Claici, Sharmistha Jat, Dan Schwartz, Nicole Rafidi, Jennifer Williams, Ashique Khudabukhsh, Chenghui Zhou, Xiao Fu, Kejun Huang, Hyun Ah Song, Evangelos Papalexakis, Witold Lipski, Alan Bush, Renato Negrinho, Manzil Zaheer, and Barun Parta. Many thanks to you all! Special thanks to Anthony Platanios who showed me that researchers can express brilliant ideas through elegantly-written code (whether it's written in Scala or not), and listened with stoicism to my complaints about TensorFlow. Undoubtedly, I am a better researcher because of him. I would also like to thank the faculty at CMU and other collaborating institutions, from whom I have learned a lot through collaborations or just interesting discussions on the 8th floor of Gates. Many thanks to Leila Wehbe, Rayid Ghani, Zach Lipton, Aaditya Ramdas, Nicholas Sidiropoulos, Christos Faloutsos, Partha Talukdar, Geoff Gordon, Alex Smola, Mike Tarr, and Mark Richardson. Of course, my PhD experience would not have been the same without the attention and care of the MLD staff, Diane Stidle, Mary Stech, Sharon Cavlovich, Dorothy Holland-Minkley, Russ O'Lare, and Kelly Widmaier. I am particularly grateful to Diane, who has made MLD feel like a home away from home!

A big thank you to my fun and caring friends who have made my PhD an unforgettable experience: Abs Saparov, Avillama Dubey, Brynn Edmunds, Daniel Bird, Mariya Toneva, Chris Dann, Sushma Anand Akoju, Maruan Al-Shedivat, Marimaya Sachan, Xun Zheng, Dan Robbie Schwartz, Zhong Zhou, Chengy Zhou, Robin Schmucker, Charvi Rastogi, Ezra Winston, and Biswajit Paria. I hope our paths will cross again soon!

Last, but not least, I would like to thank my family for their constant love and support. My parents have always encouraged me to pursue my passions, wherever they may take me, and have made great efforts to ensure I can achieve them—for which I cannot thank them enough. My dad inspired me to appreciate the beauty in math from a young age, while my mom inspired me to develop a curiosity for physics and for understanding how things work. This led to me choosing, without realizing, a field at the intersection of their passions: computer science. However, a big source of inspiration throughout my academic years has been my little brother, Cătă. His love and admiration has always pushed me to be a better version of myself. Very importantly, I want to also thank my grandparents, who worked hard to overcome adversities in life, to give their children and grandchildren a good education. Moreover, their work ethic is something that I always look back to and keeps me going, whenever I feel overloaded with work. Finally, I would like to thank my former office-mate, collaborator, best friend, partner in crime and in life, Anthony Platanios. His love, support, and constant ability to make fun of me, have kept me going through the highs and lows of PhD life and a year in COVID-19 lockdown. Looking forward to keep laughing together, wherever life may take us next!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1 INTRODUCTION

The field of artificial intelligence (AI) has witnessed an impressive leap in the last decade. Machines are now able to perform tasks never before thought to be possible, such as driving cars or interacting with humans in natural language. However, these advances were only possible through large data collection efforts. Humans, on the other hand, are very good at learning new skills efficiently using incredibly small amounts of supervision. Inspired by this, AI researchers have often attempted to create models that resemble the way the human brain works, with notable examples being convolutional neural networks (LeCun et al., 2015) and various forms of attention mechanisms (e.g., Vaswani et al., 2017). However, one key difference between human learning and machine learning (ML) that is often overlooked lies not in the model architecture, but in the way in which they learn. Unlike most ML systems, humans do not learn difficult new tasks (e.g., solving differential equations) entirely from scratch by looking at independent and identically distributed examples. Instead, new skills are often learned progressively, starting with easier tasks and gradually becoming able to tackle harder ones. For example, students first learn to perform addition, multiplication, differentiation, and solving simple equations, before starting to learn about differential equations. Thus, we can think of human learning as often being aided by a *curriculum*, which may either be provided by a teacher or chosen directly by the student.

Curriculum learning (Elman, 1993; Bengio et al., 2009) is a line of work in machine learning that attempts to devise learning strategies inspired by human learning, in which concepts are learned in the order of difficulty, from easy to hard. This relies on the assumption that, similar to humans, machines trained in this manner can *learn faster* or *perform better*. However, biological brains are different than silicon brains, and are not trained by using gradient descent, as it has become the norm in machine learning. So, can we expect human learning strategies to work for computers, too? Two decades after this idea was originally proposed (Elman, 1993), there is increasingly more evidence that curriculum learning can indeed improve learning (e.g., Bengio et al., 2009; Jiang et al., 2015; Pentina et al., 2015; Jiang et al., 2018; Zhou and Bilmes, 2018; Wu et al., 2021), by helping the model learn faster (e.g., Krueger and Dayan, 2009; Platanios et al., 2019; Li et al., 2020) or reach better final performance (e.g., Bengio et al., 2009; Jiang et al., 2015; Sachan and Xing, 2016; Platanios et al., 2019). However, there is also evidence that curriculum learning does not benefit learning, or in some cases it even harms the performance of the model (e.g., Rohde and Plaut, 1999, 2003; Avramova, 2015; Bojar et al., 2017b). Even more, taking certain curriculum learning methods that work successfully for one problem and applying them to another, often does not result in benefits for the new problem. Therefore, *when* and *how* to make curriculum learning work is still far from being understood.

In this thesis, we pose the following questions:

1. *What* exactly is curriculum learning?
2. *When* is curriculum learning useful? Does it work only for certain types of data, or only for certain models?
3. *What* kinds of benefits can curriculum learning bring to the learning process (e.g., better performance, faster convergence)?
4. *How* does a curriculum learning affect the optimization process, and *why* does it sometimes work and sometimes not?

In this work we attempt to address these questions. More concretely, our goal is to better understand in what problem settings curriculum learning is beneficial, as well as the types of curriculum learning strategies that are appropriate for each situation. Additionally, we propose new curriculum learning methods targeted at certain properties of the data, the models or the tasks at hand, from teaching an LSTM (Hochreiter and Schmidhuber, 1997) to solve basic arithmetic problems, to neural machine translation using Transformers (Vaswani et al., 2017), image classification using convolutional neural networks, and compositional multitask learning problems. We tackle the above questions empirically through extensive experimentation using various datasets and models, as well as analytically, with the goal of predicting which new learning settings could also benefit from using curriculum learning.

Moreover, we believe that one of the reasons why curriculum learning is not well understood is the lack of a common formal language in which to describe the various approaches. Each new approach to curriculum learning has its own interpretation of what *easy* and *hard* are measuring, as well as which component of the learning pipeline to modify in order to incorporate a curriculum (e.g., the data, the loss, or the model architecture). Therefore, as a first step, we introduce a formalism that allows us to describe both the existing and proposed approaches in a unified way, as well as compare curriculum learning with other related fields, such as transfer learning and continual learning.

Furthermore, to answer the questions above, we consider not only the takeaways from our own experiments, but also the results obtained so far by the entire community. Using the formalism mentioned above, we will put together and categorize the existing curriculum learning literature in the first curriculum learning survey. We believe that having a compiled version of the developments in curriculum learning over the past two decades will benefit not only the work in this thesis, but also future work in the field.

We propose the following statement:

THESIS STATEMENT: *AI systems that learn like humans, starting with easy problems and gradually tackling more and more difficult ones, have the potential to reach* ***better local optima*** *and/or* ***converge faster****. Furthermore, the learning benefits gained using a curriculum depend on the choice of* ***curriculum****, the* ***size and type of data****, and the* ***model architecture****.*

In the rest of the thesis, we provide evidence for this statement, as well as further details on the model types and data that can benefit from curriculum learning. We discovered that different problem settings can benefit more from different types of curricula. For instance, for problems on sequential data—be it natural language, numerical data, or simple as sequences of bits—we found *input space* curricula (that is, curricula that schedule when the training samples are shown to the model) to be consistently beneficial, providing faster convergence speeds and often better performance (Chapter 4). Our analysis into why this is the case (Chapter 6) suggests that curricula that correlate with sequence length can reduce the gradient noise, as well as widen the local minima in the loss landscape early on during training. On the other hand, for tasks of compositional nature, we found that *task space* curricula can significantly improve performance (Chapter 5). However, we also discovered some settings where curriculum learning is not improving, or even harming the learning process. For example, curricula that are too long can in fact perform worse than the baseline (e.g., Section 4.5). At the same time, curricula that are otherwise successful when training a model from scratch may have no impact when applied to pretrained models (Section 6.5).

In summary, this thesis aims to further our understanding about curriculum learning, as well as propose new methods appropriate for different situations. We hope that the methods and analyses presented in this thesis will provide useful insights for practical applications and for future research in the field.

1.1 THESIS OVERVIEW

This thesis is structured in the following main chapters:

1. **What is Curriculum Learning:** We introduce a definition of curriculum learning that allows us to formally describe existing and proposed curriculum learning approaches under a unified framework. We then use this definition to categorize the main lines of work in curriculum learning. Note that when this work was performed, it was the first attempt at unifying existing approaches under a common framework. Finally, we also provide a comparison of curriculum learning to other learning paradigms, such as transfer learning and active learning.

2. **Literature Survey:** We review the curriculum learning literature, from the early ideas that started in the fields of psychology and cognitive neuroscience, to state-of-the-art machine learning methods.

3. **Curriculum in Input Space:** We propose curriculum learning methods in which the curriculum acts as a filter on the data, deciding when each training example is presented to the learner based on its difficulty. We apply these to multiple problems (e.g., machine translation, multimodal image understanding, learning arithmetics) using a variety of data modalities (e.g., text, images, and numbers in the form of digit sequences).

4. **Curriculum in Task Space:** We introduce curriculum learning methods that consider a series of learning tasks with increasing difficulty. This can be done in the context of learning a single task, or learning multiple tasks together. For single task learning, we propose a curriculum learning method that can automatically create a series of auxiliary tasks—which are not explicitly provided—as intermediate goals. For multitask learning, where multiple tasks are provided, we show that by leveraging their relationships and learning them in a particular order defined by a curriculum, we can improve their performance.

5. **Understanding Curriculum Learning – A Case Study on Sequential Data:** In this chapter we focus on understanding the effect of curriculum learning on the optimization process. We consider problems involving sequences as the target problem setting because such problems have consequences for multiple application areas, and curriculum learning has already proven to work well for them. Using several case studies, we analyse the model gradients and visualize the loss landscapes, and attempt to explain the effects that we observe when using curriculum learning. Moreover, we propose a conjecture which can help us decide for future problems whether a length-based curriculum can be helpful in that setting.

1.2 BACKGROUND

We now provide a brief overview of the field and clarify what kind of approaches fit under our definition of "curriculum learning" in this thesis.

The ideas behind curriculum learning originated in the field of cognitive neuroscience. Inspired by the process of language development in children, Elman (1993) was among first advocates for the "importance of starting small" when training artificial neural networks. In this early work, he trained a simplified language model using a recurrent neural network (RNN), and showed that the network was only able to learn when the training process was guided using a curriculum. This idea was later disputed by Rohde and Plaut (1999, 2003), who showed contradictory results and argued that in fact, "less is less". Afterwards, these ideas remained mainly unexplored until the influential paper of Bengio et al. (2009) that coined the term *curriculum learning* and lead the way to a rediscovery of these earlier ideas, introducing them to the machine learning community. In the next decade, a multitude of curriculum learning methods were proposed. While all these approaches attempted to train models to solve problems in order of difficulty, "from easy to hard", *what* was being scored by difficulty and *how* the scoring was performed could vary significantly. For example, one line of work focused on presenting the training examples the model in a particular order, while the objective function the model is optimizing was kept fixed throughout training (e.g., Bengio et al., 2009; Jiang et al., 2015, 2018; Wang et al., 2018; Zhou and Bilmes, 2018; Platanios et al., 2019). Other approaches allowed the learner to see all the training examples throughout learning, but they made the learning goal of the model incrementally more difficult with time (Florensa et al., 2017; Saxena et al., 2019; Dogan et al., 2020; Stretcu et al., 2020). A less explored direction is one that alters the model during training, where the some of the model's features could purposefully be disabled to force the model to "start small". For instance, Elman (1993) trained a simple RNN language model by limiting the amount of memory the network has in the early stages of training, and allowing it to get larger and larger with every training phase. This was an attempt to replicate the fact that, when children learn their first language starting at a few months of age, their brain is simultaneously undergoing some changes, and their memory and attention span is growing during this time. Along the same lines, Sinha et al. (2020) proposed a curriculum learning approach for image understanding that controls the amount of texture information that a convolutional neural network (CNN) is able to represent during training.

While all these different learning strategies are intuitive for humans, it is not entirely clear why curriculum learning would benefit a machine learning model. In fact, a few publications have reported that *anti-curriculum* approaches (i.e., training on hard examples first) sometimes work better than the reverse (Mermer and Amasyali, 2017). Moreover, in active learning, a field closely related to curriculum learning, many approaches suggest tackling what can be considered the "hardest" samples first, such as those for which the model has the highest uncertainty, or those that are closest to the decision boundary (Fu et al., 2013). These observations, coupled with the limited amount of theoretical analyses of curriculum learning (Gong et al., 2016; Weinshall et al., 2018; Hacohen and Weinshall, 2019), raise the question of whether "easy first" is indeed the right policy.

Moreover, as seen from all examples above, there is no consensus on a definition of curriculum learning and what objective it should optimize. For this reason, one of the contributions of this thesis consists of introducing a formal definition of curriculum learning and its variants, and describing and comparing existing approaches in terms of a common terminology. We discuss this next.

2 WHAT IS CURRICULUM LEARNING?

Curriculum learning can be described informally as a strategy for training machine learning systems on incrementally more difficult learning tasks, similar to how we would teach a human. While this idea is intuitive, it is vague with respect to several aspects. What exactly is a *learning task*? How do we define incrementally more difficult tasks and how do we transition between them? In fact, different approaches in the literature have found different interpretations of curriculum learning, and it is important to have a common language to describe and compare them.

In this chapter, we propose some definitions along with some notation that allow us to define curriculum learning formally. We then use these definitions to describe several *categories* of curriculum learning which we have identified in the field.

2.1 DEFINITIONS

We start by introducing definitions and terminology that allow us to formally describe existing and proposed curriculum learning methods. To the best of our knowledge, this is the first attempt to unify all the existing curriculum learning approaches under a common terminology. We then use this terminology in the next chapter to survey the main lines of work in the curriculum learning literature.

Consider a learning task where the goal is to learn the function $f_\theta : \mathcal{X} \to \mathcal{Y}$ that operates on an arbitrary feature space $\mathcal{X}$, projects to an arbitrary target domain $\mathcal{Y}$, and has trainable parameters θ. To train f, we are provided with the training dataset $D = \{(x_1, y_1), ..., (x_N, y_N)\}$ containing N training examples with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, for $i \in \{1, ..., N\}$. For simplicity, we denote the set of training examples as two tensors, X and Y, that contain all of the training inputs and outputs, respectively, stacked along the first dimension. Moreover, we denote the marginal probability distribution of the inputs by $P(X)$, the marginal probability distribution of the labels by $P(Y)$, and the labels conditional distribution by $P(Y|X)$.

Using this notation, we define a *learning task* as:

$$\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}, \tag{2.1}$$

and the *input domain* it operates on as:

$$\mathcal{D} = \{\mathcal{X}, P(X)\}. \tag{2.2}$$

Notation	Description
$\mathcal{X}$	input/feature space
$\mathcal{Y}$	output/targets space
$P(X)$	inputs marginal distribution
$P(Y)$	labels marginal distribution
$P(Y\|X)$	labels conditional distribution
$\mathcal{T}$	learning task; $\mathcal{T} = \{\mathcal{Y}, P(Y\|X)\}$
$\mathcal{D}$	input domain; $\mathcal{D} = \{\mathcal{X}, P(X)\}$
D	training dataset; $D = \{(x_i, y_i)\|i = 1..N\}$
f	prediction function; $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$
θ	parameters of function f
$\mathcal{L}$	learning context; $\mathcal{L} = (\mathcal{D}, \mathcal{T}, D, f_\theta)$

Table 2.1: Summary of notations used in our definitions.

We will refer to all of these components together as the *learning context*, denoted as:

$$\mathcal{L} = (\mathcal{D}, \mathcal{T}, D, f_\theta) \tag{2.3}$$

Table 2.1 summarizes these definitions. Let us further ground these definitions using a concrete example. Consider an image classification task where the goal is to classify the main object shown in the image (e.g., such as in the CIFAR-10 dataset (Krizhevsky, Hinton, et al., 2009)). In this case, $\mathcal{X}$ represents the space of images, $\mathcal{Y}$ represents the space of labels, the dataset D contains example pairs of images and their corresponding labels, and f_θ is the neural network that is trained to solve the task. Moreover, $P(X)$ represents the probability of each input image to appear in the dataset, and $P(Y|X)$ represents that probability of a label assignment given an input image.

Having established our notation, we can now formally define curriculum learning, as follows:

Definition 2.1. Given a target domain $\mathcal{D}_T$ and a learning task $\mathcal{T}_T$, *curriculum learning* aims to improve the learning of the target predicting function f_{θ_T} operating over $\mathcal{D}_T$, using knowledge acquired by learning a series of K functions, f_{θ_1},..., f_{θ_K}, operating on domains $\mathcal{D}_1, ..., \mathcal{D}_K$ and solving tasks $\mathcal{T}_1, ..., \mathcal{T}_K$, respectively. The domains $\mathcal{D}_1, ..., \mathcal{D}_K$, tasks $\mathcal{T}_1, ..., \mathcal{T}_K$ and functions f_{θ_1},..., f_{θ_K}, are typically derived from $\mathcal{D}_T$, $\mathcal{T}_T$, and f_{θ_T}, respectively.

Definition 2.2. We refer to the functions f_{θ_1},..., f_{θ_K} as *auxiliary functions*, because their sole purpose is to aid the learning of the target function f_{θ_T}.

We illustrate this definition graphically in Figure 2.1. Note that the curriculum learning literature includes a variety of approaches for deriving the auxiliary functions, and it is possible, depending on the approach, that

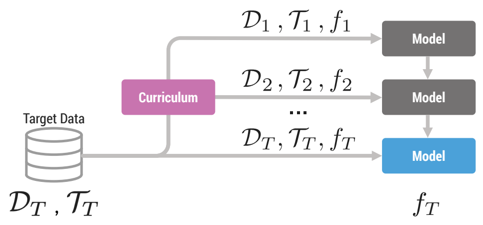

$\mathcal{D}_1, \mathcal{T}_1, f_1$
$\mathcal{D}_2, \mathcal{T}_2, f_2$
$\mathcal{D}_T, \mathcal{T}_T, f_T$
Model
Model
Model
Curriculum
Target Data
$\mathcal{D}_T, \mathcal{T}_T$
f_T

Figure 2.1: Graphic illustration of the curriculum learning definition.

either $f_{\theta_1} = f_{\theta_2} = ... = f_{\theta_K}$, or $\mathcal{T}_1 = \mathcal{T}_2 = ... = \mathcal{T}_K$, or $\mathcal{D}_1 = \mathcal{D}_2 = ... = \mathcal{D}_K$. In the next section, we categorize existing approaches based on which of these components are changed and which are kept fixed. Moreover, in order to decide on the sequence of auxiliary tasks, functions and domains, the method may use various kinds of information:

1. Data properties, such as sentence length (e.g., Platanios et al., 2019), word frequency (e.g., Bengio et al., 2009), etc.
2. Model properties, such as iteration number, history of validation accuracy, various learning signals (e.g., Graves et al., 2017).
3. Importance of each sample to current model, such as the loss on each sample in self-paced learning (e.g., Kumar et al., 2010).
4. Domain knowledge directly encoded by the model designer (e.g., in deciding how to measure the sample difficulty or how to generate the auxiliary tasks).

2.2 CATEGORIZATION OF CURRICULUM LEARNING METHODS

There are two main dimensions of comparison when discussing curriculum learning (CL) methods:

I. Learning Context: Different methods may keep fixed or change different components of the learning context $\mathcal{L} = (\mathcal{D}, \mathcal{T}, D, f_\theta)$. Thus, one way to categorize CL methods is by grouping them in terms of the components they modify. To this end, we define 3 such categories:
   a) *curriculum in input space*, which modifies the inputs domain, progressively training the model on domains $\mathcal{D}_1, ..., \mathcal{D}_K$ before finally training it on the target domain $\mathcal{D}_T$.
   b) *curriculum in task space*, which modifies the task, progressively training on tasks $\mathcal{T}_1, ..., \mathcal{T}_K$ before training on the target task $\mathcal{T}_T$.
   c) *curriculum in model space*, which modifies the learning function, progressively training the functions $f_{\theta_1}, ..., f_{\theta_K}$ before training the target function f_{θ_T}.

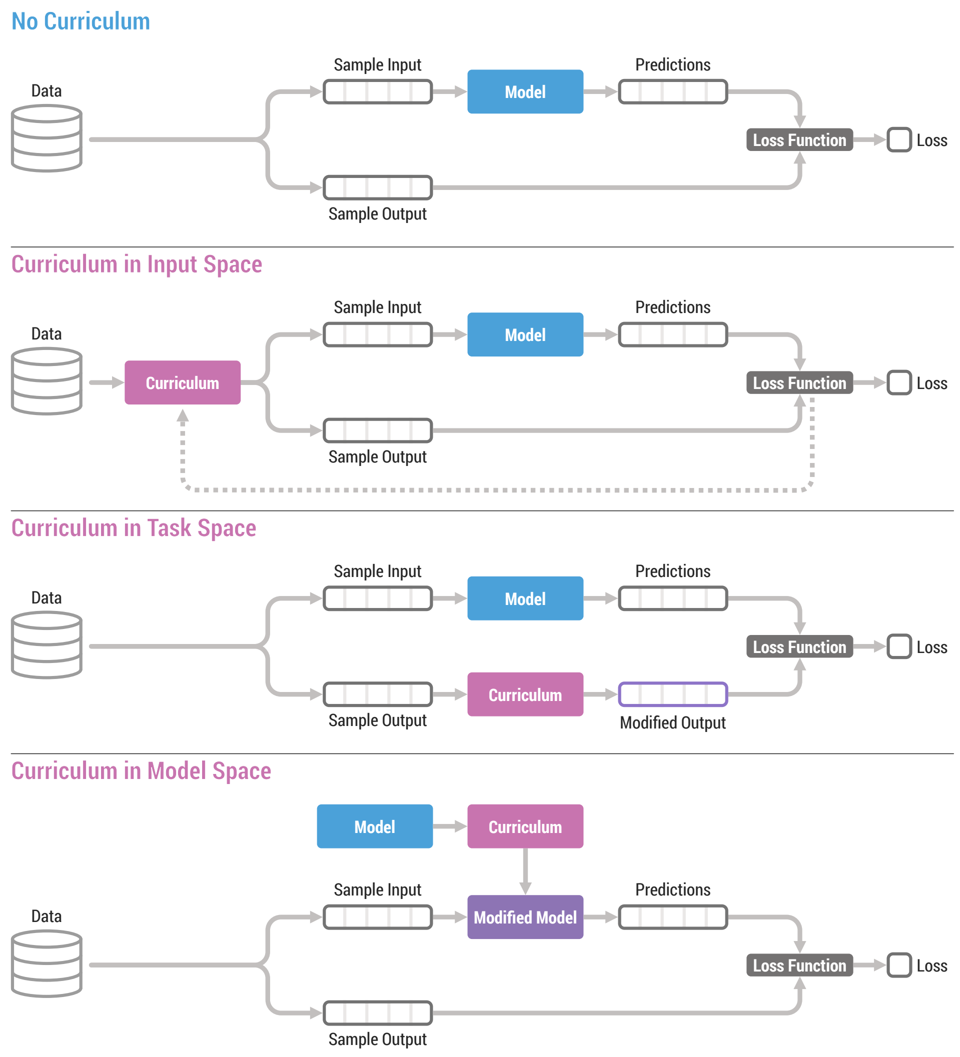


Figure 2.2: A comparison of the main types of curriculum learning approaches. This illustration is meant to showcase the main characteristics of these three types of curriculum approaches, but there can be exceptions. For instance, a curriculum in task space may also produce small modifications to the model architecture (e.g., the output layer), to be able to handle the modified targets. Similarly, different types of curriculum may also modify the loss function.

Figure 2.2 provides an illustration of these categories. Of course, it is possible for a CL method to modify multiple of the above components simultaneously (e.g., Saxena et al., 2019), but such approaches are not common.

II. Human Supervision: Different methods rely to different extents on human decisions. We can categorize the existing methods based on how much human effort is required for the creation of the auxiliary tasks, which can range from completely *hand-crafted* (i.e., a human has to decide what all the components of the learning task are), to completely *automated* (i.e., everything is generated automatically). For instance, in our object recognition example, the user might decide which images are easy and which are hard based on their own intuition (e.g., images with fewer background objects are easier), and how exactly the images will be presented to the learner based on their difficulty. Alternatively, the image difficulties, as well as they way they are presented to the learner, can be automatically inferred from the data, using metrics independent of the domain or task at hand (e.g., the difficulty can be the loss of the model on a particular example).

In the next chapter, use the first criteria of comparison—learning context—to structure our discussion on existing and proposed curriculum learning approaches. When discussing each category, we will also mention the amount of human supervision required for each approach.

2.3 COMPARISON WITH RELATED FIELDS

Curriculum Learning (CL) is often confused with other learning strategies, such as transfer learning or continual learning. Therefore, it is useful to understand the relationships between CL and such other methods. We now enumerate the closest related fields, and discuss their similarities to and differences from CL:

- **Curriculum Learning (CL):** Based on the previous definitions, we can see CL as a way of *pretraining* a model on an *altered* version of the *original learning context* before training on the target setup of interest.
- **Transfer Learning (TL):** Similar to CL, Transfer Learning (TL) is also a *pretraining* strategy. However, unlike CL, in TL the pretraining is typically done using an additional source dataset, whereas CL has no access to extra data.
- **Continual Learning (ContL):** Similar to a curriculum in input space, ContL does not train the model on the whole target dataset at once, but rather subsets of the data are presented at different points in time. However, unlike CL, the model is forced to see different slices of the data at different points in time in a way that not controlled by the model designer or by a curriculum. Instead, the data flow is controlled by the outside world, and thus it may not be sorted by difficulty. Additionally, the most successful curricula in input space do not slice the data in non-overlapping batches. As training progresses, they typically allow the model to sample harder and harder examples, while always having access to the easier ones. This is not the case for ContL.

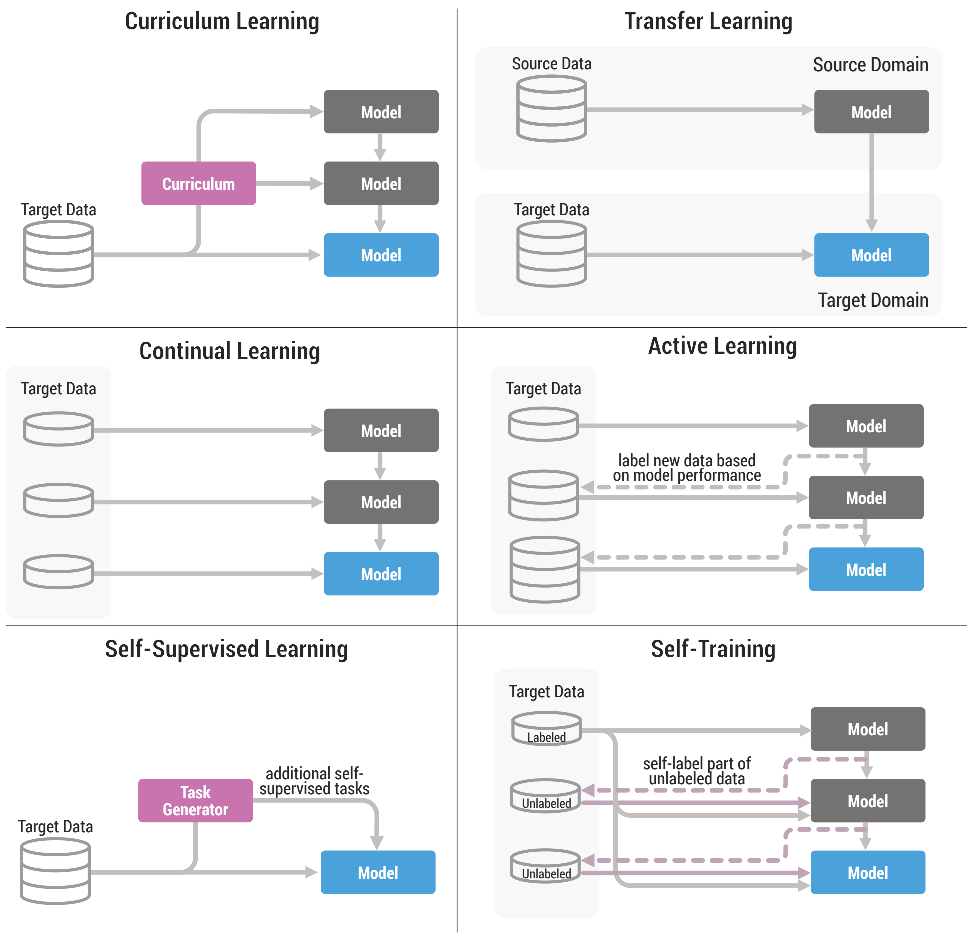


Figure 2.3: A comparison between curriculum learning and other related learning strategies.

- **Active Learning (AL):** Similar to CL, AL also trains a model on incrementally more data. However, the goal of AL is very different. AL typically starts with a few labeled data, and the model asks for labels for the examples that would be most informative to its learning process at that point in time. On the other hand, CL has access to all of the labeled target dataset from the beginning and can plan on how to use it, but it cannot ask for more labels or data.
- **Self-Supervised Learning (SL):** Similar to CL, SL also introduces some auxiliary tasks whose only purpose is to help the target task train better. These tasks are also created from altering the target dataset (e.g. predicting the relative position of image patches in image understanding (Doersch et al., 2015)). However, unlike CL, SL is not concerned with establishing the relative difficulties of these auxiliary tasks and scheduling them from easy to hard.

- **Self-Training (ST):** ST is used in the context of semi-supervised learning, where the model has access to both labeled and unlabeled data. The model is trained on the available labeled data and is then used to make predictions on unlabeled data. Some of these predictions (e.g., the most confident ones) are used to self-label some the unlabeled samples, and the process is repeated. While this bears some similarity to CL, ST does not use the notion of auxiliary tasks, and is not concerned with scheduling tasks or samples by difficulty.

These comparisons are depicted graphically in Figure 2.3.

3 LITERATURE REVIEW

In this chapter, we provide an overview of the field of curriculum learning since its beginnings. We focus on the motivations behind this idea, which started in the field of cognitive neuroscience, as well as on the main curriculum learning methods that have shaped the field. We shape the discussions around the categories of curriculum learning methods identified in Section 2.2. We cover methods and ideas proposed in the past three decades. Since this area now includes hundreds of publications, this survey is not meant to be exhaustive, but rather will focus on the main ideas that have shaped the field. We hope that this survey will inform and inspire future research in curriculum learning and related areas.

This survey consists of the following main components:

- A discussion on cognitive science ideas that have sparked interest in applying human learning strategies to machine learning systems (Section 3.1).
- A description of the curriculum learning methods that have shaped the field and provided inspiration for the approaches that followed, which we group by category: input space (Section 3.2), task space (Section 3.3) and model space (Section 3.4).

Note that this survey does not cover curriculum learning approaches for reinforcement learning (RL), since the work included in the rest of the thesis focuses on supervised and semi-supervised learning. However, curriculum learning was shown to be successful in several RL settings (e.g., Florensa et al., 2017; Narvekar et al., 2017; Svetlik et al., 2017; Fournier et al., 2018; Sukhbaatar et al., 2018). For more on this, we refer the reader an excellent blog post providing an overview of curricula for RL, at Weng (2020).

3.1 THE EARLY DEBATE BETWEEN "LESS IS MORE" AND "LESS IS LESS"

The ideas behind curriculum learning started very early, in the field of cognitive neuroscience, when Newport (1988, 1990) put forward the idea that children are better able to learn languages than adults because they have fewer cognitive resources available to them. This inspired Elman, also a psycholinguist, to test this idea using recurrent neural networks. Inspired by the process of language development in children, Elman (1993) was among of the earliest advocates for the "importance of starting small" when training artificial neural networks, at least as far as language modeling is concerned. In this work, he trained a simplified language model using a recurrent neural network, and showed that the network was only able to learn when the training process was guided using a curriculum. More specifically, this curriculum involved 5 stages, where the early stages train the model only on sentences with a simple grammatical structures, and with every phase, it transitioned towards more complicated ones[1]. In our terminology, this approach contains 5 auxiliary functions, in which the data distribution $P(X)$ is modified to be biased towards easy sentences first and towards difficult sentences later, where the difficulty of a sentence was computed based on a hand-crafted rule.

In contrast to these results, Rohde and Plaut (1999) found using a similar setup that a language model can achieve similar performance as in the case of Elman (1993) without “starting small” in terms of data or memory. In fact, they found that starting small can actually hinder the model performance when the learned language is made more realistic by introducing graded semantic constraints. In a later work (Rohde and Plaut, 2003), the same authors argue against the “less is more” theory proposed by earlier research, and provide counter arguments for why this may not be the case for language modeling. Importantly, while Rohde and Plaut (2003) disagree about the benefits of curriculum learning in language modeling, they do acknowledge that such an approach might be beneficial in other language tasks, such as language comprehension—this was later shown to be true by Bengio et al. (1994), Lin et al. (1998) and others.

In summary, early work on this topic has shown contradictory evidence regarding the benefits of curriculum learning. It is important to note though that most of the early work has focused on various language tasks, and the difficulty of the training examples, as well as the way they were presented to the learner were predefined by the authors.

3.2 CURRICULUM IN INPUT SPACE

This category includes the curriculum learning (CL) strategies that modify the input domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$ of the learning function. These approaches alter the domain by either changing the input space $\mathcal{X}$ in a way that makes the examples "easier" in some sense (e.g., blur the images), or they change the distribution $P(X)$ of the training data (e.g., images that are considered easier have higher probability of being sampled in the beginning of training). Most existing supervised and semi-supervised CL approaches lie in this category.

[1]In the same work, Elman also experimented with increasing the model capacity, which we discuss in Section 3.4

One of the most influential publications on CL in input space, which provided inspiration for the work that followed, was the work of Bengio et al. (2009). Although the idea of "starting small" had been put forward more than a decade before, "curriculum learning" became known under this name and gained attention in the machine learning community with this publication. In this work, Bengio et al. considered a formulation of curriculum learning in which the learning task $\mathcal{T}$ and function f_θ do not change, but the input distribution $P(X)$ does. This is done by starting with a $P(X)$ that puts emphasis only on the easy samples, and slowly transitions towards the target data distribution, allowing the model to train on more and more difficult examples. Their results using this strategy suggested that this type of curriculum regularizes the model, leading to a lower generalization error for the same training error, as well as bringing improvements in convergence speed. The benefits gained with curriculum learning seemed very promising, but in this case both the sample sample difficulties and the way the authors changed $P(X)$ with time was manually decided. Therefore, this left a very important question unanswered: *how can we design curricula for new problems?*

This question has been answered in many different ways by the line of work that followed, which we discuss next. But first, we start by introducing the main components of curricula in input space, which are present in most approaches in this category.

3.2.1 *The Structure of Curricula in Input Space*

The methods proposed by Bengio et al. (2009), as well as the line of work that followed on curriculum methods in input space, consist of two main components:

1. **Difficulty** – each sample is assigned difficulty score, which can be *hand-crafted* or computed *automatically*. Hand-crafted difficulties are defined by humans based on some properties of the data or problem at hand (e.g., shorter sentences are considered easier than long sentences), while automatic difficulties could be based on model performance metrics (e.g., loss function value for that sample).

2. **Schedule** (or **Pacing function**) – the sample difficulties are used to define a schedule which decides when each sample is presented to the learner. Typically, the samples are scheduled from easy to difficult, but the schedule needs to specify more precisely at which iteration and for how long the model can train on a sample. In other words, the schedule specifies the domain $\mathcal{D}$ from which batches of examples are sampled are at every training iteration. The schedule can be *predefined*

before training, or it can be *computed* on the fly and adjusted according to the progress of the learner.

The work that followed proposed multiple ways of defining the sample difficulty and schedule. We further categorize input space approaches based on the level of human-decision making they require, from hand-crafted curricula, to automated curricula and self-paced learning.

3.2.2 *Hand-Crafted Curricula*

Hand-crafted approaches often rely on some property of the data or the specific problem at hand in order to define the sample difficulty scores. Because of this, we will focus the discussion around different application areas and problem settings. We include in this category the approaches that are specific to a single data type (e.g., sentences) or problem setting (e.g., noisy or imbalanced data), and would not be as broadly applicable as the automated approaches discussed later in Section 3.2.3 and Section 3.2.4.

NLP. Hand-crafted difficulty metrics are generally very common in natural language processing (NLP), where intuitive metrics often work well. Some common difficulty metrics include sentence length (Spitkovsky et al., 2010; Platanios et al., 2019), the frequency of words in the training dataset where the samples containing only very common words are easier (Bengio et al., 1994; Platanios et al., 2019), and the specificity of the word semantics from the most semantically generic to the most specific words (Caubrière et al., 2019). Moreover, when applying curriculum learning to domain adaptation for neural machine translation, Zhang et al. (2019) estimate the sample difficulty based on its distance to the in-domain data. In another NLP application, Goldberg and Elhadad (2010) perform dependency parsing according to a curriculum, starting from easy attachment decisions to harder and harder ones, instead of simply going left to right.

COMPUTER VISION. In one of the early attempts to apply curriculum learning to image problems, Bengio et al. (2009) performed a toy experiment where the task was to classify geometric shapes into classes. They used as difficulty the variability of different classes of shapes (i.e., easy classes have less degrees of variability, occupying a small volume in the input space), and showed that a simple 2-stage curriculum using this difficulty criteria was able to generalize better. More recent approaches attempted to discover curriculum learning approaches that can improve performance on common benchmark datasets, such as CIFAR-10 or CIFAR-100 (Krizhevsky, Hinton, et al., 2009). Wu et al. (2021) performed an extensive experimental analysis, considering 3 types of sample difficulty metrics: loss value of a reference model on each sample, learned epoch/iteration, and c-score (Jiang et al., 2020). However, their analysis concluded that curriculum learning provides benefits only in the presence of noisy labels or when the budget of training iterations is limited, and not in standard training. On the other hand, other approaches applied to different computer vision problems report more success with curriculum learning. For instance, Ionescu et al. (2016) used the human response time at solving a visual search task as sample difficulty metric in weakly supervised object localization. This approach requires extra efforts and costs for collecting human annotations, and so Li et al. (2017b) proposed to use the agreement between segmentation masks and detection bounding boxes. Furthermore, Wang et al. (2018) used curriculum learning when training a ranking model for person re-identification, where they prioritize negative samples based on their distances to the anchor when using a triplet loss. Other examples of computer vision applications where curriculum learning has been successfully applied are medical images (e.g., Tang et al., 2018; Oksuz et al., 2019) and visual concept learning (Li et al., 2020).

NOISY DATA. Curriculum learning has also proved to be useful in noisy data settings. For example, Lotfian and Busso (2019) improved the performance of a speech emotion recognition system with a curriculum that uses the disagreement between human evaluators as measure of difficulty. This relied on the intuition that samples that are ambiguous for humans are also ambiguous for the model. Along similar lines, Jiang et al. (2018) used a curriculum that first focuses samples that are likely to be correctly labeled. This showed improved performance for image classification problems with noisy labels. Another similar approach was proposed by Guo et al. (2018b). The applications mentioned above considered *noise in the label space*. However, Braun et al. (2017) successfully used a curriculum when dealing with *noise in the input space*. They improved the performance of automatic speech recognition systems using a simple multi-stage curriculum using the signal-to-noise ratio of the input as difficulty.

IMBALANCED DATA. Wang et al. (2019b) proposed a curriculum learning approach for handling classification problems with imbalanced data. Their approach alters the data distribution from imbalanced to balanced, improving the accuracy on two attribute analysis datasets, CelebA (Liu et al., 2015) and RAP (Li et al., 2016a).

All in all, this variety of curriculum learning methods and applications showcases how curriculum learning can have a broad impact across many areas of machine learning, and how intuitive difficulty metrics often work well. Of course, ideally we should not rely on human intuition for defining the curriculum, both because it may be hard to propose relevant difficulty metrics for certain applications, and because difficulties based on human intuition may not be optimal. Thus, we next discuss approaches that attempt to define the curriculum automatically.

3.2.3 *Automated Curricula*

Automated curriculum learning methods aim to *learn* a curriculum along with the model parameters, which would require less human decision-making. There have been efforts to automate both the choice of difficulty metric and the pacing function, in a way that makes them more generally applicable to more problem settings and data types.

In one line of work, Graves et al. (2017) introduced a non-stationary multi-armed bandit algorithm for determining the curriculum. The algorithm is provided with signals about the learning efficiency of the model (e.g., the rate of increase in prediction accuracy, the rate of increase in network complexity), and predicts the distribution of the data from which to sample the next batch of training samples. This strategy was able to accelerate learning for a language modeling problem and the bAbI dataset (Weston et al., 2016).

A common strategy for designing automated curriculum methods is using a *teacher-student* architecture. Such approaches formulate learning as an interaction between a "student" model and a "teacher" model. The student is the model whose parameters we aim to optimize, and the teacher is an auxiliary network that assists the student network in order to improve its learning process. In terms of our notation in Section 2.1, the teacher is a trainable model that outputs the training data distribution $P(X)$ for the student model at each step during the student's training process. For example, Matiisen et al. (2019) proposed a teacher-student approach, in which a teacher model gives tasks to a student model, while also learning what makes makes the student learn better. The tasks were prioritized such that the student model would make the fastest progress (i.e., those tasks with the steepest learning curve), while at the same time the student revisited already learned tasks that were being forgotten (i.e. those whose learning curve slope is negative). The framework was formulated as a partially observable Markov decision process (Kaelbling et al., 1998), where the reward is the change in a performance criterion (i.e., the improvement in learner's performance) from one step to the next. This method was applied on two different kinds of applications: learning to add decimal numbers with LSTMs (Hochreiter and Schmidhuber, 1997) and learning to navigate in a Minecraft environment. The tasks from which the teacher had to choose were manually defined: in the addition experiment, the tasks correspond to the number of digits in the operands that the student needs to learn to add; in the Minecraft experiment, the tasks are different mazes in which the agent needs to learn to navigate. Along similar lines, Fan et al. (2018) proposed a teacher-student method in which a reinforcement learning agent teacher decides the data, loss function, and hypothesis space to train a student model. The teacher is updated using reward signals from the student (e.g., the accuracy on a validation set). This strategy was shown to achieve almost the same accuracy as a baseline without curriculum on benchmark image datasets, using less training data and fewer training iterations. Moreover, the work of Jiang et al. (2018), which discussed earlier in the context of noisy data, is also a teacher-student approach where the teacher provides the sample weights for training a student model.

It is important to note that the teacher-student curriculum learning methods are closely related to "machine teaching" (Zhu, 2015; Liu et al., 2017; Zhu et al., 2018), which also relies on the interaction between a teacher and a student model, but goal is different. In machine teaching, the goal is generally to select a minimal subset of training data (or other measures of "teaching effort") that enough to train an accurate student model fast. In such cases, there is often a trade-off between student performance and teaching effort. Moreover, a common assumption in machine teaching is that the teacher already knows the optimal model parameters, and tries to guide the student guide the student towards these *by examples*. However, in curriculum learning the teacher does not know the optimal solution, and the cost associated to using all the training data not considered. Instead, the emphasis lies on how to schedule the training data to improve the performance of the learner.

3.2.4 *Self-Paced Learning*

Self-Paced Learning (SPL) is a line of work inspired by curriculum learning, but which challenges the idea that sample are universally "easy" or "hard". Instead, in SPL the difficulty of the sample depends on the model chosen to solve this task and also, importantly, on the current knowledge of the learner. Thus, the curriculum should not be defined a priori before starting to train the model, but it should adapt to the current capabilities of the model. This direction has become known as *"self-paced learning"* (SPL), after the work of Kumar et al. (2010).

Since this work has been the inspiration for a long series of self-paced learning methods, we discuss the proposed idea in more detail. To allow the model to choose its own "pace", Kumar et al. (2010) considered that the difficulty of the samples at a particular step during training should be given by the how difficult it is for the model to predict its target output correctly at that point during training. Concretely, the difficulty of a training example i at a particular step t during training is defined as the loss value for sample i, using the current estimation of the function parameters θ (e.g., this could be the mean squared error between the predicted value and the true value). This idea is formulated using a latent variable model whose parameters are optimized jointly with the curriculum using the following mixed-integer program:

$$(\theta^*, \nu^*) = \underset{\theta \in \mathbb{R}^d, \nu \in \{0,1\}^N}{\arg\min} \left(r(\theta) + \sum_{i=1}^{N} \nu_i \mathcal{L}(f(x_i, y_i; \theta), y_i) - \frac{1}{K} \sum_{i=1}^{N} \nu_i \right) \quad (3.1)$$

where f, x, y, N and θ are defined in Section 2.1, $\mathcal{L}$ is a loss function, $r(.)$ is a regularization function, K is a weight that determines the number of samples to be considered at the current training step, and ν_i is a variable that specifies whether sample i is easy enough to be included in training at step t. The optimization problem is solved using an alternative search strategy. In a first stage, the parameters θ are fixed and Equation 3.1 is solved for ν. The solution can be computed in closed form: $\nu_i = \delta(f(x_i, y_i; \theta) < \frac{1}{K})$, where $\delta(.)$ is the indicator function. In the second stage, ν is fixed and Equation 3.1 is solved for θ. The value of K is decreased with every iteration, such that when K approaches 0, all samples are included in training. In terms of our definitions from Section 2.1, Kumar et al.'s approach to self-paced learning modifies the training data domain by altering $P(X)$ with every iteration of training. At every iteration, $P(X)$ assigns probability 0 to the samples considered difficult, and distributes the probability mass uniformly among the samples considered easy. By increasing K iteratively, $P(X)$ approaches a uniform distribution. The authors tested their approach using a latent structural Support Vector Machines (SVM) model on four types of problems (object localization, noun phrase co-reference, motif finding and handwritten digit recognition), and showed improvements in the final performance and the number of iterations it took the model to reach that performance.

