Eindhoven University of Technology

MASTER

# An Empirical Study on Dynamic Curriculum Learning in Information Retrieval

Qian, Chen

*Award date:*
2021

Link to publication

**Disclaimer**
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

**General rights**
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

# An Empirical Study on Dynamic Curriculum Learning in Information Retrieval

*Master Thesis*

Chen Qian
c.qian@student.tue.nl
1443348
Supervisor: Dr. Meng Fang

Graduation Committee:
Dr. Meng Fang
Dr. rer. nat. Morteza Monemizadeh
Dr. Yulong Pei

Final Report

Eindhoven, September 2021

# Contents

# Abstract

Most humans need to spend nearly twenty years on well-organized training before fully functioning in human society. For many people, well-organized training means following a school curriculum, which prepares the knowledge in a meaningful order, usually from easier concepts to more complex concepts. Providing the previously learned knowledge can accelerate the learning speed of new ones. Resembling the human learning process, curriculum learning (CL) has been successfully applied in many machine learning fields. Given that lack of training data has become the bottleneck of many research questions. Curriculum learning is a promising research direction to tackle this problem, as it can improve model performance without extra computational cost and requirements for additional data. Also, there remains a research gap of CL in Information retrieval (IR). We contribute to this research gap by doing empirical explorations with the state-of-the-art language model BERT in one of the complex IR tasks, conversation response ranking (CRR). The existing CL frameworks contain two main steps, first the difficulty criterion to arrange the data from easy to difficult, and second the speed criterion to control the pacing of transferring from easy to difficult data. However, the difficulty criterion or the speed criterion usually remains unchanged and static during the entire training process. For a more comprehensive exploration, we propose the dynamic rescoring method for the first step, and the noise method for the second step. The experimental results show: 1) the dynamic rescoring method has no exciting improvement over the baseline CL methods, whereas the noise method can slightly outperform the baselines with a non-excessive noise ratio; 2) the comparisons of different settings inform us that a good difficulty criterion and a proper speed criterion are essential for CL to be effective.

**Key words**: curriculum learning, information retrieval, conversation response ranking, BERT

# Chapter 1

# Introduction and Background

## 1.1 Introduction

Curriculum learning (CL) is a training strategy in machine learning. CL is inspired by the human learning process at school. The school curriculum is well-organized training, usually preparing the knowledge from easier concepts to more complex ones. Providing the previously learned knowledge can accelerate the learning speed of the new knowledge. Humans also learn much faster and better with the knowledge presented in a meaningful order instead of randomly presented. The earliest studies of applying this human-like learning process to machines started at the intersection of cognitive science and machine learning (Elman et al.[13]; Rohde & Plaut et al.[42]; Krueger & Dayan et al.[25]). Slightly later, Bengio et al.[3] formally proposed the vanilla CL in 2009 by designing CL training on three different tasks, including binary classification, shape recognition, and language modeling. They hypothesized that a well-designed curriculum works as a continuation method that helps find a better local minimum of a non-convex training criterion, and a regularizer that regularizes the process leading to a lower generalization error. Their success has led to a flourish of researches in CL with many exciting achievements in multifarious fields, including computer vision [6][20] and natural language processing [24][5][55].

Compared with random sampling, CL has been proved to accelerate the convergence and enhance model performance without extra computational costs and requirements for additional training data. CL can be a promising direction to tackle the problem that lack of data has become the bottleneck for many complex tasks with deep neural models. The idea of vanilla CL is to design a training curriculum by first sorting training data from easy to difficult and then training the model with easy samples in the early stages and gradually including in more difficult data later. There are mainly two challenges in making CL effective: **1) how to judge the difficulty of data with a good difficulty criterion; 2) how to determine the speed of transferring from easy data to difficult data according to a proper speed criterion.**

In this work, we aim to contribute to tackling the challenges by conducting empirical explorations of CL in information retrieval (IR), as there exists a research gap of applying CL in IR. We choose conversation response ranking (CRR) as the IR task, because CRR task is challenging and complex with multi-turn conversations and responses. We choose the pre-trained deep language model BERT [12] as the neural ranking model, because BERT has been proved to perform outstandingly on multiple retrieval and ranking tasks [10][7][32].

We base our work on the CL framework from Penha et al.[35], where they have experimented with several static difficulty criteria and speed criteria. However, exploration of any dynamic strategy is absent in their work. Thus in response to the two CL challenges and the absence of dynamic strategy:

- 1) For the difficulty criterion step, we design the **dynamic rescoring method** to explore if re-sorting the data several times according to the current model can help improve the performance. The motivation of the dynamic rescoring method is that while the model is learning, its evaluation of the difficulty is also changing.

- 2) For the speed criterion, in addition to determining the speed by a pacing function, we combine the idea from the work of Fang et al.[14] and apply the **noise method**. In brief, their vision is to allow the model to have more explorations in the early learning stages and gradually back to its learning goal. Likewise, in our work, by adding noise to the CL-scheduled data with a shrinking amount automatically adapted with the number of iterations, we explore if more flexibility can be beneficial.

## 1.2 Background

In this section, we introduce the basic knowledge of our research questions, including the background of information retrieval (IR), which is a sub-field of natural language processing (NLP), the model architecture and attention mechanism of Transformer, as well as the development and NLP related applications of curriculum learning.

### 1.2.1 Natural Language Processing

Natural language processing (NLP) is the intersection of linguistics and computer science (including artificial intelligence). The goal of NLP is to train the machines to understand human language and be capable of processing and analyzing massive natural language data precisely and efficiently. In NLP, popular challenging tasks involve machine translation, speech processing, natural language understanding, and natural language generation. In the research history of NLP, it has witnessed mainly three stages [51]: the first one is classical symbolic NLP from the 1950s to the early 1990s where complicated linguistic rules were added into the NLP system; the second one is empirical and statistical NLP from the 1990s to 2010s where machine learning algorithms were implemented into the system thanks to the steady growth of computational power and the gradual diminishing of the dominance of Noam Chomsky's linguistics theories; finally the Neural NLP, as the one being dominant currently, relies heavily on deep neural architectures due to the widespread of representation learning and state-of-art performance of deep neural networks in recent years.

#### Non-neural Methods

We merge the first and second stages into the category non-neural methods and briefly introduce this subsection. In the period of classical symbolic NLP, many systems were based on complex sets of hand-written rules. Traditionally [19], the entire process is decomposed into several stages, as somehow listed in a granularity order, are tokenization, lexical analysis, syntactic analysis, semantic analysis, and pragmatic analysis. Tokenization and lexical analysis are at the level of the word, while the rest are usually at the level of sentence, paragraph, or document. Tokenization is to split a sentence into words, which is not a problem for some languages (e.g., English words can be easily separated because it is space-delimited), whereas it is essential to separate words correctly for some other languages (e.g., Chinese, Japanese, and Thai). Lexical analysis is also to perform text analysis at the level of the word, of which a basic task is to relate the morphological variants of one word to its original version in the dictionary (e.g., relate does, did, done to do). For syntactic analysis, the task is to analyze the sentence according to its grammatical structure. Semantic analysis extracts the semantic or literal meaning from the sentence, and it sometimes works also at the level of the word, whereas pragmatic analysis focuses on the task to determine the meaning of the utterance or text at the paragraph or document level. Although it can be hard to split the task into these stages in real-life NLP applications, such a separation makes software

engineering tasks more manageable.