An important observation is that in self-paced learning, the notion of sample difficulty is dynamic (i.e., it is not fixed at the beginning of training, but it changes with time) and is in some sense independent of the problem at hand (i.e., signals such prediction confidence or loss do not depend on the nature of the data). However, the way in which the sample difficulties are used is more or less predefined, and is not adjusted depending on the progress of the learner.

Building on the work of Kumar et al. (2010), Jiang et al. (2014b) proposed an improvement to self-paced learning based on the intuition that the curriculum should not only prioritize easy samples, but it should also ensure some amount of "diversity" in the examples presented to the learner. This is implemented using an additional regularization term in the loss function that encourages the curriculum to select samples that belong to different clusters. The sample clustering is done one before training, using any clustering algorithm (in the paper, the authors use k-means and spectral clustering). The proposed approach was evaluated on three datasets: a multimedia event detection dataset and two video action recognition datasets. The results reported by Jiang et al. (2014b) show improvements over the SPL approach of Kumar et al. (2010) in terms of final performance.

Moreover, Jiang et al. (2015) pointed out that both hand-crafted curricula and SPL have their advantages and disadvantages. On one hand, a hand-crafted curriculum (e.g. using sentence length as difficulty score) is rigid and does not adapt to the current knowledge of the learner, but it allows us to incorporate domain knowledge into the curriculum. On the other hand, SPL is adaptive to the progress of the learner, but under the formulations of Kumar et al. (2010) and Jiang et al. (2014b), it does not provide a way to utilize any domain knowledge. To mitigate these issues, Jiang et al. (2015) proposed a model that combines the benefits of the two approaches, by modifying the objective function in Equation 3.1 to incorporate prior knowledge. This method was tested on a matrix factorization dataset and a multimedia event detection dataset, and showed boosts in performance and convergence speed compared to the baseline curriculum learning and SPL methods.

Several other approaches have been proposed, which we describe briefly. Li et al. (2016b) aimed to overcome the sensitivity to initialization of other SPL methods by introducing a multi-objective formulation, and using an evolutionary algorithm to optimize the two objectives simultaneously. Zhang et al. (2015) proposed a modification to the SPL with diversity method of Jiang et al. (2014b) by using a norm that is easier to optimize, and applied it to co-saliency detection. Avramova (2015) provided some interesting insights into the types of samples that a model finds easy or hard as training progresses, when training a convolutional neural network using SLP or SPL with diversity. Fan et al. (2017) pointed out a limitation of the SPL approaches described so far. All these approaches can expressed using the following common formulation:

$$(\theta^*, v^*) = \operatorname*{arg\,min}_{\theta \in \mathbb{R}^d, v \in \{0,1\}^N} \left(r(\theta) + \sum_{i=1}^{N} v_i \mathcal{L}(f(x_i, y_i; \theta), y_i) + g(\lambda, v_i) \right) \tag{3.2}$$

where $g(\lambda, v_i)$ that depends on the specific SPL method of choice, λ is a parameter that controls the learning pace (corresponding to $\frac{1}{K}$ in Equation 3.1), and the rest of the terms have the same interpretation as in Equation 3.1. As Fan et al. (2017) pointed out, the specific form of the regularizer $g(\lambda, v_i)$ needs to be specified *explicitly*, and choosing it manually might result in sub-optimal results. Thus, Fan et al. (2017) proposed an implicit regularizer whose minimizer function can be derived based on convex conjugacy, using the dual of a robust loss function. For details, we refer the reader to the original publication. Results on three types of problems (matrix factorization, clustering and classification) show improvements in performance compared to SPL methods that use explicit regularizers.

Inspired by the SPL methods described so far, several other approaches have been introduced that adapt these techniques to specific problem settings. For example, Zhao et al. (2015) adapted the SPL method proposed by Kumar et al. (2010) for the task of matrix factorization. This idea was inspired by Kumar et al. (2010), but the authors proposed a different version of the self-paced regularizer $g(.)$ in Equation 3.2, which allows the sample weights to take "soft" (continuous) values, as opposed to the "hard" (binary) sample weighting used by Kumar et al. (2010).

Similarly, Xu et al. (2015) also proposed a soft SPL approach for a different application, multi-view clustering. In this setting, each sample is represented though multiple sets of features (i.e. views), but the goal is to create a clustering that is unique across all views. In this case, Xu et al. (2015) assigned the notions of *easy* and *hard* not just to samples, but also to views. Moreover, they also redefined the regularizer $g(.)$ in Equation 3.2 to allow for "soft" sample weights.

Along the same lines, Jiang et al. (2014a) also introduced a modification of the SPL method of Kumar et al. (2010), adapted for multimedia retrieval. Their approach focused on the task of reranking search results using multiple data modalities. Therefore, SPL was adapted to weigh not only samples, but also different modalities. Moreover, similar to the approaches before, Jiang et al. (2014a) also proposed a "soft" SPL version.

Ma et al. (2017) proposed a SPL strategy for co-training. In this setting, two classifiers are trained on different views of the same data and provide each other labels for the unlabeled data. Similar to the SPL of Kumar et al. (2010), the training examples are allowed in the training process based on their current loss. Each classifier has its own binary vector indicating which samples are easy enough to be currently considered for training (i.e., those for which the corresponding loss value is lower than a threshold), but the two vectors are encouraged to be similar through an additional loss term added to the objective function.

Other more applied approaches propose more problem-specific versions of SPL. For example Sangineto et al. (2019), Supancic and Ramanan (2013), and Lee and Grauman (2011) defined sample difficulty measures that depend on image properties relevant to the problem at hand (e.g., an image region's difficulty score depends on the predictions made for the regions around it), but these methods are self-paced in the sense that the model trained so far is used to re-estimate the sample difficulties during training.

3.2.5 *Comparison*

Given the multitude of curriculum learning approaches in input space, it would be useful to compare and understand the pros and cons of these methods, in order to understand which one to use in different settings. While we do not run experiments to conduct such a comparison in this survey, we refer the reader to a couple of publications that have conducted such analyses.

Collier and Beel (2018) provide an empirical comparison of several curriculum learning schedules applied to training a LSTM (Hochreiter and Schmidhuber, 1997) on three synthetic sequence learning tasks: (1) Copy – the LSTM needs to memorize a sequence and retrieve it from memory; (2) Repeat Copy – same as Copy but the sequence has to be repeated k times, where k is specified at the end of the input; and (3) Associative Recall – the model needs to retrieve an item, conditioned on a provided query. The curriculum is defined as a time-varying distribution over the training data. In other words, every phase of the curriculum changes $P(X)$ such that every phase only considers sequences of a specific length, which grows with time. The tested curricula include: Naive (the model is shown only examples from the current domain), Look Back (the model is shown only examples from the current and past domains), Look Back and Forward (the model is shown only examples from past, current and future domains, with more weight in the current domain), None (no curriculum), Uniform (sample uniformly from all domains), and Prediction Gain (as defined by Graves et al. (2017)). For details on the problem setting and the curricula, we refer the readers to Collier and Beel (2018). Their analysis found that including examples from previous domains (i.e. shorter sequences) is critical in preventing catastrophic forgetting, while including examples from future domain (i.e. longer sequences) is also helpful in increasing convergence speed. Prediction Gain performed competitively to the Look Back and Forward approach.

Moreover, Cirik et al. (2016) compare the effect of curriculum learning on the LSTM internal representation on two sequence prediction tasks: a digit summation task and a sentiment analysis task. In the first task, the LSTM is presented with a sequence of digits as input, and it needs to learn to add them and output their sum. The second is a sentiment classification task. In all cases, the sample difficulty is defined as the length of the input sentence. Three curricula are considered, corresponding to the Naive and Look Back approaches of Collier and Beel (2018), as well as a curriculum where each epoch visits all the examples in the order of difficulty. The results suggest that curriculum learning is not always beneficial, and it seems to bring some advantages compared an LSTM trained without curriculum only in certain conditions: when the training data is limited, or when the model capacity (i.e., here the hidden size of the LSTM) is low. Out of the considered curricula, the Look Back approach works best, whereas the Naive approach even hurts performance for a large model and larger dataset. An interesting result is shown by observing the predicted sentiment as each word of an input sentence is presented to the model sequentially. The predictions of the Look Back curriculum are the most consistent with what a human would predict.

More recently, Wu et al. (2021) tried to answer the question "When do curricula work?" and conducted an extensive experimental analysis comparing different forms of curriculum, anti-curriculum, and random-curriculum. Their analysis performed on image classification datasets (e.g., CIFAR-10, CIFAR-100 (Krizhevsky, Hinton, et al., 2009)) concluded that "curriculum, but not anti-curriculum can indeed improve the performance either with limited training time budget or in existence of noisy data". Note that this is not necessarily the case in our later experiments in Chapter 4 where we apply curriculum learning to sequential data.

3.3 CURRICULUM IN TASK SPACE

Curriculum learning in task space refers to those approaches that operate on the learning task $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. In this section, we consider only methods targeted at learning a single task by modifying $\mathcal{T}$. We also discuss the multitask learning case in Chapter 5, where the goal is to train a system to perform well on multiple such $\mathcal{T}_1$, $\mathcal{T}_2$, etc.

Perhaps surprisingly, this category contains only a handful of methods. Saxena et al. (2019) proposed an approach targeted at classification problems, in which they introduce a class-specific parameter for each class in the dataset. The class parameters are learned together with the model parameters. When making predictions on a particular sample during training, the predicted logits are scaled by the corresponding class parameter. This has the effect of decaying the gradient with respect to the misclassified classed, thereby accelerating learning. They applied their method to image classification and object detection, reporting improvements in accuracy especially when the labels are noisy.

Moreover, Dogan et al. (2020) also proposed an approach targeted at classification problems, which is based on the label similarity. During training, the class labels that the model is supervised with are modified: instead of using a 1-hot vector for the true class, they use a probability distribution over classes, assigning a non-zero probability weight to classes that are similar to the true class. The label similarities come from prior knowledge (e.g., when the labels are word, they use the distance in word embeddings as similarity). As training progresses, the target distribution is shifted towards the 1-hot encoding of the correct label. This method is also evaluated on image classification tasks and reports accuracy improvements, especially in the low data regime.

Along similar lines, Ganesh and Corso (2020) propose another curriculum approach in the label space of a classification method. This method incrementally increases the number labels considered by the classifier, while always using the entire dataset during training. This is achieved by replacing the original correct label with a pseudo-label that is shared by multiple classes. As the model is being trained, the correct labels are incrementally revealed, finally converging to the original label distribution. The approach is also evaluated on image classification datasets.

Aside from these approaches, we also propose other curriculum learning methods in task space in Chapter 5.

3.4 curriculum in model space

This type of curriculum is different than the approaches described earlier, since the inputs and targets of the model stay fixed throughout learning. The change brought about by the curriculum takes place in *model space*, meaning that the auxiliary functions $f_1, f_2, ..., f_K$ defined in Section 2.1 modify the architecture of the model, while, in this case, the tasks $\mathcal{T}_1 = \mathcal{T}_2 = ... = \mathcal{T}_K$ and the input domains $\mathcal{D}_1 = \mathcal{D}_2 = ... = \mathcal{D}_K$ are left unchanged (unless we combine it with other forms of curriculum learning). Curricula in model space are perhaps the sparsest category of curriculum learning methods, with only a handful of publications. However, other learning strategies, that were not originally intended as curriculum learning, in fact train a model in order of difficulty, and thus can also be seen as forms of curriculum learning in model space. We review these methods next.

In the same seminal work of Elman (1993) discussed in Section 3.2, the author proposes another approach of aligning learning in humans with learning in machines. It is well known in neuroscience that in the early stages of development, children have limited memory and attention span, which then grow with time as the brain develops (Kail, 1990). Elman tried to replicate this process with recurrent neural networks (RNN) by limiting the amount of memory the network has in the early stages of training, and allowing it to become larger and larger with every training phase. This was done by limiting the RNN to propagate information only from the previous 3-4 words initially, then 4-5 words, and so on. The results showed that the network trained in this manner performed better than the one trained without a curriculum, and on par with the input space strategy described in Section 3.2.

More recently, Sinha et al. (2020) proposed a curriculum learning method in model space targeted at convolutional neural networks (CNNs). Their work is motivated by the observation that CNNs are biased towards extracting high-frequency (texture) information, and often fail to take advantage of the low-frequency (shape) information (Geirhos et al., 2019). Thus, Sinha et al. (2020) propose a curriculum learning approach that controls the amount of texture information that the model is allowed to represent during training, by convolving the output of each CNN layer with a Gaussian kernel, which acts like a low-pass filter. The standard deviation of the Gaussian kernel is reduced as training progresses, thus allowing more and more texture information to pass through. Their results show improvements in accuracy on several image datasets and several CNN architectures.

Moreover, Morerio et al. (2017) propose a different take on curriculum learning in model space: they modify the amount of dropout applied to the model during training according to a curriculum, increasing the amount of neurons that are dropped out layer-wise. This method is shown to improve performance on standard image classification datasets. This strategy can also been interpreted as a curriculum in model space, since dropout essentially changes the function f_θ being learned.

There are also other areas in machine learning which can be seen as versions of curriculum learning in model space. For example, continuation methods, also known as graduated optimization (Hazan et al., 2016), incrementally change the loss function during training, in a smooth-to-sharp manner. Smoothing the loss landscape early on during training aims to facilitate the learning of highly non-convex loss functions. Another type of category of methods related to curricula in model space are the layer-wise pretraining strategies in deep learning (Bengio et al., 2007). Neural networks that are very deep often benefit from training layer-by-layer, by starting with a small number of hidden layers and successively adding a new layers and refitting the model. This can be seen as a curriculum in model space, because the learning function f_θ is progressively changed during training.

4 CURRICULUM IN INPUT SPACE

As discussed in Section 2.2, one of the main categories of curriculum learning approaches consists of methods that modify the domain of the inputs presented to the model during training. This is most commonly done by changing the distribution of the training data, such that examples that are considered easier have higher probability of being sampled in the beginning of training. Most existing CL methods for supervised learning lie in this category, and some representative examples include the work of Bengio et al. (2009), Kumar et al. (2010), Spitkovsky et al. (2010), Jiang et al. (2015), Graves et al. (2017), Jiang et al. (2018), Zhou and Bilmes (2018), and Platanios et al. (2019).

In order to prioritize the samples by difficulty as training progresses, these approaches typically consist of two main components:

– Difficulty Function: each method needs to provide a means of assigning difficulty scores to each training sample. This can be *hand-crafted* or *computed automatically*. Hand-crafted difficulty functions are designed by humans based on some properties of the data (e.g., shorter sentences are considered easier than long sentences), while automatically computed difficulties could be based on model performance metrics (e.g., loss function value for that sample).

– Pacing Function (i.e., Schedule): the sample difficulties are used to define a schedule which decides when each sample is presented to the learner. Typically, the samples are scheduled from easy to difficult, but the pacing function needs to specify at which training step and for how long the model can train on a sample. In other words, the pacing function specifies the domain $\mathcal{D}$ from which batches of examples are sampled at each training iteration. The schedule can be *predefined* before training, or it can be *computed* on the fly and adjusted according to the progress of the learner.

Existing approaches cover multiple ways of defining the sample difficulties and pacing function.s However, despite the progress made in the literature in the last few years, defining good difficulty measures that work across different data domains is still an open question. We covered some of the most popular approaches in Section 3.2. However, for the applications included in this chapter, we found that intuitive hand-crafted difficulty measures in fact work very well. For example, sequence length worked well across all applications where the inputs are sequences (e.g., natural language sentences, sequences of digits for learning arithmetic, etc.). In Chapter 6 we provide an explanation for why this is the case. On the other hand, what had a big impact on our results was the way we *processed* the difficulties (from the original values which may be discrete and not evenly spaced), as well as the pacing function that decides when to show samples of a particular difficulty to the model. In the next section we propose a new *generic* curriculum learning framework, that can take any measures of sample difficulty (e.g., sentence length, number of objects in image, etc.) and provide a schedule for the model trainer.

We successfully applied the proposed framework to multiple applications, on various types of data. In Sections 4.2, 4.3, 4.4, and 4.5 we discuss these applications, along with their corresponding sample difficulties. We consider incrementally more difficult problems, starting with a few simple synthetic problems, such as learning to add two numbers using a recurrent neural network, and leading to some real-world applications such as neural machine translation using Transformers (Vaswani et al., 2017). Table 4.1 provides a summary of our applications and their corresponding curricula.

Application	**Models**	**Difficulty Function**	**Pacing Function**
Addition digit-by-digit	RNNs	Number of digits	Square Root
Learning arithmetic with seq2seq models	seq2seq LSTMs, Transformer	Sequence Length	Square Root
Neural Machine Translation	seq2seq LSTM, Transformer	Sentence Rarity, Sentence Length	Linear, Square Root
Multimodal Language Understanding	CNN-LSTM	Number of Objects in Image	Square Root

Table 4.1: Overview of our applications using input-space curricula, and their corresponding difficulty and pacing functions.

Note that some of these applications will be recurring in this thesis. We also use them when discussing curricula in task space in Chapter 5, and in our analysis of curriculum learning in Chapter 6.

4.1 A GENERIC CURRICULUM LEARNING FRAMEWORK

We propose *competence-based curriculum learning*, a training framework based on the idea that training algorithms can perform better if training data is presented in a way that picks examples appropriate for the model's current *competence*. This algorithm was first introduced in our NAACL paper (Platanios et al., 2018), which was tackling the topic of machine translation, but the framework itself is *generic*, meaning that it can be applied to any kind of data, as long as we can define a difficulty measure for each training sample. For ease of discussion, when introducing our framework, we use sequence length as difficulty metric, but the method does not depend on it.

Our approach is based on two key concepts:

– Difficulty: A value that represents the difficulty of a training sample. For example, sentence length is an intuitive difficulty metric for natural language processing tasks. However, the difficulty scores can take any values; the only constraint is that they are comparable across different training samples (i.e., the training samples can be ranked according to their difficulty).

– Competence: A value between 0 and 1 that represents the progress of a learner during its training. It is defined as a function of the learner's state. More specifically, we define the competence $c(t)$ of a learner at training step t as the proportion of training data it is allowed to use at that time. The training examples are ranked according to their difficulty and the learner is only allowed to use the top $c(t)$ ratio of them at step t.

They key idea of our approach is based on the balance between the difficulty and competence: at every step during training, an example from the dataset can be shown to the model *only if the difficulty of the sample is less than the competence of the model*. More precisely, we propose the following algorithm, which we refer to as *competence-based curriculum learning*. At each training step:

(i) the current competence of the model is computed,

(ii) a batch of training examples is sampled uniformly from all training examples whose difficulty is lower than that competence.

Note that, at each training step, we are not changing the relative probability of each training sample under the input data distribution, but we are rather constraining the domain of that distribution, based on the current competence of the learner. Eventually, once the competence becomes 1, the training process becomes equivalent to that without using a curriculum,

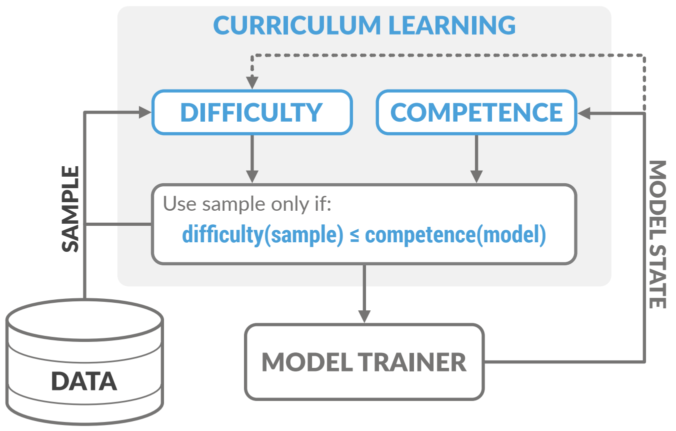


Figure 4.1: Overview of the proposed curriculum learning framework. During training, the *difficulty* of each training sample is estimated, and a decision whether to use it is made based on the current *competence* of the model.

with the main difference being that the learner should now be more capable to learn from the more difficult examples. A high-level overview of this algorithm is illustrated in Figure 4.1, an example visualization of the first two steps is shown in Figure 4.3, and an example of the interaction between difficulty and competence is shown in Figure 4.2.

There are two decisions that still need to be made:

1. If the difficulty is a hand-crafted metric, how do we make sure the difficulty and competence are comparable?

2. How do we set the competence per training step?

We address the first question by converting the sample difficulties to the $[0, 1]$ spectrum, in a way that is compatible with the competence. We describe how we can convert an arbitrary difficulty measure in Section 4.1.1. As we mentioned earlier, the competence is already defined between $[0, 1]$, but we still need to define how the competence changes with time during training. We propose several pacing functions in Section 4.1.2. Finally, we put all the pieces together and present our full algorithm in Section 4.1.3.

4.1.1 *How can we make sample difficulties comparable to model competence?*

As described earlier, the competence refers to the ratio of the training data that the learner is allowed to use at a particular training step. Of course, from a curriculum learning perspective, a percentage of x% of data will refer to the *easiest* x% of the training set. As such, we convert the sample difficulties from arbitrary values (e.g., sentence lengths, number of objects in image) to their corresponding percentiles—numbers between 0 and 1 which represent the *relative* difficulty of that sample compared to all other samples in the dataset.

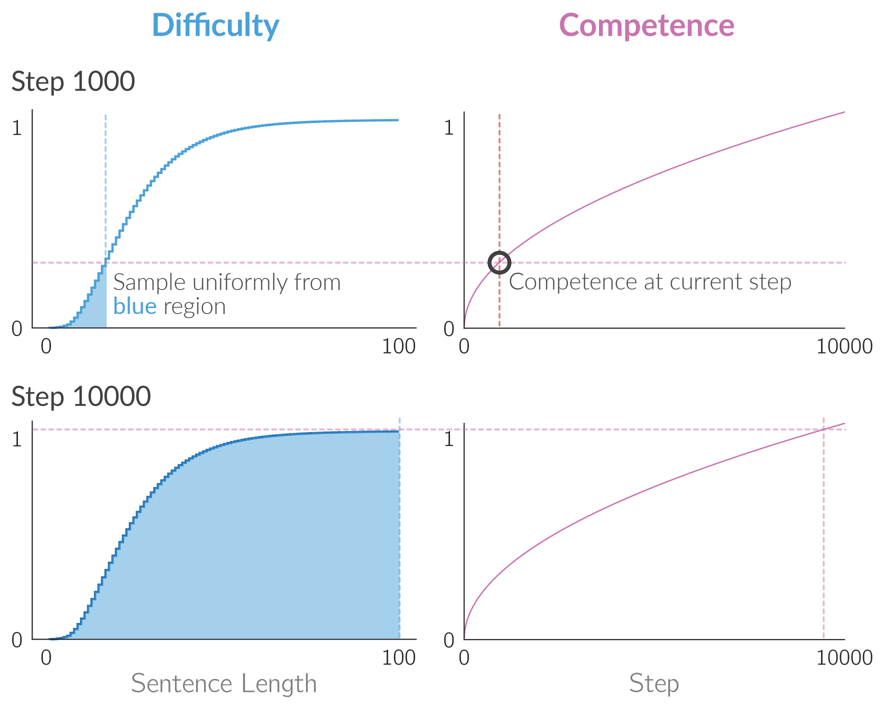


Figure 4.2: Illustration of the training data "filtering" performed by our algorithm. To showcase this idea, here we used the preprocessed data from our machine translation experiments in Section 4.4, with sentence length as measure of difficulty.

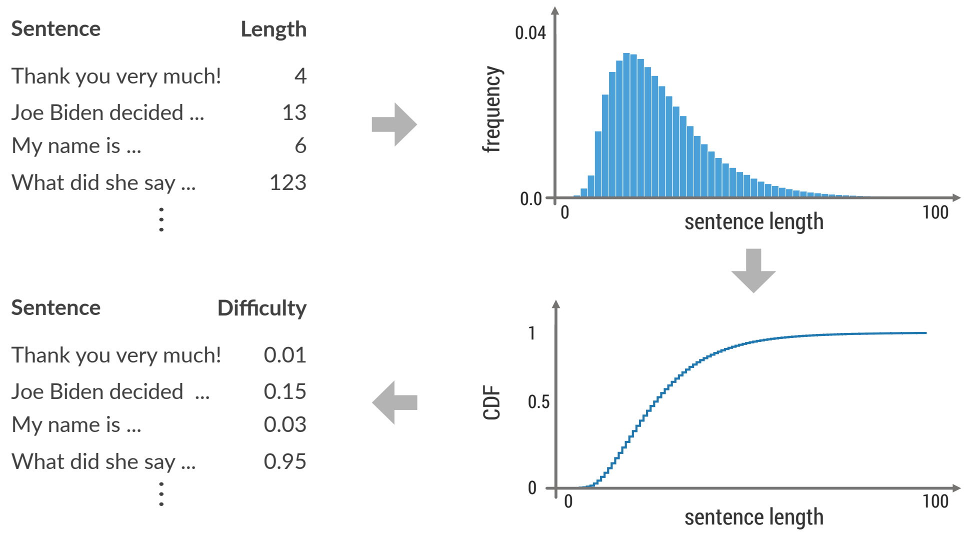
We do this using using the strategy displayed in Figure 4.3, where we first calculate the histogram for all difficulty values. Then, we convert the histogram into a cumulative distribution function (CDF), and replace each difficulty with its corresponding position in the CDF.

4.1.2 *How do we define the model competence?*

A competence function dictates how the competence of the model, $c(t)$, changes with time, which reflects how fast we introduce more and more difficult examples to the learner. In this work, we propose two simple functional forms for $c(t)$ and justify them with some intuition. More sophisticated strategies that depend on the loss function, the gradient, or on the learner's performance on held-out data, are possible, but we do not consider them in this line of work.

LINEAR: This is a simple way to define $c(t)$. Given an initial value $c_0 \triangleq c(0) \geqslant 0$ and a slope parameter r, we define:

$$c(t) \triangleq \min(1, tr + c_0) \tag{4.1}$$



Sentence	Length
Thank you very much!	4
Joe Biden decided ...	13
My name is ...	6
What did she say ...	123
⋮	

Sentence	Difficulty
Thank you very much!	0.01
Joe Biden decided ...	0.15
My name is ...	0.03
What did she say ...	0.95
⋮	

Figure 4.3: Visualization of the difficulty preprocessing strategy of our algorithm. Here we use an as example sentence length as the difficulty function. "CDF" stands for the empirical "cumulative density function" obtained from the histogram on the top.

In this case, new training examples are constantly being introduced during the training process, with a constant rate r (as a proportion of the total number of available training examples). Note that we can also define $r = (1 - c_0)/T$, where T denotes the time after which the learner is fully competent, which results in:

$$c_{\text{linear}}(t) \triangleq \min\left(1, t\frac{1 - c_0}{T} + c_0\right). \tag{4.2}$$

ROOT: In the case of the linear form, the same number of new and more difficult, examples are added to the training set, at all times t. However, as the training data grows in size, it gets less likely that any single data example will be sampled in a training batch. Thus, given that the newly added examples are less likely to be sampled, we propose to reduce the number of new training examples per unit time as training progresses to give the learner sufficient time to assimilate their information content. More specifically, we define the rate in which new examples are added as inversely proportional to the current training data size:

$$\frac{dc(t)}{dt} = \frac{P}{c(t)}, \tag{4.3}$$

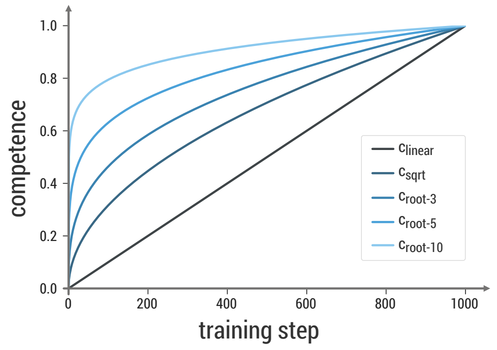


Figure 4.4: Examples of various competence functions with initial competence value $c_0 = 0.01$ and total curriculum duration $T = 1,000$.

for some constant $P \geqslant 0$. Solving this differential equation, we obtain:

$$\int c(t)dc(t) = \int P dt \Rightarrow c(t) = \sqrt{2Pt + D}, \tag{4.4}$$

for some constants P and D. Then, we consider the following constraint: $c_0 \triangleq c(0) = \sqrt{D} \Rightarrow D = c_0^2$. Finally, we also have that $c(T) = 1 \Rightarrow P = (1 - c_0^2)/2T$, where T denotes the time after which the learner is fully competent. This, along with the constraint that $c(t) \in [0, 1]$ for all $t \geqslant 0$, results in the following definition:

$$c_{\text{sqrt}}(t) \triangleq \min\left(1, \sqrt{t\frac{1-c_0^2}{T} + c_0^2}\right). \tag{4.5}$$

In our experiments, we refer to this specific formulation as the "square root" competence model. If we want to make the curve *sharper*, meaning that even more time is spent per sample added later on in training, then we can consider the following more general form, for $p \geqslant 1$:

$$c_{\text{root-p}}(t) \triangleq \min\left(1, \sqrt[p]{t\frac{1-c_0^p}{T} + c_0^p}\right). \tag{4.6}$$

We observed that best performance is obtained when $p = 2$ and then, as we increase the value of p, the performance converges to that obtained when training without a curriculum. Plots of the presented competence functions are shown in Figure 4.4.

4.1.3 *Algorithm*

Putting the pieces together, Algorithm 4.1 presents our entire method, end-to-end. In the next sections we show applications of this algorithms to various machine learning problems, and using different data modalities.

Algorithm 4.1: Competence-based Curriculum Learning

```
Inputs: Dataset, $\mathcal{D} = \{s_i\}_{i=1}^N$, consisting of N examples.
        Model trainer, that uses batches of training data for each update.
        Difficulty scoring function, d.
        Competence function, c.
1 Compute the difficulty, $d(s_i)$, for each $s_i \in \mathcal{D}$.
2 Compute the cumulative density function of the difficulty scores.
    This results in a relative difficulty score per example, $\bar{d}(s_i) \in [0, 1]$.
    Illustrated in Figure 4.3.
3 for training step $t = 1, \ldots$ do
4     Compute the model competence, $c(t)$.
5     Sample a data batch $B_t$ uniformly from all $s_i \in \mathcal{D}$, such that
        $\bar{d}(s_i) \leqslant c(t)$. Illustrated in Figure 4.2.
6     Invoke the trainer using $B_t$ as input.
Output: Trained model.
```

4.2 ADDITION DIGIT-BY-DIGIT WITH RECURRENT NEURAL NETWORKS

For our first application, we attempt a simple synthetic experiment: we teach a neural network how to add two numbers, digit by digit. While this problem in itself is not of practical interest, it provides an excellent setting for studying curriculum learning, for multiple reasons:

- We can generate arbitrarily large datasets, since the domain of the inputs is infinite, and we know how to compute the target answer for any pair of inputs. This gives us the opportunity to see how the benefits of curriculum learning vary with the amount of training data.
- Due to the sequential nature of the data, we can test the model's ability to extrapolate beyond the training distribution (with and without curriculum learning), on sequences that are longer than what it has been trained on. This can provide us with insights that are beneficial to a wide range of application areas, from language, signal processing, finance data, or any kind of sequential data.
- We know how humans are taught to do it, which gives us an intuitive difficulty metric. Children are taught to add two numbers starting with numbers with 1 digit, then 2, then 3, and so on. While humans and machine learning models may find different types of examples difficult, trying a length-based curriculum for this setting seems like a reasonable approach to attempt (more on this in Chapter 6).

To allow for numbers of arbitrary length, we use a recurrent neural network architecture, that takes as input at every time step a pair of digits from the two operands—starting with the least significant digits—and outputs the corresponding digit of their sum. The final result consists of concatenating the output of the network at all time points, in reverse. This process is illustrated in Figure 4.5.

4.2.1 *Data*

We generate training samples, each example consisting of two input operands (i.e. the two numbers we want to add), and their sum will constitute the target output. All operands have between $1-5$ digits, and thus their sums have $1-6$ digits, accounting for a possible carry at the most significant digit position. The data we is sampled as follows: (i) First we sample the number of digits for each operand, uniformly from $\{1,2,3,4,5\}$. (ii) For each operand, given a number of digits k, we sample uniformly k digits. This sampling strategy is aimed to help the baseline model trained without curriculum, by making sure that all number lengths are equally represented in

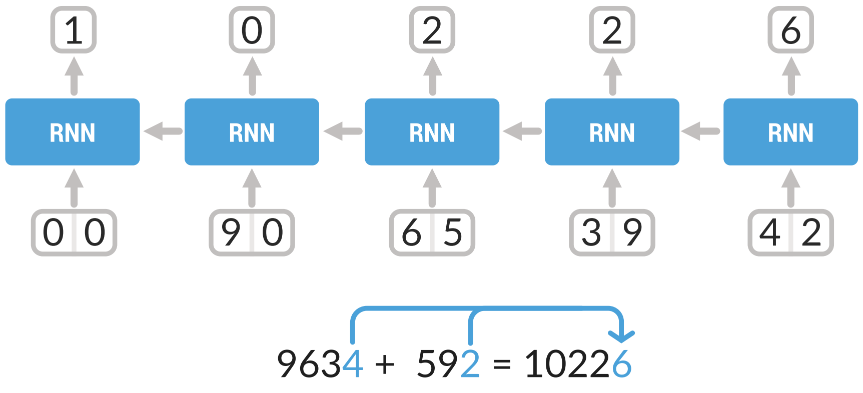


Figure 4.5: A recurrent neural network (RNN) performing addition digit by digit. We align the digits of the two operands, and we present the network pairs of digits step-by-step, starting with the least significant digits. For every pair of input digits, the network outputs the corresponding digit of the result. The carry needs to be expressed through the hidden state of the network.

the data. Using this sampling process, we generate 3 training datasets of different sizes, containing $N \in \{500, 1000, 5000\}$ instances, in order to compare how curricula perform with different amounts of data.

We also generate two test datasets of 1000 samples each, sampled from two different data distributions:

- an *interpolation* dataset, which contains 1000 examples sampled from a distribution similar to the training distribution, containing operands with 1-5 digits, and which have been held out from training.
- an *extrapolation* dataset, which contains 1000 examples sampled from a distribution that is different from the training distribution, and which contains operands with 6-10 digits.

4.2.2 *Models and Training*

We use a Long Short-Term Memory (LSTM) network (Hochreiter and Schmidhuber, 1997) which takes as input at every time step a pair of digits represented as floating point numbers (0.0, 1.0, ..., 9.0). Since the two operands can have different number of digits, we pad the shorted numbers with 0.0 in the most significant digits positions. In our experiments, we set the LSTM hidden size to 16.

We formulate the prediction problem as a classification problem, where for every time step, we predict a probability distribution over the 10 possible digits of the sum at that position. Therefore, at every time step, we pass the output of the LSTM through a hidden dense layer that projects it to a tensor of size 10 (i.e., one output for each possible digit).

For the curriculum we use the square root competence function introduced in Equation 4.5, with initial competence $c_0 = 0.2$, and we experiment with various curriculum lengths T as shown in the next section.

We train the model for 50000 epochs, using the Adam optimizer (Kingma and Ba, 2015) with batch size 128, learning rate 0.001 and momentum 0.9. We use the cross entropy loss function, averaged over the time dimension. Our code is implemented using the TensorFlow framework (Abadi et al., 2016), and we conduct all our experiments on a single Nvidia TitanX GPU.

4.2.3 *Results*

The results are presented in Figure 4.6. On each row of the figure, we display the accuracy per training step (where each training step corresponds to one batch update) for models trained using each of the three training datasets of increasing size. We evaluate the model performance on the two test settings described earlier, interpolation (first column) and extrapolation (second column). For each train-test setting, we display the performance of a model trained without a curriculum ("Baseline"), as well as of models trained with curricula of different lengths. Note that for all training settings, we trained the models for the same number of epochs (where an epoch is a complete pass over the entire training set), but because some datasets are larger, each epoch requires more training iterations (hence the different number of training steps on the x-axis in Figure 4.6).

As expected, models trained on larger training sets perform better both in the interpolation and extrapolation setting. In fact, for this simple problem setting, when the dataset is large enough all models are able to predict the test data perfectly (100% accuracy), in both interpolation and extrapolation. However, as we reduce the amount of training data, we start seeing a wider and wider gap between the performance of the model in interpolation versus extrapolation, and we also start observing increasing benefits from using curriculum learning. For both $N = 500$ and $N = 1,000$ training samples, curricula improve the final accuracy for both interpolation and extrapolation, with larger benefits on the extrapolation distribution, of up to 18% difference in accuracy.

Comparing different curriculum lengths, we notice an interesting trend. As we increase the curriculum length C, the performance gain initially increases, and then starts to decrease. This suggests that longer curricula are not necessarily better. We should then use a validation dataset to chose hyperparameter C, as we typically do with other model parameters.

In terms of the benefits gained from using curriculum learning, here we see two types of benefits:

(i) better accuracy at the end of training, for certain curriculum lengths.

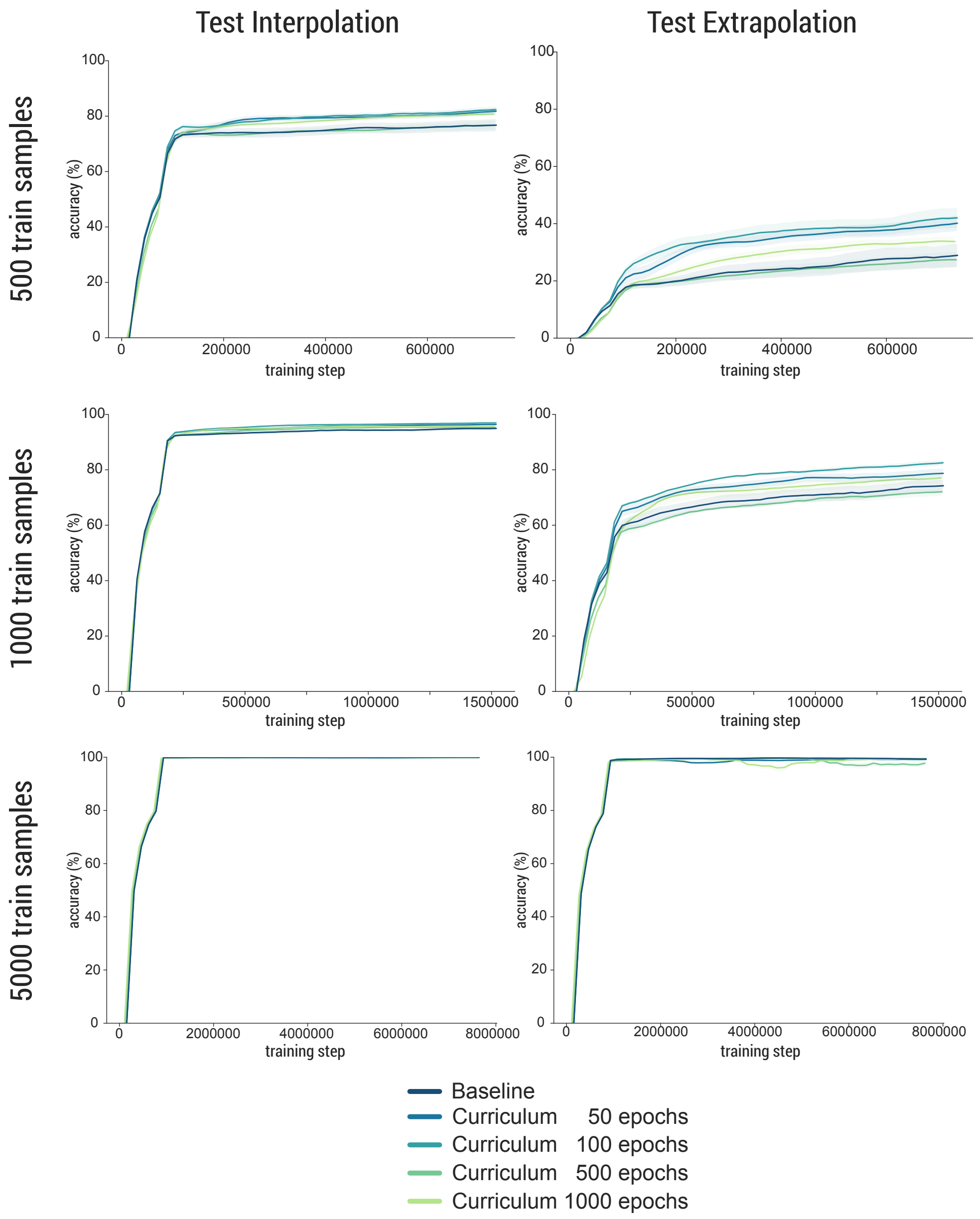


Figure 4.6: Test accuracy for an LSTM with hidden size 16, trained to add two numbers digit-by-digit. Each curve represents the accuracy mean and standard error (over 4 runs started at different random parameter initializations) per training step, under different training testing settings. Each row corresponds to a different training dataset, with 500, 1000 and 5000 samples, respectively. For each training setting, we compute the test accuracy on the interpolation (first column) and extrapolation (second column) datasets. For each train-test setting, we display the performance of a model trained without a curriculum ("Baseline"), as well as of models trained with curricula of increasing lengths.

(ii) faster training. Except for the largest dataset, the curricula on the small and medium-sized dataset train faster. We measure this in the following way. We select a point on the accuracy curve of the baseline model, and we check at which training step a certain curriculum approach achieved the same accuracy. It is easy to see that this point is much earlier for the best curricula, since the curriculum accuracy curve is always "above" that of the baseline. Thus, if we have a limited training time budget, a model trained with curriculum learning can achieve a better performance in the allotted time. Alternatively, if we want to train the model up to a minimum accuracy of $a\%$, a model trained with a curriculum can achieve this faster.

4.2.4 *Discussion*

In this case study, we applied curriculum learning to a simple synthetic problem: learning to add two numbers using an LSTM. We used a curriculum based on the number of digits of the two operands, and evaluated it on training datasets of increasing size. Our results indicated that the curriculum strategy was particularly beneficial in middle and low-data regimes. Curricula with an appropriate length helped the model train faster and achieve better accuracy at the end of training, with larger gains when tested out of distribution. However, curricula that are too long provided marginal or no benefits. Moreover, for this simple problem, with enough training data, the baseline model was able to solve the problem without any benefits from using a curriculum. These results suggest that curriculum learning can be very useful, but only for certain data regimes, and when the length of the curriculum is chosen carefully.

4.3 LEARNING ARITHMETIC WITH SEQUENCE-TO-SEQUENCE MODELS

For our second case study, we take the problem setting in the previous section, and we make it more challenging:

1. The inputs are no longer provided one digit at a time, with the two operators aligned digit-by-digit. Instead, the input is provided as a string (e.g., "963 + 59"), and thus the model also needs to learn to interpret the requested computation.

2. The inputs may contain more than two operands (e.g., "963 + 59 + 123 + 2").

3. The inputs are provided character by character as symbols from a vocabulary. Thus the model also needs to learn a meaningful representation (embedding) for each digit.

4. The target outputs are also sequences of digits (e.g., "1022"), and we no longer have one output for every pair of inputs as in Section 4.2. Thus, the model also needs to infer the appropriate output length.

To support the new data format, we take inspiration from the *sequence-to-sequence* (seq2seq) models typically used in machine translation (e.g., Sutskever et al., 2014). The model consists of an *encoder*, that processes the entire input sentence and converts it into a hidden representation, and a *decoder*, that predicts the answer character by character. We depict this setting in Figure 4.7.

This setting gives us the opportunity to test a some additional properties of curriculum learning:

- We test if curriculum learning is also beneficial for seq2seq models, which are very common in practice (e.g., in machine translation, semantic parsing, syntactic parsing, time series).

- We can test the model ability to generalize to adding more terms than it has been trained on. Unlike our previous setup from Section 4.2, now the model does not have any support from the model architecture to know how to align the digits of the operands.

- We can test the model ability to generalize to adding numbers with more digits than it has trained on. Unlike our previous setup from Section 4.2, now the model does not have any support from the model architecture to know how many digits to predict for the output.

- We experiment not only with recurrent networks, but also with Transformers (Vaswani et al., 2017), which have become the go-to models for many NLP areas.

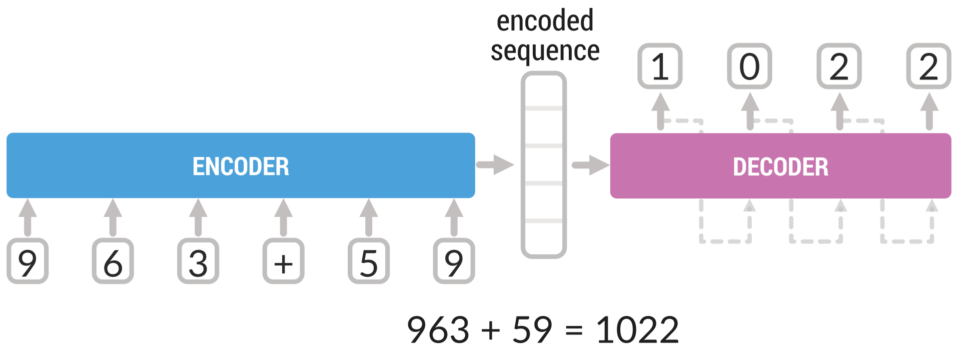


Figure 4.7: Illustrating of a sequence-to-sequence model trained to predict the sum of an arbitrary number of terms.

4.3.1 *Data*

TRAINING DATA. We generate training samples, each representing a summation of 2-4 terms, each being an integer with 1-5 digits. We generate $10,000$ samples for each number of terms (i.e., $3 \times 10,000$ samples in total). Each input is represented as a string sentence (e.g., "$12 + 3456$") that is passed to the model character by character. The target output is also a string, representing the answer (e.g., "3468").

TEST DATA. We generate multiple test datasets:

- Interpolation: we generate data from the same distribution as training, and test the model's ability to add 2, 3 and 4 terms, using $10,000$ samples for each case.
- Extrapolation in the number of terms: we generate $10,000$ samples containing $5-7$ terms to add, each term consisting of $1-5$ digits.
- Extrapolation in the number of digits: we generate $10,000$ samples representing the addition of 2 terms, each term consisting of 6 or 7 digits.

4.3.2 *Models and Training*

MODEL. The model consists of an encoder-decoder architecture, as illustrated in Figure 4.7. We experiment with two types of architectures:

- `LSTM`: The encoder is a Long Short Term Memory (`LSTM`) network (Hochreiter and Schmidhuber, 1997) with a single layer of hidden size 512 units. Similarly, the decoder is also a single layer `LSTM` with 2048 hidden units.
- `Transformer`: The encoder-decoder architecture is a `Transformer` (Vaswani et al., 2017), which has become ubiquitous in deep learning, and

which uses a special type of attention mechanism known as *self-attention*. We chose the number of layers and units to match the architecture used in Saxton et al. (2019), where a `Transformer` was also used for solving multiple mathematical problems. Concretely, both the encoder and the decoder consist of 6 self-attention layers with 512 units, 8 attention heads, and a feed-forward layer with 2048 units. The model is regularized using dropout on the outputs of each layer, with drop probability 0.1.

The inputs are parsed into individual characters and embedded into a 256-dimensional space for the `LSTM` model, and 512-dimensional space for the `Transformer`. The character embeddings are learned from scratch together with the model parameters.

IMPLEMENTATION DETAILS. We implemented our model using OpenNMT-tf [1], the TensorFlow (Abadi et al., 2016) version of the OpenNMT framework (Klein et al., 2017), which provides support for training seq2seq models.

TRAINING. We trained the model using as loss function the average cross-entropy between the predicted sequences and the targets. We used a batch size of 1024 samples and trained for $200,000$ iterations. We updated the model parameters using the Adam optimizer (Kingma and Ba, 2015) with learning rate 0.0006 and momentum 0.9. For the `Transformer` we also apply learning rate decay, with an exponential decay which multiplies the learning rate every 10 steps by a factor of 0.999, similar to Saxton et al. (2019). To guard against exploding gradients, we used gradient clipping with a global norm threshold of 10.

CURRICULUM. We apply our curriculum framework from Section 4.1, using the length of the input sequence as difficulty metric and the Square Root competence function with initial value $c_0 = 0.1$. The length of the input sequence is an intuitive difficulty score in this case, since longer sequences can mean that there are more terms to add or that the numbers that we are adding are larger. We vary the curriculum length, and report results for multiple lengths.

4.3.3 *Results*

We evaluated the model under multiple training regimes (baseline without curriculum, and multiple curriculum lengths). We plot the test accuracy per training step for both interpolation and extrapolation in Figure 4.8 for the `LSTM` architecture, and in Figure 4.9 for the `Transformer`.

[1] https://github.com/OpenNMT/OpenNMT-tf

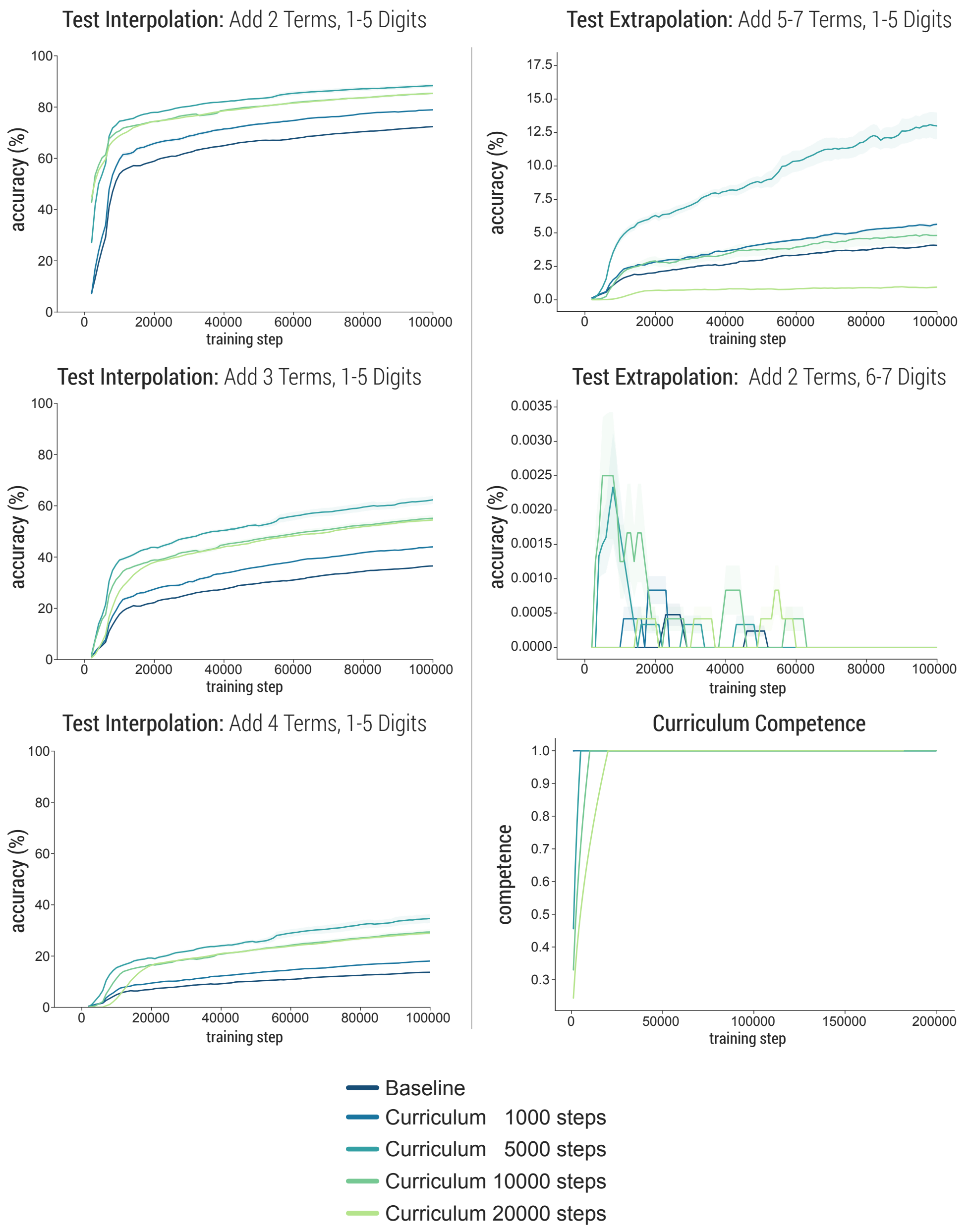


Figure 4.8: Results for the `LSTM`-based sequence-to-sequence model trained to add multiple integers. We show the results for a baseline model trained without curriculum, and multiple models trained with curricula of different lengths. We report the accuracy mean and standard error (over 4 runs with different random initializations) per training step.

As expected, adding more and more terms is more difficult for both model architectures for all training regimes. However, the models trained with curricula seem to perform better regardless of the number of terms we add, obtaining boosts in accuracy between up to a difference of 30% for the

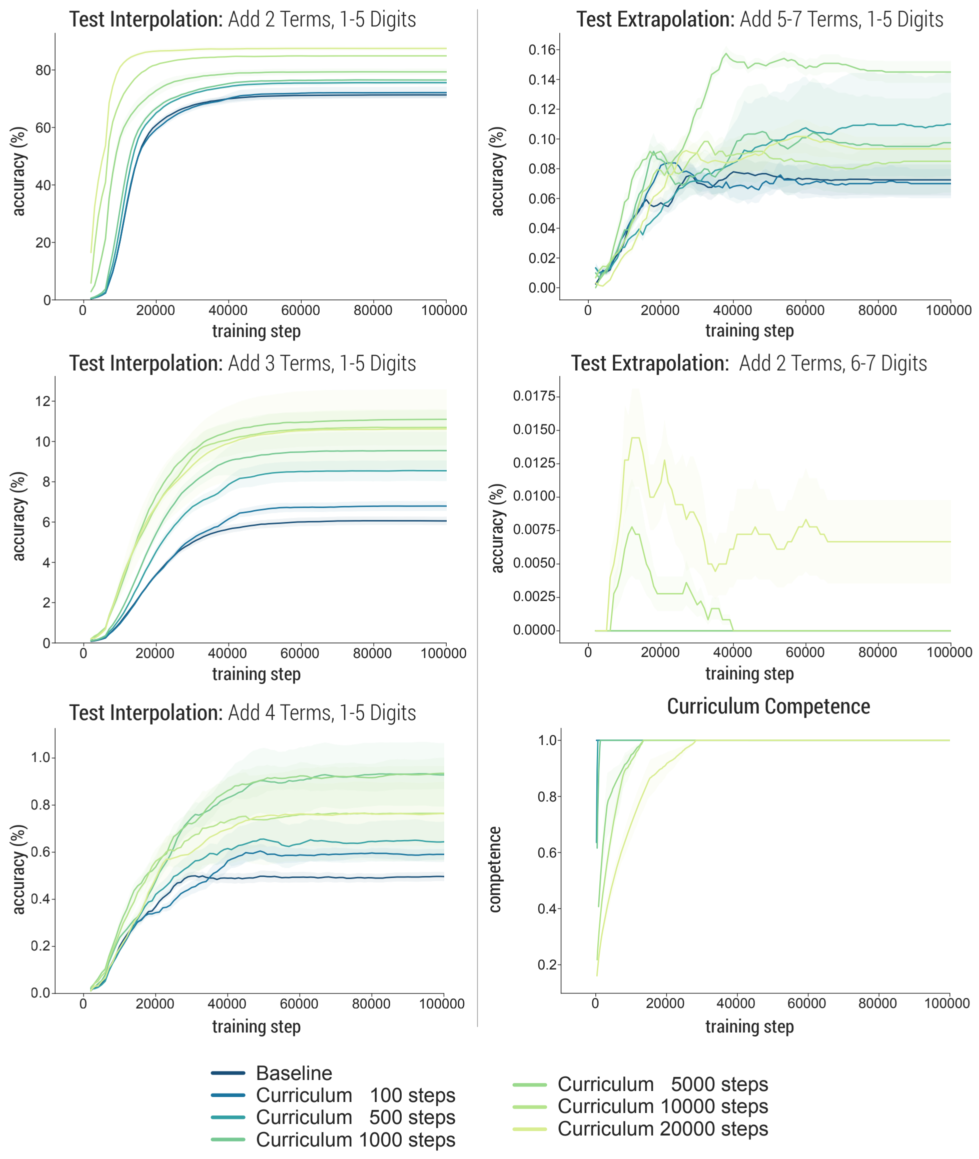


Figure 4.9: Results for the `Transformer`-based sequence-to-sequence model trained to add multiple integers. We show the results for a baseline model trained without curriculum, and multiple models trained with curricula of different lengths. We report the accuracy mean and standard error (over 4 runs with different random initializations) per training step.

interpolation case (adding $2-4$ terms), up to 6% for extrapolation (adding $5-7$ terms).

An interesting failure mode is extrapolation in the number of digits of the operands, where both the baseline and the curricula fail to make correct predictions, both settings obtaining $\sim 0\%$ accuracy, for both `LSTMs` and `Transformers`. This is because the sequence-to-sequence model is always supervised to predict the end-of-sequence token after reaching the maximum sequence length seen in training, and thus it wrongly predicts the end-of-sequence token earlier than it should on the extrapolation setting.

Comparing curricula of different lengths, the results show that curricula that are too short perform closer to the baseline model. As we increase the curriculum length, the performance improves, but only up to a point, after which the performance gains start decreasing. This is especially prevalent on the extrapolation dataset, where the longest curriculum in fact harms performance. This suggests that there is an optimal curriculum length, and it would be beneficial to use a validation dataset to choose the best hyper-parameter C. However, at least for this problem, the range of curriculum lengths C for which the curriculum performs better than the baseline is quite permissive, which makes it easy to improve the baseline performance without significant parameter tuning efforts.

In terms of the types of benefits provided by curriculum learning, in this case the benefits are two-fold. In the plots in both Figure 4.8 and Figure 4.9, the curriculum curve is always above that of the baseline. This means that for every accuracy achieved by the baseline at a training step t, the curriculum has achieved that accuracy earlier. We can therefore say that the curriculum trains the model *faster*. The gains in training speed are particularly prevalent for the `Transformer`. Moreover, at the end of training, the models trained with curriculum learning (at least up to some length C) achieve *better performance* on all datasets. For the `LSTM` it is questionable whether the baseline would have achieved the same performance if given a larger budget of training iterations, since the accuracy curves still have not yet converged despite the generous iteration budget (we discuss more on this in Chapter 6). However, for the `Transformer` the curves have flattened, and we can see a clear difference in final accuracy. Nevertheless, in practical settings we cannot allow the model to train indefinitely, and thus a method that achieves better performance after a reasonable amount of training time would be preferable.

4.3.4 *Discussion*

In this case study we considered a type of neural network architecture that if very common in modern deep learning, especially in natural language processing: sequence-to-sequence models. We experimented with two of the most common types of encoder-decoder architectures, consisting of `LSTM` and `Transformer` layers. Our results showed that both types of architectures can have significant benefits from curriculum learning, both in terms of final performance and training speed. This is particularly impressive for `Transformers`, which are notoriously hard to train and typically require specialized learning rate schedules (Vaswani et al., 2017). In our case, without modifying a learning rate schedule that had been tuned for the baseline, we were able to observe performance improvements with curriculum learning. This can have important consequences for several application areas where `Transformers` have become the de facto standard, such as machine translation (Vaswani et al., 2017), document summarization (e.g., Egonmwan and Chali, 2019; Pilault et al., 2020) and many other areas in natural language processing or signal processing. In the next case study, we consider one such application.

4.4 NEURAL MACHINE TRANSLATION

In this section, we apply our curriculum learning framework to the task of Neural Machine Translation (NMT). The work presented in this section is also included in our publication (Platanios et al., 2019).

Neural Machine Translation (Kalchbrenner and Blunsom, 2013; Bahdanau et al., 2015) now represents the state-of-the-art adapted in most machine translation systems (Crego et al., 2016; Wu et al., 2016; Bojar et al., 2017a), largely due to its ability to benefit from end-to-end training on massive amounts of data. In particular, the recently-introduced self-attentional Transformer architectures (Vaswani et al., 2017) are rapidly becoming the de-facto standard in NMT, having demonstrated both superior performance and training speed compared to previous architectures using recurrent neural networks (RNNs; Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014). However, large scale NMT systems are often hard to train, requiring complicated heuristics which can be both time-consuming and expensive to tune. This is especially true for Transformers which, when carefully tuned, have been shown to consistently outperform RNNs (Popel and Bojar, 2018), but on the other hand, also rely on a number of heuristics such as specialized learning rates and large-batch training.

In this line of work, we attempt to tackle this problem by using our generic curriculum learning framework introduced in Section 4.1 for training NMT systems, which reduces training time, reduces the need for specialized heuristics or large batch sizes, and results in overall better performance. It allows us to train both RNNs and, perhaps more importantly, Transformers, with relative ease. In this case, it can also be thought of as a means to avoid getting stuck in bad local optima early on in training.

Notably, we are not the first to examine curriculum learning for NMT, although other related approaches have met with mixed success. Kocmi and Bojar (2017) explored the impact of several curriculum heuristics on training a translation system for a single epoch, presenting the training examples in an easy-to-hard order based on sentence length and vocabulary frequency. However, their strategy introduced all training samples during the first epoch, and it is not clear how this affected learning in the subsequent epochs, with official evaluation results (Bojar et al., 2017b) indicating that final performance may indeed be hurt with this strategy. Contemporaneously to our work, Zhang et al. (2018) further proposed splitting the training samples into a predefined number of bins (5, in their case), based on various difficulty metrics. A manually designed curriculum schedule then specified the bins from which the model sampled training examples. Experiments demonstrated that the benefits of curriculum learning were highly sensitive to several hyperparameters (e.g., learning rate, number of iterations spent in each phase, etc.), and largely provided benefits in convergence speed as opposed to final model accuracy.

In contrast to these previous approaches, we define a continuous curriculum learning method (instead of a discretized regime) with only one tunable hyperparameter (the duration of curriculum learning). Furthermore, as opposed to previous work which only focuses on RNNs, we also experiment with Transformers, which are notoriously hard to train (Popel and Bojar, 2018). Finally, unlike any of the work described above, we show that our curriculum approach helps not only in terms of convergence speed, but also in terms of the learned model performance. In summary, our method has the following desirable features:

1. **Abstract:** It is a novel, generic, and extensible formulation of curriculum learning. A number of previous heuristic-based approaches, such as that of Kocmi and Bojar (2017), can be formulated as special cases of our framework.
2. **Simple:** It can be applied to existing NMT systems with only a small modification to their training data pipelines.
3. **Automatic:** It does not require any tuning other than picking the value of a single parameter, which is the length of the curriculum (i.e., for how many steps to use curriculum learning, before easing into normal training).
4. **Efficient:** It reduces training time by up to 70%, whereas contemporaneous work of Zhang et al. (2018) reports reductions of up to 46%.
5. **Improved Performance:** It improves the performance of the learned models by up to 2.2 BLEU points, where the best setting reported by Zhang et al. (2018) achieves gains of up 1.55 BLEU after careful tuning.

The proposed method uses the curriculum learning framework that we introduced in Section 4.1, and in the next section, we discuss the difficulty metrics used in conjunction with our framework.

4.4.1 *Difficulty Metrics*

There are many possible ways of defining the difficulty of translating a sentence. We consider two heuristics inspired by what we, as humans, may consider difficult when translating, and by factors which can negatively impact the optimization algorithms used when training NMT models. In the rest of this section we denote our training corpus as a collection of M sentences, $\{s_i\}_{i=1}^{M}$, where each sentence is a sequence of words, $s_i = \{w_0^i, \ldots, w_{N_i}^i\}$.

SENTENCE LENGTH: We argue that it is harder to translate longer sentences, as longer sentences require being able to translate their component parts, which often consist of short sentences. Furthermore, longer sentences are intuitively harder to translate due to the propagation of errors made early on when generating the target language sentence. Therefore, a simple way to define the difficulty of a sentence $s_i = \{w_0^i, \ldots, w_{N_i}^i\}$ is as follows:

$$d_{\text{length}}(s_i) \triangleq N_i. \tag{4.7}$$

Note that we can compute this difficulty metric on either the source language sentence or the target language sentence. We only consider the source sentence in this work [2].

WORD RARITY: Another aspect of language that can affect the difficulty of translation is the frequency with which words appear. For example, humans may find rare words hard to translate because we rarely ever see them and it may be hard to recall their meaning. The same can be true for NMT models where: (i) the statistical strength of the training examples containing rare words is low and thus the model needs to keep revisiting such words in order to learn robust representations for them, and (ii) the gradients of the rare word embeddings tend to have high variance; they are overestimates of the true gradients in the few occasions where they are non-zero, and underestimates otherwise. This suggests that using word frequencies may be a helpful difficulty heuristic. Given a corpus of sentences, $\{s_i\}_{i=1}^M$, we define relative word frequencies as:

$$\hat{p}(w_j) \triangleq \frac{1}{N_{\text{total}}} \sum_{i=1}^{M} \sum_{k=1}^{N_i} \mathbb{1}_{w_k^i = w_j}, \tag{4.8}$$

where $j = 1, \ldots, \#\{\text{unique words in corpus}\}$ and $\mathbb{1}_{\text{condition}}$ is the indicator function which is equal to 1 if its condition is satisfied and 0 otherwise. Next we need to decide how to aggregate the relative word frequencies of all words in a sentence to obtain a single difficulty score for that sentence. Previous research has proposed various pooling operations, such as minimum, maximum, and average (Zhang et al., 2018), but they show that they do not work well in practice. We propose a different approach. Ultimately, what might be most important is the overall likelihood of a sentence as that contains information about both word frequency and, implicitly, sentence length. An approximation to this likelihood is the product of the unigram probabilities, which is related to previous work in the area of active learning (Settles and Craven, 2008). This product can be thought of as an approximate language model (assuming words are sampled independently)

[2]NMT models typically first pick up information about producing sentences of correct length. It can be argued that presenting only short sentences first may lead to learning a strong bias for the sentence lengths. However, in our experiments, we did not observe this to be an issue as the models kept improving and predicting sentences of correct length throughout training.

and also implicitly incorporates information about the sentence length that was proposed earlier (longer sentence scores are products over more terms in $[0, 1]$ and are thus likely to be smaller). We thus propose the following difficulty heuristic:

$$d_{\text{rarity}}(s_i) \triangleq -\sum_{k=1}^{N_i} \log \hat{p}(w_k^i), \tag{4.9}$$

where we use logarithms of word probabilities to prevent numerical errors. Note that negation is used because we define less likely (i.e., more rare) sentences as more difficult.

These are just two examples of difficulty metrics, and it is easy to conceive of other metrics such as the occurrence of homographs (Liu et al., 2018) or context-sensitive words (Bawden et al., 2018), the examination of which we leave for future work.

4.4.2 *Experiments*

DATASETS. For our experiments, we use three of the most commonly used datasets in NMT, that range from a small benchmark dataset to a large-scale dataset with millions of sentences. Statistics about the datasets are shown in Table 4.2.

MODELS. We perform experiments using both RNNs and Transformers. For the RNN experiments we use a bidirectional LSTM for the encoder, and an LSTM with the attention model of Bahdanau et al. (2015) for the decoder. The number of layers of the encoder and the decoder are equal. We use a 2-layer encoder and a 2-layer decoder for all experiments on IWSLT datasets, and a 4-layer encoder and a 4-layer decoder for all experiments on the WMT dataset, due to the dataset's significantly larger size. For the Transformer experiments we use the BASE model proposed by Vaswani et al. (2017) It consists of a 6-layer encoder and decoder, using 8 attention heads, and 2,048 units for the feed-forward layers. The multi-head attention keys and values depth is set to the word embedding size. The word embedding size is 512 for all experiments. Furthermore, for the Transformer experiments on the two smaller datasets we do not use any learning rate schedule, and for the experiments on the largest dataset we use the default Transformer schedule. A detailed discussion on learning rate schedules for Transformers is provided near the end of this section.

SETUP. All of our experiments were conducted on a machine with a single Nvidia V100 GPU, and 24 GBs of system memory. During training, we use a label smoothing factor of 0.1 (Wu et al., 2016) and the AMSGrad optimizer

Dataset	# Train	# Dev	# Test
IWSLT-15 En→Vi	133k	768	1268
IWSLT-16 Fr→En	224k	1080	1133
WMT-16 En→De	4.5M	3003	2999

Table 4.2: Number of parallel sentences in each dataset. "k" stands for "thousand" and "M" stands for "million."

(Reddi et al., 2018), and a batch size of 5,120 tokens (due to GPU memory constraints). During inference, we employ beam search with a beam size of 10 and the length normalization scheme of Wu et al. (2016).[3]

CURRICULUM HYPERPARAMETERS. We set the initial competence c_0 to 0.01, in all experiments. This means that all models start training using the 1% easiest training examples. The curriculum length T is effectively the only hyperparameter that we need to set for our curriculum methods. In each experiment, we set T in the following manner: we train the baseline model without using any curriculum and we compute the number of training steps it takes to reach approximately 90% of its final BLEU score. We then set T to this value. This results in T being set to 5,000 for the RNN experiments on the IWSLT datasets, and 20,000 for the corresponding Transformer experiments. For WMT, we set T to 20,000 and 50,000 for RNNs and Transformers, respectively. Furthermore, we use the following notation and abbreviations when presenting our results:

– Plain: Trained without using any curriculum.
– SL: Curriculum with sentence length difficulty.
– SR: Curriculum with sentence rarity difficulty.
– Linear: Curriculum with the linear competence shown in Equation 4.2.
– Sqrt: Curriculum with the square root competence shown in Equation 4.5.

DATA PREPROCESSING. Our experiments are performed using the machine translation library released by Platanios et al. (2018). We use the same data preprocessing approach the authors used in their experiments. While training, we consider sentences up to length 200. Similar to them, for the IWSLT-15 experiments we use a per-language vocabulary which contains the 20,000 most frequently occurring words, while ignoring words that appear less than 5 times in the whole corpus. For the IWSLT-16 and WMT-16 experiments we use a byte-pair encoding (BPE) vocabulary (Sennrich et al., 2016) trained using 32,000 merge operations, similar to the original Transformer paper by Vaswani et al. (2017).

[3]We emphasize that we did not run experiments with other architectures or configurations, and thus our baseline architectures were not chosen because they were favorable to our method, but rather because they were frequently mentioned in existing literature.

		RNN					Transformer					
		Plain	SL Curriculum		SR Curriculum		Plain	Plain*	SL Curriculum		SR Curriculum	
			c_{linear}	c_{sqrt}	c_{linear}	c_{sqrt}			c_{linear}	c_{sqrt}	c_{linear}	c_{sqrt}
BLEU	En→Vi	26.27	26.57	27.23	26.72	26.87	28.06	29.77	29.14	29.57	29.03	29.81
	Fr→En	31.15	31.88	31.92	31.39	31.57	34.05	34.88	34.98	35.47	35.30	35.83
	En→De	26.53	26.55	26.54	26.62	26.62	–	27.95	28.71	29.28	29.93	30.16
Time	En→Vi	1.00	0.64	0.61	0.71	0.57	1.00	1.00	0.44	0.33	0.35	0.31
	Fr→En	1.00	1.00	0.93	1.10	0.73	1.00	1.00	0.49	0.44	0.42	0.39
	En→De	1.00	0.86	0.89	1.00	0.83	–	1.00	0.58	0.55	0.55	0.55

Table 4.3: Summary of experimental results. For each method and dataset, we present the test set BLEU score of the best model based on validation set performance. We also show the relative time required to obtain the BLEU score of the best performing baseline model. For example, if an RNN gets to 26.27 BLEU in 10,000 steps and the SL curriculum gets to the same BLEU in 3,000 steps, then the plain model gets a score of 1.0 and the SL curriculum receives a score of $3,000/10,000 = 0.3$. Plain stands for the model trained without a curriculum and, for Transformers, Plain* stands for the model trained using the learning rate schedule shown in Equation 4.10.

RESULTS. We present a summary of our results in Table 4.3 and we also show complete learning curves for all methods in Figure 4.10. The evaluation metrics we use are the test set BLEU score and the time it takes for the models using curriculum learning to obtain the BLEU score that the baseline models attain at convergence. We observe that Transformers consistently benefit from our curriculum learning approach, achieving gains of up to 2 BLEU, and reductions in training time of up to 70%. RNNs also benefit, but to a lesser extent. This is consistent with our motivation for this work, which stems from the observation that training RNNs is easier and more robust than training Transformers. Moreover, the square root competence consistently outperforms the linear one, which fits well with our intuition and motivation for introducing it. Regarding the difficulty heuristics, sentence length and sentence rarity both result in similar performance.

We also observe that, for the two small datasets, RNNs converge faster than Transformers in terms of both the number of training iterations and the overall training time. This is contrary to other results in the machine translation community (e.g., Vaswani et al., 2017), but could be explained by the fact that we are not using any learning rate schedule for training Transformers. However, they never manage to outperform Transformers in terms of test BLEU score of the final model. Furthermore, to the best of our knowledge, for IWSLT-15 we achieve state-of-the-art performance. The highest previously reported result was 29.03 BLEU (Platanios et al., 2018), in a multi-lingual setting. Using our curriculum learning approach we are able to achieve a BLEU score of 29.81 for this dataset.

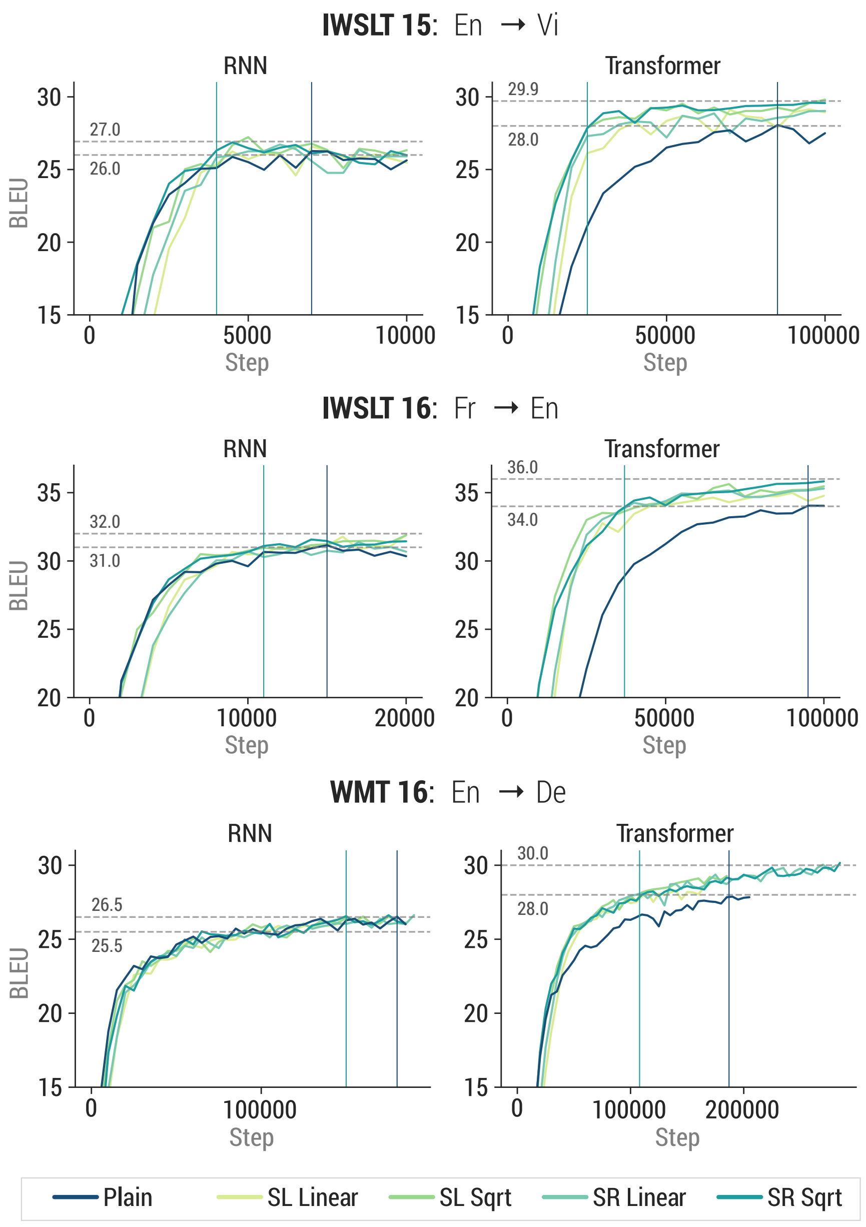


Figure 4.10: Plots illustrating the performance of various models on the test set, as training progresses. Blue lines represent the baseline methods when no curriculum is used and red lines represent the same models when different versions of our curriculum learning framework are used to train them. The vertical lines represent the step in which the models attain the BLEU score that the baseline models attain at convergence.

Overall, we have shown that *our curriculum learning approach consistently outperforms models trained without any curriculum*, in both limited data settings and large-scale settings.

LEARNING RATE SCHEDULE. In all of our IWSLT experiments so far, we have used the default AMSGrad learning rate of 0.001 and intentionally avoid using any learning rate schedules. However, Transformers are not generally trained without a learning rate schedule. Such schedules typically use a warm-up phase, which means that the learning rate starts at a very low value and keeps increasing until the end of the warm-up period, after which a decay rate is typically used. In order to show that our curriculum learning approach can act as a principled alternative to such highly tuned learning rate schedules, we now present the results we obtain when training our Transformers using the following learning rate schedule:

$$\mathrm{lr}(t) \triangleq d_{\text{embedding}}^{-0.5} \min\left(t^{-0.5}, t \cdot T_{\text{warmup}}^{-1.5}\right), \tag{4.10}$$

where t is the current training step, $d_{\text{embedding}}$ is the word embedding size, and T_{warmup} is the number of warmup steps and is set to 10,000 in these experiments. This schedule was proposed in the original Transformer paper (Vaswani et al., 2017), and was tuned for the WMT dataset. The results obtained when using this learning rate schedule are also shown in Table 4.3, under the name Plain*. *In both cases, our curriculum learning approach obtains a better model in about 70% less training time*. This is very important, especially when applying Transformers in new datasets, because such learning rate heuristics often require careful tuning. This tuning can be both very expensive and time consuming, often resulting in very complex mathematical expressions, with no clear motivation or intuitive explanation (Chen et al., 2018). Our curriculum learning approach achieves better results in significantly less time, while only requiring one parameter (the length of the curriculum). Note that even without using any learning rate schedule, our curriculum methods were able to achieve performance comparable to the Plain* in about twice as many training steps. Plain was not able to achieve a BLEU score above 2.00 even after fives times as many training steps, at which point we stopped these experiments.

IMPLEMENTATION AND REPRODUCIBILITY. We are releasing an implementation of our proposed method and experiments built on top of the machine translation library released by Platanios et al. (2018). Furthermore, all experiments can be run on a machine with a single Nvidia V100 GPU, and 24 GBs of system memory. Our most expensive experiments — the ones using Transformers on the WMT-16 dataset — take about 2 days to complete, which would cost about $125 on a cloud computing service such as Google Cloud or Amazon Web Services, thus making our results reproducible, even by independent researchers.

4.4.3 *Related Work*

We provide an extensive overview of curriculum learning methods in Chapter 3. However, in this section we review the work that is related to applying curriculum learning to machine translation (MT).

Perhaps the earliest attempt to apply curriculum learning in MT was made by Zou et al. (2013). The authors employed a curriculum learning method to learn Chinese-English bilingual word embeddings, which were subsequently used in the context of phrase-based machine translation. They split the word vocabulary in 5 separate groups based on word frequency, and learned separate word embeddings for each of these groups in parallel. Then, they merged the 5 different learned embeddings and continued training using the full vocabulary. While this approach makes use of some of the ideas behind curriculum learning, it does not directly follow the original definition introduced by Bengio et al. (2009). Moreover, their model required 19 days to train. There have also been a couple of attempts to apply curriculum learning in NMT that were discussed earlier.

There also exists some relevant work in areas other than curriculum learning. Zhang et al. (2016a) propose training neural networks for NMT by focusing on hard examples, rather than easy ones. They report improvements in BLEU score, while only using the hardest 80% training examples in their corpus. This approach is more similar to boosting by Schapire (1999), rather than curriculum learning, and it does not help speed up the training process; it rather focuses on improving the performance of the trained model. The fact that hard examples are used instead of easy ones is interesting because it is somewhat contradictory to curriculum learning. Also, in contrast to curriculum learning, no ordering of the training examples is considered.

Perhaps another related area is that of active learning, where the goal is to develop methods that request for specific training examples. Haffari et al. (2009), Bloodgood and Callison-Burch (2010), and Ambati (2012) all propose methods to solicit training examples for MT systems, based on the occurrence frequency of n-grams in the training corpus. The main idea is that if an n-gram is very rare in the training corpus, then it is difficult to learn to translate sentences in which it appears. This is related to our sentence rarity difficulty metric and points out an interesting connection between curriculum learning and active learning.

Regarding training Transformer networks, Shazeer and Stern (2018) perform a thorough experimental evaluation of Transformers, when using different optimization configurations. They show that a significantly higher level of performance can be reached by not using momentum during optimization, as long as a carefully chosen learning rate schedule is used. Such learning rate schedules are often hard to tune because of the multiple seemingly arbitrary terms they often contain. Furthermore, Popel and Bojar

(2018) show that, when using Transformers, increasing the batch size results in a better model at convergence. We believe this is indicative of very noisy gradients when starting to train Transformers and that higher batch sizes help increase the signal-to-noise ratio. We show that our proposed curriculum learning method offers a more principled and robust way to tackle this problem. Using our approach, we are able to train Transformers to state-of-the-art performance, using small batch sizes and without the need for peculiar learning rate schedules, which are typically necessary.

4.4.4 *Discussion*

In this case study, we applied our curriculum learning framework to training neural machine translation models. Our approach is able to boost performance of existing NMT systems, while at the same time significantly reducing their training time. It differs from previous approaches in that it does not depend on multiple hyperparameters that can be hard to tune, and it does not depend on a manually designed discretized training regime. Perhaps most interestingly, we show that our method makes training Transformers faster and more reliable, but has a much smaller effect in training recurrent neural networks (RNNs).

As future directions for this line of work, we are mainly interested in: (i) exploring more difficulty heuristics, such as measures of alignment between the source and target sentences (Kocmi and Bojar, 2017), sentence length discrepancies, or even using a pre-trained language model to score sentences, which would act as a more robust replacement of our sentence rarity heuristic, and (ii) exploring more sophisticated competence metrics that may depend on the loss function, the loss gradient, or on the learner's performance on held-out data. Furthermore, it would be interesting to explore applications of curriculum learning to multilingual machine translation (e.g., it may be easier to start with high-resource languages and move to low-resource ones later on).

4.5 multimodal language understanding

In the previous sections, we have successfully applied curriculum learning on various types of sequential data, and we used difficulty metrics that were in some sense characteristic for sequences. In this case study, we tackle a different scenario with a different data modality and difficulty metric.

We consider the problem of multimodal language understanding, where the goal is to train a model that can integrate information from multiple modalities. In particular, we focus on the task of integrating visual information from images with natural language sentences. This is an active research area, with numerous practical applications, from Visual-Question Answering (VQA; Antol et al., 2015; Zhang et al., 2016b; Goyal et al., 2017), image captioning (Lin et al., 2014; Hossain et al., 2019), image search (Thomee and Lew, 2012), to robots that live in the real world and interact with humans.

In our experiments, we use the ShapeWorld framework introduced by Kuhnle and Copestake (2017) for generating custom Visual Question Answering (VQA) datasets. Here, each sample consists of an image and a caption, and the goal is to predict whether the caption agrees with the image. Some examples are shown in Figure 4.11. We chose this framework because it allows us to create both images and captions of varying degrees of difficulty, requiring the model to solve different kinds of tasks (e.g., conjunctions, disjunctions, spatial relations, etc.), and thus we chose it primarily to study curricula in task space in Chapter 5. However, in this chapter we are interested to find out whether a curriculum in input space would also work for this scenario. We could attempt a curriculum based on properties of the caption like in the previous case studies (e.g., sentence length, sentence rarity). However, for diversity, we decided to experiment with a curriculum based on the image properties, which we discuss further.

4.5.1 *Data*

The data consists pairs of images and sentences. Each image contains one or more shapes of geometrical figures (e.g., circle, rectangle, cross, polygon, etc.) of different colors, sizes and rotations, displayed on a black background. For this experiment, we generated images of shape 64×64 pixels containing 1-8 shapes.

While the caption generator can create complex sentences, in this chapter we focus on a simple task—testing existential statements—where the caption states that a specific shape, color or shape-color combination exists in the image, such as the examples in Figure 4.11.

For training, we generated $10,000$ images with 5 different captions for each (i.e. a total of $50,000$ training instances). For testing, we we generated $5,000$ images with 5 captions each (i.e. a total of $25,000$ test instances). We

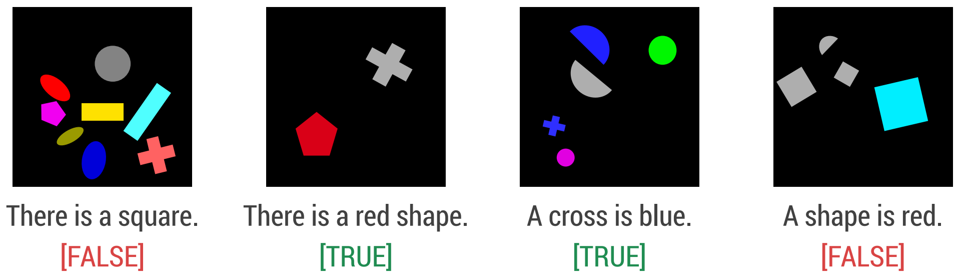


Figure 4.11: Examples from the ShapeWorld dataset, for an existential task.

also generated $1,000 \times 5$ instances for validation, used to tune the model hyperparameters. For all datasets, the ratio of positive and negative labels is 50%, thus the chance accuracy is also 50%.

4.5.2 *Model and Training*

MODEL. We use the CNN-LSTM framework also used by Kuhnle and Copestake (2017) and illustrated in Figure 4.12. The image is processed using a 3 layer convolutional neural network (CNN) with filter sizes 32, 64, and 128, respectively, kernel size 3×3 and ReLU activations. After the last layer, we apply global max pooling to obtain an image embedding of size 128.

The sentence is first split into words and converted to a list of 1-hot word representations. The words are then passed through an embedding layer of size 200, and processed using an LSTM model with hidden size 128.

Then, the image and sentence embeddings are multiplied element-wise, and passed through a multi-layer perception (MLP) with 2 hidden layers of sizes 256 and 128, respectively, and ReLU activation. Finally, a dense projection layer maps the 128 vector to a single output, which is interpreted as the unnormalized probability that the image and sentence agree.

TRAINING. We frame the problem of predicting image-caption agreement as a binary classification task, and we use the binary cross-entropy loss function. We train the model using the Adam optimizer (Kingma and Ba, 2015) with learning rate $3e-4$ and momentum 0.9. The model was regularized using a weight decay value of $1e-5$ and dropout rate 0.2. We train all models for $200,000$ iterations, using batches of size 128. Our code is implemented using Tensorflow (Abadi et al., 2016) and we performed all our experiments using a single Nvidia V100 GPU.

CURRICULUM. We apply a curriculum in input space that is based on the difficulty of the image. The intuition was that it is easier to test existential statements, such as *"There is a circle in the image."*, when there are fewer shapes in the image. Thus, we use the number of shapes in the image as

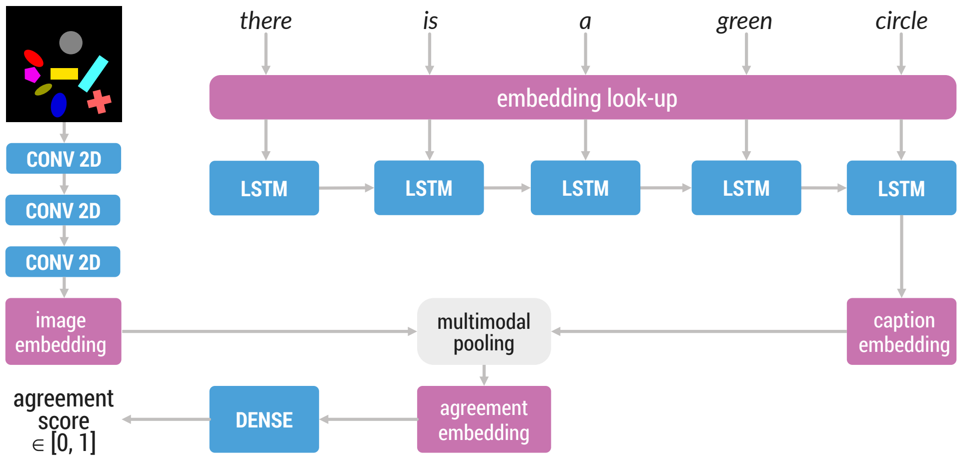


Figure 4.12: Model used for the ShapeWorld dataset. Figure adapted from Kuhnle and Copestake (2017).

difficulty metric, and we apply our curriculum learning framework introduced in Section 4.1 using the Square Root competence function.

4.5.3 *Results*

We evaluated the baseline method trained without a curriculum, as well as curricula of different lengths C. We show their corresponding test accuracy per training step in Figure 4.13.

These results show that, as we increase the curriculum length C the model starts performing better than the baseline, both in terms of final accuracy after $50,000$ steps, but also in terms of training speed. Similar to the previous sections, we say that the curriculum approach is training *faster* than the baseline if for a particular accuracy a of the baseline, the curriculum approach has reached that accuracy earlier. We can see that this is also the case for this experiment. However, if we increase the curriculum length C beyond a point ($C \geqslant 30,000$), the model starts performing worse than the baseline. This is consistent with our observations from the previous sections. As before, there may be multiple causes for this. This may be due to the fact that the slow pace of the curriculum allows difficult examples very late, and thus the model may not have had the chance to learn them within the allotted iteration budget. Alternatively, a curriculum that is too slow may cause the model to get stuck in a local minimum before the schedule allows the model to train on all samples, which would require an adjustment of the learning rate or other strategies. We attempt to understand this behavior in Chapter 6.

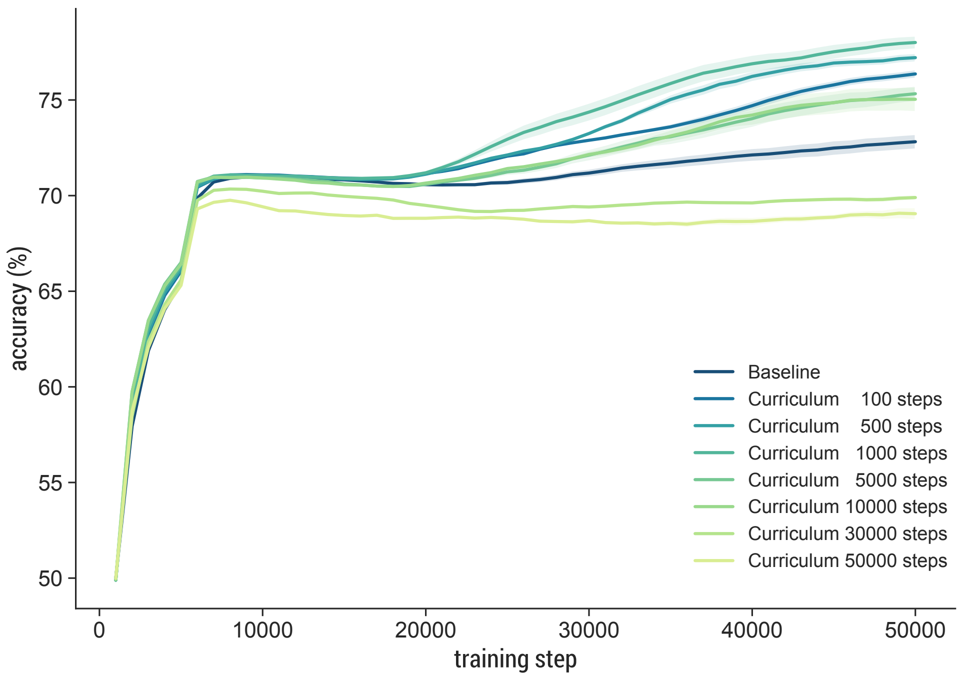


Figure 4.13: Results for the ShapeWorld dataset, using a CNN-LSTM model. We show the results for a baseline model trained without curriculum, and for input space curricula of different lengths C. The difficulty was the number of shapes per image. We report the mean and standard error over 5 runs.