In the time of empirical and statistical NLP, many methods were adopted from machine learning algorithms, including supervised (e.g., naive Bayes, support vector machine, and logistic regression), unsupervised (e.g., mixture models, and expectation-maximization models), and methods used in sequence analysis (e.g., hidden Markov chain, conditional random field). With the increase in computational power, machine learning methods achieved exciting breakthroughs in many NLP tasks, where those methods outperformed the classical symbolic methods and dominated in the fields of NLP. Although many researchers at that time were fascinated by the incredible power of machine learning methods, some still saw the prospective future of incorporating linguistics rules into statistical methods.

### Neural Methods

The neural methods are based on the deep neural architectures, and thus in this subsection, we give an intuition of the neural methods by illustrating a fundamental and general neural network [11] with one hidden layer (it is not a deep network, but it can be expanded easily with more hidden layers). This network, shown in figure 1.1, is a simple fully-connected feed-forward network with an input layer ($x_i$, $i \in 1, 2, \ldots, n_0$), one hidden layer ($h_i$, $i \in 1, 2, \ldots, n_1$) and an output layer ($y_i$, $i \in 1, 2, \ldots, n_2$). It is worth noting that a feed-forward network is a multi-layer network whose units connect with no circles. Moreover, fully connected means each unit in each layer takes all the units from the previous layer as input and has connected with all of them.

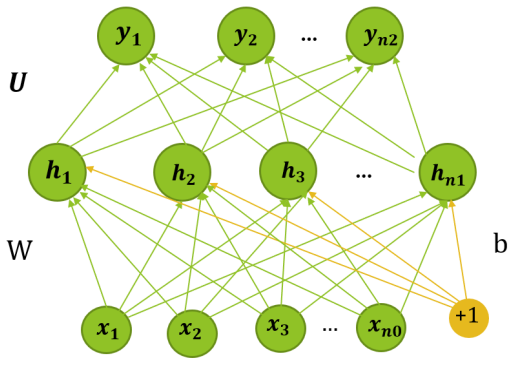


Figure 1.1: A basic example of neural network.

Take a neural unit of the hidden layer as an example (a blue circle with symbol $h_i$ in figure 1.1), the unit takes a weighted sum of its input with an additional term called bias. As shown in the equation 1.1, $b$ denotes the bias term, $w$ denotes a vector of weights.

$$h_i = b + \sum_i w_i x_i \tag{1.1}$$

And then an activation function ($f$) is applied to $h_i$ in equation 1.2, usually $f$ could be **sigmoid** function, **ReLU** or **tanh** function.

$$y = f(h_i) = f(b + \sum_i w_i x_i) \tag{1.2}$$

Then, by expanding the above equations on a single unit to the entire 3-layer network, we can have the following equations 1.3 [11]. $W$ is a matrix consists of the weight vectors $w$ of every unit in the hidden layer, $b$ denotes a vector of bias items, $U$ is a similar weight matrix of the output layer, $\sigma$ denotes the **sigmoid** function as an activation function, and finally, a **softmax** function is applied as the final output should be a probability between 0 and 1.

$$
\begin{aligned}
h &= \sigma(Wx + b) \\
z &= Uh \\
y &= \text{softmax}\,(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} 1 \leq i \leq d
\end{aligned}
\tag{1.3}
$$

**Information Retrieval**

According to [29], an academic definition of information retrieval (IR) can be: IR is usually to find documents of an unstructured nature (e.g., text) from massive collections stored on computers to answer an information query. Following the above definition, when the query and documents are in natural language (without an easy-for-a-computer structure), IR can be seen as a subfield of NLP. Distinguished by the operating scale, IR systems can be categorized into three categories: web search, enterprise/institutional/domain-specific search, and personal search, from big scale to small scale. In web search, the IR system needs to search over billions of documents within a satisfying short responding time. Thus many issues need to be tackled, including good indexing, building an efficient system to work with enormous scale, and against the tricks played by site providers to boost the flow to their pages. The task conversational response ranking (CRR), which we are interested in, is covered by web search. In brief, CRR ranks the retrieved documents (responses) according to their relevance to the query (conversation) by analyzing the content.

### 1.2.2 Neural Language Model

A language model [2] is a function or an algorithm for gaining such a function that can capture the probability distributions of natural language sequences of words. Usually, a language model can predict the next word when the previous words are given. Specifically, a neural language model is based on neural networks, and it alleviates the curse of dimensionality by learning distributed representations. Generally speaking, the curse of dimensionality means that with the number of input features increasing, the number of training data needed grows exponentially. Specifically, in language models, the curse of dimensionality comes from the explosion in the number of possible different word sequences. For example, with a 5000-word vocabulary, a 15-word sequence can have $5000^{15}$ variants. In the following subsections, we introduce one state-of-the-art language model of interest called Transformer [49].

**Architecture of Transformer**

Transformer was proposed by Vaswani et al.[49]. Complying with the paper name "Attention is all you need", the Transformer has a new architecture relying entirely on the attention mechanism. The architecture of the Transformer is shown in figure 1.2, which consists of two parts, the left half is the encoder, while the right half is the decoder. Both the encoder and decoder have a stack of $N = 6$ identical blocks as the corresponding one shown in figure 1.2. For the encoder, the block is composed of two sub-layers. The first one is a multi-head attention layer, and the second is a position-wide fully-connected feed-forward layer. While for the decoder, in addition to the same two sub-layers as the encoder, one extra multi-head attention sub-layer is added to attend the output of the encoder, also a masking strategy is applied to guarantee that the prediction at position $i$ only depends on the previous positions less than $i$. Besides, for each sub-layer, a residual connection followed by layer normalization (Add & Norm) is applied, which can be expressed as $LayerNorm(x + Sublayer(x))$. They also note that to facilitate these residual connections, the dimensions of the outputs of all the layers in the model is $d_{model} = 512$.

**Attention of Transformer**

Generally speaking, an attention function takes a query and a set of key-value pairs as input and produce output correspondingly. With the query, keys and values all being vectors, firstly dot

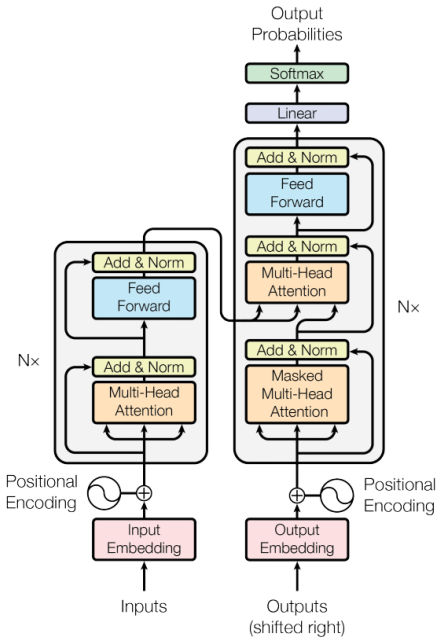


Figure 1.2: The architecture of Transformer [49].

products are taken between query and each key, then the dot products serve as the weights to compute a weighted sum of the values, which is the output. The attention mechanism used in Transformer can be seen in figure 1.3, including Scaled Dot-Product Attention and Multi-Head Attention. A mathematics expression of the Scaled Dot-Product Attention is shown in equation 1.4, a scale of $\sqrt{d_k}$ is added in case that a large value of $d_k$ can make the gradients of the softmax function extremely small. Whereas the Multi-Head Attention is mathematically shown in equation 1.5, where the projections are parameter matrices $W_i^Q \in R^{d_{\text{model}} \times d_k}, W_i^K \in R^{d_{\text{model}} \times d_k}, W_i^V \in R^{d_{\text{model}} \times d_v}$ and $W^O \in R^{hd_v \times d_{\text{model}}}$. Multi-Head Attention is proved to be beneficial and the authors assume that the model can jointly attend to different representation subspaces at various positions. In Transformer, $h = 8$ parallel attention heads are implemented with the dimension of each head reduced as $d_k = d_v = d_{\text{model}}/h = 64.$, so that multi-head strategy has a similar computational cost as the single-head attention with full dimensionality.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.4}$$

$$\begin{aligned} \text{MultiHead}(Q, K, V) = &\ \text{Concat}\ (\ \text{head}_1, \ldots,\ \text{head}_h)\, W^O \\ \text{where head}_i = &\ \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{1.5}$$

**Position-wise Feed-Forward Networks**

The feed-forward layer as mentioned in the architecture consists of two linear transformations with a ReLU activation in the middle, as shown in equation 1.6.

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right)W_2 + b_2 \tag{1.6}$$

As Transformer has no recurrent or convolutional layers, it is essential to infuse positional information into the model. That is the reason why "positional encoding" is added to the embedding. And the type of "positional encoding" in Transformer can be seen in equation 1.7, where *pos* is the position, and $i$ is the dimension.

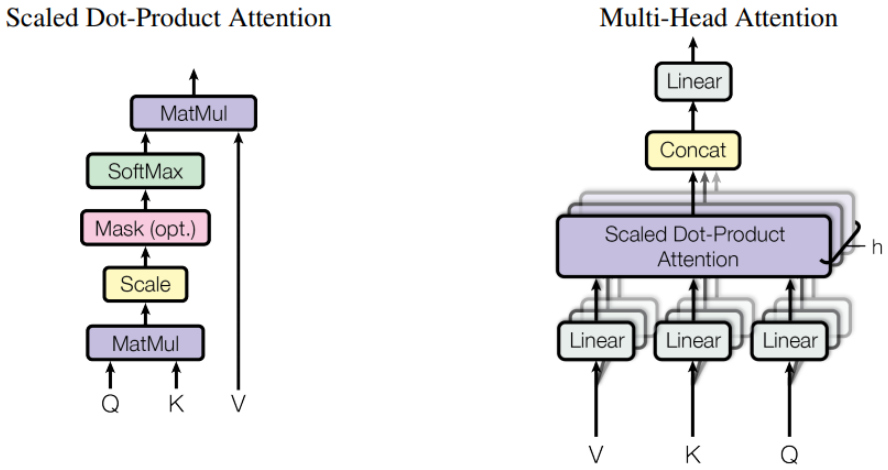


Figure 1.3: The attention mechanism of Transformer [49].

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \end{aligned} \tag{1.7}$$

### 1.2.3 Non-convex Optimization

A generic definition of optimization problem is shown in equation 1.8 [21], where x is the variable of the problem, function $f : R^p \to R$ is the objective function, and $C \subseteq R^p$ is the constraint set of the problem.

$$\begin{aligned} &\min_{x \in R^p} f(x) \\ &\text{s.t. } x \in C \end{aligned} \tag{1.8}$$

Concerning the optimization problem in machine learning, the objective function are usually encoded with some specific behavior wanted by the algorithm designer, for example, fitting the model well to the training data by minimizing a proper loss function. Moreover, the constraint set adds restrictions to the model, for instance, limiting the model size.

In order to be called a convex optimization problem, one must have both convex objective function and convex constraint set, otherwise it is a non-convex optimization problem. So first we will introduce some of the basic definitions of convex function, as well as the definition of convex set. For a continuously differentiable function $f : R^p \to R$ to be considered convex, it should satisfy that $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle$ for every $x, y \in R^p$, where $\nabla f(x)$ is the gradient of $f$ at $x$. Also, there is a simpler definition without considering notions of derivatives, which defines convex functions $f : R^p \to R$ as following $f((1-\lambda) \cdot x + \lambda \cdot y) \leq (1-\lambda) \cdot f(x) + \lambda \cdot f(y)$ for every $x, y \in R^p$ and every $\lambda \in [0, 1]$. Whereas for a set $C \in R^p$ to be considered convex, it should follow that $(1-\lambda) \cdot x + \lambda \cdot y \in C$ for every $x, y \in C$ and $\lambda \in [0, 1]$. A visualization of several kinds of convex function is shown in figure 1.4, whereas a visual representation of convex and non-convex set is displayed in figure 1.5.

In real life, it is more common for people to encounter non-convex optimization problems, which makes it necessary to focus more on non-convex optimization problems. Especially in recent years, more and more applications that require learning algorithms to deal with high dimensional data have been emerging rapidly, such as large-scale web documents classification, recommendation systems with both millions of items and customers, and medical image analysis. The blooming of

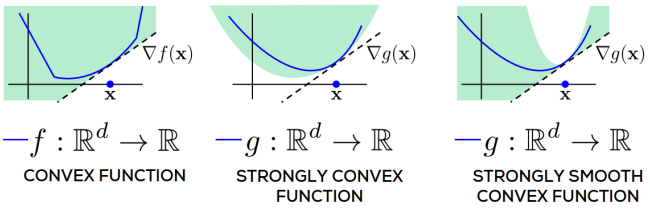


Figure 1.4: Several different types of convex functions [21].

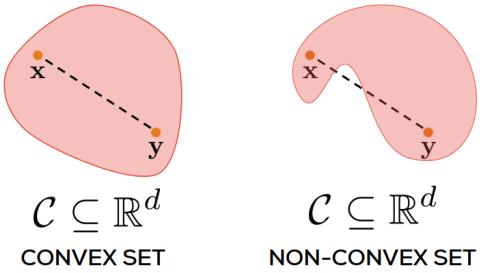


Figure 1.5: Some examples of convex and non-convex set [21].

such modern applications makes it necessary to facilitate the learning model with well-designed structural constraints from estimating the training data, and such structural constraints usually are in the non-convex set. On the other hand, deep neural architecture, as one of the most leading and popular models for tackling these applications, always leads to a non-convex objective function. The conversation response ranking task selected in this work is also a non-convex optimization problem. A visualization of two simple non-convex objective function examples are shown in figure 1.6.

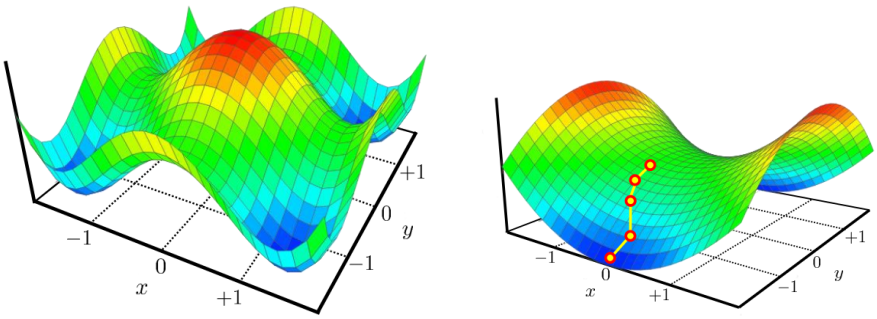

Figure 1.6: Examples of non-convex objective function surface: multiple local minima (left), saddle point (right) [21].