4.5.4 *Discussion*

These results provide us with a few interesting insights:

- Curricula in input space also work on problems with images and CNN-style networks, if we choose an appropriate difficulty metric.
- There is an optimal range for the curriculum length C that will allow the model to converge faster and achieve better performance than the baseline. Curricula that are too short perform similar to the baseline, while curricula that are too long can in fact harm the performance. For this reason, we would typically use the results on the validation set to select the appropriate curriculum length.
- Our framework proposed in Section 4.1 was successfully applied to another problem setting without changing the competence function. The only hyperparameter we have to tune is the curriculum length C.

We will also revisit this problem in the next chapter, when discussing curriculum learning methods in task space.

4.6 KEY TAKEAWAYS

In this chapter we have proposed algorithms for performing curriculum learning on the *input space* of a model. In this scenario, the training samples are ranked by some measure of difficulty, and presented to the model for training according to a schedule. In Section 4.1, we proposed a curriculum learning framework that allows us to take any sample difficulty measures and preprocess them such that they are compatible with our proposed model competence functions (i.e. pacing functions). Using this framework, we were able to easily apply curriculum learning to multiple problems, where we used different models that operate on different types of data (sequences of numbers, sequences of characters, natural language, images). This addresses the part of the thesis statement in which we try to understand whether curriculum learning is beneficial only for certain types of data, or only for certain models. Moreover, we also considered datasets of various sizes, to investigate in what data regime the curriculum is most useful. Taking all the results together, we observed the following:

1. For all considered problems, curricula *can work* if chosen carefully. This is true for a variety of model architectures and data types.

2. The benefits can be both in terms of improvements in the final performance at the end of training (even with a generous number of training iterations), as well as faster training (i.e. models trained with a curriculum can reach the baseline performance much earlier).

3. The benefits gained with curriculum learning followed a consistent trend with respect the curriculum length, across our case studies: as we increase the curriculum length (i.e. the number of iterations until the model is allowed to train on the full dataset) the benefits (both in terms of performance and speed) first increase up to an optimal curriculum length, then start decreasing.

4. Curricula that are excessively long can in fact harm learning, making it slower to train, overall requiring more iterations to reach the baseline performance.

5. For easy problems (e.g., addition digit by digit in Section 4.2), curriculum learning is most useful with less training data, since the baseline models can learn well on their own when enough data is provided.

6. For more difficult problems (e.g. machine translation in Section 4.4), we still obtained performance and speed gains even when using the entire available training dataset.

In Chapter 6, we revisit some of these experiments, as well as new input-space curricula, and attempt to understand *why* we observed these effects.

5 CURRICULUM IN TASK SPACE

5.1 OVERVIEW

As discussed earlier, curriculum learning refers to training strategies that learn a difficult task by pretraining on a series of auxiliary learning goals of increasing difficulty. However, which components of the model or the training procedure are modified in order to derive easier auxiliary tasks can vary significantly across different approaches. Most efforts are focused around scheduling the order in which training data is presented to learner (similar to the work presented in Chapter 4), and rely on the assumption that the provided training datasets contain examples of varying degrees difficulty. Such strategies are particularly appropriate for some domains such as machine translation, where we can easily assume that some training examples are easier than others.

However, we argue that for many common learning tasks, the errors that a model makes can be mostly attributed to the difficulty of the learning task itself, and less so to the difficulty of specific examples. To use a concrete example, in classification tasks the errors that the model makes may be due to the *similarity of the classes* being considered (e.g., it may be harder to distinguish between a cat and a dog than between a mammal and a reptile), rather than the absolute difficulty of a sample independent of its class. To exemplify this, Figure 5.1 shows the confusion matrix of a convolutional neural network (CNN) classifier trained on the popular CIFAR-10 dataset (Krizhevsky, Hinton, et al., 2009). This confusion matrix shows that the errors that our model makes are not uniformly distributed among all pairs of classes. Instead, they are mostly dominated by a select few class pairs that are difficult to distinguish (e.g., `dog` and `cat`). Moreover, certain classes like `automobile` are mainly confused with only a few other classes, suggesting that a sample is in many cases difficult to classify correctly because of its similarity to a few specific other classes, rather than because of being an inherently difficult image (e.g., because the input image has a lower signal-to-noise ratio). Therefore, in such cases it may be beneficial to consider the difficulty of classes—rather than that of the data samples—to more effectively perform curriculum learning.

Another potential disadvantage of input space curricula is that training datasets usually contain only examples for the difficult target task (e.g., distinguishing between multiple different species of animals), but no examples at all for easy intermediate goals (e.g., distinguishing between mam-

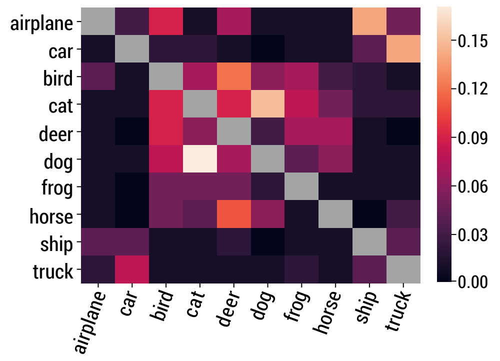


Figure 5.1: Confusion matrix for a CNN classifier on the CIFAR-10 dataset. Each element at position (i, j) indicates the ratio of times the model wrongly classifies an image as class j instead of the correct class i. The diagonal elements have been removed for visualization purposes.

mals and reptiles). For example, this is true for ImageNet (Russakovsky et al., 2015), a popular image classification dataset. In these situations, we would ideally like for a system to be able to break a difficult learning task down into a sequence of easier sub-tasks that better facilitate learning. We would further like for the system to be able to do this without requiring any additional human supervision. This motivates the design of curriculum algorithms that operate on the learning tasks themselves, rather than on the order in which training data is presented to the learner.

For such scenarios, we look at curriculum learning strategies in *task space* (or in *output space*), which we introduced in Section 2.2, and which gradually change the learning task $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. Thus, the auxiliary functions $f_{\theta_1}, ..., f_{\theta_K}$ learn to perform increasingly more difficult tasks $\mathcal{T}_1, ..., \mathcal{T}_K$, transferring the knowledge acquired for performing task $\mathcal{T}_k$ to perform $\mathcal{T}_{k+1}$.

However, we should make an important distinction between curriculum methods in task space for single task learning versus multitask learning.

single task learning. In this setting, the goal is to learn a single function $f : \mathcal{X} \rightarrow \mathcal{Y}$. Therefore, a curriculum in task space needs to create a series of auxiliary tasks (which are not provided) as intermediate goals. Perhaps surprisingly, this idea has been underexplored in machine learning. This category contains only a handful of approaches (Han and Myaeng, 2017; Saxena et al., 2019; Dogan et al., 2020; Ganesh and Corso, 2020), which we discussed in more detail in Section 3.3. In Section 5.2, we propose such a curriculum strategy in task space that shows considerable benefits for image classification tasks.

multitask learning. In this setting, the goal is to learn multiple functions, $f_1 : \mathcal{X}_1 \rightarrow \mathcal{Y}_1, ..., f_M : \mathcal{X}_M \rightarrow \mathcal{Y}_M$ that are in some sense related. Instead of training each function in isolation, multitask learning approaches take advantage of the relationships between the tasks and train them jointly (Caruana, 1997; Ruder, 2017). Common strategies for sharing information between the tasks include sharing a part of the model architecture, or by enforcing some constraints between the tasks. In standard training (i.e., without a curriculum), the tasks are usually learned simultaneously, by sampling batches alternatively from each task. With curriculum learning, we can alter the order in which the tasks are trained, prioritizing the easier tasks first. While this is more common in the area of reinforcement learning (which we are not covering in this thesis), there have also been a few attempts in supervised learning (Pentina et al., 2015; Li et al., 2017a; Murugesan and Carbonell, 2017; Guo et al., 2018a). We also consider this setting in Section 5.3.

In what follows, we propose curriculum learning approaches for the two scenarios above, single task and multitask learning.

5.2 COARSE-TO-FINE CURRICULUM LEARNING

In this section we present a novel algorithm for performing curriculum learning *on the output space of a model*, rather than on its input space. The methods and results discussed in this section can also be found in our publications at Stretcu et al. (2020) and Stretcu et al. (2021). Our algorithm is targeted at classification problems and allows learners to set their own easier goals, towards learning to solve a difficult classification problem. The intuition is that learners may benefit from learning to classify labels in stages, starting with coarse-grained concepts (e.g., learning to distinguish between `animal` and `object`), before moving on to more fine-grained concepts (e.g., `dog`, `cat`, `car`, `truck`).

This idea was inspired by human learning. For example, when a baby encounters a dog for the first time, her parents teach her that it is simply a "dog," rather than specifying its breed. Only later on, they start helping her distinguish between different dog breeds. Aside from the human teaching strategies, there is also ample evidence in neuroscience that the human ability to understand concepts and infer relationships among them is acquired progressively. Warrington (1975) was amongst the first to suggest that children first learn abstract conceptual distinctions, before progressing to finer ones. Subsequent studies have also found evidence consistent with this coarse-to-fine progression (Mandler, 1992; Mandler and McDonough, 1993; Mandler, 2000; Pauen, 2002; McClelland and Rogers, 2003; Keil, 2013). Moreover, McClelland and Rogers (2003) found evidence that human learning for classification problems follows a coarse-to-fine order. In fact, the same authors also showed that semantic dementia causes cognitive degradation in the reverse order. Thus, we can think of human learning as being driven by a *curriculum* that is either explicitly provided by a teacher, or implicitly learned by the student.

In this line of work, we propose such a *coarse-to-fine curriculum* algorithm for ML systems. This algorithm is aimed at classification problems and enables learners to decompose difficult problems into sequences of *coarse-to-fine* classification problems, that improve learning for the original difficult problem. Our main goal is to answer the following questions:

– How can we automatically construct a sequence of learning tasks from *coarse*-grained to *fine*-grained?

– How can knowledge acquired by learning coarse-grained tasks transfer to fine-grained tasks?

– How does such a curriculum learning approach affect the generalization ability of a model?

The proposed algorithm allows us to answer these questions and can be applied to any classification problem without requiring any additional human supervision.

Our main contribution is a novel algorithm for curriculum learning that answers these questions and that can be applied to any classification problem without requiring additional human supervision, thus making it broadly applicable to many areas of machine learning. Furthermore, it is model-independent, and can thus be used to train arbitrary models. We perform an empirical evaluation on several established classification datasets and using several types of models, and show that it consistently helps boost performance. The gains are especially prevalent on classification problems with many labels. We further introduce this algorithm.

5.2.1 *Method*

The standard strategy for learning f_θ is to initialize θ using random values and iteratively update it by performing gradient descent on a loss function that is defined over $\mathbf{X}$ and $\mathbf{Y}$. In this work, we propose a different approach: we learn a series of auxiliary functions $f_{\theta_1}, f_{\theta_2}, \dots, f_{\theta_M}$, sequentially, where the final function f_{θ_M} corresponds to our target function, f_θ. These functions operate on the same input domain as f_θ, but the task they are learning is coarser, meaning that they each learn to classify samples into fewer classes than the function that comes after them. This means that f_{θ_1} is learning an easier task than f_{θ_2}, f_{θ_2} an easier task than f_{θ_3}, etc., up until f_{θ_M}, which is learning our actual target task.

Our method thus consists of two parts:

(i) deciding what the auxiliary tasks should be and providing a way to automatically generate them along with training data for them.

(ii) providing a way for each learned function to transfer its acquired knowledge to the next function in the chain.

We propose a solution for part (i) in Section 5.2.2, followed by a method for transferring knowledge in Section 5.2.3. An illustration of the proposed approach is shown in Figure 5.2.

5.2.2 *Generating Auxiliary Tasks*

Our main requirement for the auxiliary learning tasks is that they form a sequence of increasing difficulty. We posit that grouping similar classes into coarse clusters will lead to an easier classification task. But how can we automatically decide which classes are similar?

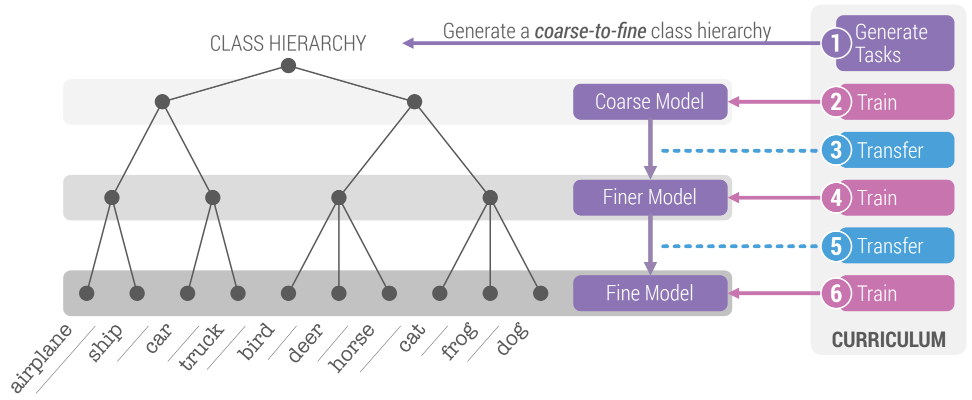


Figure 5.2: High-level illustration of the proposed algorithm.

MEASURING CLASS SIMILARITY. There exists a natural heuristic for gauging how similar classes are, and that is the confusion matrix of a trained classification model. However, it turns out that using this as our class similarity metric results in a degenerate case that we discuss in Appendix A.1.1. Thus, we consider another similarity metric that also encodes what the trained model might find confusing: the *class embedding similarity*. We define the embedding of a target class as the parameters of the final layer of the trained model, that are associated with that class. In a neural network setting, the final layer typically consists of a linear projection using a weight matrix $\boldsymbol{W} \in \mathbb{R}^{E \times K}$ that maps from the last hidden layer of size E to the predicted logits, for each of the K classes. We use $\boldsymbol{W}_{\cdot k}$, the k-th column of $\boldsymbol{W}$, to represent the embedding of class k. Thus, we can measure the distance between two classes, k_1 and k_2, as $d(k_1, k_2) = \cos(\boldsymbol{W}_{\cdot k_1}, \boldsymbol{W}_{\cdot k_2})$, where cos refers to the cosine distance, and their similarity as $1 - d(k_1, k_2)$.

DEFINING A COARSE CLASSIFICATION TASK. Given the original set of classes and their computed similarities, we expect that:

(i) grouping together similar classes to form coarse clusters, and

(ii) defining a new classification task where the goal is to predict the cluster instead of the specific class,

should result in an easier learning problem.

Using an example from the CIFAR-100 dataset, we could group the classes `willow_tree`, `oak_tree` and `pine_tree` into a single `tree` cluster, and all samples that belong to either of these three classes receive a new label associated with this cluster. The clusters allow us to define a new *coarser* classification problem that the auxiliary function $f_{\theta_{M-1}}$ is responsible for learning. Note that it is easy to automatically generate training data for the new task given the data of the original task: for every training example (x_i, y_i) in the original dataset, we replace the label y_i with the index of its cluster. We repeat this process for $f_{\theta_{M-2}}, \ldots, f_{\theta_1}$.

DEFINING COARSE-TO-FINE TASK SEQUENCES. It remains to show how we generate *sequences* of such auxiliary tasks with increasing difficulty, leading to the original task. Extending the previous idea of clustering classes, we consider a *hierarchical clustering* algorithm. An example of a class hierarchy is shown in Figure 5.2. If such a hierarchy forms a tree, then we can consider each level of the tree as a separate task, and form a sequence of tasks by iterating over these levels in top-down order. In Figure 5.2, the first auxiliary task, which corresponds to f_{θ_1} is a binary classification problem where the 2 classes correspond to the clusters {`airplane`, `ship`, `car`, `truck`} and {`bird`, `deer`, `horse`, `cat`, `frog`, `dog`}. The subsequent task, which corresponds to f_{θ_2}, further splits these clusters resulting in 4 classes. Intuitively, we expect the auxiliary tasks built in this manner to be sorted by difficulty. In other words, f_{θ_1} should be easier than f_{θ_2}, f_{θ_2} than f_{θ_3}, etc. This is because, using our example from Figure 5.2, f_{θ_3} would need to be able to tell whether a sample is a `car` or a `truck`, as opposed to f_{θ_2} which only needs to be able to tell if it is a `road vehicle`. Formally, we have a cluster hierarchy of depth M where the bottom level corresponds to the original classes. Training data for each of these tasks can be generated automatically using the approach described in the previous paragraph. The concrete algorithm is shown in Algorithm 5.1. These tasks will be trained in order, starting with the top level in the tree, and transferring acquired knowledge from each level to the next, using the approach described in Section 5.2.3.

Algorithm 5.1: Transform Labels

```
// This algorithm replaces the original sample labels with their corresponding
   cluster index.
Inputs: Original labels $\{y_i\}_{i=1}^{N}$.
        Set of clusters $\{c_{\hat{k}}\}_{\hat{k}=1}^{K}$, where
        each cluster $c_{\hat{k}}$ is a set of labels.
1 originalToNew ← Zero-initialized array of length $K$.
2 for $\hat{k} \leftarrow 1, \ldots, \hat{K}$ do
3     foreach Label $l \in c_{\hat{k}}$ do
4         originalToNew[l] ← $\hat{k}$
5 newLabels ← Zero-initialized array of length $N$.
6 for $i \leftarrow 1, \ldots, N$ do
7     newLabels[i] ← originalToNew[$y_i$]
Output: newLabels.
```

GENERATING CLASS HIERARCHIES. We still need to show how to automatically generate our hierarchies without human supervision. We already mentioned that we would like to use a hierarchical clustering algorithm using the similarity metric defined earlier. Most existing hierarchical clustering algorithms (Sibson, 1973; Defays, 1977; Kaufman and Rousseeuw, 2009) do not directly fit our setting because they typically output binary trees, where each non-leaf node has exactly two children clusters in the next level (e.g., `cat`, `frog`, `dog` would not be directly grouped together in Figure 5.2). This can be important if we want our curriculum to visit all levels because, in the worst case, the depth of generated hierarchies will be $\mathcal{O}(K)$, where K is the original number of classes. To address this, we adopt the *affinity clustering* algorithm proposed by Bateni et al. (2017), which is based on Borůvka's algorithm for minimum spanning trees. This algorithm has several desirable properties—including the fact that it is parallelizable—but the property that is most important for our approach is that the depth of the hierarchy is at most $\mathcal{O}(\log K)$, where K is the original number of classes. In summary, for every level l in the hierarchy, affinity clustering starts with the clusters from level $l+1$ and then joins each cluster with the one closest to it from the same level, thus forming a larger cluster. This means that in each level the size of the smallest cluster at least doubles relative to the next level. An overview of our algorithm for generating class hierarchies is shown in Algorithm 5.2.

Algorithm 5.2: Generate Class Hierarchy

```
// This algorithm generates a class hierarchy.
Inputs: Number of classes $K$.
        Training data $\{x_i, y_i\}_{i=1}^N$.
        Trained baseline model $f_\theta$.
1 Estimate class distance matrix $\mathbf{D}$ by computing pairwise cosine distances between
    the columns of the projection matrix in $\theta^{\text{pred}}$.
2 Compute the class hierarchy, $\mathcal{H}$, using affinity clustering with $\mathbf{D}$ distance matrix
    between samples.
3 clustersPerLevel $\leftarrow$ []
4 for $l \leftarrow 1, \ldots,$ depth($\mathcal{H}$) do
5     clustersPerLevel[$l$] $\leftarrow$ []
6     foreach $n \in \mathcal{H}$.nodesAtDepth[$l$] do
7         Create cluster $c$ by grouping the leaves of the sub-tree rooted at $n$.
8         clustersPerLevel[$l$].append(c)
Output: clustersPerLevel.
```

5.2.3 *Transferring Acquired Knowledge*

A direct approach to transferring knowledge from a trained classifier at one level of the hierarchy to the next is via the model parameters. We can initialize the parameters of $f_{\theta_{l+1}}$ based on the parameters of the trained f_{θ_l}, for $l \in \{1, \ldots, M-1\}$. However, since $f_{\theta_{l+1}}$ and f_{θ_l} make predictions for different number of classes, the number of parameters in θ_{l+1} does not directly match that of θ_l. One solution is to transfer only the subset of parameters that θ_{l+1} and θ_l can have in common (e.g., all except for the very last layer), and train from scratch the final prediction layer at each level of the hierarchy. We have attempted this approach and discuss it in detail in Appendix A.1.2. We refer to this as the *staged* variant of our curriculum learning algorithm. However, its main disadvantage is that the prediction layer, being re-initialized at each hierarchy level, can potentially lose valuable information—this is often called "representational collapse" in pre-training literature (e.g., Aghajanyan et al., 2020). Ideally, we would like to be able to reuse the knowledge captured by the prediction layer used for the coarse labels when initializing the prediction layer for finer labels. We can achieve this using the following strategy.

Let us assume that we use the same model for all stages, and that it is a model that predicts the probability of each class from our target task, while not being aware of the cluster hierarchy. When training the model at hierarchy level ℓ, we want to use that level's cluster assignments as the target labels (instead of the original classes), and we need to define a way to supervise the model with that information. Intuitively, when we are told that the label for an example is cluster k, we know that the underlying class is one that belongs to cluster k, but we do not know which one. Therefore, while we are doing maximum likelihood optimization during training (e.g., by minimizing the cross-entropy function), we propose to marginalize out the class variable which is unobserved. Given that all classes are mutually exclusive, this results in the following objective for f_{θ_ℓ} (i.e., the negative log-likelihood):

$$\mathcal{L}_\ell = -\sum_i \log \sum_{c \in \mathcal{C}_\ell(y_i)} \exp\{f_{\theta_\ell}(x_i)\}, \tag{5.1}$$

where $\mathcal{C}_\ell(y_i)$ is the cluster in level ℓ that class y_i belongs to. Using this formulation, the coarse-to-fine algorithm proceeds as follows (shown in more detail in Algorithm 5.3):

1. We start by initializing the parameters θ_1 randomly.
2. We learn f_{θ_1} using $\mathcal{L}_1$ as the loss function.
3. We initialize $\theta_2 = \theta_1$, and continue training using $\mathcal{L}_2$, to learn f_{θ_2}. We can do this because the function being learned is the same for all levels.
4. We iterate over this process until we go through all the levels of the hierarchy. That is, for each level ℓ, we initialize $\theta_\ell = \theta_{\ell-1}$ and learn f_{θ_ℓ} by optimizing $\mathcal{L}_\ell$.

Algorithm 5.3: Coarse-To-Fine Curriculum Learning

```
// This is an overview of the proposed continuous curriculum algorithm.
Inputs: Number of classes $K$.
        Training data $\{x_i, y_i\}_{i=1}^N$.
        Trainable baseline model $f_\theta$.
1  Train $f_\theta$ on the provided training data $\{x_i, y_i\}_{i=1}^N$.
2  bestEpoch ← epoch where $f_\theta$ reached its best validation accuracy
   // Assign the number of epochs to spend on the curriculum either manually, or
      automatically based on the baseline accuracy per epoch.
3  numEpochsCurriculum ← round(bestEpoch $* 0.9$)
4  clustersPerLevel ← GenerateClassHierarchy( $K, \{x_i, y_i\}_{i=1}^N, f_\theta$ )
5  M ← clustersPerLevel.length
6  numEpochsPerLevel ← round(numEpochsCurriculum / M)
   // Train the model at each level of the hierarchy.
7  originalLabels ← [1,...,K]
8  $\theta_0$ ← random()
9  for $l$ ← 0,...,M - 1 do
10     clusters ← clustersPerLevel[$l+1$]
11     newLabels ← TransformLabels( $\{y_i\}_{i=1}^N$, clusters)
12     Train $f_{\theta_{l+1}}$ using newLabels as the target labels, and summing the predicted
         probabilities for all labels in the same cluster to obtain the cluster probability,
         when computing the loss function. Please refer to our code repository for
         implementation details (e.g., how to make this calculation numerically stable).
Output: $f_{\theta_{[M]}}$.
```

This allows us to learn a single function f_θ by going through the levels of our class hierarchy sequentially and adjusting the objective function we are optimizing appropriately.

5.2.4 *Algorithm Properties*

HYPERPARAMETERS. The proposed approach introduces a single hyperparameter: the total number of epochs to be spent on the curriculum before training on the original classes (i.e., the last level in the cluster hierarchy), which we denote as T. We split this budget of T epochs equally among the levels of the hierarchy. In our experiments, we found that T is easy to set using a heuristic that, albeit not optimal, consistently results in an accuracy boost across all datasets: following Platanios et al. (2019), we set T to the number of epochs it takes for the baseline model to reach 90% of its best validation accuracy. Note that T set this way tends to be only a small fraction of the total number of training epochs (~5-10%).

COMPUTATIONAL COMPLEXITY. Let $\mathcal{C}$ be the computational complexity required to train the baseline model to convergence. The computational cost per training iteration of our coarse-to-fine model is approximately the same as that of the original model (label marginalization is implemented efficiently as a matrix multiplication). Also, even though a computational overhead could come from the need to train the baseline model first in order obtain the class similarity matrix, if one is not provided, in our experiments we observed that this can be avoided. Specifically, we observed that the relative class similarities were mostly consistent among different models (e.g., small CNN, WideResnet, Resnet), even when training on only a subset of training examples and for only a small number of epochs.

HUMAN SUPERVISION. Our algorithm does not require supervision for deciding on the class hierarchy or other measures of data difficulty. This is in contrast to many existing curriculum methods (e.g., Bengio et al., 2009; Spitkovsky et al., 2010; Platanios et al., 2019). However, prior knowledge can still be incorporated into our method by either replacing the task generation module with a provided hierarchy, or by providing a custom class dissimilarity matrix to the hierarchical clustering algorithm. We consider this flexibility an advantage of our approach.

RELATIONSHIP TO HIERARCHICAL CLASSIFICATION. The algorithm proposed in this paper bears some conceptual similarities to hierarchical classification. They both leverage a label hierarchy in order to obtain a better classifier. However, there is one fundamental distinction between hierarchical classification and curriculum learning more generally: hierarchical classification methods typically use the class hierarchy, not just during training, but also while making predictions at inference time (see Section 5.2.6 for more details). On the other hand, curriculum learning is purely a training strategy. Its goal is to provide a better means of training a model, without introducing any additional memory or computational cost when the model is deployed. We propose a curriculum learning approach. Nevertheless, it could be converted to a hierarchical classification method by creating an ensemble using the classifiers trained at each level, but this is beyond the scope of this paper. We discuss this more extensively in Section 5.2.6.

5.2.5 *Experiments*

We performed experiments on both synthetic and real datasets, using multiple different neural network architectures: a convolutional neural network with 3 convolution layers followed by a single densely connected layer (which we refer to as `CNN`), as well as the common larger models `Resnet-18`, `Resnet-50` (He et al., 2016) and `WideResnet-28-10` (Zagoruyko and Komodakis, 2016). Details on the hyperparameters we used and our training pipeline can be found in Appendix A.1.3.

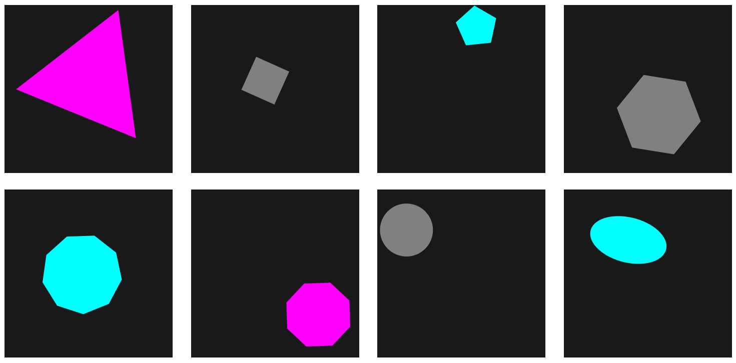


Figure 5.3: Example images from the Shapes dataset.

5.2.5.1 *Synthetic Datasets*

In order to study the properties of our method, we created a synthetic dataset that is easy to analyse, and where a natural coarse-to-fine curriculum might arise. The questions we investigate are: (i) how our method performs with varying amounts of training data, (ii) what the class hierarchy looks like, and (iii) how the class embedding distance metric compares to other metrics.

DATASET GENERATION. We refer to this dataset as Shapes. The inputs consist of 64×64 images depicting one of 10 geometrical shapes (circles, ellipses, and regular polygons with 3-10 vertices) in one of three colors (magenta, cyan, or grey) against a black background (see Figure 5.3). The dataset contains 50,000 images (5,000 per shape), out of which 10,000 are set aside for testing. The goal is to predict the shape and its color for each image (i.e., we have 30 classes). Our motivation for the design of this dataset is that shapes with similar colors and number of vertices look more alike and are thus more likely to confuse the model. Therefore, we expect our method to help in this setting.

RESULTS. For this dataset, we performed experiments using the CNN architecture. Our results, shown in Figure 5.4, indicate that our method consistently outperforms the baseline. Furthermore, we observe that the curriculum method provides the biggest boost over the baseline in the middle regime, when there are not enough samples for the baseline to reach high accuracy, but there is enough to make it a sufficiently good learner that our curriculum learning algorithm can improve upon. These observations also agree with prior results showing that pre-training is most beneficial in problems where labeled data is scarce. To further understand where the gains are coming from, we also inspected the generated label hierarchies. The most common hierarchy generated during our experiments separates all shapes by color on the first level, and by shape similarity on the second

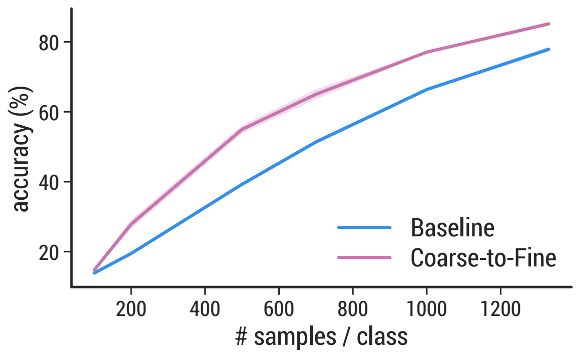


Figure 5.4: Accuracy mean and standard error (shown in tight shaded bands around the mean) for a CNN trained with and without our curriculum approach, averaged over 5 runs, on the Shapes dataset.

level (i.e., circles and ellipses are grouped together, and polygons are also grouped together). This is intuitive and compatible with what we might have manually constructed.

Distance metric evaluation. A natural question to ask is whether using the class embedding distance as a distance metric between classes is better than alternative approaches. For example, what if we force classes that are similar to be separated into different clusters early on, such as in Figure 5.6? Could this help the model recognise subtle differences better, by focusing on them early on? Another natural concern is whether the curriculum itself indeed matters, or the model can simply benefit from any clustering of the labels. This could be because assigning coarse group labels to samples still requires learning feature representations that distinguish these groups (e.g., edges, corners), no matter what the grouping is. To answer these questions, we tested our approach using different types of class distance matrices as input to the hierarchical clustering algorithm:

(i) the class confusion matrix[1] (Confusion)
(ii) a distance matrix defined as 1 minus the confusion matrix (ConfusionDist)
(iii) the class embedding distance used in the rest of our experiments (EmbeddingDist)
(iv) the class embedding similarity, defined as 1 minus embedding distance (EmbeddingSim)
(v) a random symmetric matrix, whose elements are drawn from a Gaussian distribution with mean 0 and standard deviation 1, which will lead to a random grouping of the classes (Random).

By using either ConfusionDist or EmbeddingDist as distances between classes we obtain a hierarchy similar to Figure 5.5, where the most similar classes

[1]In practice, we add its transpose to it, since a distance metric needs to be symmetric.

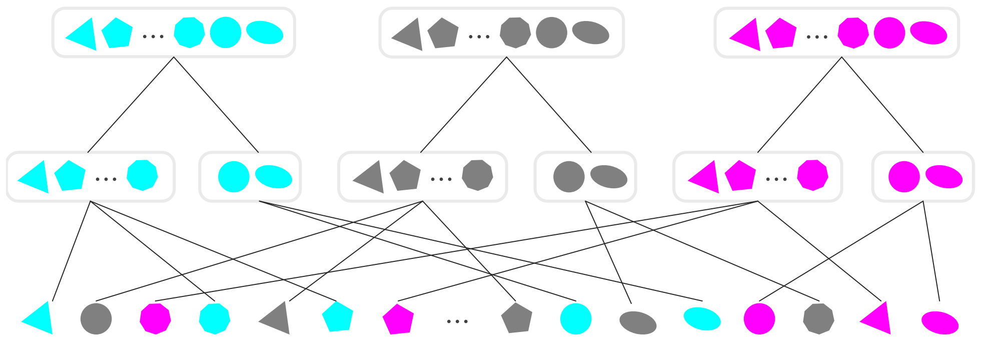


Figure 5.5: Hierarchy generated using EmbeddingDist as class distance measure. The hierarchy is created bottom-up, starting by first connecting the shapes that have the lowest embedding *distance* (i.e. those that are most similar) at the bottom of the hierarchy.

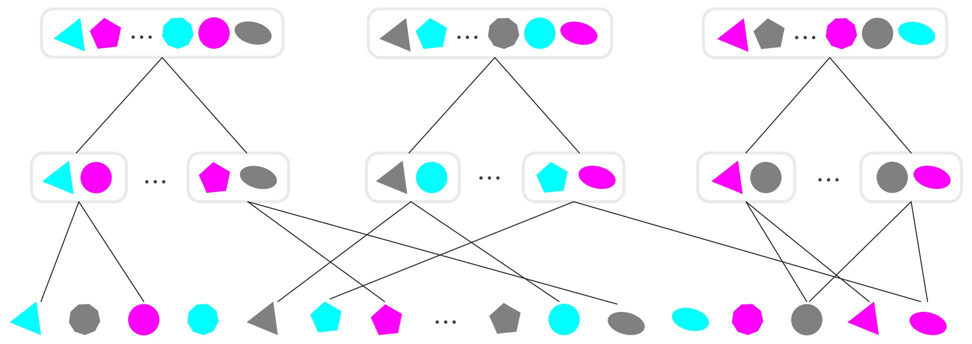


Figure 5.6: Hierarchy generated using EmbeddingSim as class distance measure. The hierarchy is created bottom-up, starting by first connecting the shapes that have the lowest embedding *similarity* (i.e. those that are most dissimilar) at the bottom of the hierarchy.

(e.g. gray circle and gray ellipse) are separated in the very last level of the hierarchy (i.e. later on during training, according to our coarse-to-fine curriculum). On the contrary, reverting these distances, and using Confusion or EmbeddingSim as inputs to the clustering algorithm, leads to a hierarchy as shown in Figure 5.6, where the most similar classes are separated at the first level of the hierarchy.

We evaluate these different approaches and present the in Table 5.1. We observe that Random hurts performance, as do the metrics that group dissimilar classes early on. On the other hand, EmbeddingDist and ConfusionDist, which group the most similar classes first, both result in accuracy gains, with the former resulting in the largest gain. This suggests that using arbitrary hierarchies along with our curriculum is not sufficient; *the actual choice of class hierarchy matters*.

Table 5.1: Results on Shapes with 500 samples per class. We show the accuracy mean and standard error of the baseline, our curriculum approach, and their difference (calculated separately per run and then averaged).

Distance Metric	**Accuracy**		
	Baseline	**Curriculum**	**Difference**
ConfusionDist	39.36±0.52	52.72±0.98	13.49 ± 1.03
EmbeddingDist	39.36±0.52	54.96±1.37	15.70 ± 1.34
Confusion	39.36±0.52	39.59±1.32	0.04 ± 1.71
EmbeddingSim	39.36±0.52	34.91±2.02	-4.46 ± 1.84
Random	39.36±0.52	39.00±1.33	-0.32 ± 1.85

5.2.5.2 *Real Datasets*

We performed experiments on the popular CIFAR-10, CIFAR-100 (Krizhevsky, Hinton, et al., 2009), Tiny-ImageNet (Li et al., 2015) and ImageNet (Russakovsky et al., 2015) datasets. The CIFAR-100 dataset contains labels at two levels of granularity: the original 100 classes, as well as 20 coarse-grained classes. Tiny-ImageNet is a subset of the larger ImageNet dataset that contains 200 categories and only 500 training examples per category. This low ratio of samples per class is what makes it an interesting test case for our method. Dataset statistics are shown in Table 5.2. The data and specific train/test splits for CIFAR-10, CIFAR-100, and CIFAR-100 Coarse were obtained from the `tensorflow_datasets` package [2] from Tensorflow. For Tiny-ImageNet, we used the provided train/validation/test splits as Li et al. (2015).

Table 5.2: Statistics for the classification datasets used in our experiments.

Dataset	**# Classes**	**# Train**	**# Test**
Shapes	30	40,000	10,000
CIFAR-10	10	50,000	10,000
CIFAR-100 Coarse	20	50,000	10,000
CIFAR-100	100	50,000	10,000
Tiny-ImageNet	200	100,000	10,000
ImageNet	1000	1,281,167	50,000

VARYING THE AMOUNT OF TRAINING DATA. Similar to our experiments on synthetic data, we compare the baseline method with our curriculum-based approach, while varying the amount of labeled examples. Our results, reported in Table 5.3, show that our approach is able to boost baseline performance for all models and across all datasets. As expected, the accuracy gains are most significant when we have a large number of labels. A more controlled setting for observing this effect is to compare the gains on CIFAR-100 and CIFAR-100 Coarse, where the input images and the number of samples are similar, but where one dataset contains more fine-grained labels. Moreover, when comparing results on the same dataset between the smaller

[2] https://www.tensorflow.org/datasets/catalog/

Table 5.3: Results on real datasets, showing the accuracy mean and standard error for the baseline model, computed over 5 runs, as well as the accuracy gain achieved by the our curriculum approach, computed per run and then averaged.

Model	Dataset	#Train	Baseline Accuracy	Curriculum Accuracy Gain (%)
CNN	CIFAR-10	50k	70.92 ± 0.37	0.69 ± 0.32
	CIFAR-10	20k	64.66 ± 0.53	1.28 ± 0.60
	CIFAR-10	10k	59.52 ± 0.35	1.24 ± 0.46
	CIFAR-10	5k	53.64 ± 0.19	1.57 ± 0.39
	CIFAR-100 Coarse	50k	49.63 ± 0.35	1.22 ± 0.38
	CIFAR-100 Coarse	20k	42.04 ± 0.29	1.84 ± 0.51
	CIFAR-100 Coarse	10k	36.61 ± 0.19	1.77 ± 0.56
	CIFAR-100 Coarse	5k	31.80 ± 0.28	1.38 ± 0.22
	CIFAR-100	50k	35.87 ± 0.23	3.31 ± 0.59
	CIFAR-100	20k	27.83 ± 0.34	2.27 ± 0.37
	CIFAR-100	10k	21.96 ± 0.49	2.67 ± 0.68
	CIFAR-100	5k	17.20 ± 0.20	1.92 ± 0.24
	Tiny-ImageNet	100k	21.94 ± 0.19	2.73 ± 0.49
	Tiny-ImageNet	50k	16.33 ± 0.32	3.06 ± 0.33
	Tiny-ImageNet	20k	10.16 ± 0.22	2.02 ± 0.34
	Tiny-ImageNet	10k	7.38 ± 0.11	1.14 ± 0.19
Resnet18	CIFAR-100	50k	76.11 ± 0.20	1.08 ± 0.12
	CIFAR-100	20k	61.24 ± 0.21	2.73 ± 0.41
	CIFAR-100	10k	46.01 ± 0.91	4.61 ± 0.40
	CIFAR-100	5k	20.98 ± 0.35	2.32 ± 0.97
Resnet50	CIFAR-100	50k	77.21 ± 0.40	2.20 ± 0.53
	CIFAR-100	20k	63.31 ± 0.38	0.52 ± 0.45
	CIFAR-100	10k	51.21 ± 0.22	0.39 ± 1.01
WRN-28-10	CIFAR-100	50k	80.10 ± 0.20	0.55 ± 0.13
	CIFAR-100	10k	58.72 ± 0.38	1.29 ± 0.35
	CIFAR-100	5k	43.77 ± 0.98	2.49 ± 0.90
	CIFAR-100	1k	14.87 ± 0.14	0.54 ± 0.56

and the larger models, we observe that the gains are larger for models with lower baseline performance, which is expected given that in these cases there is more room for improvement. Interestingly, the `WideResnet-28-10` performance on CIFAR-100 follows the same trend as our observations on the shapes dataset: we see the largest gains in the middle regime (neither too much nor too little training data).

INSPECTING THE CURRICULUM. To understand how our curriculum has been trained, we show the generated class hierarchy for CIFAR-10 in Figure 5.2, and for CIFAR-100 in the list below. Most clusters are intuitive, matching our expectation based on semantic similarity (e.g., `{crab, lobster}`, `{bicycle, motorcycle}`), but there are a few interesting examples (e.g., `{lamp, cup}`, `{skyscraper, rocket}`) that are based on visual similarity as interpreted by the model. The latter examples stress the importance of automatically generating hierarchies based on what the model itself finds confusing, rather than using hand-crafted ones.

Generated label hierarchy for CIFAR-100:

- <u>Level 1:</u>

Cluster 1 :	apple, pear, sweet_pepper, orange, aquarium_fish, sunflower, rose, orchid, tulip, poppy, crab, lobster
Cluster 2 :	baby, woman, girl, hamster, boy, man, fox, lion, snail, camel
Cluster 3 :	flatfish, ray, shark, turtle, dolphin, whale, bear, chimpanzee, skunk, cattle, dinosaur, elephant, seal, otter
Cluster 4 :	crocodile, lizard, shrew, beaver, porcupine, mushroom, kangaroo, tiger, leopard, trout, possum, wolf, mouse, raccoon, squirrel, rabbit
Cluster 5 :	lamp, cup, worm, chair, bed, table, keyboard, couch, snake, bicycle, motorcycle, can, telephone, television, bottle, wardrobe, bowl, plate, clock
Cluster 6 :	caterpillar, bee, butterfly, cockroach, spider, beetle
Cluster 7 :	willow_tree, forest, oak_tree, palm_tree, pine_tree, maple_tree, skyscraper, rocket, tractor, train, tank, castle, bridge, house, streetcar, pickup_truck, bus, lawn_mower, mountain, cloud, road, sea, plain

- <u>Level 2:</u>

Cluster 1 :	apple, pear, sweet_pepper, orange
Cluster 2 :	aquarium_fish, sunflower, rose, orchid, tulip, poppy
Cluster 3 :	bowl, plate, clock
Cluster 4 :	castle, bridge, house
Cluster 5 :	streetcar, pickup_truck, bus, lawn_mower
Cluster 6 :	fox, lion, snail, camel
Cluster 7 :	skunk, cattle, dinosaur, elephant
Cluster 8 :	mountain, cloud, road, sea, plain
Cluster 9 :	crab, lobster
Cluster 10 :	crocodile, lizard
Cluster 11 :	lamp, cup
Cluster 12 :	flatfish, ray, shark, turtle, dolphin, whale
Cluster 13 :	baby, woman, girl, hamster, boy, man
Cluster 14 :	willow_tree, forest, oak_tree, palm_tree, pine_tree, maple_tree
Cluster 15 :	mushroom, kangaroo, tiger, leopard, trout
Cluster 16 :	possum, wolf, mouse, raccoon
Cluster 17 :	seal, otter
Cluster 18 :	squirrel, rabbit
Cluster 19 :	skyscraper, rocket
Cluster 20 :	tractor, train, tank
Cluster 21 :	bear, chimpanzee
Cluster 22 :	shrew, beaver, porcupine
Cluster 23 :	worm, chair, bed, table, keyboard, couch, snake
Cluster 24 :	caterpillar, bee, butterfly
Cluster 25 :	cockroach, spider, beetle
Cluster 26 :	bicycle, motorcycle
Cluster 27 :	can, telephone, television, bottle, wardrobe

- <u>Level 3:</u>
  Each class has its own cluster.

Moreover, we show the progression of the accuracy per epoch, with and without curriculum, in Figure 5.7,

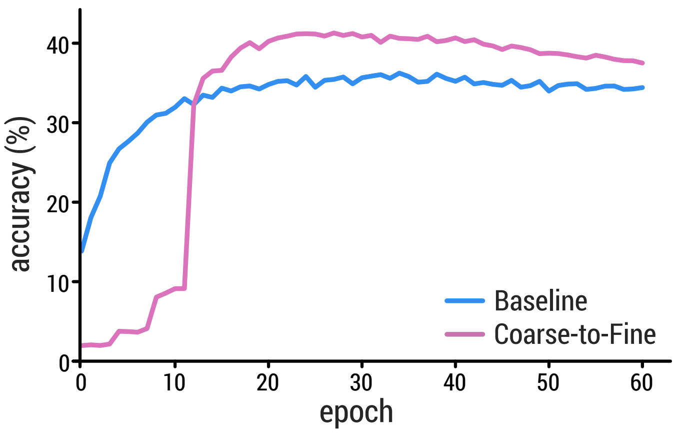


Figure 5.7: Test accuracy per epoch on the CIFAR-100 using a CNN. The curriculum here had 3 levels, all visited in the first 11 epochs, which is the time it took the baseline to reach 90% of its accuracy.

comparing to other approaches. We also compare our approach to other related methods:

1. a recent method by Saxena et al. (2019) which is, to the best of our knowledge, the only other existing curriculum approach operating in label space. This method also supports using a curriculum in both the label space and the input space simultaneously. We refer to the two variants of this method as DP-L (label space) and DP-LI (label and input space).

2. a recent hierarchical classification method (Wan et al., 2021), termed NBDT, that uses a label hierarchy generated automatically, similar to our EmbeddingSim.