### 1.2.4 Continuation Method

Continuation method [1] is an optimization strategy that tackles minimizing non-convex criteria by first optimizing a smoothed (simplified) objective and gradually moving to a less smoothed objective. It is with the intuition that a smoothed, or in other words, a simplified version, can give an overview of the original problem. According to one definition by Bengio et al.[3], suppose $C_\lambda(\theta)$ is a single-parameter family of cost functions, where $C_0$ is a smoothed version that can

be optimized easily, and $C_1$ is the real target criterion. So one starts from optimizing $C_0$, then increases $\lambda$ slowly towards the target $C_1$. During this process, as $C_0$ is a simplified version of $C_1$, $\theta$ is likely to end in the basin of attraction of a dominant minimum of $C_1$, a local or sometimes global minimum.

### 1.2.5 Curriculum Learning

The majority of humans need to spend nearly twenty years on well-organized training before they are fully functional and ready to start their careers. Usually, for many people, well-organized training means following a school curriculum designed and adjusted constantly based on numerous educational researches. The school curriculum prepares the knowledge in a meaningful order, usually from easier concepts to more difficult concepts, providing that the previously learned knowledge can accelerate the learning speed of the new ones. Inspired by the human learning process, some researchers (Elman et al.[13]; Rohde & Plaut et al.[42]; Krueger & Dayan et al.[25]) at the intersection of cognitive science and machine learning started to wonder whether this curriculum learning strategy would be effective for machines.

To answer this question and explore when and why a curriculum can make machine learning more effective than random sampling, Bengio et al.[3] proposed vanilla curriculum learning in 2009. They hypothesized that a well-established curriculum could function as a continuation method. That is to say, at a high level, a curriculum can also be seen as a sequence of training criteria. In each sequence stage, the corresponding training criterion assigns a different set of weights to the training samples. Typically, the early stages favor easier examples, while the later stages pay more attention to more difficult examples. Finally, the re-weighting of the examples is uniform that equals training on the target training set or distribution. One can define curriculum learning as follows. Let $z$ be a random variable representing a training example for the machine, $P(z)$ is the target training distribution of the interest for the learner, $0 \leq w_\lambda(z) \leq 1$ is the weight assigned to the example $z$ at step $\lambda$ in the curriculum stages, with $0 \leq \lambda \leq 1$ (monotonically increasing from 0 to 1), and $W_1(z) = 1$. The corresponding training distribution at step $\lambda$ is

$$Q_\lambda(z) \propto W_\lambda(z)P(z) \quad \forall z \tag{1.9}$$

such that $\int Q_\lambda(z)dz = 1$, then it follows that

$$Q_1(z) = P(z) \quad \forall z \tag{1.10}$$

They also give a simple definition of curriculum as follows. For a distribution $Q_\lambda$ satisfying equation 1.9 and 1.10, if its entropy follows equation 1.11 and weight $W_{\lambda+\epsilon}(z)$ is monotonically increasing in $\lambda$ as shown in equation 1.12, then $Q_\lambda$ is called a curriculum. Intuitively speaking, increasing $\lambda$ means new samples (weight) are added so that training data's diversity (entropy) is also increasing. They want the training to start with a relatively small set of easy examples and end with the target training set. In a word, curriculum learning proposed by Bengio et al.[3] is an optimization method targets at benefiting the process of minimizing a non-convex criterion by organizing the training data in an easy-to-difficult order. However, "easy examples" or "difficult example" are sometimes hard to define and highly depends on the problem. Thus more exploration is needed in this direction.

$$H\left(Q_\lambda\right) < H\left(Q_{\lambda+\epsilon}\right) \quad \forall \epsilon > 0 \tag{1.11}$$

$$W_\lambda(z) \leq W_{\lambda+\epsilon}(z) \quad \forall z, \forall \epsilon > 0 \tag{1.12}$$

To explore the effectiveness of CL, Bengio et al.[3] have experimented on three different problems. The first one was a simple toy problem of binary classification, where they artificially generated training data to train a Perceptron and used two simple ways to define easiness. One defined the examples that were correctly classified by a linear SVM model as easy examples and the rest as

difficult examples, while the other introduced noisy examples whose inputs were irrelevant to the label as difficult examples. For the result that the curriculum was significantly better, they argued that although difficult examples are more informative, they can confuse the model compared with easier examples. The second problem was shape recognition (3-class multi-classification), where they generated easy examples with less variability in object shape, size, position, orientation, and greyscales of the foreground and background. They trained a neural network (3 hidden layers) with a two-stage curriculum training for this problem. At the first stage, the model was trained only on the easy examples, and then when it reached the switch epoch, the training switched to the stage with only difficult examples. For curriculum training, the testing error on the difficult examples was much lower. The third case is on language modeling, where they designed the curriculum training with four stages based on vocabulary size. Specifically, for the first stage, only text containing the 5000 most frequent words was used for training, then the vocabulary was enlarged by 5000 words per stage until it reached 20000 words. A significant improvement by curriculum training also applied to this problem.

Since then, curriculum learning has aroused massive research interest. Very recently, Soviany et al.[44] have written a survey to summarize these works. They formalized the curriculum learning from a great diversity of studies into a generic formulation, which is shown in Algorithm 1.

**Algorithm 1** General curriculum learning algorithm [44].

$M$ – *a machine learning model*;
$E$ – *a training data set*;
$P$ – $performance\ measure$;
$n$ – $number\ of\ iterations\ or\ epochs$;
$C$ – $curriculum\ criterion\ or\ difficulty\ measure$;
$l$ – *curriculum level*;
$S$ – *curriculum scheduler*;
**for** $t \in 1, 2, \ldots, n$ **do**
  $p \leftarrow P(M)$
  **if** $S(t, p) = True$ **then**
    $M,\ E^*,\ P \leftarrow C(l,\ M,\ E,\ P)$
  **end if**
  $E^* = Select(E)$
  $M = Train(M,\ E^*,\ P)$
**end for**

The original definition from Begio et al.[3] only focuses on the training dataset ($E$), while it can be seen in Algorithm 1 that subsequent studies have expanded curriculum learning to model ($M$) and performance measure ($P$) as well. According to Soviany et al.[44], applying curriculum learning on training dataset ($E$), model ($M$), or performance measure ($P$) all have the same goal, namely to smooth the loss function. The curriculum criterion or difficulty measure ($C$) in the algorithm is the methodology of how to determine the ordering, and it modifies the training dataset ($E$) by, for example, sorting the data from easy to difficult; the model ($M$) by for example increasing model capacity; the performance measure ($P$) by for example adding complexity to the objective function. The curriculum is a sequence of training criteria that differ on different levels ($l$). Finally, the curriculum scheduler ($S$) is another critical component determining when to update $C$ for each level.

A more intuitive example of curriculum learning over the training dataset (E) is shown in figure 1.7, where there are three curriculum levels based on the time from t-k to t+k. The data is sorted and separated into three difficulty levels by curriculum criterion. These subgroups of data are then fed to the model one by one at the corresponding time stage, and the performance of the

trained model is evaluated each time.

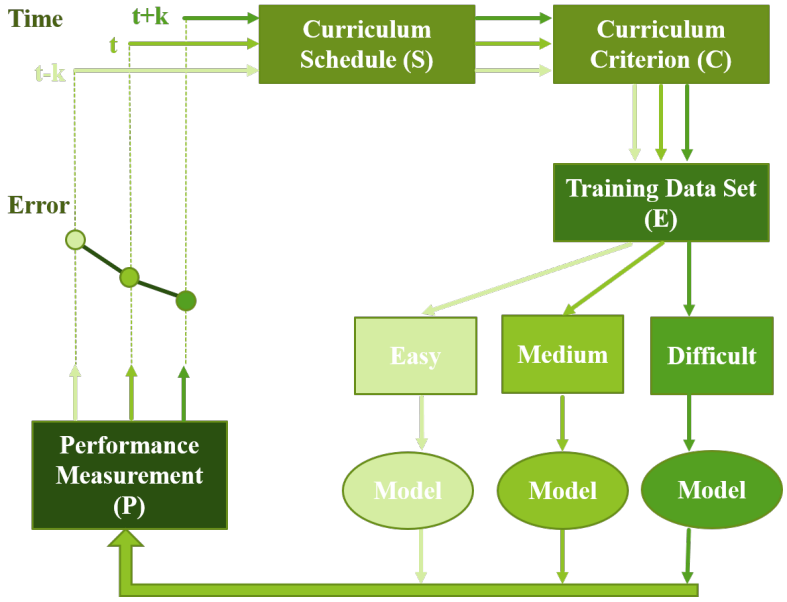


Figure 1.7: General framework of curriculum learning on training dataset [44].

Curriculum learning has been explored in various domains and proved to be effective in many of them, such as natural language processing (including speech processing), computer vision, medical imaging, and reinforcement learning. In addition to the vanilla curriculum learning proposed by Bengio et al.[3], other researches can be categorized into the following groups by a multi-perspective categorization: self-paced learning (SPL), self-paced CL, balanced curriculum, progressive CL, teacher-student CL, and implicit CL. For a brief introduction, unlike vanilla CL pre-defines the curriculum criterion, **SPL** re-evaluates the difficulty of the training data during the training process and resorts the data accordingly; **self-paced CL** is a combination of vanilla CL and SPL with both a pre-defined criterion and dynamical adjustments of the criterion; **balanced curriculum** guarantees the training data to be diverse enough so that the training can cover a balanced diversity; **progressive CL** modifies the task settings progressively, for example, Morerio et al., [31] apply progressive CL by proposing a time scheduling to mutate the dropout probability; **teacher-student CL** improves performance by jointly training two models, for example [22], one model evaluates the importance of the training samples (teacher) and the primary model (student) follows the evaluation while training; **implicit CL** is utilizing CL without sorting the training data in an apparent order.

# Chapter 2

# Problem Formulation

In this chapter, we first motivate the research question that we focus in this thesis, including the prospect of utilizing curriculum learning (CL) and the research gap in applying CL in information retrieval (IR). Then we give a formal definition of the CL framework in this work.

## 2.1 Motivation

Curriculum learning (CL), as a training strategy, can be applied to the training dataset, model, or performance measure, with the common goal to smooth the loss function or the objective function. Different CL methods are introduced in chapter 1. This paper mainly focuses on the vanilla CL [3], that is utilizing CL on the training dataset by sorting the data in a meaningful order (e.g., from easy to difficult according to a difficulty criterion) and then training the model with the sorted data. Compared with random sampling, CL can improve the training process (e.g., accelerate the convergence) and enhance model performance without extra computational costs and requirements for additional training data. These advantages or effectiveness of CL can be a promising research direction to tackle the following existing problem:

With recent breakthroughs and growing interest in models with deep neural architectures and million (e.g., BERT [12]) or even billion (e.g., GPT-3 [4]) parameters, the demand for datasets of massive size is dramatically increasing as well. However, it is sometimes hard to satisfy this longing for more training data, thus creating an obstacle to improving the model performance. For example, Talmor et al.[45], who have conducted an empirical investigation of generalization and transferring on ten reading comprehension (RC) datasets with the model DocQA [8] and BERT, believe that the bottleneck is the dataset size rather than the models.

Besides the benefits of utilizing CL, another motivation is the research gap in applying CL in information retrieval (IR). Although CL has been proved to work effectively on many natural language processing (NLP) fields, including machine translation [24][57][26] and speech processing [5][59][50], yet the effectiveness of CL has rarely been explored in IR [35]. Also, as far as we know, CL has rarely been studied in neural ranking models, except for the work by Penha et al.[35], where they accomplished an empirical study of CL on conversation response ranking (CRR) task with BERT. Their work tried seven different scoring functions and seven different pacing functions to compare if there is significant improvement with CL-scheduled training to baseline. It is worth noting that they name the difficulty criterion as scoring function, and the speed of transferring from easy data to difficult data as pacing function. Their results show that CL can improve model performance significantly. They also argue that neural ranking models are more suitable for utilizing CL than traditional Learning to Rank (LTR) models because no efficiency improvement over random sampling is found on the LTR with CL [15]. Also, traditional LTR models create representations based on manually engineered input features, whereas the neural ranking models

learn the representations automatically. In addition, Penha et al.[35] mainly focus on how to predefine the curriculum, which remains unchanged during the entire training process. Still, there is no exploration of any dynamic strategy. The dynamic strategy here means the difficulty criterion or speed criterion can change during the training.

## 2.2 Formalization

Corresponding to the motivations, this work explores how to apply curriculum learning (CL) dynamically on a neural language model to improve the model performance in IR. Following the choices of Penha et al.[35], we choose BERT as the neural language model as BERT has performed outstandingly on a wide range of NLP tasks. We choose conversation response ranking (CRR) as the task in IR not only as CRR is challenging and complex with multi-turn conversations and responses, but also that there are suitable large-scale datasets available. The formalization of the CRR task and the CL framework in this work is defined in the following subsections. The details of our dynamic methods are elaborated in chapter 4.

### 2.2.1 Conversation Response Ranking

A general procedure of conversation response ranking task is for any given conversation (query), to rank the responses (documents) from the most relevant to the least relevant with a measurement of relevance. We can formalize the dataset of CRR task $D$ consists of $N$ examples into $D = \{(C_i, R_i, Y_i)\}_{i=1}^{N}$, where $C_i$ denotes the conversation, $R_i$ denotes the responses, and $Y_i$ denotes the measurement of relevance. Every conversation $C_i$ usually consists of multiple utterances that can be formulated as $C_i = \{u_1, u_2, \ldots, u_\tau\}$. Each conversation $C_i$ is followed by a list of candidate responses $R_i = \{r_1, r_2, \ldots, r_k\}$, with a list of $k$ corresponding relevance labels $Y_i = \{y_1, y_2, \ldots, y_k\}$ indicating the relevance of each response $r_j$ to the conversation $C_i$. In our case, the response can be either relevant or irrelevant, thus the label is binary with each value $y_j = 1$ representing relevant and $y_j = 0$ representing irrelevant. Then the task is to learn a model or function $f(.)$ to make predictions of label denoted as $f(C_i, r)$ based on the content of conversation $C_i$ and response $r$. A visualization of the CRR task is in figure 2.1.