3. the popular self-paced learning (SPL) method of Kumar et al. (2010), which allows us to compare with an established input space curriculum approach.

4. a multitask learning approach which trains in parallel all levels of the hierarchy (Multitask). This is meant to test if training the tasks *sequentially* (as given by the curriculum) matters, or having a contribution from each of them throughout training is enough.

Note that there is no standardized evaluation setting for curriculum learning methods, and thus most of the published approaches use their own custom setup. For a fair comparison, we replicated the setup from NBDT when training our approach (including the `Resnet` and `WideResnet` architectures, learning rate and other hyperparameters), and implemented other methods (SPL and Multitask) from scratch where code was not available. We also implemented our `CNN` architecture within the published code repositories of

Table 5.4: Results on real datasets, showing the accuracy mean and standard error for the baseline model, computed over 5 runs, as well as the accuracy gain achieved by the various curriculum approaches, computed per run and then averaged. The missing numbers are due to the fact that we were only able to run competing methods using the CNN, due to limited computational resources. For the larger models, we report the numbers published in the respective papers, and do not include standard errors as they were not reported. CIFAR-100 C refers to the coarse version of the CIFAR-100 dataset.

Model	Dataset	Accuracy (%)	Accuracy Gain (%)					
		Baseline	SPL	DP-L	DP-LI	Multitask	NBDT	C2F (Ours)
CNN	CIFAR-10	70.92 ± 0.37	-0.04 ± 0.19	0.26 ± 0.20	0.53 ± 0.30	0.12 ± 0.25	0.04 ± 0.38	0.69 ± 0.32
	CIFAR-100 C	49.63 ± 0.35	-0.27 ± 0.09	-0.65 ± 0.34	-0.75 ± 0.47	-0.08 ± 0.24	–	1.22 ± 0.38
	CIFAR-100	35.87 ± 0.23	0.94 ± 0.61	0.14 ± 0.25	0.26 ± 0.31	0.69 ± 0.15	0.45 ± 0.45	3.31 ± 0.59
	Tiny-ImageNet	21.94 ± 0.19	-0.97 ± 0.97	-0.05 ± 0.14	-0.08 ± 0.15	0.33 ± 0.33	0.22 ± 0.28	2.73 ± 0.49
Resnet18	CIFAR-100 C	84.57 ± 0.14	-0.21 ± 0.82	–	–	0.53 ± 0.02	–	0.69 ± 0.11
	CIFAR-100	76.11 ± 0.20	0.51 ± 0.45	–	–	0.01 ± 0.37	-1.19	1.08 ± 0.12
	Tiny-ImageNet	65.03 ± 0.09	-0.19 ± 0.15	–	–	-0.76 ± 0.42	-0.80	0.12 ± 0.14
Resnet50	CIFAR-100 C	84.68 ± 0.47	-1.21 ± 0.56	–	–	-0.97 ± 0.51	–	0.49 ± 0.41
	CIFAR-100	77.21 ± 0.40	-1.68 ± 0.55	–	–	0.43 ± 0.89	–	2.20 ± 0.53
WRN-28-10	CIFAR-100	80.10 ± 0.20	0.50 ± 0.41	–	0.70 ± 0.33	0.21 ± 0.12	2.77	0.55 ± 0.13
	Tiny-ImageNet	64.14 ± 0.15	–	–	–	–	2.52 [3]	1.01 ± 0.22

the methods DP-L and NBDT, and ran their training pipeline. We tried several hyperparameter configurations starting with the ones reported in the original publications (DP-L and DP-LI have 3 and 6 curriculum-specific hyperparameters, respectively). For the larger models we report the numbers from their respective publications, where available.

The results are shown in Table 5.4. Surprisingly, DP-L and DP-LI generally do not perform well for the small CNN. They do, however boost the accuracy of the larger model by 0.7%. The results for SPL are inconsistent; it sometimes improve the performance of the baseline and it sometimes does not (note that we have also attempted to tune its pace parameter, denoted as K in the original publication, in order to allow for a fair comparison). This result is interesting, because SPL has been shown to work well for other types of problems (though generally involving data of a sequential nature). One possible explanation for this result is that input space methods such as SPL are better suited for problems where the difficulty of training examples can be quantified (e.g., some examples involve longer sequences or are noisier than others), and less so for datasets like the ones we consider, where the difficulty is possibly given by the inter-class similarities. This is possibly why DP-L and DP-LI are also very similar in terms of performance. NBDT results in some improvements for CNN and WideResnet, but also incurs a significant loss in accuracy for Resnet-18, as also reported in the original paper. Multitask generally performs similarly to or slightly better than the baseline, suggesting that there is value in adding a loss function contribu-

[3] This difference is with respect to our baseline accuracy of 64.14%, but the difference would be -0.99% relative to their own baseline, which we could not reproduce. Similarly, for CIFAR-100, the difference relative to their baseline is 0.78%.

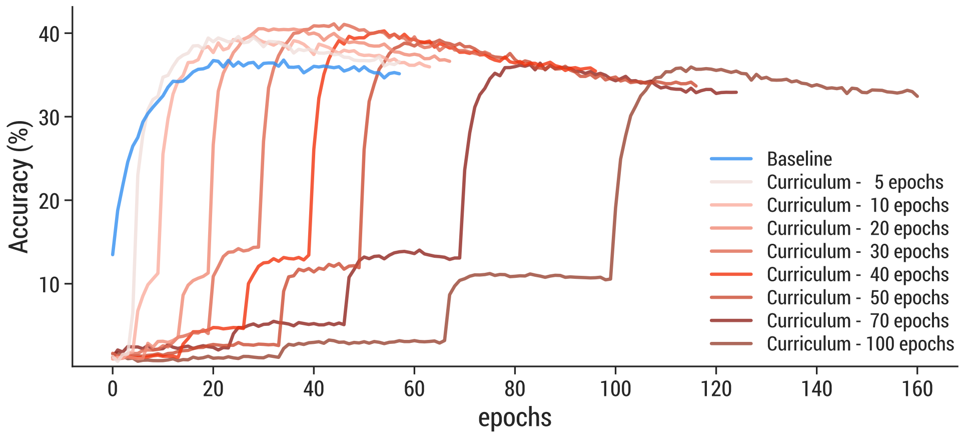


Figure 5.8: Accuracy of our coarse-to-fine curriculum strategy with different curriculum lengths on CIFAR-100.

tion for each level of label granularity. However, our approach results in the largest improvements, suggesting that learning these tasks *sequentially* in a coarse-to-fine order, is important.

HYPERPARAMETER SENSITIVITY. We evaluate the sensitivity of our curriculum approach to the length of the curriculum. In Figure 5.8, we show the accuracy of the CNN on the CIFAR-100 dataset, using various curriculum lengths. For a fixed curriculum length (in number of epochs), we divide it equally among the hierarchy levels. Once the final hierarchy level is reached, we train until the validation accuracy has not improved for the last 50 epochs. As can be seen in Figure 5.8, this threshold is enough for both the baseline model and the curriculum models to reach their best point. The results suggest that there is an optimal middle curriculum length: too few epochs and the coarse hierarchy levels do not have enough time to train and provide good initial points for the next level; too many epochs and the auxiliary hierarchy levels start overfitting, and their performance starts dropping before we move on to the next level. However, our curriculum method is robust overall, because despite the large range of curriculum lengths we used across all experiment runs, the performance of the curriculum trained model consistently outperformed or at least matched the baseline model performance.

5.2.6 *Related Work*

Most prior work in curriculum learning focuses on the notion of example difficulty, rather than task difficulty (e.g., Bengio et al., 2009; Jiang et al., 2015; Guo et al., 2018b; Jiang et al., 2018; Wang et al., 2018; Zhou and Bilmes, 2018). The difficulty of the examples is estimated based on problem-specific rules (e.g., sentence length in natural language processing; Bengio et al., 2009; Platanios et al., 2019) or based on the progress of the learner (e.g., the loss on each sample in SPL; Kumar et al., 2010; Jiang et al., 2015). Using the sample difficulties, these methods then decide when a sample should be shown to the model, starting with the easy ones first. Perhaps most related to our work is the work of Saxena et al. (2019), where the logits predicted for each training example are scaled by a corresponding class weight, which is learned together with the model parameters. While this can be seen as a form of curriculum in output space, the core idea is different.

There also exist a few methods that consider curricula in task space in the context of multitask learning. For example, Pentina et al. (2015) propose a multitask learning approach where a set tasks are learned one at a time, as opposed to jointly, by sharing knowledge between subsequent tasks, instead of solving all of them jointly. Other curriculum approaches for multitask learning include Guo et al. (2018a) and Sarafianos et al. (2018). However, multitask learning is different than our setting because: (i) the tasks are provided, rather than being automatically generated, and (ii) the goal is typically to perform well on all of the tasks and not just one of them.

A line of work related to curriculum in *output* space and which can be viewed as a curriculum in *task* space is in the area of reinforcement learning, where agents are trained to achieve incrementally more difficult goals (e.g., Florensa et al., 2017; Narvekar et al., 2017; Svetlik et al., 2017; Sukhbaatar et al., 2018). While these approaches do alter the target task, the setting is very different from ours and therefore these methods could not be directly translated to the supervised classification regime.

The idea of solving tasks in a coarse-to-fine order has previously been explored in computer vision and signal processing (e.g., for object detection and recognition Fleuret and Geman, 2001; Amit et al., 2004; Moreels and Perona, 2005; Ren et al., 2018), head pose estimation (Wang et al., 2019c), or more general computer vision tasks (Sahbi and Boujemaa, 2002; Lu et al., 2011; Zambanini and Kampel, 2012; Mazić et al., 2015; Wu et al., 2019). Similarly, coarse-to-fine ideas have also been used for various tasks in natural language processing (e.g., Dong and Lapata, 2018; Lee et al., 2018; Yao and Al-Dahle, 2019). Our approach is different in that it is widely applicable; it does not depend on the problem space at hand, but can rather be applied as-is to any classification problem. In a different line of work, Srivastava and Salakhutdinov (2013) proposed a probabilistic learning method using a class hierarchy. A tree-based prior over the last layer of a neural network is used to encourage classes closer in the hierarchy to have similar parameters. Moreover, Bilal et al. (2017) obtain coarse clusters of classes from the confusion matrix and introduce extra branches to a neural network architecture during training that classify the coarse classes simultaneously during training (similar to our Multitask baseline, but the branches are introduced at custom positions in the model).

Finally, some of the ideas in our paper bear some resemblance to hierarchical classification (e.g., Bennett and Nguyen, 2009; Ramaswamy et al., 2015; Ramírez-Corona et al., 2016; Xu and Geng, 2019). As discussed in Section 5.2.4, there is a fundamental difference between our approach, which is purely a training algorithm, and the various HC methods (that also leverage the hierarchy during inference, such as NBDT; Wan et al., 2021). For example, some HC methods propose special architectures that can make predictions at all levels of the hierarchy and combine them (e.g., Wehrmann et al., 2018), or use the predictions for coarse labels as inputs to a classifier for finer labels (e.g., Bennett and Nguyen, 2009). We provide a more detailed comparison with different HC approaches in Appendix A.1.4. However, the important take-away is that our method is: (i) general purpose, meaning that it can be used to train any baseline model without requiring a special architecture, and (ii) does not affect the model during inference, meaning that it does not require any extra memory or computation.

5.2.7 *Discussion*

In this section, we proposed a curriculum learning algorithm in *task space* aimed at single-task classification problems. Our approach:

(i) breaks down complex classification tasks into sequences of simpler tasks, and

(ii) goes through these tasks in order of increasing difficulty, training a classifier for each task and transferring the acquired knowledge between the trained classifiers.

We showed that our approach achieves significant performance gains on both synthetic and real data, using multiple neural network architectures. Our approach shows especially great promise for settings with low amounts of training data, where machine learning models are prone to overfitting, and pretraining (even on the same dataset) can bring large boosts in performance. Finally, our approach is purely a training strategy, and does not incur any additional memory or computational costs during inference.

5.3 curricula for compositional multitask learning

Multitask learning refers to a problem setting in which we want to jointly learn multiple related tasks (Caruana, 1997). It is a very active research area in machine learning, with numerous methods proposed in the past couple of decades (e.g., Xue et al., 2007; Collobert and Weston, 2008; Misra et al., 2016; Zhao et al., 2020), and with numerous uses across many application domains (e.g., Ji et al., 2009; Xu and Yang, 2011; Peng et al., 2017; Kendall et al., 2018). In this setting, the learning tasks are usually trained in parallel, for example by alternating between tasks in random order and updating the model with a batch of examples from each task. Therefore, it is natural to consider whether a curriculum would be useful in this case, and if there is any benefit in learning the tasks in order of difficulty. This problem has been investigated in prior work, such as Pentina et al. (2015), Guo et al. (2018b), and Sarafianos et al. (2018), where the models are applied to image classification tasks.

However, in this thesis we focus on a particular class of multitask problems: problems with *compositional tasks*. Since "composition" is an overloaded term used with different meanings, let us define what it means in our setting. Consider a multitask-learning setting where we aim to learn a set of functions $f_1, f_2, ..., f_K$. We assume that each of these functions can be expressed as a composition (in the mathematical sense, where $(f \circ g)(x) = f(g(x))$ for two functions f and g) of a small set of primitive functions that need to be learned, and that are shared among functions $f_1, f_2, ..., f_K$.

We chose to focus on compositional tasks, because this is a setting that is very common in the real world and yet very difficult for machine learning methods nowadays, and where we expect that curriculum learning can have a large impact. In the next sections, we consider two problems, in which the composition between the tasks is modeled either *explicitly* (i.e., the relationships between the tasks are explicitly used by the model architecture to improve its predictions) or *implicitly* (some difficult tasks implicitly require the model to be able to solve easier ones, and learning the easier tasks first seems to be beneficial despite the fact that task relationships are not explicitly modeled in the architecture). We propose some ideas of how curriculum learning can be applied in each of these settings, and we evaluate what kind of benefits we can expect.

5.3.1 *Learning Arithmetic using Explicit Task Composition*

In this section we illustrate an approach of combining curriculum learning with semi-supervised and self-supervised learning. We apply this on a learning setting very common in human learning—learning arithmetic—and which provides a textbook definition of compositional tasks.

In human learning, students are introduced to arithmetic operations in a specific order: first they are taught to count, then to add, then to multiply, etc. The operations are taught in this order not only because counting is easier than adding, which is easier than multiplying, and so on, but also because the compositional relationship between these operations provides additional benefits when taught in this order. Once a student has learned how to count, her teacher tells her that "adding x and y simply means counting y times starting at x". Similarly, once the student knows how to add, her teacher will explain that "multiplying x and y simply means adding x, y times". Using the relationships between these different learning tasks gives the students a means of verifying their answers (e.g., use addition to verify a multiplication result), as well as a means of "generating their own examples" for practicing on their own.

In this line of work, we replicate this setting using neural networks. The goal is to train a machine learning system to perform basic arithmetic operations (counting, addition, multiplication) using very little labeled data. We refer to the process of learning each operation as a separate *learning task*. Thus we operate in a *multitask learning* (MTL) setting. We represent the compositional relationships between the learning tasks *explicitly* in the architecture. We show that, by taking advantage of these relationships, and by training on these tasks in a particular order as driven by a curriculum, the system can take advantage of human-like strategies such as self-training, and perform better. In what follows, we introduce our data and experimental setting.

5.3.1.1 *Multitask Framework*

PROBLEM FORMULATION. We use the same problem formulation as introduced in Section 4.2 for performing addition digit-by-digit. However, in this case we have multiple functions that we want to learn, one for learning each operation: $f_{\mathrm{count}} : \mathcal{X}_{\mathrm{count}} \to \mathcal{Y}_{\mathrm{count}}$, $f_{\mathrm{add}} : \mathcal{X}_{\mathrm{add}} \to \mathcal{Y}_{\mathrm{add}}$, $f_{\mathrm{mult}} : \mathcal{X}_{\mathrm{mult}} \to \mathcal{Y}_{\mathrm{mult}}$. In what follows we use f_{op} to refer to any of these functions, where "op" stands for an arbitrary arithmetic operation.

All functions operate on the similar input domains, with addition and multiplication operating over pairs of integers, and counting takes a single integer input. The addition and multiplication functions estimate the sum and product of the provided input numbers, respectively, while counting es-

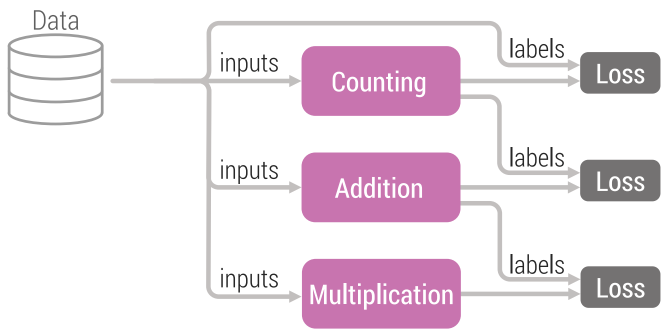


Figure 5.9: A self-supervised learning setting for learning arithmetic operations. The system is provided with labeled examples only for learning how to count. Once the model has learned how to count, it can self-label training examples for learning addition, using the compositional relationship between counting and addition. Similarly, once the model has learned how to add, it can self-label data for learning multiplication, etc.

timates the next consecutive number (i.e., for an input number x, it should predict $x+1$). All functions predict integer numbers. The inputs are provided to the corresponding model digit-by-digit, and the model also makes predictions digit-by-digit. Thus, the inputs and outputs are in fact *sequences of digits* of arbitrary length, instead of natural numbers. We formally define the input and output domains as:

$$\mathcal{X}_{\mathrm{count}} = \{(x_n)_{n\in\mathbb{N}} | x_n \in \{0,...,9\}\} \tag{5.2}$$

$$\mathcal{X}_{\mathrm{add}} = \mathcal{X}_{\mathrm{mult}} = \{((x_n^{(1)})_{n\in\mathbb{N}}, (x_m^{(2)})_{m\in\mathbb{N}}) | x_n^{(1)}, x_m^{(2)} \in \{0,...,9\}\} \tag{5.3}$$

$$\mathcal{Y}_{\mathrm{count}} = \mathcal{Y}_{\mathrm{add}} = \mathcal{Y}_{\mathrm{mult}} = \{(y_n)_{n\in\mathbb{N}} | y_n \in \{0,...,9\}\} \tag{5.4}$$

where $(x_n)_{n\in\mathbb{N}}$ represents a sequence of natural numbers of arbitrary length.

To train this system we are provided with labeled and unlabeled examples from each task. Let N^L_{op} and N^U_{op} refer the to the number of labeled and unlabeled samples for task op. For addition and multiplication, N^L_{op} can be 0.

BASELINE TRAINING. We train the baseline model only on the labeled dataset for each task. In a typical MTL fashion, we train the model by sampling batches of examples alternatively from each task, and updating the corresponding model parameters. We update the model parameters using the average cross-entropy loss for each predicted digit at every time step, similar to Section 4.2. That is, for a batch of samples of size $B < N^L_{\mathrm{count}}$, indexed by i:

$$\mathcal{L}^{\mathrm{labeled}}_{\mathrm{op}} = \frac{1}{B}\sum_i \sum_{j=1}^{n_i} \mathrm{cross\_entropy}(f_{\mathrm{count}}(x_i), y_{ij}). \tag{5.5}$$

where x_i and y_i are the input and output of labeled sample i of task op, n_i is the number of digits in y_i, and y_{ij} is the j-th digit of y_i.

CURRICULUM TRAINING. The curriculum model is trained by prioritizing certain tasks in the beginning of training, instead of alternating between all tasks uniformly. We start by training the system only on the counting task, using labeled data. When counting reaches a target validation accuracy threshold ($\mathtt{thresh}_{count}$), we allow the model to also train on the addition task. However, for addition we can use not only the labeled training set, but we can also self-label the available unlabeled data using the counting module. We describe how we perform the self-labeling in the paragraph below. Similarly, once the addition task reaches a target validation accuracy threshold ($\mathtt{thresh}_{add}$), we can use the addition module to self-label data for multiplication, and start training this task as well. In this work, we set the two thresholds manually, as a proof of concept that such a setup could work. For future work, it would be interesting to develop a curriculum approach that can automatically derive these thresholds. When multiple tasks are enabled for training, we alternate between the allowed tasks in random order and update one batch from each task at a time, as in standard multi-task learning. An illustration of the described self-training setting is shown in Figure 5.9.

SELF-LABELING. We can generate labels for the unlabeled data for addition and multiplication, using the already trained counting and addition modules, respectively, by taking advantage of the known compositional relationship between these operations:

- $a + b =$ count up starting at b, a times $= \underbrace{f_{count}(f_{count}(...(f_{count}(b))))}_{\text{a times}}$
- $a \times b =$ add b, a times $= \underbrace{f_{add}(b, f_{add}(b, ...(f_{add}(b, 0))))}_{\text{a times}}$

The algorithms for generating labels in this manner are presented in Algorithm 5.4 and Algorithm 5.5.

MODEL. We use the `LSTM` model introduced in Section 4.2 for performing addition digit by digit. However, in this case we have multiple such `LSTM` models, one for learning each operation. Concretely, for each pair of numbers that we want to add or multiply, we provide the two operands as input to the corresponding LSTM model one digit at the time (from least significant to most significant digits), and the model predicts the result, also digit by digit. For counting, the LSTM is provided with a single input number x digit by digit, and the model needs to predict $x + 1$, digit by digit.

Algorithm 5.4: Generating Labels for Addition using Counting

```
// This algorithm generates samples for addition using the counting module.
Inputs: Trained counting model $f_{count}$.
        Unlabeled data for addition $\{(x^{(1)}, x^{(2)})_i\}_{i=1}^{N^U_{add}}$, each sample consisting of
        two sequences of digits $x^{(1)}$ and $x^{(2)}$ .
1 $y \leftarrow$ []
2 for $i \leftarrow 1, \dots, N^U_{add}$ do
      // Count up starting at $x_i^{(1)}$, $x_i^{(2)}$ times.
3     $y_i \leftarrow x_i^{(1)}$
4     for iter $\leftarrow 1, \dots,$ integer$(x^{(2)})$ do
5         logits $\leftarrow f_{count}(x_i^{(1)})$
6         $y_i \leftarrow$ argmax(logits, axis=-1)
Output: $y$.
```

Algorithm 5.5: Generating Labels for Multiplication using Addition

```
// This algorithm generates samples for multiplication using the addition module.
Inputs: Trained addition model $f_{add}$.
        Unlabeled data for multiplication $\{(x^{(1)}, x^{(2)})_i\}_{i=1}^{N^U_{mult}}$, each sample
        consisting of two sequences of digits $x^{(1)}$ and $x^{(2)}$ .
1 $y \leftarrow$ []
2 for $i \leftarrow 1, \dots, N^U_{mult}$ do
      // Add $x_i^{(1)}$, $x_i^{(2)}$ times.
3     $y_i \leftarrow (0, 0, 0, \dots, 0)$
4     for iter $\leftarrow 1, \dots,$ integer$(x^{(2)})$ do
5         logits $\leftarrow f_{add}(x_i^{(1)}, y_i)$
6         $y_i \leftarrow$ argmax(logits, axis=-1)
Output: $y$.
```

5.3.1.2 *Training*

We train the system using stochastic gradient descent, with batch size 128, and update the model parameters using the mean cross-entropy loss over the predicted digits probability at every time step, similar to Section 4.2. We use the Adam optimizer (Kingma and Ba, 2015) with learning rate 0.001 and momentum 0.9. Similar to the previous experiments, we implemented our system using TensorFlow (Abadi et al., 2016).

5.3.1.3 *Data*

TRAIN. We generate 1,000 labeled and 10,000 unlabeled samples per task. Given our self-supervised setting, in principle we do not need any labeled data, except for learning to count. However, we decided to provide the system with a small amount of labeled data for each task, in order to be able to compare the curriculum approach with a baseline method, which otherwise could not be trained without supervision. In the training data, all input numbers have between $1-5$ digits.

TEST INTERPOLATION. We generate $1,000$ test instances from the same distribution as the training data, where the input terms consist of $1-5$ digits, which allows us to test the model's ability to *interpolate* between the training samples.

TEST EXTRAPOLATION. We generate $1,000$ test instances where the input terms consist of $6-10$ digits. This allows us to test the model's ability to *extrapolate* beyond the training distribution, on more difficult examples.

5.3.1.4 *Results*

We ran experiments with $\mathtt{thresh}_{\mathtt{add}} = \mathtt{thresh}_{\mathtt{mult}} = 90\%$ and we present the results in Figure 5.10. We evaluate the models in terms of exact match accuracy (i.e., each prediction is either awarded a score of 1.0 if all the digits are predicted correctly, or 0.0 otherwise).

As can be observed from the figure, counting is a very easy task, and the baseline quickly achieves 100% accuracy, both for interpolation and extrapolation. For addition, the baseline also achieves good results, with 95% accuracy in the interpolation case and around 80% accuracy for extrapolation. The curriculum approach starts later, waiting for counting to reach $\mathtt{thresh}_{\mathtt{add}} = 90\%$ accuracy, and then quickly outperforms the baseline, with 98% accuracy for interpolation and 87% accuracy for extrapolation. Thus the benefits are larger on out-of-distribution data.

For the multiplication task, the baseline achieves 1.5% accuracy in the interpolation case, while the curriculum approach obtains 4.0% accuracy. However, neither the baseline nor the curriculum can extrapolate in this case. These results suggest that multiplication is a difficult very difficult task for this type of model, and that we should be considering model architectures more suitable for this problem. In fact, there are active areas of research in machine learning focused on developing models that can perform multiplication (Trask et al., 2018; Madsen and johansen, 2020).

5.3.1.5 *Discussion*

In this case study we have seen how curriculum learning can be combined with other learning strategies such as semi-supervised learning and self-training. By taking advantage of the compositional relationships between tasks, and by applying a curriculum that keeps track of each task's prerequisites, we can improve the performance of the model by self-labeling unlabeled data. We obtained good results both for interpolation and extrapolation. However, the low accuracy for multiplication suggests that while

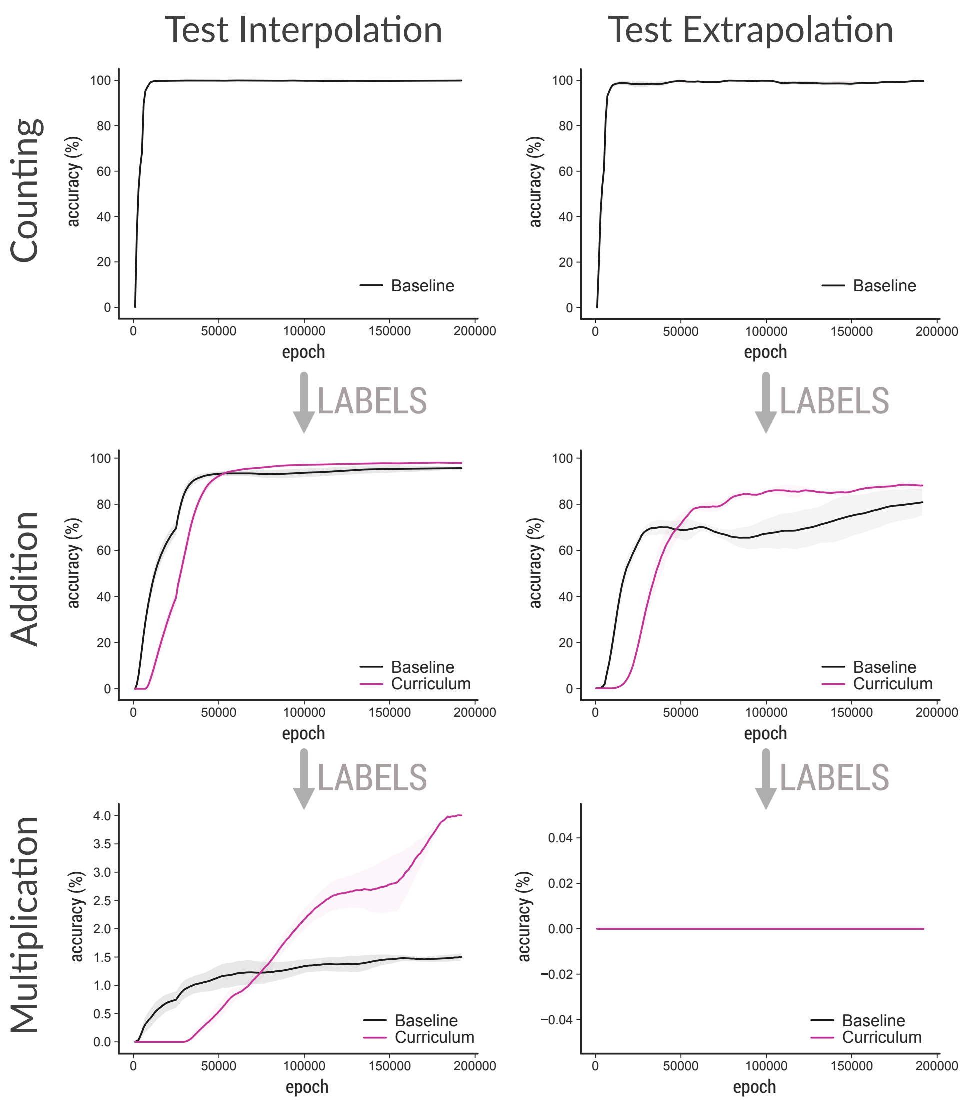


Figure 5.10: Results for semi-supervised learning model trained to perform arithmetic tasks (counting, addition, multiplication). We show the performance of two models—one trained without a curriculum ("Baseline") and one trained using curriculum learning in task space ("Curriculum")—evaluated in two settings: on an interpolation dataset ($1-5$ digits), and on a extrapolation dataset ($6-10$ digits). For counting we only show the baseline model, since we always train it in a fully-supervised way using the provided labeled dataset.

our strategy can bring some improvements, it is not a replacement for better architectures that can model this task better.

5.3.2 *Implicit Task Composition for Multimodal Image Understanding*

In the previous section we considered a multitask setting, where the compositional relationship between the tasks of interest was known, and we could *explicitly* represent it in our system. However, in some cases it is hard to know how to represent this composition explicitly. Nevertheless, even in this case curriculum learning can be used in a different way. We further illustrate this with another case study. We start from the multimodal language understanding problem we introduced in Section 4.5, where we trained a model to predict the agreement between images and their natural-language captions on the ShapeWorld dataset (Kuhnle and Copestake, 2017). In that case, we only considered one type of task: an `existential` task where the captions always specified the *existence* of a shape, or a color, or a shape-color combination (e.g., *"There is a red shape."*, *"A cross is blue."*), and applied an *input space* curriculum on the number of shapes in the image.

We now convert the problem into a multitask learning setting, by introducing more types of tasks. We then take advantage of the *implicit* task composition and design a *task space* curriculum to learn these tasks in order. We consider the following tasks:

– `existential`: similar to Section 4.5, as described above.

– `conjunctions`: consisting of conjunctions of `existential` tasks (e.g., *"There is a gray square and there is a gray ellipse."*).

– `disjunctions`: consisting of disjunctions of `existential` tasks (e.g., *"A shape is a yellow cross or there is a gray ellipse."*).

– `conjunctions & disjunctions`: consisting of a combination of conjunctions and disjunctions of `existential` tasks. Therefore, a model solving this task would need to learn the difference between *"and"* and *"or"*.

– `quantitative`: consisting of quantitative statements, using both numeric attributes (e.g., *"More than five shapes are red ellipses."*), as well as fractional descriptions (e.g., *"Exactly half the crosses are magenta."*).

– `spatial`: consisting of statements related to the position of objects (e.g., *"A magenta rectangle is to the right of a cross."*, *"A red shape is below an ellipse."*).

– `selection`: consisting of descriptions that identify and describe a specific object in the image. The entity can be identified through its absolute position, relative position, size, color, or any combination of these (e.g., *"The upper red shape is a rectangle."*, *"The magenta shape farther from the cyan cross is a cross.'*, *"The biggest green shape is an ellipse."*).

It is easy to notice that certain tasks depend on others. For instance, the `selection` task requires the model to be able to identify the existence of one or more objects with certain attributes (`existential`, `conjunctions`), but also to understand their relative positioning (`spatial`). However, the way the `selection` captions are phrased cannot necessarily be expressed in terms of a composition of `existential`, `conjunctions` and `spatial` captions. Instead, the composition may take place at a higher level of representation, and not directly on the model inputs. Therefore, we pose the following question: can a curriculum in task space, through which the tasks are learned in a particular order, help the model perform better, despite the fact that we do not explicitly model the task compositions? We refer to this means of taking advantage of the dependencies between tasks without explicitly representing the composition function as *implicit* task composition. We further analyse this empirically.

5.3.2.1 *Method*

We consider multiple learning settings:

1. single-task learning, where each task is trained in isolation, using a separate model for each task.

2. multitask learning, where parts of the model architecture are shared, and all tasks are trained in parallel, alternating between them in random order (sampled from a uniform distribution).

3. multitask curriculum learning, where parts of the model architecture are shared among tasks, and the tasks are trained in an order specified by a curriculum.

MULTITASK MODEL. In multitask learning, part of the model architecture is shared among the tasks. In our experiments, we chose to share the first two layers of the convolutional network (i.e., those performing the more low-level processing), as well as the word embeddings. The rest of the architecture is kept separate for each task, in order to allow it to adapt to the task requirements. This is illustrated in Figure 5.11.

BASELINE TRAINING. For both the single task and multitask models without curriculum we train the models in the following way. At every training step, we sample uniformly a task from all tasks. We then sample a batch of training samples from the chosen task, and update the model. We repeat for a fixed number of training steps, or until all tasks have converged.

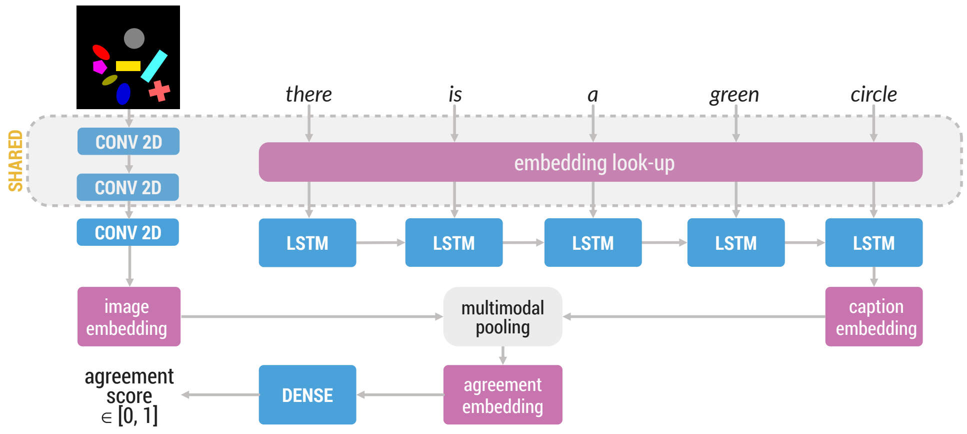


Figure 5.11: Model used for multitask learning on the ShapeWorld dataset. The gray area enclosed by a dashed line marks the part of the model architecture that is shared among tasks.

CURRICULUM TRAINING. Our proposed curriculum training algorithm consists of the following steps:

1. Decide the task dependencies. For each task, we define a set of *prerequisites*, consisting of those tasks that the model should learn (at least to some extent) before attempting the current task (e.g., `existential` could be a prerequisite for `conjunctions`). Thus, the task dependencies can be represented as a directed acyclic graph (DAG), as the one showed in Figure 5.12.

2. For each pair of tasks (t_1, t_2) where t_1 is a prerequisite of t_2, decide an accuracy (or any other meaningful metric) threshold, $\mathrm{thresh}_{t_1 \to t_2}$, such that the multitask model is allowed to train on task t_2 only after t_1 has reached $\mathrm{thresh}_{t_1 \to t_2}$ accuracy (e.g., start training on `conjunctions` when `existential` reaches 70% accuracy).

3. Start training. At every training step, sample uniformly a task from the *allowed tasks*. Sample a batch of training samples from the chosen task, and update the model.

4. Evaluate regularly (on a validation set, or even on the training set) and update the *allowed tasks*. A task is allowed during training once all its prerequisites have been satisfied.

5.3.2.2 *Data*

For each of the 7 tasks described earlier, we generate equal amount of training and test data, as follows. As before, all images contain $1-8$ shapes of different colors. For training we generate $10,000$ images, each associated

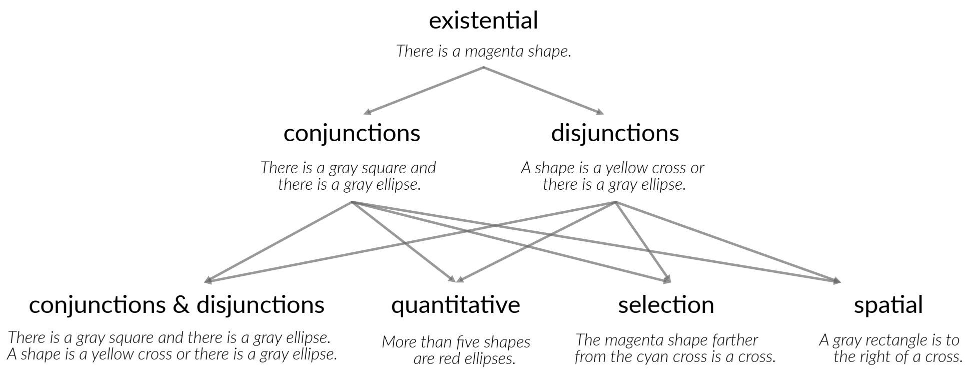

existential
There is a magenta shape.
conjunctions
There is a gray square and there is a gray ellipse.
disjunctions
A shape is a yellow cross or there is a gray ellipse.
conjunctions & disjunctions
There is a gray square and there is a gray ellipse. A shape is a yellow cross or there is a gray ellipse.
quantitative
More than five shapes are red ellipses.
selection
The magenta shape farther from the cyan cross is a cross.
spatial
A gray rectangle is to the right of a cross.

Figure 5.12: Example of task dependencies for a task space curriculum. An arrow between two tasks $t_1 \rightarrow t_2$ indicates that t_1 is a *prerequisite* of t_2, and thus t_1 should be prioritized before t_2.

with 5 different captions (i.e., $50,000$ training samples per task). We also generate $1,000$ images with 5 captions each for validation, per task. For testing, we generate $5,000$ images with 5 captions each, per task.

5.3.2.3 *Models and Training*

We use the same CNN-LSTM model architecture and sizes as described in Section 4.5, except part of the architecture is shared in the multitask learning case, as described earlier. We train all models using the Adam optimizer (Kingma and Ba, 2015) with learning rate 0.001, momentum 0.9, and batch size 512.

5.3.2.4 *Results*

We run experiments with single task learning, multitask learning without a curriculum, and multitask learning with 4 different curricula. The 4 curriculum DAGs and the corresponding accuracy thresholds are displayed in Figure 5.13. The results for all training settings on the test datasets are shown in Figure 5.14.

For the easier tasks such as existential, conjunctions, disjunctions, conjunctions & disjunctions, or quantitative, the single task and multitask models without curriculum perform on par, with accuracies between $70-75\%$. For the harder tasks such as spatial and selection, the accuracies are lower ($56-58\%$), but the multitask model is able to take advantage of the parameter sharing, and performs better than the single-task one.

The 4 curriculum settings perform overall better on all tasks, obtaining up to $6-7\%$ additional accuracy points. Interestingly, even the existential task, which is a prerequisite for all other tasks and thus does not benefit from pretraining on other easier tasks, performs better in the multitask curriculum setting. However, depending on the prerequisite requirements,

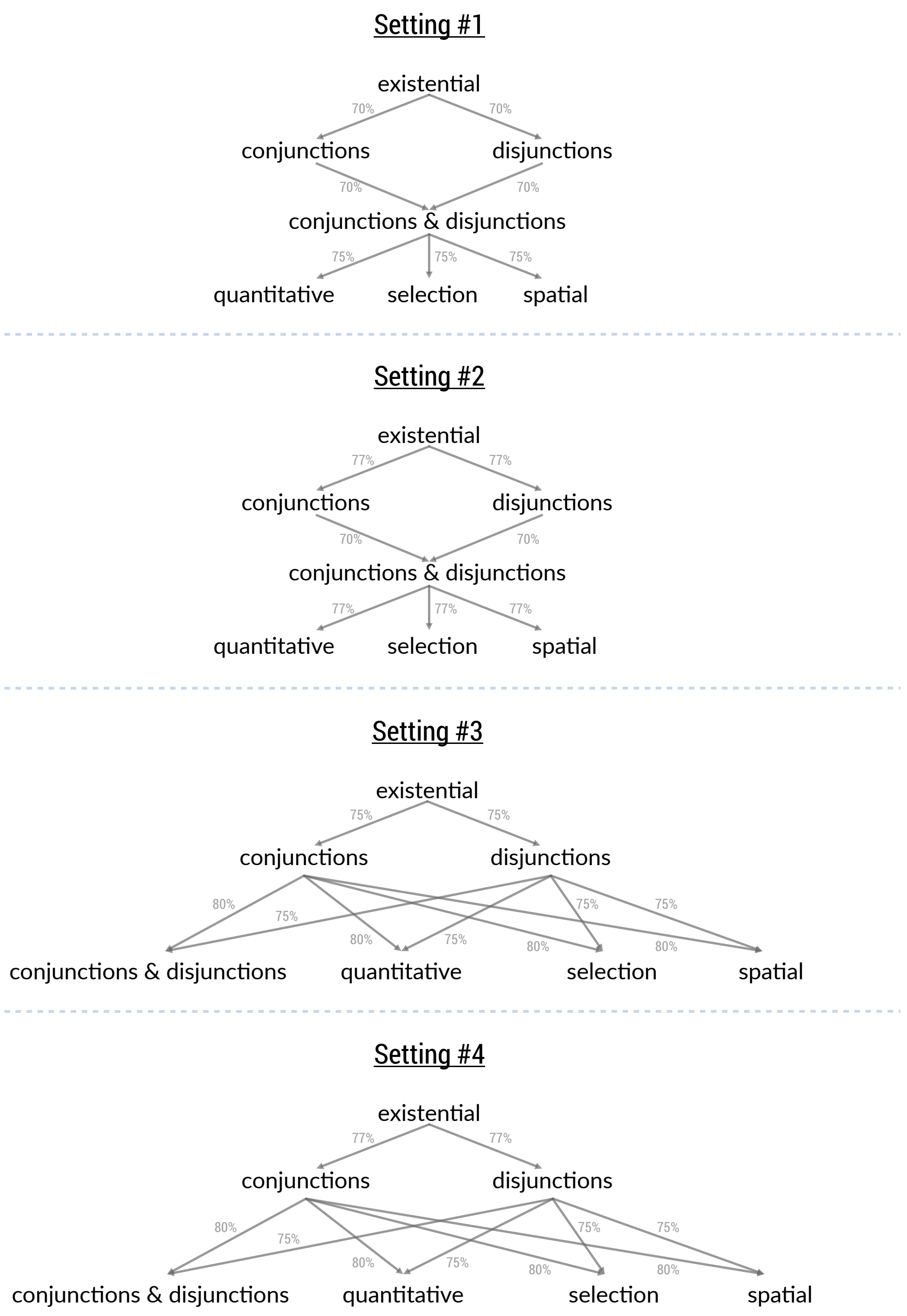


Figure 5.13: The four curriculum configurations whose results are presented in Figure 5.14. Each edge in a DAG shows a task perquisite, together with its corresponding accuracy threshold.

certain tasks start training much later. Therefore, there is a trade-off between the gain in performance and total training time. Another interesting observation is that, for some tasks, the curriculum approaches reach a better accuracy than the standard training approaches, but soon start overfitting. It is thus important to use the validation set in order to keep track of the best performing model parameters.

5.3.2.5 *Discussion*

In this section, we have introduced a curriculum learning strategy for multitask learning. This approach takes advantage of the dependencies between tasks and prioritizes training certain tasks early on. Our goal was to provide a proof of concept that such a strategy can improve model performance. Indeed, for the problem at hand, the curriculum strategies reached better performance, which will hopefully motivate future work at the intersection of curriculum and multitask learning. For the future, it would be beneficial to develop curriculum methods that can automatically detect the task dependencies, to reduce the overhead of validating the curriculum hyperparameters.

Moreover, note that when switching from a single task model to a multitask model, we chose which components of the architecture to share among tasks based on intuition, without considering what is more beneficial for a model trained with or without curriculum. For future work, it would be interesting to analyze how the benefits gained with curriculum learning vary relative to the amount of parameter sharing among tasks.

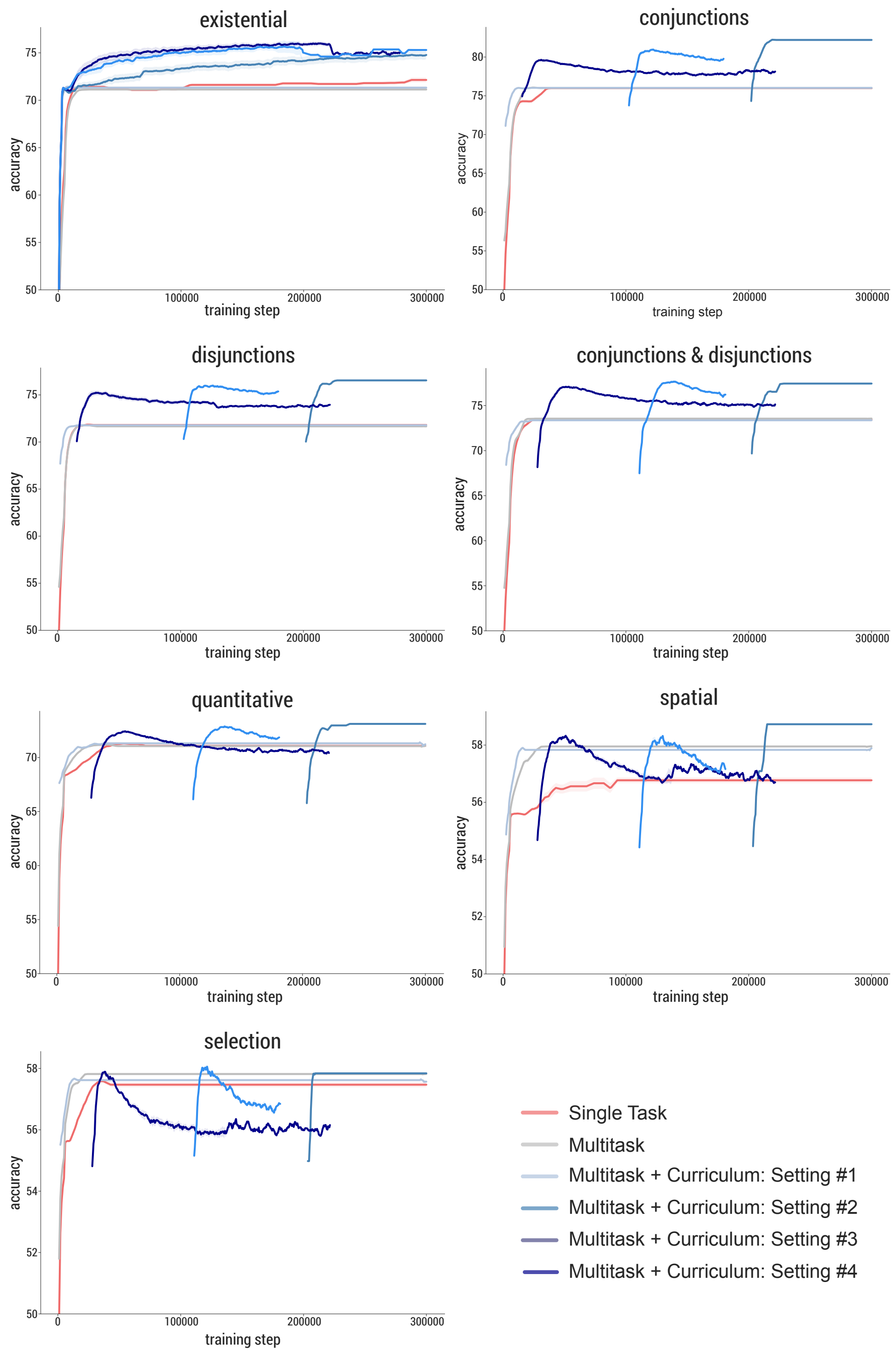


Figure 5.14: Results on the ShapeWorld dataset, for multiple learning settings: single task learning (red), multitask learning without a curriculum (gray), and multitask learning with various curricula (shades of blue). The precise curricula for each setting are described in Figure 5.13. We report the accuracy mean and standard error per training step, averaged over 5 runs. Note that not all curves start at training step 0. This is because some of the tasks wait until their prerequisites are satisfied, before starting to train.

5.4 KEY TAKEAWAYS

In this chapter we presented another approach to curriculum learning by designing curricula that operate on the *task space* of a model, as opposed to scheduling the training examples. We showed that this can be done both in the context of single task learning and in multitask learning.

In single task learning, the data and model architecture are structured with the goal of learning a single function of interest. Thus, a curriculum in task space needs to create a series of auxiliary tasks, which are not provided and which are easier than the task of interest. In Section 5.2 we proposed such an approach aimed at multiclass classification problems. Our approach automatically derives a series of easier classification problems operating on coarser labels. It then trains a classifier sequentially on these tasks, transferring the knowledge acquired about the coarser labels through the model parameters. We applied this training strategy to several common model architectures (`CNN`, `Resnet`, `WideResnet`) on popular image classification datasets (e.g., CIFAR-100, Tiny-Imagenet), and found that it achieves significant performance gains, especially on low data regimes and on classification problems with many labels.

Moreover, we also considered curricula for multitask learning. Here, we decided to focus on compositional problems, where current machine learning methods still struggle, and where we believe there is much to gain from curriculum learning. In Section 5.3.1, we used the *explicit* compositional relationships between tasks and we combined curriculum learning with semi-supervised and self-supervised learning to improve the system performance. We applied this idea to a synthetic scenario where we want to train a neural network basic arithmetic (counting, addition, multiplication). With our proposed strategy, we were able to boost the results for both interpolation (when the test data is sampled from the same distribution as the training data) and extrapolation (when the test data is sampled from a different distribution than the training data).

Finally, we also applied a different task-space curriculum learning approach on a multimodal image understanding problem. We generated multiple related tasks using the ShapeWorld framework (Kuhnle and Copestake, 2017). Here, the task compositions were *implicit*: for the model to be able to solve more complex tasks (such as reasoning about the relative positioning of objects in images), it must first be able to solve an easier task (such as detecting if that object exists in the image). Without explicitly modeling the relationships between tasks, we observed that just by training the tasks in a particular order given by a curriculum, we were able to improve the overall system performance on all tasks.

Taken together, these results show that there are various ways to apply curriculum learning on the task space of a model to improve performance.

6 UNDERSTANDING CURRICULUM LEARNING: A CASE STUDY ON SEQUENTIAL DATA

In this thesis so far, we have reviewed prior work on curriculum learning, proposed several novel methods for performing curriculum learning, and presented multiple successful use cases. However, while there has been been a significant amount of work on the practical usefulness of curriculum learning, not much progress has been made towards a better theoretical understanding of the effect of curriculum learning on optimization algorithms. One of the main issues that thwarts progress on this front is that curricula have been shown to bring benefits mainly for deep neural networks, which are notoriously hard to analyze theoretically. Existing theoretical work mostly focuses on self-paced learning (Fan et al., 2017; Ma et al., 2017), or makes strong assumptions about the studied curriculum (e.g., Hacohen and Weinshall (2019) prove that an *ideal* curriculum selected with respect to an *optimal* hypothesis can indeed improve learning). While these studies are excellent first steps, more work is needed to fully understand the effects of curriculum approaches that are commonly used in practice. For instance, in our work in Chapter 4, curricula based on intuition inspired by human learning were able to provide significant benefits in both training speed and final performance, without relying on any knowledge about the optimal model parameters.

Our goal in this chapter is to better understand how curriculum learning affects training and why it benefits learning in some cases, while it hinders it in others. Specifically, we are going to focus on a problem setting where we have seen many successful applications of curriculum learning: problems involving sequential data. Several publications (e.g., Bengio et al., 2009; Spitkovsky et al., 2010; Sachan and Xing, 2016; Xu et al., 2020), including our work presented in Chapter 4 used input space curricula on various types of sequential data, and showed how such curricula can help neural networks train faster and also sometimes obtain higher accuracy. Additionally, many of these approaches use as the sample difficulty measure either the sequence length, or other measures that correlate to some extent with sequence length (e.g., sentence probability in Section 4.4). For this reason, we focus on understanding the effect of length-based curricula on sequential data problems, and hope that our insights for this problem setting prove to be beneficial to a wide variety of domains that involve sequential data, including natural language processing, signal processing, finance, medical applications, etc.

To study the effect of curriculum learning on such problems, in this chapter we consider several case studies of increasing complexity:

1. Learning the parity function using recurrent neural networks (RNNs).
2. Learning to add two numbers digit by digit using Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997).
3. Learning arithmetic using encoder-decoder networks based on LSTMs and Transformers (Vaswani et al., 2017).
4. Fine-tuning BERT (Devlin et al., 2019) on tasks included in the GLUE benchmark datasets (Wang et al., 2019a).

For each of these scenarios, we try to understand the effect of curriculum learning on the training process by analyzing the parameter gradients computed during training and the loss function landscape.

6.1 ANALYTICAL STUDY

In this section we inspect analytically the parameter gradients equation for the some of the models considered in this chapter, and discuss the effect that we expect to see with curriculum learning. We then verify that we do indeed observe this effect empirically in each of the case studies. Before computing gradients for specific model architectures, in Section 6.1.1 we consider a more abstract model representation, that applies to recurrent neural networks (RNNs), as well as to other compositional models more broadly. Then, in Section 6.1.2, we inspect the concrete gradients for the simple RNN used in the first case study. Finally, based on the observations from the analytical study, in Section 6.1.3 we propose two hypotheses about when and why length-based curricula work.

6.1.1 *Gradients for Composed Functions*

Consider a scenario where the function that we want to learn, f parameterized by θ, can be expressed as the composition of two other functions g and h with parameters θ_g and θ_h, respectively, where $\theta = \theta_g \cup \theta_h$ and θ_g and θ_h may share parameters ($\theta_g \cap \theta_h \neq \emptyset$):

$$f(x) = g(h(x)) \tag{6.1}$$

This type of composition applies to RNNs, as well as other models. For example, for RNNs this is a composition of the same function, applied repeatedly, once for each element of the sequence being modeled. However, we are now going to focus on the simplified function f for simplicity of exposition and without loss of generality.

Let us first inspect the gradients of f with respect to θ_g. By applying the chain rule, we have that:

$$\frac{\partial f(x)}{\partial \theta_g} = \frac{\partial g(h(x))}{\partial \theta_g} = \frac{\partial g(y)}{\partial \theta_g}\Big|_{y=h(x)} + \frac{\partial g(h(x))}{\partial h(x)} \frac{\partial h(x)}{\partial \theta_g} \tag{6.2}$$

Here we have decomposed the gradient of $g(h(x))$ with respect to θ_g into a component that depends only on the transformation performed by function g, and a component that depends on the nested function h due to the fact that θ_g and θ_h may share parameters.

If the function $g(y)$ applies some transformation on the inputs y using the parameters θ_g, as is typically the case in neural networks, then the gradient $\frac{\partial g(y)}{\partial \theta_g}$ depends on y (i.e., $\frac{\partial g(h(x))}{\partial \theta_g}$ depends on the value of $h(x)$). Thus the update that we apply to θ_g depends on the estimate of $h(x)$ using the current θ_h parameters. This can be a problem early on during training while the parameters θ_h are far from converged and their distribution may be shifting. This a well known problem in machine learning, known as *internal covariate shift*, which refers to "the change in the distribution of network activations due to the change in network parameters during training" (Ioffe and Szegedy, 2015). More generally, in neural networks, the predictions of the first layer are passed as input to the second layer, the predictions of the second layer are passed as input to the third layer, etc. Because of this, when the parameters of a particular layer (especially an early one) change, the input distribution of all subsequent layers also changes. This makes deep models harder to train, and requires special tricks such as learning rate schedules that progressively lower the learning rate, careful initialization, or using methods designed to mitigate this such as batch normalization (Ioffe and Szegedy, 2015). Such strategies can ameliorate the problem, but slow down training and also require additional tuning efforts. RNNs can be seen as deep networks by "unrolling" the time dimension, and thus are also prone to this issue, even more so when training on longer sequences. A curriculum based on sequence length may be able to improve the internal covariate shift issue, because the predictions for the shorter sequences may have stabilized by the time the model is allowed to see longer ones.

Let us now consider the gradients of f with respect to θ_h:

$$\frac{\partial f(x)}{\partial \theta_h} = \frac{\partial g(h(x))}{\partial \theta_h} = \frac{\partial g(h(x))}{\partial h(x)} \frac{\partial h(x)}{\partial \theta_h} \tag{6.3}$$

Early on during training, while θ_g is far from its correct value, the prediction for $g(h(x))$, and consequently the value of $\frac{\partial g(h(x))}{\partial h(x)}$, is likely far from its optimal value. Therefore, the update that we apply to θ_h via $\frac{\partial f(x)}{\partial \theta_h}$ will also be noisy. For RNNs this means that the updates that we apply to correct the predictions for the beginning of the sequence will be noisy due to the multiplier coming from the wrong predictions at the end of sequence. Therefore, it is worth considering training h (which, for RNNs means shorter sequences) in isolation if possible, before training the composed function jointly. In fact, for many RNN problems, datasets typically have examples of different lengths which would allow us to do this using a length-based curriculum.

Taken together, these observations suggest that the coupling between the two composed functions, f and g, can have a negative impact on learning each of them. Therefore, it may be beneficial to decouple their parameter updates, if possible, in order to avoid internal covariate shift and compounding errors.

6.1.2 *Gradients for a Simple RNN*

We now inspect the gradients of a basic RNN architecture (which we also use in Section 6.2 in our first case study), to verify that the issues described earlier indeed apply in this case. Moreover, for this specific RNN architecture there is an additional issue, which we will discuss after introducing its gradients. We start by introducing the model architecture, followed by a discussion on its gradients and the problems that can arise.

6.1.2.1 *Model*

We consider a simple RNN architecture and input sequence x of length T, which is passed through the RNN step by step. After the last input at time[1] step T, the network predicts the output of interest, which is used to update the model parameters. We depict this in Figure 6.1 where, for illustration purposes, the data is a sequence of bits as used in Section 6.2.

To facilitate our analysis, we chose a basic recurrent neural network (RNN) architecture, with the following hidden state update equation:

$$h_t = x_t W_{xh} + \sigma(h_{t-1}) W_{hh} + b \tag{6.4}$$

where x is a sequence of length T, $t \in \{1, ..., T\}$ represents the time step indexing the position in the input sequence, x_t is the m-dimensional input at time t, h_t is the network hidden state at step t, $W_{xh} \in \mathbb{R}^{m \times d}$, $W_{hh} \in \mathbb{R}^{d \times d}$ and $b \in \mathbb{R}^d$ are the RNN parameters with hidden size d, and σ is a non-linear activation function

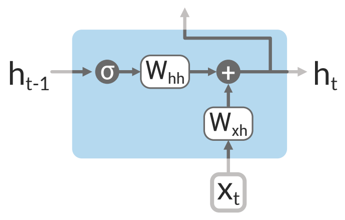


Figure 6.2: A basic RNN cell.

[1]Borrowing terminology from signal processing, it has become standard in recurrent neural networks to refer to sequence elements as "time steps", because they are provided to the network one at a time. We follow this convention, although the sequence elements in our case do not refer to time per se.

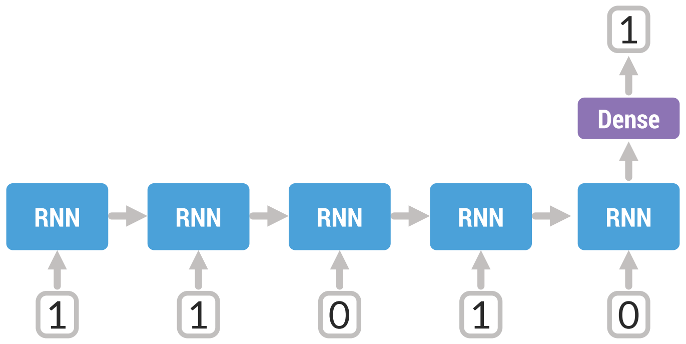


Figure 6.1: Using a recurrent neural network (RNN) to compute the parity function for a sequence of binary inputs.

(we use *tanh* in our experiments). An illustration of the recurrent network cell is shown in Figure 6.2. Note that this formulation is equivalent to the more popular alternative $h_t = \sigma(x_t W_{xh} + h_{t-1} W_{hh} + b)$, and we chose it to facilitate the derivation of the gradients, as did Pascanu et al. (2013) in a related line of work.

The last RNN output is passed to a dense layer, that projects to the output space of size k:

$$o = h_t W_{ho} + b_o \tag{6.5}$$

where $W_{ho} \in \mathbb{R}^{d \times k}$ and $b_o \in \mathbb{R}$ are the dense layer parameters.

When we perform classification, we further normalize the logits o into a valid probability distribution as $\hat{y} = \text{softmax}(o)$. We train the network by minimizing a loss function $\mathcal{L}(\hat{y}, y)$ between the predicted outputs $\hat{y}$ and the target y.

6.1.2.2 *Gradients*

Let us consider the gradient of the loss function with respect to the recurrent weights of the model, W_{hh}. Using the chain rule, we can calculate the gradient as:

$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \frac{\partial \mathcal{L}}{\partial o_T} \frac{\partial o_T}{\partial h_T} \frac{\partial h_T}{\partial W_{hh}} \tag{6.6}$$

Taking into account the recurrent relationship between h_t and h_{t-1} for all t, we have that:

$$\frac{\partial h_T}{\partial W_{hh}} = \sum_{1 \leqslant k \leqslant T} \frac{\partial h_T}{\partial h_k} \frac{\partial^+ h_k}{\partial W_{hh}} \tag{6.7}$$

$$\frac{\partial h_T}{\partial h_k} = \prod_{k < i \leqslant T} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{k < i \leqslant T} W_{hh}^T \text{diag}(\sigma'(h_{i-1})) \tag{6.8}$$

where we have used the notation of Pascanu et al. (2013) in $\frac{\partial^+ h_k}{\partial W_{hh}}$ to refer to the "immediate" partial derivative of h_k with respect to W_{hh} in which h_{t-1} is taken as a constant with respect to W_{hh}. Also, $diag$ converts a vector into a diagonal matrix, and σ' is the element-wise derivative of σ with respect to its only argument. Moreover, the value of a row in the tensor $\frac{\partial^+ h_k}{\partial W_{hh}}$ is $diag(\sigma(h_{k-1}))$.

From Equation 6.4 and Equation 6.7 we can observe two important issues that make training difficult for RNNs:

1. **Vanishing Gradients:** Equation 6.7 exposes a well known difficulty in training recurrent neural networks known as the "vanishing gradients problem" (Hochreiter, 1991). This refers to the fact that the contribution of the time steps T-1, T-2, ..., 1 to the gradient keeps decreasing in this order, and thus when T is large the gradient cannot help the model correct its predictions for the early time steps. This is due to the multiplication in $\frac{\partial h_T}{\partial h_k}$ in Equation 6.7 containing more and more factors with values between [0,1] as k decreases, that cause that term to "vanish". For instance, if σ is the tanh function, then $\sigma'(f_t) = 1 - (\underbrace{\sigma(f_t)}_{\in[-1,1]})^2 \in [0,1]$. Similarly, depending on the norm of W_{hh}, the power of W_{hh} coming from the product in Equation 6.7 can also contribute to this problem. A more detailed discussion on this issue can be found at Pascanu et al. (2013).

2. **Compounding Errors:** The value of the hidden state h_t depends on all the estimated values for the prior states $h_{t-1}, ..., h_1$. Therefore, during training, until the network is good enough at predicting $h_1, ..., h_{t-1}$, due to compounding errors, we can expect h_t to be noisy.

Putting the pieces together, we notice that the terms with large impact on the gradient (i.e., those corresponding to the latter hidden states h_{T-1}, h_{T-2}, etc.) depend on having good hidden states propagated from the earlier time points. At the same time, the early hidden states h_1, h_2, etc., contribute little to gradient update, especially when the input sequence is long, and thus it would be difficult for the model to "fix" its predictions for the early states using parameter gradients computed on a long sequence. Therefore, we can expect that *the gradients for long sequences will be noisy while the model is not good at predicting shorter sequences*.

6.1.3 *Hypotheses on* When *and* Why *Curriculum Learning Works*

Based on the observations in Section 6.1.1 and Section 6.1.2, we propose the following hypotheses:

- **Hypothesis 1:** When training composed models (in particular recurrent networks) without a curriculum on sequential data, there is some degree of correlation between the sequence length and the relative time during training when that sequence is learned.
- **Hypothesis 2:** When there is such a correlation even without a curriculum, a length-based curriculum can help the model converge faster.

Hypothesis 1 (H1) is stating that a model during training will start performing well first on shorter sequences, and only afterwards on longer and longer ones. Of course, depending on the concrete application, there may be other criteria besides sequence length that make a sequence easier or harder (e.g., in Section 4.4 we used the intuition that if a sentence contains rare words, this makes it harder to translate). H1 is accounting for this issue by specifying that there only "some degree" of correlation with the time during training when the model starts performing well on sequences of that length—only "some" degree because part of the difficulty of the sample may be caused by other criteria as well. In other words, *the sequence length only contributes to the difficulty of a training sample*, and thus can be used in combination with other difficulty criteria to establish a combined sample difficulty.

Hypothesis 2 refers to curricula that focus on shorter sequences early on. There are multiple ways to "focus" on shorter sequences. For example, at a training iteration i, we can sample uniformly from all sequences of length $T < \mathtt{threshold}(i)$, as we did in Chapter 6 using the competence function. Another way is to perform weighted sampling from all training data, but put more weight on shorter sequences early on during training.

In the following sections we investigate empirically whether these hypotheses hold on multiple case studies.

6.2 CASE STUDY #1: PARITY FUNCTION

For our first case study, we consider learning the *parity function*, also known as the XOR problem. This is a famous problem that is often used as a textbook example when introducing neural networks, because it is a very simple classification problem that is not "solvable" using a linear model (i.e., it is not linearly separable). The XOR function over binary inputs x_1 and x_2 is defined as follows:

$$x_1 \oplus x_2 = \begin{cases} 0, & \text{if } x_1 = x_2 \\ 1, & \text{otherwise} \end{cases} \tag{6.9}$$

In this case study, we are interested in sequences and so we consider the n-bit parity function $f : \{0, 1\}^n \to \{0, 1\}$, where $f(x) = 1$ if and only if the total number of ones in the vector $x \in \{0, 1\}^n$ is odd. We denote this as:

$$f(x) = x_1 \oplus x_2 \oplus \cdots \oplus x_n \tag{6.10}$$

Therefore, in this first case study, we will train a neural network to compute the parity of bit sequences of arbitrary length. In the following sections we provide details on the model and the training algorithm that we use.

MODEL. To support variable-length sequences, we use a recurrent neural network (RNN). In fact, we use the precise architecture described in Section 6.1.2.1. In this case, the inputs at every time step are binary, $x \in \{0, 1\}^T$, and thus the dimension m of x_t is 1. After the last input at time step T, the network predicts the parity of the sequence or, in other words, it predicts whether the sequences contains an odd number of 1's. Finally, since the network is performing binary classification, we use the binary cross entropy loss function to train it:

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})) \tag{6.11}$$

CURRICULUM. As discussed earlier, we use an input-space curriculum that uses the sequence length to measure the sample difficulty. The network is trained by sampling batches of examples and updating the model parameters according to the batch loss (we discuss concrete optimizers and learning rates later in Section 6.2.1). The curriculum acts as a filter on the data that is selected during training. To keep the analysis easy to interpret, we use a simple form of curriculum learning that still shows good results on this problem: a linear curriculum with its length being the single hyperparameter of this algorithm (equivalent to the "linear" competence function from Section 4.1). We set the curriculum length to a prespecified number of training steps, C. For the first C training steps, we sample batches of examples uniformly from the k% easiest training samples (i.e., shortest sequences), where k linearly increases to 100% during the first C steps. After the first C training steps, we sample batches uniformly from the whole training set (i.e., similar to training without a curriculum).

6.2.1 *Experimental Setup*

DATA. The model is trained on bit sequences with length $T \in \{1, ..., T_{max}\}$. In what follows we report results for $T_{max} = 10$. We also experimented with longer training sequences, but this did not seem to affect our results and key takeaways that we present at the end of this section. We used the following dataset splits:

– Train: For each training batch, we first sample a sequence length T uniformly from $\{1, ..., T_{max}\}$, and we then sample a batch of 128 examples of length T. Each bit in the input sequences is sampled from a Bernoulli distribution with parameter $p = 0.5$. This strategy ensures that all sequence lengths appear equal probability during training.

– Test: We construct two datasets: one for testing the ability of the model to *interpolate*, and one for testing its ability to *extrapolate*. For the interpolation dataset we use the training data sampling strategy to sample 100 batches of examples, with each batch containing 16 examples. Note that each batch contains examples of equal length. For the extrapolation dataset we also sample 100 batches of examples of size 16, but in this case the sequence lengths we consider are between 101 and 120. This is so that we can evaluate the model on sequences that are significantly longer than the sequences it was trained on. This large difference between the train and test data distribution will allow us to evaluate whether the model has learned the parity function rule that can generalize to arbitrary lengths, as opposed to just overfitting on the training data distribution.

TRAINING. We train our model using stochastic gradient descent with a constant learning rate of 0.001, and a batch size 128. We also tried using other optimizers like Adam (Kingma and Ba, 2015) and other learning rates, but while these affected the convergence speed of the models, they did not affect the relative performance of the models trained with and without curriculum, and thus do not affect our conclusions.

CURRICULUM. We use the linear curriculum described earlier. Concretely, the curriculum changes the probability distribution of the training data by adjusting the value of T_{max} (i.e., the maximum sequence length that we consider when sampling training data batches), starting at $T_{max} = 1$, and linearly increasing it to 10 during the first C epochs. Figure 6.3 shows the T_{max} value every training epoch depending on total curriculum length C.

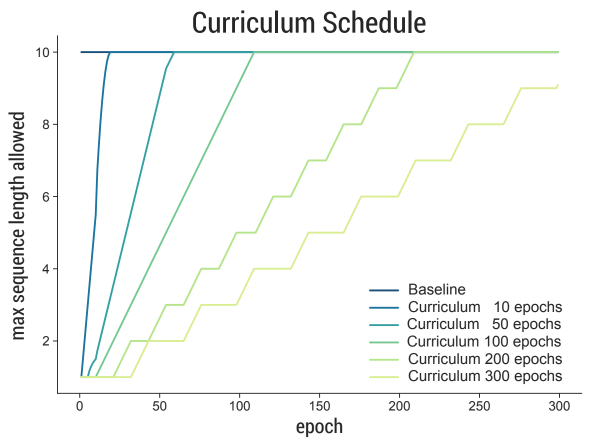


Figure 6.3: Maximum sequence length allowed to be sampled at each epoch during training, depending on the curriculum length, C.

6.2.2 *Results*

For our experiments we consider two training regimes: a *baseline* regime where we train without using a curriculum, and a *curriculum* regime where train using a linear curriculum for the first C epochs, after which we switch back to the baseline training regime. We consider multiple values for C and we also use two variants of our model: one with a hidden size of $d = 4$ and one with $d = 128$. Note that the hidden size $d = 4$ was the smallest network that we were able to train successfully under any training regime. Our results are shown in Figure 6.6. Across all of our experiments, we observe that models trained using a curriculum reach better performance and learn faster than the corresponding baselines. We now summarize a few more of our observations:

- Small vs Large Model: The larger RNN seems easier to train even without a curriculum (i.e. it both performs better and trains faster than the smaller model), but the curriculum helped in both cases. Interestingly, the small baseline was not able to extrapolate at all in any of the 5 runs, while the model trained using curriculum learning was able to extrapolate.

- Curriculum Length: We observe that the optimal curriculum learning for these experiments seems to be between 100 and 200 epochs. We also notice that curricula which are shorter or longer than that range result in smaller gains. This suggests that there is an optimal curriculum length beyond which the benefits of curriculum learning start to decay and may even harm overall performance. We attempt to provide an explanation for this effect in the following sections.

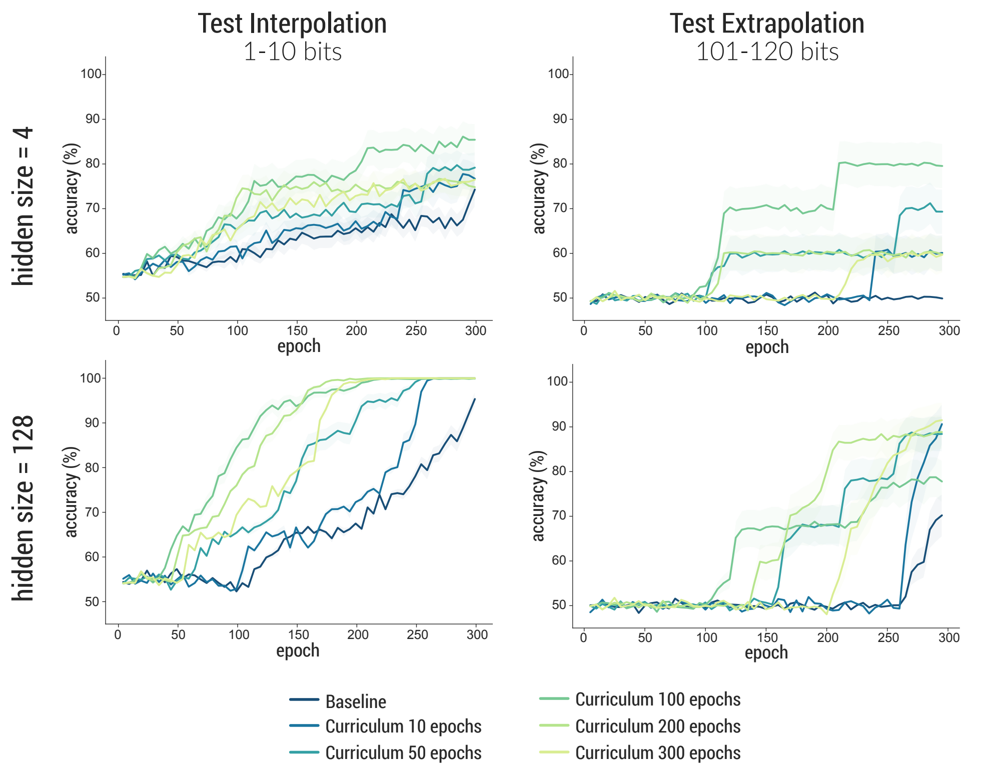


Figure 6.4: Accuracy of RNNs trained on sequences of length 1-10, with and without a curriculum. Top row: hidden state $d = 4$. Bottom row: hidden state $d = 128$. First column: the test dataset contains sequences of length 1-10 to test interpolation. Second column: test sequences of length 101-120, to test extrapolation. For each experiment configuration, we report the accuracy mean and standard error over 5 runs with different random initializations.

– Training Loss: Observing the loss of the two models in Figure 6.5, we notice an interesting behavior for the curriculum experiments: at the beginning of each new curriculum level, when longer sequences are introduced, the loss shows a sharp increase followed by a steady decrease until the next curriculum level. Also, after 300 epochs we observe that, for $d = 4$, the loss of the model trained with curriculum is significantly lower than that of the baseline. On the contrary, for $d = 128$ the two values are close, but the curriculum allowed to the model to reach that value earlier in training.

6.2.3 *Evidence for the Proposed Hypotheses*

HYPOTHESIS 1. To test Hypothesis 1, we first train the network without a curriculum and compute the accuracy on a test set as training progresses. We keep track of the test accuracy per sequence length (i.e. we separately average the accuracies for test sequences of length 1, 2, etc.), in order to

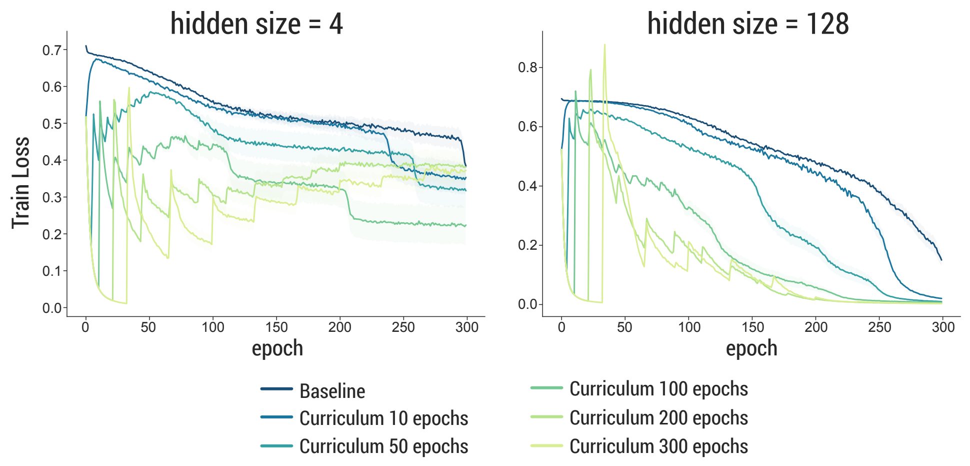


Figure 6.5: Train loss of RNNs trained on sequences of length 1-10, with and without a curriculum. Left: hidden state $d = 4$. Right: hidden state $d = 128$. For each experiment configuration, we report the accuracy mean and standard error over 5 runs with different random initializations.

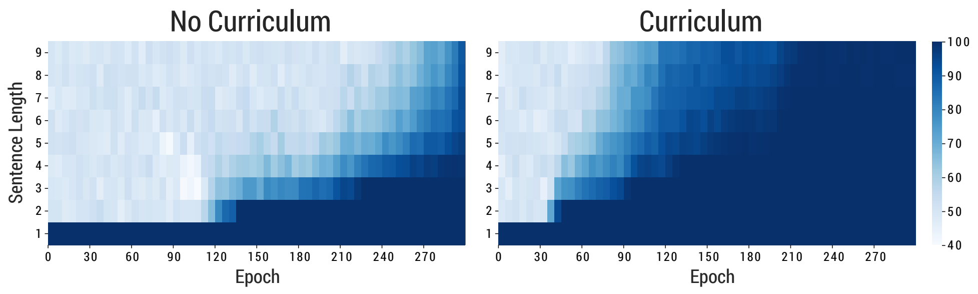


Figure 6.6: Accuracy per sequence length for an RNN with hidden size 128, trained on sequences of length 1-10. We report the accuracy on a validation set, averaged separately for samples of different sequence length. All accuracies are the average over 5 runs initialized with different random seeds.

observe how well the network performs on sequences of different length as training progresses. We perform this experiment using an RNN with hidden size $d = 128$ trained on sequences of length 1-10, and tested on sequences of the same length. The results are shown in Figure 6.6, left side. We can clearly see a pattern where the sequences are learned in order of length, meaning that at every point during training the model performs on average better on sequences of length t than length $t + 1$, which is consistent with Hypothesis 1.

HYPOTHESIS 2. We perform the same experiment with a model trained with curriculum learning. We set $C = 100$, which was one of the best performing curricula in Figure 6.4. The results are shown in Figure 6.6, right side. We notice the same pattern of accuracies increasing in order of length as displayed by the baseline model, but the all accuracies increase faster.

Using curriculum learning, the model reaches a good performance even for the longest sequences much faster, which is consistent with Hypothesis 2. In fact, Figure 6.4, where the accuracies for all lengths are aggregated, also supports this hypothesis.

6.2.4 *The Effect of the Curriculum on the Loss Landscape*

In the previous sections we have seen that a model trained with curriculum performs better than the equivalent baseline, and we conjectured in Section 6.1 that this is because it avoids the noisy gradients from longer sequences early on during training. In this section we visualize how the curriculum changes the trajectory of the parameters during training. To this purpose, we use a recent method of visualizing the loss landscape of deep neural networks (Goodfellow et al., 2014; Im et al., 2016; Li et al., 2018). We follow the method described in Li et al. (2018) to visualize the training loss along a 2D projection of the model trajectory. We further summarize the visualization approach, and then use it on our parity function problem. Note that we will also use the same visualization technique in the following case studies.

VISUALIZATION METHOD. Let θ represent all the model parameters flattened and concatenated into a single 1-dimensional vector. The model trajectory during training is a sequence of such parameters $\theta_0, \theta_1, ..., \theta_n$, etc., starting at the initial parameters θ_0 and converging to θ_n at the end of training. For every parameter configuration θ, we obtain a different loss value on the training dataset, $\mathcal{L}(\theta)$. However, we would like to visualize the loss value not only at the exact points on the trajectory, but also on a grid around them, in order to better understand what the *loss landscape* looks like. To be able to visualize it, the grid needs to be in a lower-dimensional subspace (e.g., 1D or 2D) that we can easily plot. Thus, we follow the approach described in Li et al. (2018) and select a center point θ_* for our grid (e.g., $\theta_* = \theta_0$), as well as two direction vectors, δ and η. We then calculate the loss value at multiple locations in the subspace given by the directions δ and η: $\ell(\alpha, \beta) = \mathcal{L}(\theta_* + \alpha\delta + \beta\eta)$. The directions δ and η need to be chosen carefully, to preserve as much as possible of the variation in the trajectory. Section 7 of Li et al. (2018) shows examples of good and bad directions. We follow their approach of applying principal component analysis (PCA) on the matrix $M = [\theta_0 - \theta_*, \theta_1 - \theta_*, ..., \theta_n - \theta_*]$ to find two directions that explain as much as possible of the variance.

RESULTS. We display the loss landscape for one of the discussed experimental settings: an RNN with hidden state 128, trained on sequences of length 1-10. We consider 4 individual trajectories: a baseline model trained

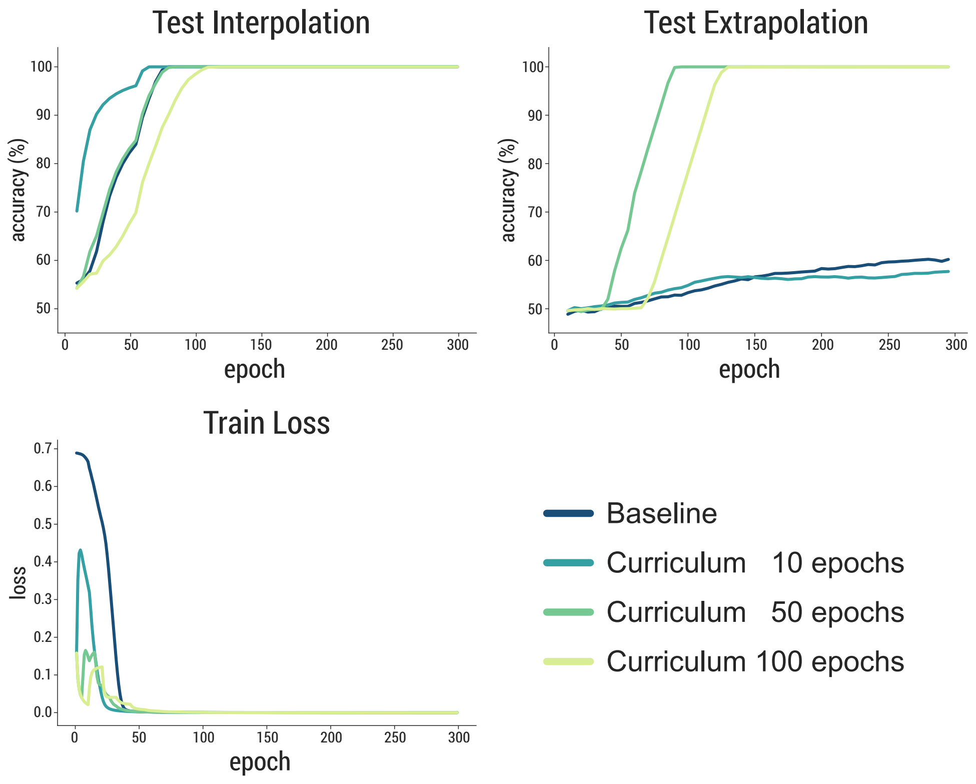


Figure 6.7: Results for the 4 model trajectories displayed on the loss landscape in Figure 6.8. Left: interpolation test accuracy evaluated on sequences of length 1-10, similar to the training distribution. Middle: extrapolation test accuracy evaluated on sequences of length 101-120. Right: training loss.

without a curriculum, and three curricula trajectories for curriculum lengths $C \in \{10, 50, 100\}$. All 4 experiments start from the same initial parameters θ_0. We select $\theta_* = \theta_0$ as the reference point for the loss plot. We display the loss landscape in Figure 6.8, and the corresponding test accuracies and training loss curves in Figure 6.7.

The loss landscape depends on the dataset it is evaluated on. The model trained without curriculum is always updated based on the landscape corresponding to "Sequence length 1-10", at every step along the trajectory. The models trained with curriculum learning are updated based on each of the landscapes sequentially, starting with sequences of length 1, followed by $1-2$, then $1-3$, etc. Thus, the initial gradient descent direction is driven by the earlier loss landscapes, but after the curriculum is complete, the model continues training on the original train distribution. We also display the loss landscape of the test distribution, for sequences of length $101-120$, on the last row of Figure 6.8.

Interestingly, we can observe that the loss surface seems *smoother* with *wider local minima* when evaluated on shorter sequences. As the sequences become longer, the loss space becomes sharper, with narrower local minima.

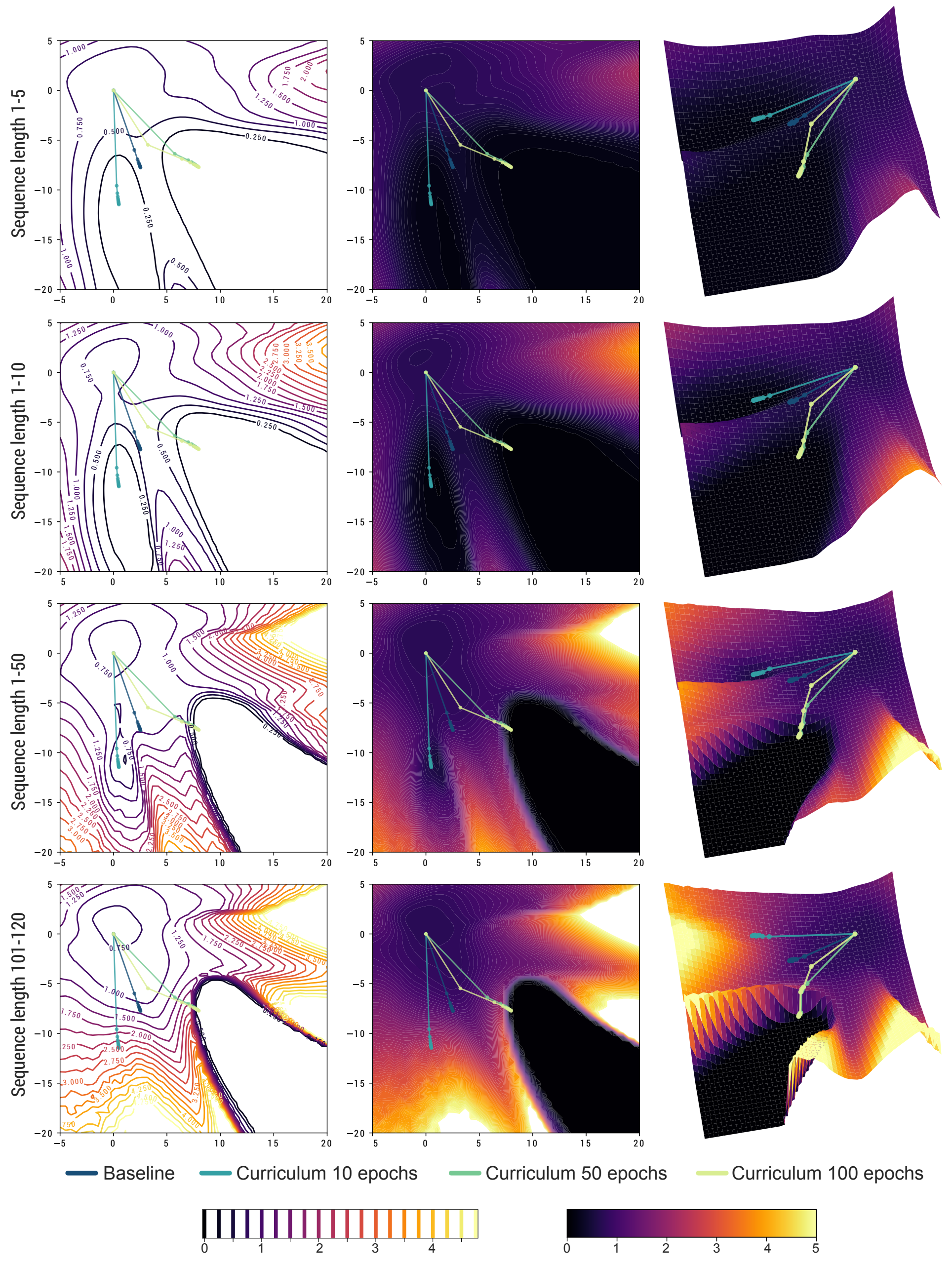


Figure 6.8: Loss landscape for an RNN with hidden state 128, trained to predict the parity function. Each row shows the loss landscape evaluated on datasets with different sequence lengths. Each column provides a different view of the corresponding loss landscape: a contour plot (left), a heatmap with contour lines (middle), a 3D view of the heatmap where the height corresponds to the value of the loss (right). We overlay over each plot the trajectories of three models: a baseline trained without curriculum, and three curricula of lengths 10, 50 and 100 epochs. The corresponding test accuracies for these trajectories are shown in Figure 6.7.

A possible explanation for the narrowing of the local minima with length is that, for the xor problem, samples consisting of short sequences can be prefixes of longer sequences. Therefore, a parameter configuration that can solve a long sequence, is likely to be a good solution for short sequences as well. Thus, local minima in the loss landscape of long sequences, will likely be local minima for short sequences as well. This effect can probably be seen for other problems where the data has this property, such as natural language processing problems using text data or our addition experiments. The wideness of the local minima, as well as the smoothness of the loss landscape, may enable a model trained with a length-based curriculum to "see" local minima that are farther from the initial parameters. Therefore, the gradient descent direction during the early stages of the curriculum may point towards local minima that are not in the immediate descent direction according to the full training distribution.

Another interesting observation is that the baseline model in this case is slowed down in a flatter region of the loss surface. We see from the accuracy plot in Figure 6.7 that is has not converged, but it is moving very slowly around a flatter region of the loss. The models trained with curricula, on the other hand, quickly start in the direction of local minima and reach a better accuracy faster. The shorter of the curricula reaches the local minimum that is closest to the baseline method, but faster. The longer curricula find a different local minimum which is further away.

Moreover, when considering that type of benefits we can expect from curriculum learning, a natural question to ask is whether the curriculum is just helping the model train faster and the difference in performance at the end of training is entirely due to the fact that the baseline may need more iterations to reach the same point. The loss landscape plots suggest that this is not necessarily the case, and that the models trained with and without curriculum may not necessarily converge to the same point.

6.3 CASE STUDY #2: ADDITION DIGIT-BY-DIGIT

In this case study, we use the same setup as in Section 4.2 to perform digit-by-digit addition using an LSTM. In this setting, we observed that a curriculum using the number of digits as difficulty measure (equivalent to using the sequence length) was beneficial for both interpolation and extrapolation, as reported in Figure 4.6. Next, we attempt to understand better what led to these effects, using the same type of analyses as in the previous section.