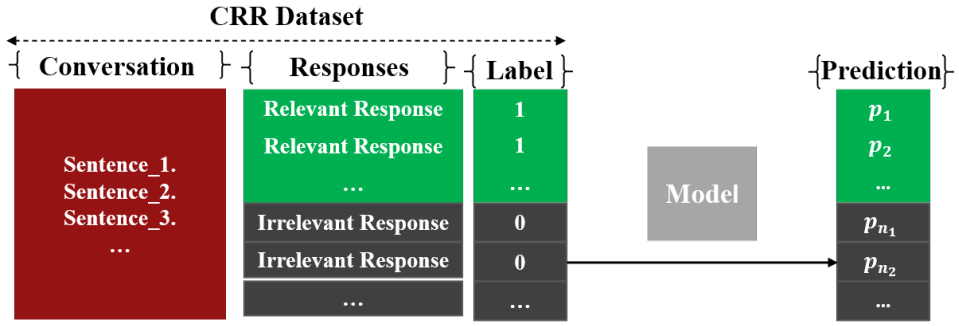


Figure 2.1: A visualization of the CRR task

### 2.2.2 Curriculum Learning

The procedure of a normal (non-CL) training of a language model is defined as follows. First the dataset $D$ is split into training dataset $D_{train}$, validating dataset $D_{val}$ and testing dataset $D_{test}$. Then for the training process, the model updates its parameters to maximize its objective function or minimize its loss function on the training dataset $D_{train}$ with size $N$. For each update, the model goes over a mini-batch of training samples denoted as $B = \{(C_i, R_i, Y_i)\}_{i=1}^{k}$ from $D_{\text{train}}$ where $k \ll N$, which is usually randomly sampled from $D_{train}$. We define one iteration $s$ as the

model updates its parameters once with a mini-batch of training samples $B$. We define training one epoch $e$ as the model sees all the training samples exactly once. Thus a one-epoch training is a sequence of iterations $[s_1, s_2, \ldots, s_{N/k}]$ and the model can be trained for multiple epochs until reaching the convergence.

Following the framework from Penha et al.[35], applying CL consists of two steps. The first is to sort the samples of the training dataset $D_{train}$ according to a difficulty criterion, and the second is to control the speed of transferring from easy to difficult data following a speed criterion. Specifically, for the second step, we gradually increase the size of accessible training data for mini-batch random sampling according to a speed criterion function. An intuitive illustration of the two CL framework steps is in figure 2.2. The CL-scheduled training of a language model can be formalized as follows. For the first step of CL, the training dataset with N samples $D_{train} = [d_1, d_2, \ldots, d_N]$ is sorted by a difficulty criterion function $f_{score}(.)$ into an ordered version of training dataset $D_{train\_ordered} = [d_{x_1}, d_{x_2}, \ldots, d_{x_N}]$. For the second step of CL, a speed criterion function $f_{pace}(.)$ takes the sequence of iterations $[s_1, s_2, \ldots, s_{N/k}]$ as the input, and then outputs a corresponding sequence of pacing values $[p_1, p_2, \ldots, p_{N/k}]$ (monotonically increasing) to gradually involve more difficult samples in the training. More details of difficulty criterion and speed criterion are elaborated in the following parts.

### Difficulty Criterion

Penha et al.[35] have compared seven different so-called scoring functions as difficulty criteria, including based on the conversation (number of sentences, number of words in conversation), response (number of words in response), the similarity of conversation and response (semantic cosine similarity, BM25[41]), the output of the model (loss, prediction). Moreover, their results show that only the scoring function based on the model's output can significantly improve over the non-CL trained baseline. Based on their observations, we select the best scoring method from their work as the difficulty criterion. This best method resembles the idea of Born Again Network[16], which has a teacher-student knowledge distillation structure. Born Again Network shows that although they share the same model architecture, the student can outperform the teacher. Now back to the best scoring method, it first fine-tunes the baseline BERT model on the training dataset, and then uses the BERT predictions of the label to calculate the score. A formalization of this scoring method is given as follows.

For each conversation $C_i$ in the training dataset $D_{train}$, there are one relevant (positive) response $r^+$ and one irrelevant (negative) response $r^-$. The difficulty score $score_i$ for sample $(C_i, R_i = \{r^+, r^-\}, Y_i = \{1, 0\})$ is defined as $score_i = \text{BERT}_{\text{pred}}(C_i, r^+) - \text{BERT}_{\text{pred}}(C_i, r^-)$. Given that any prediction satisfies $\text{BERT}_{\text{pred}}(C_i, r_i) \in [0, 1]$, the score $score_i$ is between -1 to 1 in theory. There are two extreme cases: one is $score_i = 1$, then BERT classifies both $r^+$ and $r^-$ correctly; the other is $score_i = -1$, then BERT is totally confused by the sample. To some extent, this score can represent how difficult the sample from the perspective of BERT.

### Speed Criterion

Penha et al.[35] have tried seven different so-called pacing functions as the speed criterion. Definitions of the pacing functions are in table 2.1, with the above seven (including the stand_training as the baseline) adopted from [35] and the last two (sigmoid and scurve) implemented by us. Figure 2.3 is a visualization of these pacing functions. Generally speaking, the pacing function outputs a value according to the current number of iterations. The output determines the range of training data used for random sampling mini-batches. Take the linear pacing function in figure 2.2 as an example, when the iteration number is $s = 500$, the output is $f_{pace}(500) = 0.70$ which means currently the first 70% of the ordered training dataset $D_{train\_ordered}$ can be used for sampling, while the rest are still inaccessible. Now we can explain further with the notations in the equations. $\delta$ denotes the percentage of dataset accessible at the beginning, and $T$ denotes the number of itera-