6.3.1 *Analytical Study*

In Section 6.1 we inspected the gradients of RNNs and of compositional functions more generally, and pointed out a few issues related to training on long sequences, such as compounding errors and vanishing gradients. LSTMs (Hochreiter and Schmidhuber, 1997) are in fact a special type of RNN that has been designed explicitly to overcome the vanishing gradients issue. Indeed, LSTMs provide a big improvement over simple RNNs, making them popular across many machine learning areas, and especially those that deal with sequential data (e.g., natural language processing, signal processing). Nevertheless, they are still recurrent networks which are compositional in nature, and so our discussion in Section 6.1.1 applies to them as well. That is, when updating the model parameters on a long sequence early on, the gradient component trying to correct the predictions at later time steps will still be affected by the wrong hidden state propagated from earlier time steps. And similarly, the gradient component trying to correct the predictions at early time steps will still be affected by the gradient component propagated from the later time steps. Therefore, using the same argument as before, we expect to see that the sequences are being learned in order of length (i.e., Hypothesis 1), and that using length-based curricula helps them learn faster and more effectively (i.e., Hypothesis 2).

6.3.2 *Evidence for the Proposed Hypotheses*

We now investigate further what property of the curriculum brought about the improvements observed in Figure 4.6. Similar to the parity function experiments, we record the accuracy of the model during training, and aggregate it over samples with the same number of digits. We report these accuracies per number of digits in Figure 6.9. Also similar to the parity function experiments, we observe that the accuracies per training step increase in order of number of digits, suggesting that the model first learns to add the samples with fewer digits, and then with more and more digits. This provides further evidence for Hypothesis 1.

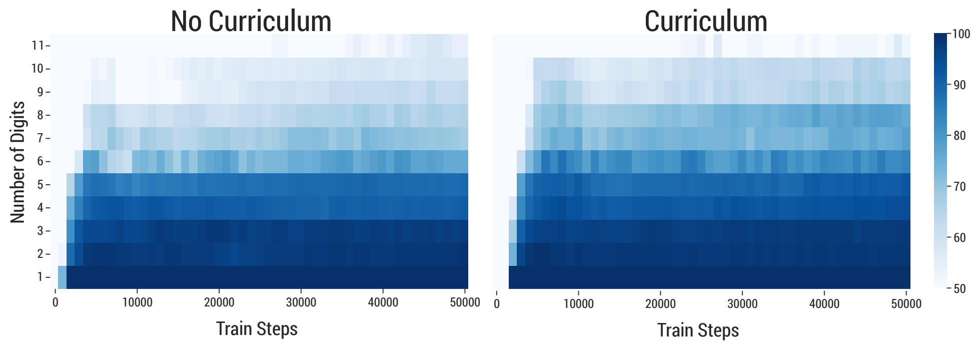


Figure 6.9: Accuracy per number of digits for an LSTM with hidden size 16, trained on adding numbers with up to 5 digits, and evaluated on a held-out test set containing operands with up to 10 digits. We report the accuracy on a validation set, averaged separately for samples with different number of digits. All accuracies are the average over 5 runs initialized with different random seeds. Since we are adding two operands with potentially different number of digits, the number of digits in the figure refers to the number of digits of their sum (thus it goes to 11).

We also observe that the final accuracies (at the last step) are also decreasing with sequence length, which can be explained by the compounding errors argument described in Section 6.1.2. Unlike the parity function experiment, here the model learns the training data distribution (1-6 digits on the y-axis in the figure) very quickly for both the baseline and the curriculum training regimes. However, the curriculum version improves accuracy much faster for long sequences that are outside the training data distribution (7-10 digits), which is in agreement with Hypothesis 2. Further evidence for Hypothesis 2 is provided by the results in Figure 4.6, where we the model trained with curriculum learning not only reaches better final accuracy, but also achieves its best accuracy earlier than the baseline.

6.3.3 *The Effect of the Curriculum on the Loss Landscape*

We also inspect the loss landscape along the learning trajectories of the two models, using the process described in Section 6.2.4. In Figure 6.10, we show contour plots of the loss landscape around the model trajectory during training. We also project the trajectories of two models: a baseline trained without curriculum learning, and a model trained with a length-based curriculum, whose losses are illustrated in Figure 6.9. As we can see from Figure 6.10, the two trajectories both start at the same initial point, but they converge to different local minima. The loss landscape marked by the contour plots depends on the dataset it is being computed over. The model trained without curriculum is always updated according to the "Full Train" loss landscape, at every step along the trajectory. The curriculum model, on the other hand, is updated based on each of the landscapes sequentially starting with the 5% easiest data, increasing the amount of data allowed

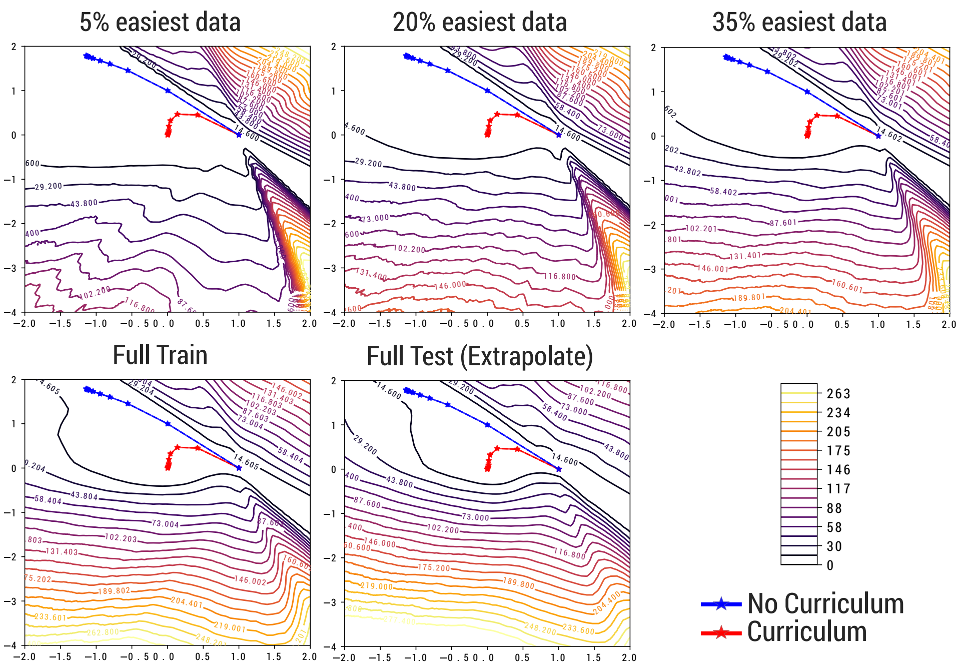


Figure 6.10: Loss landscape for an LSTM with hidden size 16, performing digit-by-digit addition. The value of the loss depends on the dataset that it is evaluated on. In each of the figures above we evaluate the loss on different subsets of data, as specified in the title of the plot. The contour plots mark different levels of the loss value—the levels are similar in all plots. The x and y axes represent the 2D projection of two high-dimensional directions in the parameter space. On the loss landscape, we plot the trajectories of two models during training: one trained without a curriculum (in blue) and one trained with a length-based curriculum (in red).

as the competence of the curriculum increases. The training distribution converges to "Full Train" within 500 steps (the length of the curriculum). Thus, the initial descent direction is driven by the earlier loss landscapes, but after the curriculum is complete, the model continues training on the original distribution.

We also make some interesting observations for the loss landscapes. Firstly, similar to the parity function, we see how the local minima for easier data are *wider*, and they become narrower and narrower as the difficulty of the examples increases. Secondly, unlike the parity function, the loss landscape here is *not necessarily smoother* for easier data. For example, the loss peak around position $(2, -3)$ on the plots is in fact sharper for shorter sequences. Note that we can make these comparisons between the contour plots because contour lines marking the loss level are kept constant for all plots.

Taking all these results together, the two case studies so far suggest that *the curriculum makes local minima wider early on*, which may be helping the model start off in a better direction, and reach local minima that may not otherwise be easily reachable by gradient descent from the starting point.

This is also consistent with our conjecture about why curricula that are too long could be harmful: *training too long on easy data can get the curriculum stuck in a different region of the wider local minimum*, that is not necessarily a local minimum for the full training distribution.

6.4 case study #3: addition sequence-to-sequence

In this case study, we use the same setup as in Section 4.3, where we trained an sequence-to-sequence encoder-decoder model to perform an addition problem provided as a sequence of characters (e.g., "123+45+7"). We replicate the same experiments, where we previously observed that a curriculum using the sequence length as difficulty measure is beneficial for both interpolation and extrapolation. We reported these in Figure 4.8 and Figure 4.9, where the curricula helped the model train faster and reach better final performance than the baseline.

We further show the same types of visualizations as in the previous case studies, to verify that the results in this case are also compatible with the proposed hypotheses. Note that in this case we were not able to provide visualizations of the loss landscape. Since the models in this case study are much larger than the ones used in the previous experiments, we could not accurately project the model parameters along the training trajectory down to a 2D space.

6.4.1 *Evidence for the Proposed Hypotheses*

Similar to the case studies before, we evaluate the accuracy per sequence length during training, with and without curriculum learning. The results are presented in Figure 6.11. The plots confirm the same trends observed in the previous case study, where the baseline model learns sequences in order of length, supporting Hypothesis 1. The models trained using a curriculum also learn sequences in the same order, but faster (especially prevalent for longer sequences), supporting Hypothesis 2.

Interestingly, these trends hold not only for LSTMs, which have the compositional nature discussed in Section 6.1.1, but also for Transformers which have a different type of compositional structure than LSTMs and do not suffer from the vanishing gradients problem in the same manner. Specifically, the self-attention mechanism (Vaswani et al., 2017) that Transformers rely on seems to also benefit from first training the model on shorter sequences. This is further supported from our results in machine translation using Transformers, where we also observed that a length-based curriculum was beneficial (Section 4.4.2).

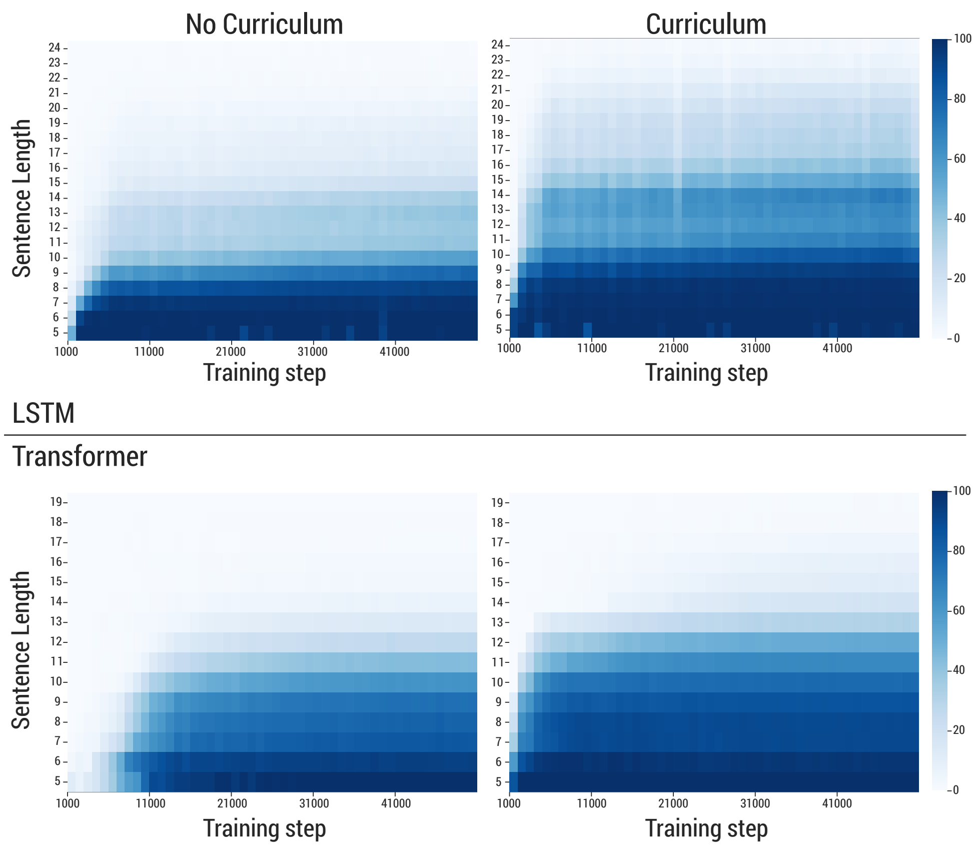


Figure 6.11: Accuracy per sequence length during training, for two sequence-to-sequence models, based on LSTMs (top row) and Transformers (bottom row). The left column displays the accuracies for models trained without a curriculum, and on the right for models trained with a length-based curriculum.

6.5 CASE STUDY #4: FINE-TUNING BERT ON GLUE DATA

In this case study we investigate the effect of curriculum learning on fine-tuning a model that has already been pretrained. This is very different than the previous case studies, where our arguments on why curriculum learning should help relied on the fact that predictions are noisy for a randomly initialized model. Nevertheless, we are interested to understand if there is still a benefit from using curriculum learning in this case, perhaps for different reasons.

Concretely, we consider a very popular Natural Language Processing (NLP) setting: fine-tuning a BERT model (Devlin et al., 2019) on the popular GLUE benchmark datasets (Wang et al., 2019a). GLUE (General Language Understanding Evaluation) is a collection of datasets aimed at evaluating language understanding capabilities of NLP models, including QNLI (Stanford Question Answering Dataset; Rajpurkar et al., 2016), SST (Stanford Sentiment Treebank; Socher et al., 2013), and RTE (Recognizing Textual Entailment; Dagan et al., 2005; Bar-Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009). BERT is based on the `Transformer` architecture (Vaswani et al., 2017), for which we have already seen some successful results with length-based curricula in Section 4.3 and Section 4.4.

We found that, in this case a length-based curriculum does not help the model in terms of speed or performance. In the following sections, we use the analysis tools we introduced earlier in this chapter to understand why this is the case.

6.5.1 *Experimental Setup*

MODELS. We follow the experimental setup from the official BERT repository[2], which reports results on the GLUE datasets using the original BERT model (`BERT-Base`) from Devlin et al. (2019), as well as using several other smaller versions of BERT (e.g., `BERT-Tiny`, `BERT-Small`, etc.; Turc et al., 2019). We use pretrained BERT models downloaded from TensorFlow Hub[3], and fine-tune them on GLUE with and without a curriculum. We provide a high-level visualization of BERT in Figure 6.12

DATA. We consider a subset of the GLUE datasets for which we analyze in detail the effect of curriculum learning. We chose a few diverse datasets where BERT performs significantly better than chance—SST-2, RTE, and QNLI—which we obtained from the TensorFlow datasets catalog[4]. SST-2 is the second version of the Stanford Sentiment Treebank dataset (Socher et al., 2013).

[2] https://github.com/google-research/bert
[3] https://www.tensorflow.org/hub
[4] https://www.tensorflow.org/datasets

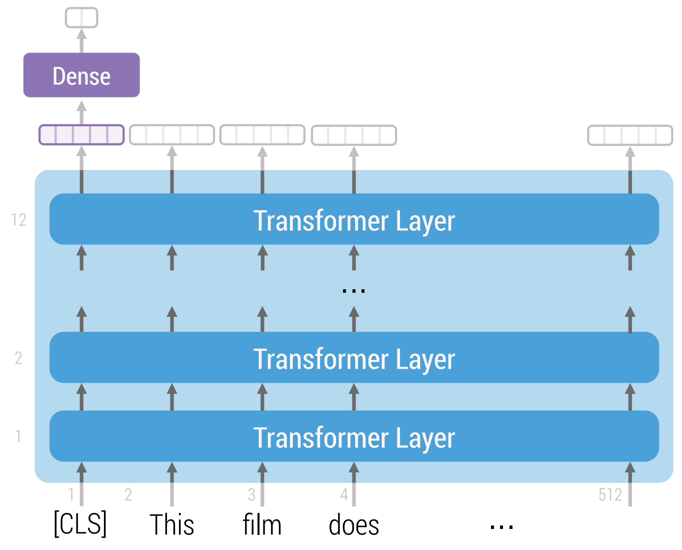


Figure 6.12: Visualization of the BERT-Base model architecture. BERT consists of multiple Transformer layers (Vaswani et al., 2017) that are pretrained on a very large corpus, as described in Devlin et al. (2019). The first output of the final Transformer layer is passed as input to a dense layer, which is initialized from scratch for each task.

It consists of sentences from movie reviews along with their corresponding human annotated sentiment, and the goal is to predict the sentiment of a given sentence. RTE is the Recognizing Textual Entailment dataset, combining multiple textual entailment datasets from a series of annual challenges: RTE1 (Dagan et al., 2005), RTE2 (Bar-Haim et al., 2006), RTE3 (Giampiccolo et al., 2007) and RTE5 (Bentivogli et al., 2009). Here, the samples come from news and Wikipedia text. QNLI, which stands for Question-answering NLI, is a modified version of the Stanford Question Answering Dataset (SQuAD; Rajpurkar et al., 2016), and consists of (question, context sentence) pairs. The goal is to determine if the context sentence contains the answer to the question. All three datasets are binary classification tasks. We fine-tune our models on the provided training splits,[4] and we report the accuracy on the validation dataset. Note that GLUE provides a leaderboard where one can submit the predictions on a test set, for which labels are not publicly available. We did not participate in this competition, and thus the results reported here are on the development set (which we further refer to as Dev) for which the correct labels have been released. Therefore, the results for the baseline may differ from those reported on the test set from the original sources. However, we tried to replicate the hyperparameter tuning setup described in Devlin et al. (2019) and `https://github.com/google-research/bert` to obtain the best results on the Dev set.

Table 6.1: Result for fine-tuning BERT models on a subset of the GLUE datasets. We report the accuracy mean and standard error over 4 runs, where the final projection layer is randomly initialized with different seeds, and the rest of the model layers are initialized with the pretrained weights of the corresponding BERT model.

Model	Dataset	Accuracy (%)	
		Baseline	Curriculum
BERT-Tiny	RTE	59.93 ± 0.75	58.93 ± 1.71
	QNLI	78.90 ± 0.14	79.23 ± 0.19
	SST-2	82.48 ± 0.11	82.26 ± 0.37
BERT-Medium	RTE	65.25 ± 0.93	64.26 ± 1.16
	QNLI	89.04 ± 0.04	88.92 ± 0.09
	SST-2	89.54 ± 0.21	89.39 ± 0.15
BERT-Base	RTE	67.37 ± 1.76	69.32 ± 1.46
	QNLI	90.36 ± 0.22	90.25 ± 0.21
	SST-2	92.03 ± 0.07	92.12 ± 0.12

training. We replicate the training setup from Devlin et al. (2019) for `BERT-Base` and from Turc et al. (2019) for the smaller versions, including the batch size, optimizer, and learning rate schedule. The official repository results are obtained by fine-tuning the smaller BERT models for only 4 epochs, and `BERT-Base` for 3 epochs. We replicate this setting.

curriculum. We used the curriculum learning framework proposed in Section 4.1, which was successful for all the applications in Chapter 4. Here, we use the sentence length as our difficulty metric, and apply the Square Root competence function (a setting which was successful in all experiments in Chapter 4). We evaluate multiple curriculum lengths C.

results. We report results for 3 model sizes: `BERT-Tiny`, `BERT-Medium` and `BERT-Base` in Table 6.1. We tested multiple values for the curriculum length C, we report the results for the best performing one in Table 6.1. We also inspect if the curriculum helps the model converge faster. For this we plot the accuracy on the Dev set per training step in Figure 6.13. In this case, the curriculum is not helping the model converge faster. Interestingly, the baseline on its own converges very quickly, achieving an accuracy close to its maximum within 1 epoch.

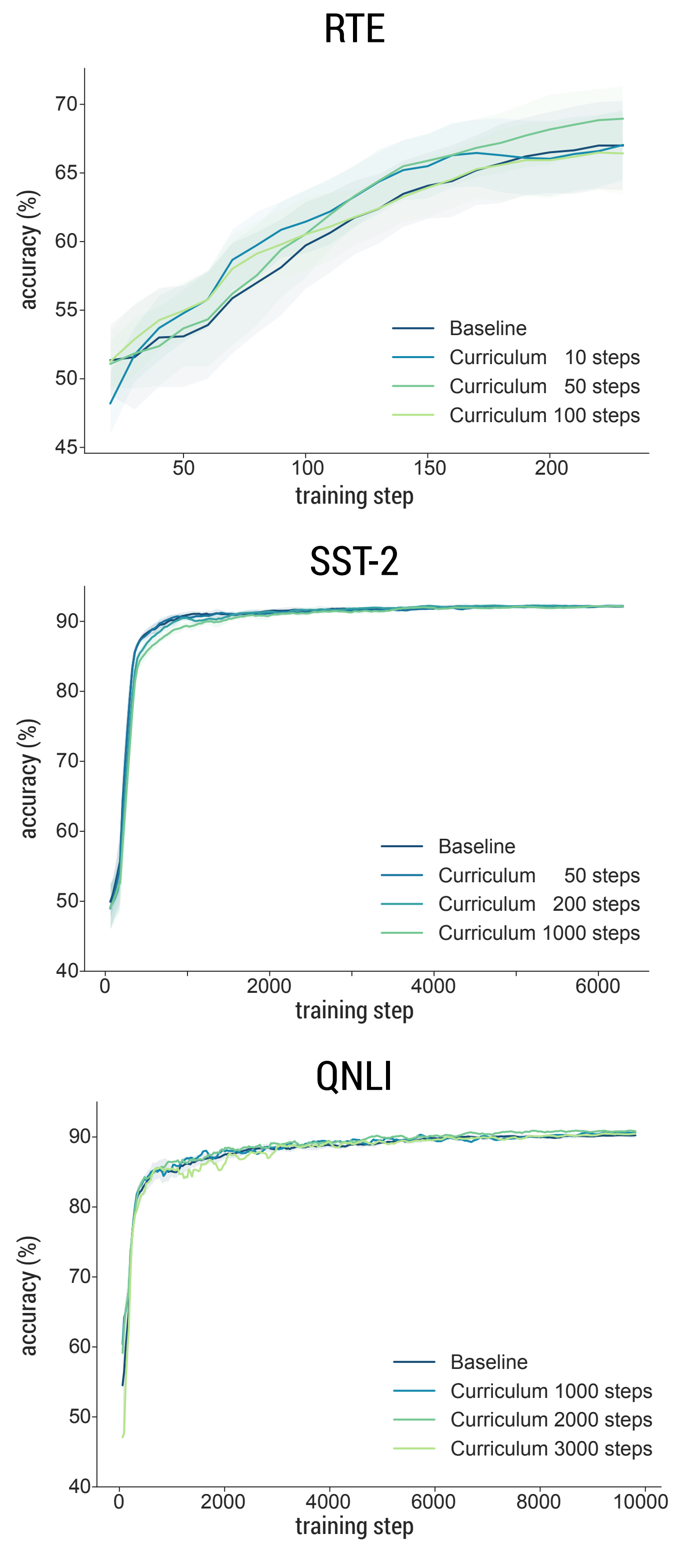


Figure 6.13: Accuracy mean and standard error per training step for `BERT-Base` on the RTE (top), SST-2 (middle) and QNLI (bottom). The model was fine-tuned for 3 epochs as in the original publication (Devlin et al., 2019), but to get a better resolution here we display the accuracy per training step.

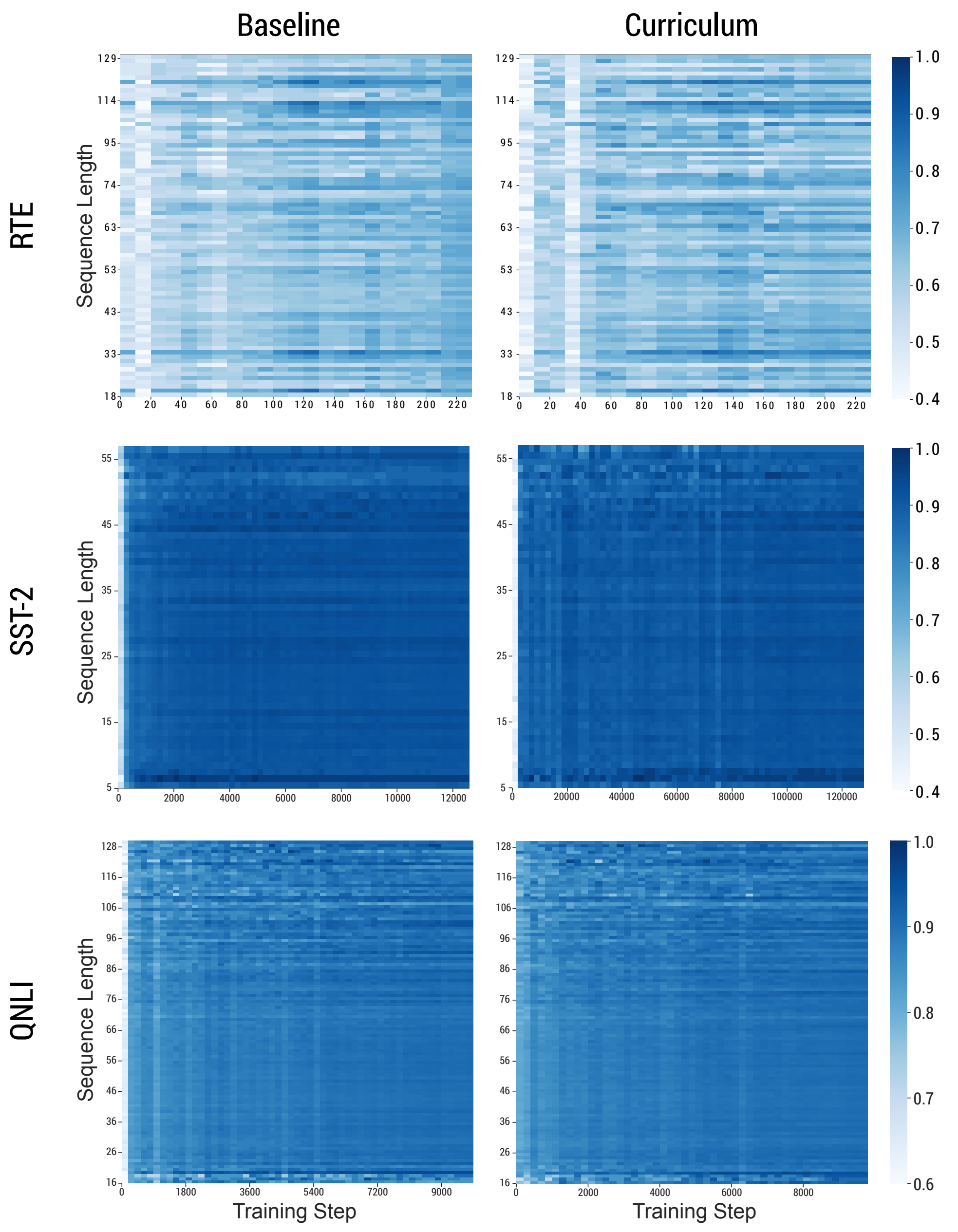


Figure 6.14: Accuracy per train step, aggregated for each input sequence length separately, for BERT-Base. Each row corresponds to a different GLUE dataset. On the left column, we show the accuracies for the baseline method trained without curriculum. On the right, we have the accuracies for the corresponding curriculum approach (with the same length C as the results in Table 6.1).

6.5.2 *Learning Sequences of Different Lengths*

As in the previous case studies, we plot the accuracy of the model during training, aggregated for each sequence length separately. This is displayed in Figure 6.14. Interestingly, neither the baseline model nor the curriculum show the pattern of decreasing accuracy with length, as we have seen in the previous experiments. This may be explained by the fact that we use BERT which is a pretrained model that already knows how to embed long sequences, despite the fact that the final layer is not fine-tuned specifically for the GLUE tasks that we are evaluating on. This could also be the reason why a length-based curriculum is no longer impactful in this case.

6.5.3 *The Effect of the Curriculum on the Loss Landscape*

We plot the loss landscape for `BERT-Tiny`. We chose the smallest version of BERT, because having less parameters helps us obtain more accurate loss plots, since it is harder to preserve the variance between the trajectory points when projecting larger parameter spaces to 2D. This is shown in Figure 6.15 for the RTE dataset, for both a model trained with and without curriculum. The plot indicates that the initial model parameters already start in the attraction basin of a local minimum, perhaps due to the fact that the majority of the model parameters—which are in the Transformer layers—are already pretrained. Given this initial position and loss landscape shape, widening the local minimum would not provide further benefits, which explains why a length-based curriculum is not useful in this case.

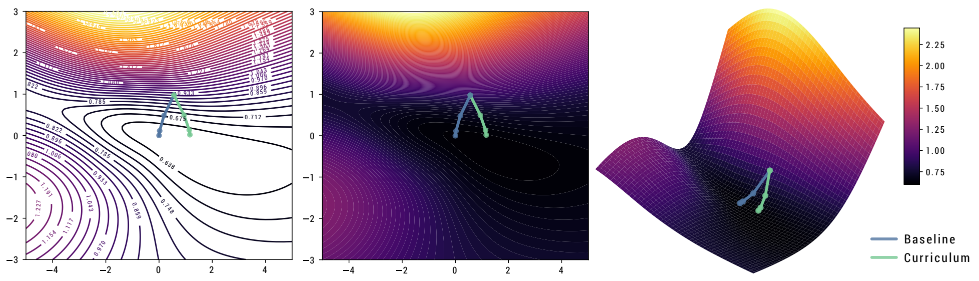


Figure 6.15: Loss landscape of the `BERT-Tiny` model on the RTE dataset.

6.5.4 *Discussion*

In this case study, we applied a length-based input-space curriculum to a popular NLP setting: fine-tuning a pretrained BERT model on the GLUE datasets. For this scenario, we found that our previously successful curriculum strategies no longer bring any improvements. We were able to provide an explanation based on the observation that, when using a pretrained BERT model, the sequences are no longer learned in order of length, which was the case when training from scratch. Moreover, through our loss landscape visualizations we were able to observe that the widening effect of the local minima that was helpful before, does not seem to matter in this case. This is because the pretrained model is already in a wide attraction basin of a local minimum. All in all, these results suggest that curricula are most beneficial when training a model from scratch.

6.6 KEY TAKEAWAYS

In this chapter, our goal was to understand what happens to the optimization process when using curriculum learning, and why it is beneficial to use in some cases. We focused on a problem setting where we have seen (from our work in Chapter 4 and from the literature) that curricula consistently improve the training process: applying length-based input-space curricula on problems with sequential data. We recapitulate our findings:

1. We considered the gradient update equations of composed models and recurrent neural networks. We observed that, when training a model from scratch on long sequences, there are several difficulties that hinder learning, such as compounding errors, internal covariate shift or vanishing gradients. These lead to a negative interaction between the gradient components corresponding to the early and the late time steps, and are particularly prevalent for long sequences.

2. The analytical discussion lead us to propose two hypotheses that apply to training recurrent neural networks (of any type) on sequences:
   (i) **Hypothesis 1:** When training composed models (in particular recurrent networks) without a curriculum on sequential data, there is some degree of correlation between the sequence length and the relative time during training when that sequence is learned.
   (ii) **Hypothesis 2:** When there is such a correlation even without a curriculum, a length-based curriculum can help the model converge faster.

3. These hypotheses were supported by the three experimental settings that we considered: learning the parity function (xor) using a basic RNN, learning to add two numbers digit by digit with LSTMs, and learning arithmetic in a sequence-to-sequence fashion with LSTMs and Transformers.

4. Using visualizations of the loss landscapes produced at different points in the curriculum, we were able to observe an interesting effect of training on incrementally longer sequences. The loss landscape on short sequences has *wider local minima*, giving the model the possibility to find a descent direction towards farther local minima that otherwise would not be accessible from the initialization point via gradient descent. Then, gradually allowing longer and longer sequences drives the model parameters to a point that is a local optimum for the entire training distribution.

5. We also applied a length-based curriculum on a fine-tuning problem. We fine-tuned the BERT model (Devlin et al., 2019) on a subset of the

GLUE datasets (Wang et al., 2019a). In this case, the curriculum was not able to provide any improvements to the training process. We were able to understand these effects by noticing that Hypothesis 1 no longer applies for pretrained BERT models, and by observing using loss landscape visualizations that the pretrained model already starts in the attraction basin of a local minimum.

7 CONCLUSION

In the introduction of this thesis we made the following statement about curriculum learning:

THESIS STATEMENT: *AI systems that learn like humans, starting with easy problems and gradually tackling more and more difficult ones, have the potential to reach* ***better local optima*** *and/or* ***converge faster****. Furthermore, the learning benefits gained using a curriculum depend on the choice of* ***curriculum****, the* ***size and type of data****, and the* ***model architecture****.*

Throughout this thesis, our goal was to provide evidence that supports this statement, by discovering different problem settings in which different forms of curriculum learning are beneficial, and understanding the types of benefits they provide.

We started this quest by first defining formally what curriculum learning is (Section 2.1), and how it differs from other learning paradigms (Section 2.2). Using these definitions, we were able to identify three broad categories of curriculum learning methods: curriculum in input space, in task space, and in model space. We then used this classification to discuss existing methods from the curriculum learning literature, and provide an overview of the field, from early ideas that started in the field of cognitive neuroscience to state-of-the-art methods (Chapter 3).

Moreover, also proposed new curriculum learning methods and applied them to a variety of models and problem settings, from teaching an LSTM to solve basic arithmetic problems, to neural machine translation using Transformers, image classification using convolutional neural networks, and compositional multitask learning problems. We structured the discussion around the categories identified earlier, and introduced curricula in input space in Chapter 4 and curricula in task space in Chapter 5.

Furthermore, we also conducted analyses to understand why curriculum learning leads to the observed effects (Chapter 6). We focused this discussion on a problem setting where curriculum learning has proved to be consistently successful, both in our work and related literature: curriculum learning in input space for problems with sequences.

We hope that the work included in this thesis will provide insights for future research in the field, as well as useful ideas for how to use CL in practical applications.

7.1 KEY RESULTS

We summarize the key results obtained in this thesis:

1. We introduced a definition of curriculum learning, which allows us to formally describe what curriculum learning is and how it compares to other learning paradigms (Section 2.1).

2. We identified three broad categories of curriculum learning methods: curriculum in input space, curriculum in task space, and curriculum in model space (Section 2.2).

3. We provided a survey of curriculum learning methods, from early ideas in cognitive neuroscience, to state-of-the-art approaches (Chapter 3). We structured the discussion around the three categories identified above.

4. We introduced a curriculum learning framework that can take any sample difficulty metrics, and combine them with our proposed pacing functions. We applied this framework successfully to multiple problems, including learning addition digit-by-digit with LSTM networks, learning arithmetic using sequence-to-sequence models (LSTM, Transformer), neural machine translation with LSTMs and Transformers, and multimodal image understanding combining images and captions using a CNN-RNN network.

5. Through these applications, we learned that curricula can work well if chosen carefully, for a variety of models, data modalities, and data regimes.

6. We found that curriculum learning can help the model not only train faster, but also obtain a better performance at the end of training.

7. For all experiments, there seems to be a consistent trend in the benefits (i.e. gain in accuracy or training speed) obtained with curriculum learning versus the curriculum length: as we increase the curriculum length, the benefits steadily increase up to an optimal point, and then start decreasing. Moreover, curricula that are excessively long can in fact harm learning, making it slower to train and potentially getting stuck in bad local optima.

8. In Chapter 5 we explained why for some types of problems (e.g., image classification) curricula in input space have limitations, and proposed the use of curricula in task space. Moreover, we introduced a curriculum learning algorithm targeted at multiclass classification

problems. Our algorithm automatically derives a series of easier auxiliary tasks which can be used to pretrain the model before tackling the target task (Section 5.2). This approach was able to improve the performance of several image classification models (CNN, Resnet, WideResnet), on multiple standard datasets (`CIFAR100`, `Tiny-Imagenet`, etc.).

9. We also showcased how curriculum learning can be used in the context of multitask learning on compositional tasks, both in situations where we model the task composition explicitly (e.g., in Section 5.3.1 we expressed multiplication through addition, and addition through counting), or implicitly (e.g., in Section 5.3.2 we do not explicitly represent the compositional relationship between tasks). In both scenarios, curricula in task space were able to improve model performance.

10. In Chapter 6, we set out to understand why curriculum learning helps in some cases, and why it does not in others. We considered a setting where we apply a length-based curriculum on sequential data. By inspecting the gradient update equations of composed models and recurrent neural networks, we proposed two hypotheses:
    (i) **Hypothesis 1:** When training composed models (in particular recurrent networks) without a curriculum on sequential data, there is some degree of correlation between the sequence length and the relative time during training when that sequence is learned.
    (ii) **Hypothesis 2:** When there is such a correlation even without a curriculum, a length-based curriculum can help the model converge faster.

11. These hypotheses were confirmed for three case studies (learning the parity function with basic RNNs, learning addition digit-by-digit with LSTMs, and learning arthmetic using an encode-decoder architecture). Furthermore, by visualizing the loss landscape at different points during a curriculum, we observed that length-based curricula widen the local minima, allowing the model to reach certain points of the parameter space during gradient optimization that otherwise it would not be able to.

12. We also applied a length-based curriculum on a popular problem setting: fine-tuning `BERT` (Devlin et al., 2019) on a few benchmark language datasets. In this case, we did not observe any improvements from using curriculum learning, and we explained this result by the fact that the pretrained model no longer follows Hypothesis 1. Moreover, from the loss landscape visualizations, we observed that the pretrained `BERT` model already starts in the attraction basin of a wide local minimum.

7.2 FUTURE WORK

While the field of curriculum learning is not new, it is still at the beginning, with several directions to explore and improve.

Out of the types of curricula discussed in this thesis, curriculum in input space is by far the most popular category. However, this type of curriculum is more common in certain application areas, such as natural language processing, and has not gained ground in many other areas yet. This is partially due to the difficulty of proposing intuitive sample difficulty metrics for certain fields (e.g. computer vision). For such areas, more automated approaches (e.g., self-paced learning, automated curricula) are more suitable. While significant progress has been made in this direction, there is still a need for more adaptable automated curriculum learning methods that can be applied to new fields "out-the-box", without significant re-tuning efforts. Additionally, for practical reasons, such methods should be easy to implement and should not add significant computation and memory overhead to the original system.

Moreover, we have seen in this thesis that curricula in task space show promise, both for single-task and multitask learning scenarios. However, for our settings the multitask curricula were designed manually. For future work, it would be impactful to design task-space curricula that can automatically derive auxiliary tasks (for both single and multi-task learning), or can automatically derive the relationships between provided tasks (in multitask learning) and how to schedule them for training.

Furthermore, we discussed in our literature review in Chapter 3 about curricula in model space. The idea of modifying the model architecture or model capabilities during training has been explored in various ares of machine learning, but very few approaches have considered it in the context of curriculum learning. Future work may look back at the original inspiration for curriculum learning—the human brain—to further develop curricula in model space.

Finally, while this thesis has touched on understanding the effects of curriculum learning on the optimization process, more work is needed to truly explain the effects of various forms of curriculum learning from a theoretical perspective.

A CURRICULUM IN TASK SPACE: SUPPLEMENTARY

A.1 COARSE-TO-FINE CURRICULUM LEARNING: SUPPLEMENTARY

A.1.1 *Confusion Matrix vs Embedding Similarity*

In Section 5.2.2, we considered two different measures of class similarity: one based on the confusion matrix and one based on the class embedding distance. We experimented with both and observed a few disadvantages to using the confusion matrix. This made us opt for a measure that is based on the class embedding distance. In what follows, we discuss these disadvantages. We recommend reading Section 5.2.2 before proceeding with this section, as some of the issues discussed here are related to the way we intend to use the class similarity measure.

First, we need to define how the confusion matrix is estimated from the data. We define the confusion matrix as $\mathbf{C} \in [0,1]^{K \times K}$, where K is the number of classes and $\mathbf{C}_{ij}$ is the probability that the model predicts class j when it should have predicted class i, and $\sum_{j=1}^{K} \mathbf{C}_{ij} = 1$. Given an existing model, this matrix can be approximated using the sample estimate of each probability on a validation dataset. Using $\mathbf{C}$ as similarity between the classes in the hierarchical clustering algorithm, we encountered the following issues:

– If the training set is imbalanced, and one class dominates in the number of training examples, it is possible that the classifier often confuses all other classes for the dominating class, instead of mistaking them for more semantically similar classes. This is because making such a mistake during training is likely to incur a lower loss. Thus, using the confusion matrix as similarity measure, the most similar class to all other classes could be the dominating class, regardless of its semantics. As a consequence, affinity clustering (Bateni et al., 2017) will connect all classes to the dominating class in the first level of the hierarchy, resulting in a degenerate case with no auxiliary functions.

– The confusion matrix is not a proper distance metric, as is not symmetric and does not necessarily satisfy the triangle inequality. Although this did not necessary pose a problem for our implementation of the affinity clustering algorithm, being a proper distance metric is important for other hierarchical clustering algorithms, and could cause issues if the users of our algorithm chose to use a different clustering method. In our experiments, we made it symmetric by adding its transpose to itself (i.e. $\mathbf{C} + \mathbf{C}^\top$).

- If the classifier does not confuse two classes i and j at all in the validation set, their confusion count will be 0, and they will thus be considered highly dissimilar. Using an example from CIFAR-100, the classes `willow_tree`, `oak_tree`, `palm_tree` and `pine_tree` are similar semantically and so, intuitively we would expect them to be grouped in the same cluster. However, suppose that in our validation set `willow_tree` and `palm_tree` are always confused only with each other, and the same happens for `oak_tree` and `pine_tree`. Then `willow_tree` and `palm_tree` classes will be grouped together early on, and `oak_tree` and `pine_tree` will also be grouped together early. After this grouping, the confusion between the two new clusters will be 0 and so they will only be merged at the top of the hierarchy. The class embedding similarity, being the cosine distance between two high dimensional vectors, does not suffer from the same issue.

A.1.2 *A Staged Coarse-to-Fine Approach*

As mentioned in Section 5.2.3, there is also another version of our algorithm, where we can train a different classifier f_{θ_ℓ} at each level ℓ of the hierarchy. The main difficulty is that $f_{\theta_{\ell+1}}$ and f_{θ_ℓ} make predictions for different number of classes, and thus the number of parameters in $\theta_{\ell+1}$ does not directly match that of θ_ℓ.

Let us assume that for any level ℓ,

$$f_\theta = \underbrace{f^H_{\theta^H}}_{\texttt{predictor}} \circ \underbrace{f^{H-1}_{\theta^{H-1}} \circ \cdots \circ f^1_{\theta^1}}_{\texttt{encoder}}, \tag{A.1}$$

where $\circ$ denotes function composition, H is the number of layers in the network, and $\theta = \{\theta^1, \ldots, \theta^H\}$. This is a simple decomposition that applies to most deep learning models that are used in practice. Intuitively, the encoder converts its input to a latent representation (i.e., embedding), that is then processed by the predictor to produce a probability distribution over classes. Let us further denote the parameters of the predictor by θ^{pred} and those of the encoder by θ^{enc} (we have that $\theta = \theta^{\text{pred}} \cup \theta^{\text{enc}}$). We suggest decomposing f_θ such that most of the model parameters are part of the encoder. In our experiments, the predictor is simply the output layer of a neural network, whose output dimensionality changes at every hierarchy level, depending on the number of clusters in that level. When training $f_{\theta_{\ell+1}}$, we initialize its encoder parameters as $\theta^{\text{enc}}_{\ell+1} = \theta^{\text{enc}}_{\ell}$, and its predictor parameters $\theta^{\text{pred}}_{\ell+1}$ randomly. Thus, knowledge transfer in this case happens through the initialization of the encoder parameters, which often include

Algorithm A.1: Coarse-To-Fine Curriculum: A Staged Approach

```
   // This is an overview of the proposed staged curriculum algorithm.
   Inputs: Number of classes $K$.
           Training data $\{x_i, y_i\}_{i=1}^N$.
           Trainable baseline model $f_\theta$.
1  Train $f_\theta$ on the provided training data $\{x_i, y_i\}_{i=1}^N$.
2  clustersPerLevel ← GenerateClassHierarchy( K,$\{x_i, y_i\}_{i=1}^N$,$f_\theta$)
3  M ← clustersPerLevel.length
   // Train the model at each level of the hierarchy.
4  originalLabels ← [1,...,K]
5  for $l \leftarrow 0, \ldots,$ M - 1 do
6      clusters ← clustersPerLevel[$l+1$]
7      newLabels ← TransformLabels( $\{y_i\}_{i=1}^N$, clusters)
8      if $l = 0$ then
9          $\theta_{l+1}^{\texttt{encoder}} \leftarrow$ random().
10     else
11         $\theta_{l+1}^{\texttt{encoder}} \leftarrow \theta_{l}^{\texttt{encoder}}$
12     $\theta_{l+1}^{\texttt{predictor}} \leftarrow$ random().
13     Train $f_{\theta_{l+1}}$ using newLabels as the target labels.
   Output: $f_{\theta_{[M]}}$.
```

most of the model parameters. The main intuition behind this decision is that lower level processing (e.g., converting pixels to edges and potentially to abstract semantic features) is a step that is necessary for most levels of granularity. However, the predictor parameters are specific to the each task (i.e., the predictor decides how the higher-level features are assembled together to solve each task). θ_1 is initialized randomly in our experiments. Putting the pieces together, first we generate a class hierarchy as described in Algorithm 5.2, and then we train a classifier at each level of the hierarchy, transferring knowledge via the model parameters as described above. These steps are detailed in Algorithm A.1. We refer to this approach as the *staged* variant of our curriculum learning algorithm, and we refer to the approach discussed in the main paper as the *continuous* variant.

The two proposed training algorithms, staged and continuous, come with advantages and disadvantages. Here we discuss the trade-offs in terms of hyperparameters and computational complexity.

HYPERPARAMETERS. An advantage of the staged approach is that it introduces no extra hyperparameters that we need to tune. The number of levels in the hierarchy is automatically determined by the output of the hierarchical clustering algorithm and we train until convergence for each level. The continuous approach introduces a hyperparameter, which is total number of epochs to be spent on the curriculum, T, and which is then split equally among the levels of the hierarchy.