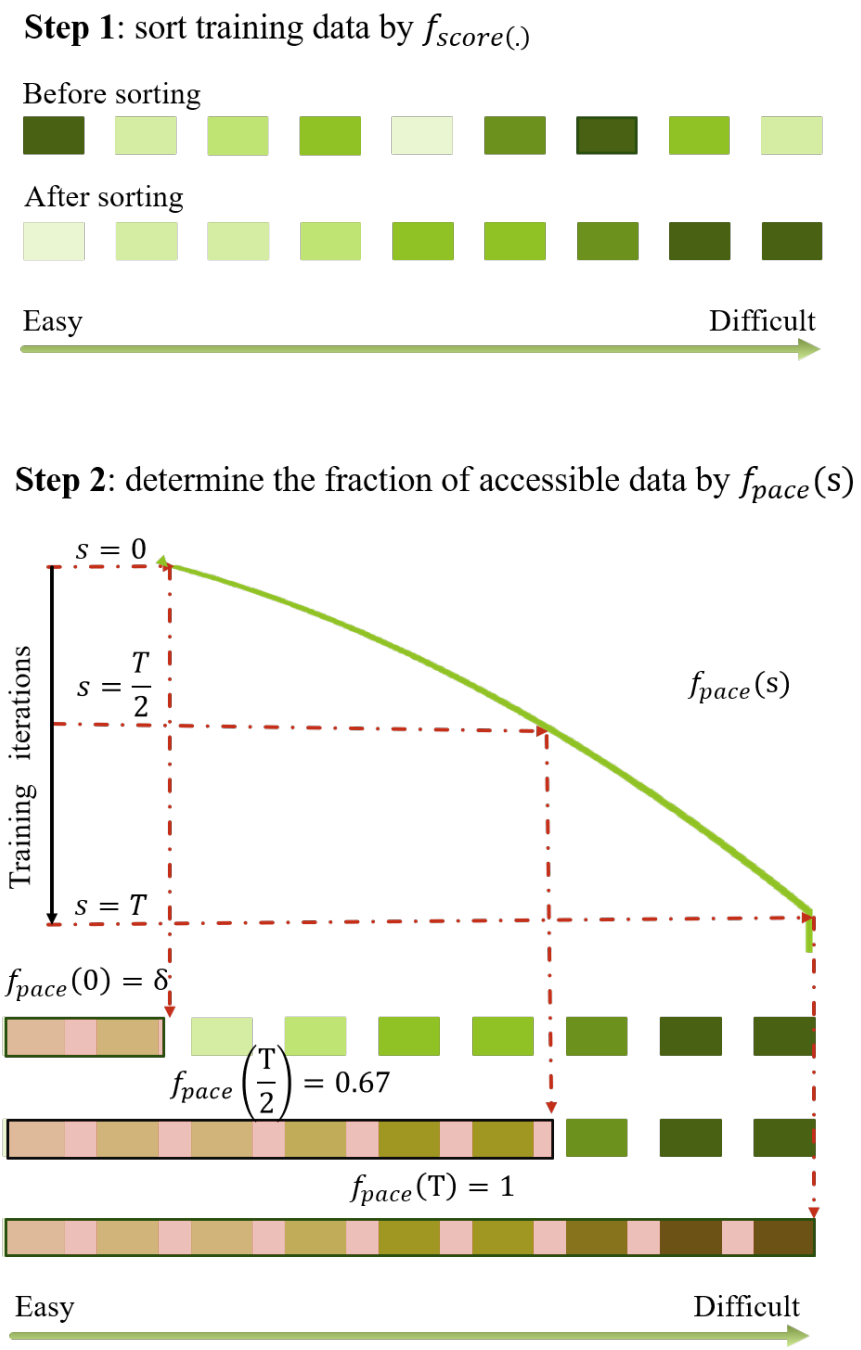


Figure 2.2: An intuitive illustration of the CL framework from [35], $\delta$ and $T$ are hyper-parameters.

tions after which the complete dataset will be used. In theory, $\delta$ should be at least no less than the mini-batch size, otherwise not enough data for compositing a mini-batch. Besides, all the pacing functions are monotonic non-decreasing functions with the value ranges from $\delta$ to 1, which means more difficult data are included in the training as the number of iterations increases. It is worth noting that baseline (standard_training) always has the pacing value 1. Thus all mini-batches are sampled from the entire dataset during training, and it equals random sampling.

Table 2.1: Definitions of pacing functions, $\delta$ and $T$ are hyper-parameters.

| **Pacing function** | **Definition** | **Shape** |
|---|---|---|
| standard_training | $f_{pace}(s) = 1$ | / |
| root | $f_{pace}(s,n) = \min\left(1, \left(s\frac{1-\delta^n}{T} + \delta^n\right)^{\frac{1}{n}}\right)$ | / |
| linear | $f_{pace}(s,n) = root(s,1)$ | straight line |
| root_n (2,5,10) | $f_{pace}(s,n) = root(s,n)$ | concave |
| geom_progression | $f_{pace}(s) = \min\left(1, 2^{\left(s\frac{\log_2 1-\log_2 \delta}{T} + \log_2 \delta\right)}\right)$ | convex |
| step | $f_{pace}(s) = \begin{cases} \delta, & if\ s \leq T*0.33 \\ 0.66, & if\ s > T*0.33,\ s \leq T*0.66 \\ 1, & if\ s > T*0.66 \end{cases}$ | stepped |
| sigmoid | $f_{pace}(s) = \frac{1}{1+\exp\left(\frac{-10*s}{T} + \ln 2\right)}$ | s-shape |
| scurve | $f_{pace}(s) = \begin{cases} \delta, & if\ s = 0 \\ \frac{1-\delta}{\left(\frac{T}{s}-1\right)^3+1} + \delta, & else \end{cases}$ | |

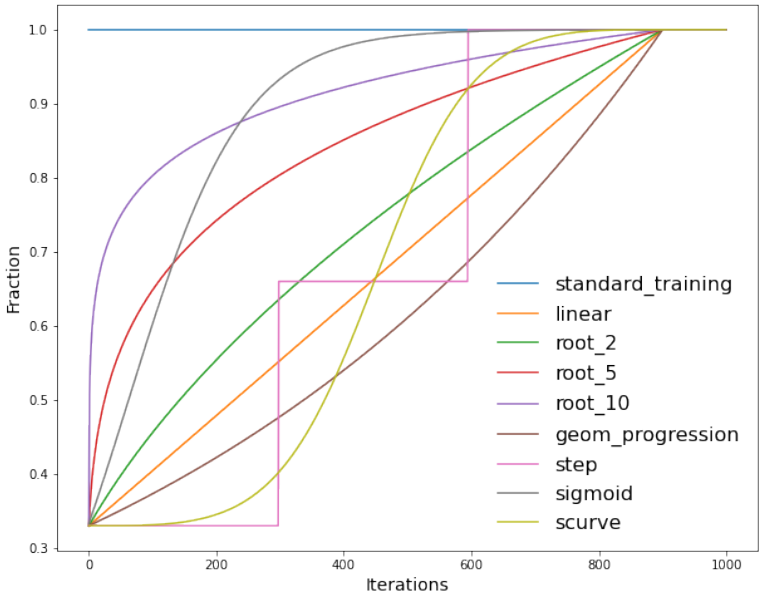


Figure 2.3: Pacing functions with $T = 900,\ \delta = \frac{1}{3}$.

# Chapter 3

# Related Work

In this chapter, we first review the related neural ranking models to motivate the choice of BERT. Then we introduce some previous works on the two main steps of vanilla curriculum learning (CL) as stated in chapter 2, namely the difficulty criterion and the speed criterion. Finally, we summarize the remaining challenges.

## 3.1 Neural Ranking Models

Retrieval ranking models retrieve and rank relevant documents by measuring the relevance between the documents and the query. Different relevance measurement approaches can be used to categorize these retrieval ranking models into the following types. Traditional text ranking models [48] usually compute the relevance or similarity by counting the frequency of the presence of query words in the documents, for example, BM25 [40]. Non-neural machine learning ranking models, also known as learning to rank (LTR) models, are learned by taking hand-crafted features as input. The requirement for hand-crafted features is a limitation because designing these features is domain-specific and time-consuming. Whereas neural architectures can take raw text data as input, they have gained more and more attention. Neural ranking models can be roughly categorized into representation-focused [18][43], interaction-focused [17] and representation-interaction-mixed [30]. In brief, representation-focused models match query and document into feature vectors separately and compute the similarity of the vectors, whereas interaction-focused models create a matrix of query and documents and extract the important interaction pattern.

Recent researches have shown the prospective future of attention-based neural language models. Transformer, as one of the most leading and influential attention-based models, was proposed by Vaswani et al.[49]. Transformer is the first sequence transduction model to rely only on the attention mechanism, as it has disposed of recurrences and convolutions completely. In machine translation, Transformer can learn explicitly faster than recurrent or convolutional neural networks. Based on the architecture of the Transformer encoder, Devlin et al.[12] proposed the model known as ***B***idirectional ***E***ncoder ***R***epresentations from ***T***ransformers (BERT), which is pre-trained on a great amount of unlabeled data with the next sentence prediction task and masked language task. BERT can be fine-tuned on one output layer and generalized to various tasks, showing state-of-the-art performance on eleven natural language processing (NLP) tasks. In information retrieval, the sentence pair classification setting in BERT has solved multiple retrieval and ranking tasks [10][7][32]. The outstanding competence of BERT motivates us to use it as a strong non-curriculum learning baseline in this work.

## 3.2 Curriculum Learning Frameworks

Inspired by the human learning process, Bengio et al.[3] proposed vanilla curriculum learning in 2009. They made a breakthrough by showing several cases with deep architectures where curriculum learning could significantly reduce the generalization error compared with non-curriculum one. They also tried to explain the effectiveness of curriculum learning by introducing a hypothesis that a well-designed curriculum works as a continuation method and a regularizer. That means such a curriculum helps find a better local minimum of a non-convex training criterion, and meanwhile, it regularizes the process leading to a lower generalization error. As CL is a promising method to improve the model performance and make performance converge faster without requiring for extra data, it has been widely explored in various complex NLP tasks, such as neural machine translation [24][57][57], speech processing [5][59][50], natural language generation [28][47][55], and so on.