COMPUTATIONAL COMPLEXITY. Let $\mathcal{C}$ be the computational complexity required to train the baseline model to convergence. Since affinity clustering guarantees that our class hierarchy will have at most $\log K$ levels, where K is the original number of classes, the computational complexity of the staged approach will be at most $\mathcal{C} \log K$. However, in practice we observed that the cost is significantly less for two reasons: (i) training the coarse-grained classifiers converges much faster than the fine-grained baseline, and (ii) after f_{θ_1}, all other levels are already pre-trained by being initialized with the parameters from the previous level, and thus require very few training iterations. Evidence of this behavior can be seen in Figure A.1. As discussed in Section 5.2.4, our continuous curriculum has roughly the same computational complexity as the baseline. Using our heuristic for setting the number of epochs T, the continuous curriculum typically requires about as many training iterations as the baseline model.

EXPERIMENTS USING STAGED COARSE-TO-FINE CURRICULUM. We show results for the staged curriculum, on similar settings to the experiments reported in the main paper, using the CNN model. Note that, for all these experiments, we do not use any image augmentation techniques or specialized learning rate schedules, since we wanted to understand the effect of our methods without extra help from such techniques. The results are reported in Table A.1 and Figure A.1 and Figure A.2.

Table A.1 shows that the staged approach also provides a significant boost over the baseline method, occasionally even better than the continuous approach (albeit at a larger computational cost).

Figure A.1 shows the training curve of the staged curriculum at each level of the hierarchy, as well as a comparison with the baseline and the continuous method shown in Figure 5.7. Importantly, even for CIFAR-100 where we have 100 labels, the hierarchy only contains two auxiliary levels with 6 and 27 clusters, respectively. This means that the staged approach can achieve accuracy improvements in the order of 3-4% with at most 3 times the computational cost. In practice, the actual cost is much less than that, because each hierarchy level now needs much fewer iterations to converge than the baseline, as shown in Figure A.1. This figure also confirms our intuition that the auxiliary tasks obtained with our approach are indeed sorted in order of difficulty, in the sense that the accuracy of the model at solving levels 3, 2, and 1 is monotonically increasing. Additionally, we directly notice the benefits of pre-training by looking the accuracy of the staged models after one epoch.

Dataset	#Class	#Samples	Accuracy	Accuracy Gain	
			Baseline	Coarse-to-Fine-Staged	Coarse-to-Fine-Continuous
CIFAR-10	10	50,000	70.92 ± 0.37	0.92 ± 0.32	0.69 ± 0.32
CIFAR-10	10	20,000	64.66 ± 0.53	1.86 ± 0.23	1.28 ± 0.60
CIFAR-10	10	10,000	59.52 ± 0.35	1.01 ± 0.13	1.24 ± 0.46
CIFAR-10	10	5,000	53.64 ± 0.19	2.22 ± 0.42	1.57 ± 0.39
CIFAR-100 Coarse	20	50,000	49.63 ± 0.35	0.91 ± 0.37	1.22 ± 0.38
CIFAR-100 Coarse	20	20,000	42.04 ± 0.29	1.03 ± 0.22	1.84 ± 0.51
CIFAR-100 Coarse	20	10,000	36.61 ± 0.19	1.38 ± 0.53	1.77 ± 0.56
CIFAR-100 Coarse	20	5,000	31.80 ± 0.28	1.17 ± 0.46	1.38 ± 0.22
CIFAR-100	100	50,000	35.87 ± 0.23	3.99 ± 0.24	3.31 ± 0.59
CIFAR-100	100	20,000	27.83 ± 0.34	4.40 ± 0.32	2.27 ± 0.37
CIFAR-100	100	10,000	21.96 ± 0.49	2.70 ± 0.22	2.67 ± 0.68
CIFAR-100	100	5,000	17.20 ± 0.20	2.35 ± 0.29	1.92 ± 0.24
Tiny-ImageNet	200	100,000	21.94 ± 0.19	3.79 ± 0.34	2.73 ± 0.49
Tiny-ImageNet	200	50,000	16.33 ± 0.32	3.64 ± 0.47	3.06 ± 0.33
Tiny-ImageNet	200	20,000	10.16 ± 0.22	2.94 ± 0.36	2.02 ± 0.34
Tiny-ImageNet	200	10,000	7.38 ± 0.11	2.01 ± 0.32	1.14 ± 0.19

Table A.1: Results on real datasets using the `CNN` architecture, showing the accuracy mean and standard error for the baseline model, computed over 5 runs, as well as the accuracy gain achieved by the two versions of our coarse-to-fine curriculum (staged and continuous), computed per run and then averaged. Note that these results were obtained without any image augmentation techniques or specialized learning rate schedules.

A.1.3 *Experimental Details*

architecture details. The Convolutional Neural Network (`CNN`) used in our experiments consists of these layers:

1. Convolution: 2D convolution using a 3×3 filter with 32 channels, followed by ReLU activation.
2. Pooling: Max pooling using a 2×2 window.
3. Convolution: 2D convolution using a 3×3 filter with 64 channels, followed by ReLU activation.
4. Pooling: Max pooling using a 2×2 window.
5. Convolution: 2D convolution using a 3×3 filter with 64 channels, followed by ReLU activation.
6. Projection: Fully connected layer performing a linear projection to the output space dimensionality (i.e., number of classes), returning logits.

The `WideResnet-28-10` and `Resnet18` architectures were implemented after the code released by Wan et al. (2021), and are similar to the original publications (He et al., 2016; Zagoruyko and Komodakis, 2016).

training. We implemented our method using the TensorFlow framework (Abadi et al., 2016). All models were trained by minimizing the softmax cross-entropy loss function.

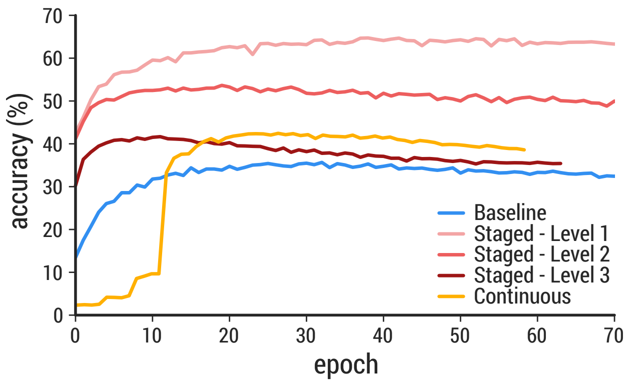


Figure A.1: Accuracy per epoch for the baseline and our algorithm, on the CIFAR-100 dataset.

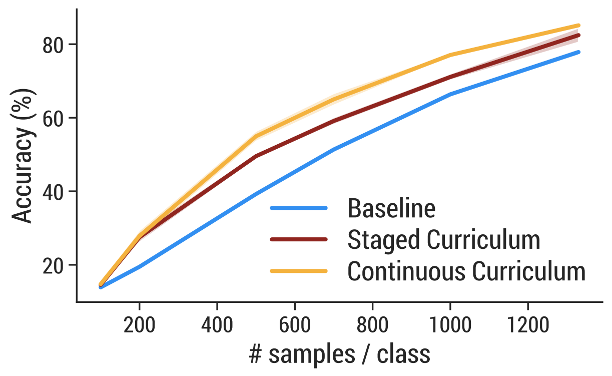


Figure A.2: Accuracy mean and standard error for the baseline and the curriculum model, averaged over 5 runs, on Shapes.

All the CNN experiments used the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.001 and a batch size of 512 samples. We also employed early stopping by terminating training when validation accuracy did not improve within the last 50 epochs (we made sure that this number is large enough by visually inspecting the validation curve). We also made sure that the baseline is allowed to perform at least as many epochs as the curriculum. We report test accuracy statistics for the iteration that corresponds to the best validation set performance. The validation dataset is obtained by setting aside 20% of the training examples, chosen uniformly at random.

For the Resnet and WideResnet experiments we replicated the setting of Wan et al. (2021) from https://github.com/alvinwan/neural-backed-decision-trees, in order to be able to directly compare with their results. Concretely, we used a Stochastic Gradient Descent (SGD) optimizer with momentum 0.9, batch size 128, and weight decay value of $5e-4$, and trained the models for 200 epochs. The learning rate schedule starts with a learning rate of 0.1 and is divided by 10 twice: $\frac{3}{7}$ and $\frac{5}{7}$ of the way through training. Because of this step-wise learning rate, our heuristic for choosing the curriculum length (when the baseline has reached 90% of its peak accuracy) no longer applies, so in this case the curriculum length was chosen based on validation set performance among the choices {5, 10, 20, 30, 40, 50}. First we randomly split the training set in a 90% train and 10% validation and trained the models with each of these curriculum lengths. We then chose the best performing curriculum length on the validation set, and retrained on the full training set using this length. Importantly, in all our experiments we only allowed both the baseline and the corresponding curriculum models to train for *exactly the same number of epochs*.

To match the implementation of Wan et al. (2021), for the large models we used the same data augmentation techniques they did, while for the CNN experiments we opted out of data augmentation to see just how much the curriculum impacts this simple model.

Finally, all our experiments were performed using a single Nvidia Titan X GPU, and the code for reproducing our results is available at https://github.com/otiliastr/coarse-to-fine-curriculum.

A.1.4 *Other Related Work*

In Section 5.2.4, we discussed how our coarse-to-fine approach may be related to hierarchical classification. Here, we expand on the main directions in hierarchical classification and compare them with our method. A popular survey on hierarchical classification (Silla and Freitas, 2011) organizes the hierarchical classification literature in three main types of methods:

– *flat classification approaches*, which ignore the class hierarchy and consider only the fine-grained leaf-node classes. This is equivalent to any standard classification problem.

– *local classification approaches*, which typically go top-down through the label hierarchy and train multiple classifiers along the way that take into account only local information (e.g., Bennett and Nguyen, 2009; Ramaswamy et al., 2015; Ramírez-Corona et al., 2016). These approaches use various ways of incorporating local information, such as having a classifier per node (Jin et al., 2008; Valentini and Re, 2009), per parent node (Gauch et al., 2009), or per hierarchy level (Clare and King, 2003). Knowledge can be passed down through the hierarchy in various ways, for example using the predictions of the parent node as input to the current classifier (Bennett and Nguyen, 2009; Holden and Freitas, 2009). A more recent approach (Xu and Geng, 2019) is based on the idea that two labels with a common ancestor in the class hierarchy are correlated, and explicitly models this correlation in the label distribution. Approaches in this category are typically computationally more expensive, and tend to be sensitive to error propagation along the hierarchy levels.

– *global classification approaches*, which train a single classification function that takes into account the entire class hierarchy at once (Cai and Hofmann, 2004; Wang et al., 2009; Xiao et al., 2011; Cerri et al., 2012). For example, Cai and Hofmann (2004) do so by designing a generalization of Support Vector Machines (SVM) using discriminant functions that decompose into contributions from different levels of the hierarchy. Xiao et al. (2011) train a hierarchical SVM which consists of a classifier at each node, but information about the whole hierarchy is encoded in a regularization term that encourages the normal vector of the classifying hyperplane at each node to be orthogonal to those of its ancestors. The more recent work of Wehrmann et al. (2018) proposes a new neural network architecture for class hierarchies, which can make predictions at different levels of the hierarchy and is trained by combining multiple losses: a local loss, a global loss, and a loss that penalizes predictions that violate the hierarchy. This approach is not merely a mechanism for training arbitrary models, but is tied together to the proposed architecture.

Our approach is similar to local node classification approaches in that we also train a classifier at each level. It is also in some sense similar to global classification approaches, because at the end of training, the model at the final level of the hierarchy is able to preserve knowledge about the rest of the hierarchy through the parameters that have been propagated through the levels. However, to the best of our knowledge, none of these hierarchical classification approaches pass information across levels only through the model parameters. Typically this information is encoded directly in the model itself, which often means that the hierarchy is also used during inference, not just during training.

BIBLIOGRAPHY

[1] Robin Sibson. "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method." In: *The Computer Journal* 16.1 (1973), pp. 30–34.

[2] Elizabeth K Warrington. "The selective impairment of semantic memory." In: *The Quarterly Journal of Experimental Psychology* 27.4 (1975), pp. 635–657.

[3] Daniel Defays. "An Efficient Algorithm for a Complete Link Method." In: *The Computer Journal* 20.4 (1977), pp. 364–366.

[4] Elissa L Newport. "Constraints on learning and their role in language acquisition: Studies of the acquisition of American Sign Language." In: *language Sciences* 10.1 (1988), pp. 147–172.

[5] Robert Kail. *The Development of Memory in Children*. WH Freeman/-Times Books/Henry Holt & Co, 1990.

[6] Elissa L Newport. "Maturational constraints on language learning." In: *Cognitive science* 14.1 (1990), pp. 11–28.

[7] Sepp Hochreiter. "Untersuchungen zu Dynamischen Neuronalen Netzen." In: *Diploma, Technische Universität München* 91.1 (1991).

[8] Jean M Mandler. "How to build a baby: II. Conceptual primitives." In: *Psychological Review* 99.4 (1992), p. 587.

[9] Jeffrey L Elman. "Learning and Development in Neural Networks: The Importance of Starting Small." In: *Cognition* 48.1 (1993), pp. 71–99.

[10] Jean M Mandler and Laraine McDonough. "Concept formation in infancy." In: *Cognitive Development* 8.3 (1993), pp. 291–318.

[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent Is Difficult." In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[12] Rich Caruana. "Multitask Learning." In: *Machine learning* 28.1 (1997), pp. 41–75.

[13] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." In: *Neural Computation* 9 (1997), pp. 1735–1780.

[14] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and Acting in Partially Observable Stochastic Domains." In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[15] Tsungnan Lin, Bill G Horne, and C Lee Giles. "How Embedded Memory in Recurrent Neural Network Architectures Helps Learning Long-Term Temporal Dependencies." In: *Neural Networks* 11.5 (1998), pp. 861–868.

[16] Douglas LT Rohde and David C Plaut. "Language Acquisition in the Absence of Explicit Negative Evidence: How Important is Starting Small?" In: *Cognition* 72.1 (1999), pp. 67–109.

[17] Robert E. Schapire. "A Brief Introduction to Boosting." In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'99. Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999, pp. 1401–1406. URL: http://dl.acm.org/citation.cfm?id=1624312.1624417.

[18] Jean M Mandler. "Perceptual and conceptual processes in infancy." In: *Journal of Cognition and Development* 1.1 (2000), pp. 3–36.

[19] Francois Fleuret and Donald Geman. "Coarse-to-fine face detection." In: *International Journal of Computer Vision* 41.1-2 (2001), pp. 85–107.

[20] Sabina Pauen. "The global-to-basic level shift in infants' categorical thinking: First evidence from a longitudinal study." In: *International Journal of Behavioral Development* 26.6 (2002), pp. 492–499.

[21] Hichem Sahbi and Nozha Boujemaa. "Coarse-to-fine support vector classifiers for face detection." In: *Object recognition supported by user interaction for service robots*. Vol. 3. IEEE. 2002, pp. 359–362.

[22] Amanda Clare and Ross D King. "Predicting gene function in Saccharomyces cerevisiae." In: *Bioinformatics* 19.suppl_2 (2003), pp. ii42–ii49.

[23] James L McClelland and Timothy T Rogers. "The Parallel Distributed Processing Approach to Semantic Cognition." In: *Nature Reviews Neuroscience* 4.4 (2003), pp. 310–322.

[24] Douglas L. T. Rohde and David C. Plaut. "Less is Less in Language Acquisition." In: *Connectionist Modelling of Cognitive Development*. Psychology Press, 2003.

[25] Yali Amit, Donald Geman, and Xiaodong Fan. "A coarse-to-fine strategy for multiclass shape detection." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.12 (2004), pp. 1606–1621.

[26] Lijuan Cai and Thomas Hofmann. "Hierarchical document categorization with support vector machines." In: *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*. 2004, pp. 78–87.

[27] Ido Dagan, Oren Glickman, and Bernardo Magnini. "The PASCAL recognising textual entailment challenge." In: *Machine Learning Challenges Workshop*. Springer. 2005, pp. 177–190.

[28] Pierre Moreels and Pietro Perona. *Probabilistic coarse-to-fine object recognition*. Tech. rep. Technical report, 2005.

[29] Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. "The second pascal recognising textual entailment challenge." In: *Proceedings of the second PASCAL challenges workshop on recognising textual entailment*. Vol. 6. 1. Venice. 2006, pp. 6–4.

[30] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. "Greedy layer-wise training of deep networks." In: *Advances in neural information processing systems* 19 (2007), p. 153.

[31] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. "The third pascal recognizing textual entailment challenge." In: *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*. Association for Computational Linguistics. 2007, pp. 1–9.

[32] Ya Xue, X. Liao, L. Carin, and B. Krishnapuram. "Multi-Task Learning for Classification with Dirichlet Process Priors." In: *J. Mach. Learn. Res.* 8 (2007), pp. 35–63.

[33] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning." In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 160–167.

[34] Bo Jin, Brian Muller, Chengxiang Zhai, and Xinghua Lu. "Multilabel literature classification based on the Gene Ontology graph." In: *BMC Bioinformatics* 9.1 (2008), p. 525.

[35] Burr Settles and Mark Craven. "An Analysis of Active Learning Strategies for Sequence Labeling Tasks." In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Honolulu, Hawaii: Association for Computational Linguistics, Oct. 2008, pp. 1070–1079. URL: http://www.aclweb.org/anthology/D08-1112.

[36] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. "Curriculum Learning." In: *International Conference on Machine Learning*. ACM. 2009, pp. 41–48.

[37] Paul N Bennett and Nam Nguyen. "Refined experts: improving classification in large taxonomies." In: *Proceedings of the 32nd international ACM SIGIR Conference on Research and Development in Information Retrieval*. 2009, pp. 11–18.

[38] Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. "The Fifth PASCAL Recognizing Textual Entailment Challenge." In: *TAC*. 2009.

[39] Susan Gauch, Aravind Chandramouli, and Shankar Ranganathan. "Training a hierarchical classifier using inter document relationships." In: *Journal of the American Society for Information Science and Technology* 60.1 (2009), pp. 47–58.

[40] Gholamreza Haffari, Maxim Roy, and Anoop Sarkar. "Active Learning for Statistical Phrase-based Machine Translation." In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. NAACL '09. Boulder, Colorado: Association for Computational Linguistics, 2009, pp. 415–423. ISBN: 978-1-932432-41-1. URL: http://dl.acm.org/citation.cfm?id=1620754.1620815.

[41] Nicholas Holden and Alex A Freitas. "Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation." In: *Soft Computing* 13.3 (2009), pp. 259–272.

[42] S. Ji, D. Dunson, and L. Carin. "Multitask Compressive Sensing." In: *IEEE Transactions on Signal Processing* 57 (2009), pp. 92–106.

[43] Leonard Kaufman and Peter J Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Vol. 344. John Wiley & Sons, 2009.

[44] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. Massachusetts Institute of Technology and New York University, 2009.

[45] Kai A Krueger and Peter Dayan. "Flexible shaping: How learning in small steps helps." In: *Cognition* 110.3 (2009), pp. 380–394.

[46] Giorgio Valentini and Matteo Re. "Weighted True Path Rule: a multilabel hierarchical algorithm for gene function prediction." In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases: 1st International Workshop on learning from Multi-Label Data*. 2009.

[47] Junhui Wang, Xiaotong Shen, and Wei Pan. "On large margin hierarchical classification with multiple paths." In: *Journal of the American Statistical Association* 104.487 (2009), pp. 1213–1223.

[48] Michael Bloodgood and Chris Callison-Burch. "Bucking the Trend: Large-Scale Cost-Focused Active Learning for Statistical Machine Translation." In: *ACL*. 2010.

[49] Yoav Goldberg and Michael Elhadad. "An efficient algorithm for easy-first non-directional dependency parsing." In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. 2010, pp. 742–750.

[50] M Pawan Kumar, Benjamin Packer, and Daphne Koller. "Self-paced learning for latent variable models." In: *Advances in Neural Information Processing Systems*. 2010, pp. 1189–1197.

[51] Valentin I. Spitkovsky, Hiyan Alshawi, and Dan Jurafsky. "From Baby Steps to Leapfrog: How "Less is More" in Unsupervised Dependency Parsing." In: *HLT-NAACL*. 2010.

[52] Yong Jae Lee and Kristen Grauman. "Learning the Easy Things First: Self-Paced Visual Category Discovery." In: *CVPR* (2011), pp. 1721–1728.

[53] Le Lu, Meizhu Liu, Xiaojing Ye, Shipeng Yu, and Heng Huang. "Coarse-to-fine classification via parametric and nonparametric models for computer-aided diagnosis." In: *Proceedings of the 20th ACM international conference on Information and knowledge management*. 2011, pp. 2509–2512.

[54] Carlos N Silla and Alex A Freitas. "A survey of hierarchical classification across different application domains." In: *Data Mining and Knowledge Discovery* 22.1-2 (2011), pp. 31–72.

[55] Lin Xiao, Dengyong Zhou, and Mingrui Wu. "Hierarchical Classification via Orthogonal Transfer." In: *ICML*. 2011.

[56] Qian Xu and Qiang Yang. "A survey of transfer and multitask learning in bioinformatics." In: *Journal of Computing Science and Engineering* 5.3 (2011), pp. 257–268.

[57] Vamshi Ambati. "Active Learning and Crowdsourcing for Machine Translation in Low Resource Scenarios." AAI3528171. PhD thesis. Pittsburgh, PA, USA, 2012. ISBN: 978-1-267-58215-7.

[58] Ricardo Cerri, Rodrigo C Barros, and Andre CPLF de Carvalho. "A genetic algorithm for hierarchical multi-label classification." In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. 2012, pp. 250–255.

[59] Bart Thomee and Michael S Lew. "Interactive Search in Image Retrieval: A Survey." In: *International Journal of Multimedia Information Retrieval* 1.2 (2012), pp. 71–86.

[60] Sebastian Zambanini and Martin Kampel. "Coarse-to-fine correspondence search for classifying ancient coins." In: *Asian Conference on Computer Vision*. Springer. 2012, pp. 25–36.

[61] Yifan Fu, Xingquan Zhu, and Bin Li. "A survey on instance selection for active learning." In: *Knowledge and information systems* 35.2 (2013), pp. 249–283.

[62] Nal Kalchbrenner and Phil Blunsom. "Recurrent continuous translation models." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013, pp. 1700–1709.

[63] Frank C Keil. *Semantic and conceptual development: An ontological perspective*. Harvard University Press, 2013.

[64] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318.

[65] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. "Recursive deep models for semantic compositionality over a sentiment treebank." In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642.

[66] Nitish Srivastava and Russ R Salakhutdinov. "Discriminative transfer learning with tree-based priors." In: *Advances in neural information processing systems*. 2013, pp. 2094–2102.

[67] James Steven Supancic and Deva Ramanan. "Self-Paced Learning for Long-Term Tracking." In: *IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 2379–2386.

[68] Will Y. Zou, Richard Socher, Daniel Cer, and Christopher D. Manning. "Bilingual Word Embeddings for Phrase-Based Machine Translation." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2013, pp. 1393–1398. URL: http://aclweb.org/anthology/D13-1141.

[69] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. "Qualitatively characterizing neural network optimization problems." In: *International Conference on Learning Representations*. 2014.

[70] Lu Jiang, Deyu Meng, Teruko Mitamura, and Alexander G Hauptmann. "Easy Samples First: Self-paced Reranking for Zero-Example Multimedia Search." In: *Proceedings of the 22nd ACM International Conference on Multimedia*. 2014, pp. 547–556.

[71] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. "Self-paced learning with diversity." In: *Advances in Neural Information Processing Systems*. 2014, pp. 2078–2086.

[72] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft COCO: Common objects in context." In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[73] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks." In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[74] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. "VQA: Visual Question Answering." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2425–2433.

[75] Vanya Avramova. "Curriculum Learning with Deep Convolutional Neural Networks." In: *Master's Thesis in Computer Science*. 2015.

[76] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *International Conference on Learning Representations*. 2015. URL: https://arxiv.org/pdf/1409.0473.

[77] Carl Doersch, Abhinav Gupta, and Alexei A Efros. "Unsupervised visual representation learning by context prediction." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1422–1430.

[78] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[79] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. "Self-Paced Curriculum Learning." In: *AAAI Conference on Artificial Intelligence*. 2015.

[80] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *Proceedings of the 3rd International Conference for Learning Representations*. ACM. 2015.

[81] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[82] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. *Tiny ImageNet Visual Recognition Challenge*. https://tiny-imagenet.herokuapp.com/. 2015.

[83] Z. Liu, Ping Luo, Xiaogang Wang, and X. Tang. "Deep Learning Face Attributes in the Wild." In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 3730–3738.

[84] Igor Mazić, Mirjana Bonković, and Barbara Džaja. "Two-level coarse-to-fine classification algorithm for asthma wheezing recognition in children's respiratory sounds." In: *Biomedical Signal Processing and Control* 21 (2015), pp. 105–118.

[85] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H Lampert. "Curriculum Learning of Multiple Tasks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 5492–5500.

[86] Harish Ramaswamy, Ambuj Tewari, and Shivani Agarwal. "Convex calibrated surrogates for hierarchical classification." In: *International Conference on Machine Learning*. 2015, pp. 1852–1860.

[87] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[88] Chang Xu, Dacheng Tao, and Chao Xu. "Multi-view Self-Paced Learning for Clustering." In: *IJCAI*. 2015.

[89] Dingwen Zhang, Deyu Meng, Chao Li, Lu Jiang, Qian Zhao, and Junwei Han. "A Self-Paced Multiple-Instance Learning Framework for Co-Saliency Detection." In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 594–602.

[90] Qian Zhao, Deyu Meng, Lu Jiang, Qi Xie, Zongben Xu, and Alexander G. Hauptmann. "Self-Paced Learning for Matrix Factorization." In: *AAAI*. 2015.

[91] Xiaojin Zhu. "Machine Teaching: An Inverse Problem to Machine Learning and an Approach Toward Optimal Education." In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[92] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: a system for large-scale machine learning." In: *Arxiv e-prints* (Mar. 2016).

[93] Volkan Cirik, Eduard H. Hovy, and Louis-Philippe Morency. "Visualizing and Understanding Curriculum Learning for Long Short-Term Memory Networks." In: *ArXiv* abs/1611.06204 (2016).

[94] Josep Maria Crego, Jungi Kim, Guillaume Klein, Anabel Rebollo, Kathy Yang, Jean Senellart, Egor Akhanov, Patrice Brunelle, Aurelien Coquard, Yongchao Deng, Satoshi Enoue, Chiyo Geiss, Joshua Johanson, Ardas Khalsa, Raoum Khiari, Byeongil Ko, Catherine Kobus, Jean Lorieux, Leidiana Martins, Dang-Chuan Nguyen, Alexandra

Priori, Thomas Riccardi, Natalia Segal, Christophe Servan, Cyril Tiquet, Bo Wang, Jin Yang, Dakun Zhang, Jing Zhou, and Peter Zoldan. "SYSTRAN's Pure Neural Machine Translation Systems." In: *CoRR* abs/1610.05540 (2016). URL: https://arxiv.org/abs/1610.05540.

[95] Tieliang Gong, Qian Zhao, Deyu Meng, and Zongben Xu. "Why curriculum learning & self-paced learning work in big/noisy data: A theoretical perspective." In: *Big Data & Information Analytics* 1.1 (2016), p. 111.

[96] Elad Hazan, Kfir Yehuda Levy, and Shai Shalev-Shwartz. "On graduated optimization for stochastic non-convex problems." In: *International conference on machine learning*. PMLR. 2016, pp. 1833–1841.

[97] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[98] Daniel Jiwoong Im, Michael Tao, and Kristin Branson. "An empirical analysis of the optimization of deep network loss surfaces." In: *arXiv preprint arXiv:1612.04010* (2016).

[99] Radu Tudor Ionescu, Bogdan Alexe, Marius Leordeanu, Marius Popescu, Dim P Papadopoulos, and Vittorio Ferrari. "How hard can it be? Estimating the difficulty of visual search in an image." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2157–2166.

[100] Dangwei Li, Z. Zhang, Xiaotang Chen, Haibin Ling, and K. Huang. "A Richly Annotated Dataset for Pedestrian Attribute Recognition." In: *ArXiv* abs/1603.07054 (2016).

[101] Hao Li, Maoguo Gong, Deyu Meng, and Qiguang Miao. "Multi-objective self-paced learning." In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[102] Ishan Misra, Abhinav Shrivastava, A. Gupta, and M. Hebert. "Cross-Stitch Networks for Multi-task Learning." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 3994–4003.

[103] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. "Squad: 100,000+ questions for machine comprehension of text." In: *arXiv preprint arXiv:1606.05250* (2016).

[104] Mallinali Ramírez-Corona, L Enrique Sucar, and Eduardo F Morales. "Hierarchical multilabel classification based on path evaluation." In: *International Journal of Approximate Reasoning* 68 (2016), pp. 179–193.

[105] Mrinmaya Sachan and Eric Xing. "Easy questions first? a case study on curriculum learning for question answering." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016, pp. 453–463.

[106] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. 2016, pp. 1715–1725. URL: http://www.aclweb.org/anthology/P16-1162.

[107] J. Weston, Antoine Bordes, S. Chopra, and Tomas Mikolov. "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks." In: *arXiv: Artificial Intelligence* (2016).

[108] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." In: *CoRR* abs/1609.08144 (2016). URL: https://arxiv.org/abs/1609.08144.

[109] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks." In: *The British Machine Vision Conference*. 2016.

[110] Dakun Zhang, Jungi Kim, Josep Crego, and Jean Senellart. "Boosting neural machine translation." In: *arXiv preprint arXiv:1612.06138* (2016).

[111] Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. "Yin and Yang: Balancing and Answering Binary Visual Questions." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[112] Mohammadhossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. "Affinity Clustering: Hierarchical Clustering at Scale." In: *Advances in Neural Information Processing Systems*. 2017, pp. 6864–6874.

[113] Alsallakh Bilal, Amin Jourabloo, Mao Ye, Xiaoming Liu, and Liu Ren. "Do convolutional neural networks learn class hierarchy?" In: *IEEE transactions on visualization and computer graphics* 24.1 (2017), pp. 152–162.

[114] Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, et al. "Findings of the 2017 Conference on Machine Translation (WMT17)." In: *Proceedings of the Second Conference on Machine Translation*. 2017, pp. 169–214.

[115] Ondřej Bojar, Jindřich Helcl, Tom Kocmi, Jindřich Libovický, and Tomáš Musil. "Results of the WMT17 Neural MT Training Task." In: *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*. Association for Computational Linguistics, 2017, pp. 525–533. URL: http://www.aclweb.org/anthology/W17-4757.

[116] S. Braun, D. Neil, and S. Liu. "A curriculum learning method for improved noise robustness in automatic speech recognition." In: *2017 25th European Signal Processing Conference (EUSIPCO)*. 2017, pp. 548–552.

[117] Yanbo Fan, Ran He, Jian Liang, and Baogang Hu. "Self-paced learning: An implicit regularization perspective." In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[118] Carlos Florensa, David Held, Markus Wulfmeier, and Pieter Abbeel. "Reverse Curriculum Generation for Reinforcement Learning." In: *The 1st Conference on Robot Learning*. 2017.

[119] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. "Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[120] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. "Automated Curriculum Learning for Neural Networks." In: *International Conference on Machine Learning*. 2017.

[121] Sanggyu Han and Sung-Hyon Myaeng. "Tree-structured Curriculum Learning based on Semantic Similarity of Text." In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2017, pp. 971–976.

[122] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. "OpenNMT: Open-Source Toolkit for Neural Machine Translation." In: *Proceedings of ACL 2017, System Demonstrations*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 67–72. URL: https://www.aclweb.org/anthology/P17-4012.

[123] Tom Kocmi and Ondřej Bojar. "Curriculum Learning and Minibatch Bucketing in Neural Machine Translation." In: *Proceedings of the International Conference Recent Advances in Natural Language Processing*. 2017, pp. 379–386. DOI: 10.26615/978-954-452-049-6_050. URL: https://doi.org/10.26615/978-954-452-049-6_050.

[124] Alexander Kuhnle and Ann Copestake. "Shapeworld-a new test methodology for multimodal language understanding." In: *arXiv preprint arXiv:1704.04517* (2017).

[125] Changsheng Li, Junchi Yan, Fan Wei, Weishan Dong, Qingshan Liu, and Hongyuan Zha. "Self-paced multi-task learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[126] Siyang Li, Xiangxin Zhu, Qin Huang, Hao Xu, and C-C Jay Kuo. "Multiple instance curriculum learning for weakly supervised object detection." In: *British Machine Vision Conference*. 2017.

[127] Weiyang Liu, Bo Dai, Ahmad Humayun, Charlene Tay, Chen Yu, Linda B Smith, James M Rehg, and Le Song. "Iterative machine teaching." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2149–2158.

[128] Fan Ma, Deyu Meng, Qi Xie, Zina Li, and Xuanyi Dong. "Self-paced co-training." In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. JMLR.org. 2017, pp. 2275–2284.

[129] Melike Nur Mermer and Mehmet Fatih Amasyali. "Scalable Curriculum Learning for Artificial Neural Networks." In: *IPSI BGD Transactions On Internet Research* 13.2 (2017).

[130] Pietro Morerio, Jacopo Cavazza, Riccardo Volpi, René Vidal, and Vittorio Murino. "Curriculum dropout." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3544–3552.

[131] Keerthiram Murugesan and Jaime Carbonell. "Self-Paced Multitask Learning with Shared Knowledge." In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. 2017.

[132] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. "Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning." In: *IJCAI*. 2017.

[133] Hao Peng, Sam Thomson, and Noah A. Smith. "Deep Multitask Learning for Semantic Dependency Parsing." In: *ACL*. 2017.

[134] Sebastian Ruder. "An overview of multi-task learning in deep neural networks." In: *arXiv preprint arXiv:1706.05098* (2017).

[135] Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. "Automatic Curriculum Graph Generation for Reinforcement Learning Agents." In: *AAAI*. 2017.

[136] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is All you Need." In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008. URL: `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

[137] Rachel Bawden, Rico Sennrich, Alexandra Birch, and Barry Haddow. "Evaluating Discourse Phenomena in Neural Machine Translation." In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1304–1313. URL: http://www.aclweb.org/anthology/N18-1118.

[138] Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. "The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*. Association for Computational Linguistics, 2018, pp. 76–86. URL: http://aclweb.org/anthology/P18-1008.

[139] Mark Collier and Joeran Beel. "An Empirical Comparison of Syllabuses for Curriculum Learning." In: *Proceedings of the 26th Irish Conference on Artificial Intelligence and Cognitive Science*. 2018.

[140] Li Dong and Mirella Lapata. "Coarse-to-Fine Decoding for Neural Semantic Parsing." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 731–742. DOI: 10.18653/v1/P18-1068. URL: https://www.aclweb.org/anthology/P18-1068.

[141] Yang Fan, Fei Tian, Tao Qin, Xiuping Li, and Tie-Yan Liu. "Learning to Teach." In: *International Conference on Learning Representations*. 2018.

[142] Pierre Fournier, Olivier Sigaud, Mohamed Chetouani, and Pierre-Yves Oudeyer. "Accuracy-based Curriculum Learning in Deep Reinforcement Learning." In: *ArXiv* abs/1806.09614 (2018).

[143] Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. "Dynamic task prioritization for multitask learning." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 270–287.

[144] Sheng Guo, Weilin Huang, Haozhi Zhang, Chenfan Zhuang, Dengke Dong, Matthew R Scott, and Dinglong Huang. "Curriculumnet: Weakly supervised learning from large-scale web images." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 135–150.

[145] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. "MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels." In: *International Conference on Machine Learning*. 2018.

[146] Alex Kendall, Y. Gal, and R. Cipolla. "Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 7482–7491.

[147] Kenton Lee, Luheng He, and Luke Zettlemoyer. "Higher-Order Coreference Resolution with Coarse-to-Fine Inference." In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 687–692. DOI: 10.18653/v1/N18-2108. URL: https://www.aclweb.org/anthology/N18-2108.

[148] Hao Li, Zheng Xu, G. Taylor, and T. Goldstein. "Visualizing the Loss Landscape of Neural Nets." In: *Advances in Neural Information Processing Systems*. 2018.

[149] Frederick Liu, Han Lu, and Graham Neubig. "Handling Homographs in Neural Machine Translation." In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1336–1345. URL: http://www.aclweb.org/anthology/N18-1121.

[150] Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. "Contextual Parameter Generation for Universal Neural Machine Translation." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2018. URL: https://arxiv.org/abs/1808.08493.

[151] Martin Popel and Ondřej Bojar. "Training Tips for the Transformer Model." In: *The Prague Bulletin of Mathematical Linguistics* 110.1 (2018), pp. 43–70. URL: https://content.sciendo.com/view/journals/pralin/110/1/article-p43.xml.

[152] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. "On the Convergence of Adam and Beyond." In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=ryQu7f-RZ.

[153] Xutong Ren, Lingxi Xie, Chen Wei, Siyuan Qiao, Chi Su, Jiaying Liu, Qi Tian, Elliot K Fishman, and Alan L Yuille. "Generalized Coarse-to-Fine Visual Recognition with Progressive Training." In: *arXiv preprint arXiv:1811.12047* (2018).

[154] Nikolaos Sarafianos, Theodoros Giannakopoulos, Christophoros Nikou, and Ioannis A Kakadiaris. "Curriculum Learning of Visual Attribute Clusters for Multi-task Classification." In: *Pattern Recognition* 80 (2018), pp. 94–108.

[155] Noam Shazeer and Mitchell Stern. "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, July 2018, pp. 4596–4604. URL: http://proceedings.mlr.press/v80/shazeer18a.html.

[156] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. "Intrinsic motivation and automatic curricula via asymmetric self-play." In: *ICLR*. 2018.

[157] Yuxing Tang, Xiaosong Wang, Adam P Harrison, Le Lu, Jing Xiao, and Ronald M Summers. "Attention-guided curriculum learning for weakly supervised classification and localization of thoracic diseases on chest radiographs." In: *International Workshop on Machine Learning in Medical Imaging*. Springer. 2018, pp. 249–258.

[158] Andrew Trask, Felix Hill, Scott E. Reed, Jack W. Rae, Chris Dyer, and P. Blunsom. "Neural Arithmetic Logic Units." In: *NeurIPS*. 2018.

[159] Cheng Wang, Qian Zhang, Chang Huang, Wenyu Liu, and Xinggang Wang. "Mancs: A Multi-Task Attentional Network with Curriculum Sampling for Person Re-Identification." In: *European Conference on Computer Vision*. 2018, pp. 365–381.

[160] Jonatas Wehrmann, Ricardo Cerri, and Rodrigo Barros. "Hierarchical multi-label classification networks." In: *International Conference on Machine Learning*. 2018, pp. 5075–5084.

[161] Daphna Weinshall, Gad Cohen, and Dan Amir. "Curriculum learning by transfer learning: Theory and experiments with deep networks." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 5238–5246.

[162] Xuan Zhang, Gaurav Kumar, Huda Khayrallah, Kenton Murray, Jeremy Gwinnup, Marianna J Martindale, Paul McNamee, Kevin Duh, and Marine Carpuat. "An Empirical Exploration of Curriculum Learning for Neural Machine Translation." In: *CoRR* abs/1811.00739 (2018). URL: https://arxiv.org/abs/1811.00739.

[163] Tianyi Zhou and Jeff A Bilmes. "Minimax Curriculum Learning: Machine Teaching with Desirable Difficulties and Scheduled Diversity." In: *International Conference on Learning Representations*. 2018.

[164] Xiaojin Zhu, Adish Singla, Sandra Zilles, and Anna N. Rafferty. "An Overview of Machine Teaching." In: *ArXiv* abs/1801.05927 (2018). A tutorial document grown out of NeurIPS 2017 Workshop on Teaching Machines, Robots, and Humans.

[165] Antoine Caubrière, N. Tomashenko, Antoine Laurent, E. Morin, Nathalie Camelin, and Y. Estève. "Curriculum-based transfer learning for an effective end-to-end spoken language understanding and domain portability." In: *INTERSPEECH*. 2019.

[166] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *NAACL-HLT*. 2019.

[167] Elozino Egonmwan and Yllias Chali. "Transformer-based model for single documents neural summarization." In: *Proceedings of the 3rd Workshop on Neural Generation and Translation*. 2019, pp. 70–79.

[168] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. "ImageNet-Trained CNNs are Biased Towards Texture; Increasing Shape Bias Improves Accuracy and Robustness." In: *International Conference on Learning Representations*. 2019.

[169] Guy Hacohen and Daphna Weinshall. "On the power of curriculum learning in training deep networks." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2535–2544.

[170] MD Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. "A comprehensive survey of deep learning for image captioning." In: *ACM Computing Surveys (CsUR)* 51.6 (2019), pp. 1–36.

[171] Reza Lotfian and Carlos Busso. "Curriculum learning for speech emotion recognition from crowdsourced labels." In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.4 (2019), pp. 815–826.

[172] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. "Teacher-Student Curriculum Learning." In: *IEEE transactions on neural networks and learning systems* (2019).

[173] Ilkay Oksuz, Bram Ruijsink, Esther Puyol-Antón, James R Clough, Gastao Cruz, Aurelien Bustin, Claudia Prieto, Rene Botnar, Daniel Rueckert, Julia A Schnabel, et al. "Automatic CNN-based detection of cardiac MR motion artefacts using k-space data augmentation and curriculum learning." In: *Medical image analysis* 55 (2019), pp. 136–147.

[174] Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M Mitchell. "Competence-based Curriculum Learning for Neural Machine Translation." In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019.

[175] Enver Sangineto, Moin Nabi, Dubravko Culibrk, and Nicu Sebe. "Self Paced Deep Learning for Weakly Supervised Object Detection." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41 (2019), pp. 712–725.

[176] Shreyas Saxena, Oncel Tuzel, and Dennis DeCoste. "Data parameters: A new family of parameters for learning a differentiable curriculum." In: *Advances in Neural Information Processing Systems*. 2019, pp. 11095–11105.

[177] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. "Analysing mathematical reasoning abilities of neural models." In: *International Conference on Learning Representations*. 2019.

[178] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Well-Read Students Learn Better: On the Importance of Pre-training Compact Models." In: *arXiv preprint arXiv:1908.08962v2* (2019).

[179] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding." In: In the Proceedings of the International Conference on Learning Representations (ICLR). 2019.

[180] Yiru Wang, Weihao Gan, Wei Wu, and J. Yan. "Dynamic Curriculum Learning for Imbalanced Data Classification." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 5016–5025.

[181] Yujia Wang, Wei Liang, Jianbing Shen, Yunde Jia, and Lap-Fai Yu. "A deep Coarse-to-Fine network for head pose estimation from synthetic data." In: *Pattern Recognition* 94 (2019), pp. 196–206.

[182] Zuxuan Wu, Caiming Xiong, Yu-Gang Jiang, and Larry S Davis. "LiteEval: A Coarse-to-Fine Framework for Resource Efficient Video Recognition." In: *Advances in Neural Information Processing Systems*. 2019, pp. 7778–7787.

[183] Changdong Xu and Xin Geng. "Hierarchical classification based on label distribution learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 5533–5540.

[184] Jian Yao and Ahmad Al-Dahle. "Coarse-to-fine optimization for speech enhancement." In: *arXiv preprint arXiv:1908.08044* (2019).

[185] Xuan Zhang, Pamela Shapiro, Gaurav Kumar, Paul McNamee, Marine Carpuat, and Kevin Duh. "Curriculum learning for domain adaptation in neural machine translation." In: *NAACL-HLT*. 2019.

[186] Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. "Better Fine-Tuning by Reducing Representational Collapse." In: *ArXiv* abs/2008.03156 (2020).

[187] Ürün Dogan, Aniket Anand Deshmukh, Marcin Bronislaw Machura, and Christian Igel. "Label-similarity curriculum learning." In: *European Conference on Computer Vision*. Springer. 2020, pp. 174–190.

[188] Madan Ravi Ganesh and Jason J Corso. "Rethinking Curriculum Learning with Incremental Labels and Adaptive Compensation." In: *arXiv preprint arXiv:2001.04529* (2020).

[189] Ziheng Jiang, Chiyuan Zhang, Kunal Talwar, and Michael C Mozer. "Exploring the memorization-generalization continuum in deep learning." In: *arXiv preprint arXiv:2002.03206* (2020).

[190] Qing Li, Siyuan Huang, Yining Hong, and Song-Chun Zhu. "A Competence-aware Curriculum for Visual Concepts Learning via Question Answering." In: *European Conference on Computer Vision*. Springer. 2020, pp. 141–157.

[191] Andreas Madsen and alexander rosenberg johansen. "Neural Arithmetic Units." In: *International Conference on Learning Representations*. 2020.

[192] Jonathan Pilault, Raymond Li, Sandeep Subramanian, and Christopher Pal. "On Extractive and Abstractive Neural Document Summarization with Transformer Language Models." In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020, pp. 9308–9319.

[193] Samarth Sinha, Animesh Garg, and Hugo Larochelle. "Curriculum By Texture." In: *arXiv preprint arXiv:2003.01367* (2020).

[194] Otilia Stretcu, Emmanouil Antonios Platanios, Tom M. Mitchell, and Barnabás Póczos. "Coarse-to-Fine Curriculum Learning for Classification." In: *International Conference on Learning Representations (ICLR) Workshop on Bridging AI and Cognitive Science (BAICS)*. 2020.

[195] Lilian Weng. *Curriculum for Reinforcement Learning*. Jan. 2020. URL: https://lilianweng.github.io/lil-log/2020/01/29/curriculum-for-reinforcement-learning.html.

[196] Benfeng Xu, Licheng Zhang, Zhendong Mao, Quan Wang, Hongtao Xie, and Yongdong Zhang. "Curriculum learning for natural language understanding." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 6095–6104.

[197] Han Zhao, Otilia Stretcu, Alexander J Smola, and Geoffrey J Gordon. "Efficient multitask feature and relationship learning." In: *Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 777–787.

[198] Otilia Stretcu, Emmanouil Antonios Platanios, Tom M. Mitchell, and Barnabás Póczos. *Coarse-to-Fine Curriculum Learning*. 2021. arXiv: 2106.04072 [cs.AI].

[199] Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E. Gonzalez. "NBDT: Neural-Backed Decision Trees." In: *International Conference on Learning Representations*. 2021.

[200] X. Wu, Ethan Dyer, and Behnam Neyshabur. "When Do Curricula Work?" In: *International Conference on Learning Representations*. 2021.

COLOPHON

This document was typeset using a modified version of the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*." `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Final Version as of June 21, 2021 (Version).