Meanwhile, many different directions of curriculum difficulty criterion and speed criterion have been explored. In the work of Kocmi et al.[24], they judged the difficulty of each translation sample pair based on various linguistics features (e.g., sentence length, number of conjunctions, number of nouns, number of verbs). Based on the difficulty criterion, they separated the data into mini-batch buckets and sampled data from easy to difficult. In addition to linguistic features, Zhang et al.[57] used the confidence of an associate model on its prediction as the difficulty criterion. From explorations of several different CL-schedules, they believe that CL can reduce convergence time without any trade-off in translation quality, but it is essential to have a good difficulty criterion. Regarding the speed criterion, they separated the training into several stages where easy data had a higher probability of being sampled in the early stages. Inspired by Zhang et al.[57], Zhou et al.[60] proposed a so-called uncertainty-aware curriculum learning on multiple translation tasks with two intuitions. The first is that higher cross-entropy and perplexity of a translation pair means it has a higher complexity. The second is that the uncertainty of the model's prediction can be used to determine the curriculum level.

The above-reviewed methods all define the training into discrete stages, whereas Platanios et al.[37] proposed the first continuous curriculum learning framework. Again linguistic features (e.g., sentence length, word rarity) was used as the difficulty criterion, but the speed criterion was a simple continuous function with the value determined by one parameter (the duration of learning). Compared with the previous CL discrete regimes, their framework requires no manually designed stages, and it is easy to tune with one parameter and highly adaptive to other tasks. Based on the framework of Platanios et al.[37], Penha et al.[35] have tried seven different scoring functions and seven different pacing functions to compare if there is significant improvement with CL-scheduled training to baseline on the conversation response ranking task. Furthermore, they found that the only difficulty criterion that achieves statistically significant improvement over non-CL baseline is the output of the model BERT.

## 3.3 Remaining Challenges

The various curriculum learning applications in processing language have all witnessed training or model performance improvement with CL strategies. To our knowledge, there is a research gap in applying CL in information retrieval (IR). Also, CL has rarely been studied in neural ranking models, except for the work by Penha et al.[35]. However, the two steps of CL are both static in their work. Namely, either the difficulty criterion or the speed criterion remains unchanged and static during the entire training process.

For the first step, they use only the same scoring file (generated by fine-tuning BERT) from the beginning of the training till the end. We argue that while the model is learning, its evaluation of the difficulty of the same data is also changing. Thus we design the **dynamic rescoring method**

to explore if re-sorting the data several times according to the current model can help improve the performance. For the second step, in addition to determining the speed by a pacing function, we combine the idea from the work of Fang et al.[14] and apply the **noise method**. Fang et al.[14] proposed the "Goal-and-Curiosity-driven Curriculum (GCC) Learning" in off-policy deep reinforcement learning with the human-like learning strategy. In brief, their idea is to allow the model to have more exploration in the early learning stage and gradually back to its learning goal. Likewise, in our work, by adding noise to the CL-scheduled data of a shrinking amount automatically adapted with the number of iterations, we explore if more flexibility can be beneficial. The details are further explained in chapter 4.

# Chapter 4

# Methodology

This chapter first illustrates the curriculum learning (CL) framework in this work, then the dynamic method for the first CL step, and the noise method for the second CL step.

## 4.1 Curriculum Learning Framework

The curriculum learning framework in this work contains two main steps. First, the training dataset is sorted according to a scoring file generated by fine-tuning the model, and then the training follows a pacing function to determine the range of training data that can be sampled at each iteration. In addition to the CL framework [35], we also propose the dynamic rescoring method as a substitute to the baseline CL strategy with fixed scoring file, and the noise method adding noise to the CL-scheduled data determined by the pacing function.

### 4.1.1 Generating Scoring File

As elaborated before in chapter 2, we take the knowledge from the BERT as the difficulty criterion. Namely, we fine-tune the baseline BERT on the training dataset to generate a scoring file for sorting. The generated scoring file contains a score for each conversation which is the difference between the BERT's predictions of its relevant response and irrelevant response. Naturally, the bigger the score is, the easier the conversation-responses sample is.

### 4.1.2 Sampling with Pacing Function

With the ordered training dataset sorted by the scoring file, we then random sample a mini-batch of data for each iteration. The fraction of the dataset serving as the sampling candidates pool is determined by the output of a pacing function, which has been formalized and visualized in chapter 2. As any of the pacing functions is monotonically increasing, more and more difficult data is added in the pool, thus increases the difficulty of the curriculum. We set two hyper-parameter $T$ and $\delta$ to control the pacing, where $\delta$ denotes the percentage of data accessible at the beginning, and $T$ denotes the number of iterations after which the complete dataset will be used.

## 4.2 Dynamic Rescoring Method

For the first step, we propose the dynamic rescoring method to compare with the method that uses the same scoring file and sorts the data only once. For the dynamic rescoring method, the best model with the highest validating MAP is used to rescore the training samples every few iterations. We propose this method because we think that while the model is learning, its evaluation of the difficulty is also changing. Therefore, we want to check if the newly learned information can be helpful in the subsequent training. A visualization of the dynamic rescoring process is shown in

figure 4.1, each square represents a sample, and the darker color indicates the sample is more difficult. The first row means the original dataset with no order, then the fine-tuned baseline BERT generates the starting scoring file and sorts the data into the second row, after which the color changes because the parameters of the model have been updated and the updated model generates new scores.


Figure 4.1: Rescoring by the best model at different training time.

## 4.3 Noise Method

For the second step, we propose the noise method resembling the idea of a trade-off between exploitation and exploration from the work of Fang et al.[14]. For this method, we use a small part of difficult data as noise to replace some of the easy data from the sampling candidate pool. Moreover, the noise part is shrinking with the number of iterations increases as shown in figure 4.2, where the blue part represents the original cl-scheduled data by the pacing function root_2, and the orange part represents the noise data. This noise method resembles a human-like learning strategy where humans tend to explore more in the early learning stages and then gradually focus on their learning goal.

We have two parameters $\lambda$ (**noise_lambda**) and $q$ (**noise_difficult_ratio**) to support this method. The parameter $\lambda$ is for adjusting the amount of noise along with the iterations, and the parameter $q$ determines the range of difficult data that serves as the noise candidates pool. We combine $\lambda \in [0.99, 0.995, 0.999]$ and $q \in [1, 0.66, 0.5]$ and show nine situations in figure 4.2. We further illustrate this method by the pseudo-code of algorithm 2, where $s$ denotes the number of iterations, $p$ denotes the output of pacing function, $ns$ denotes the size of the noise part, $cls$ denotes the size of the CL-scheduled part, and $bs$ denotes the batch size. It is worth noting that the size of the noise part $ns$ decreases exponentially with the number of iterations, so a relatively bigger value of $\lambda$ is preferred. Otherwise, the noise disappears too early. Also, we include the difficult data that starts from the end of the dataset as noise, so $q$ means the last $q$ of samples. In a word, we add a shrinking amount of noise data to replace part of the CL-scheduled data in the early training stages, and the noise data is randomly sampled from the relatively more difficult samples. Also, we give a visualization of the noise method in figure 4.3.

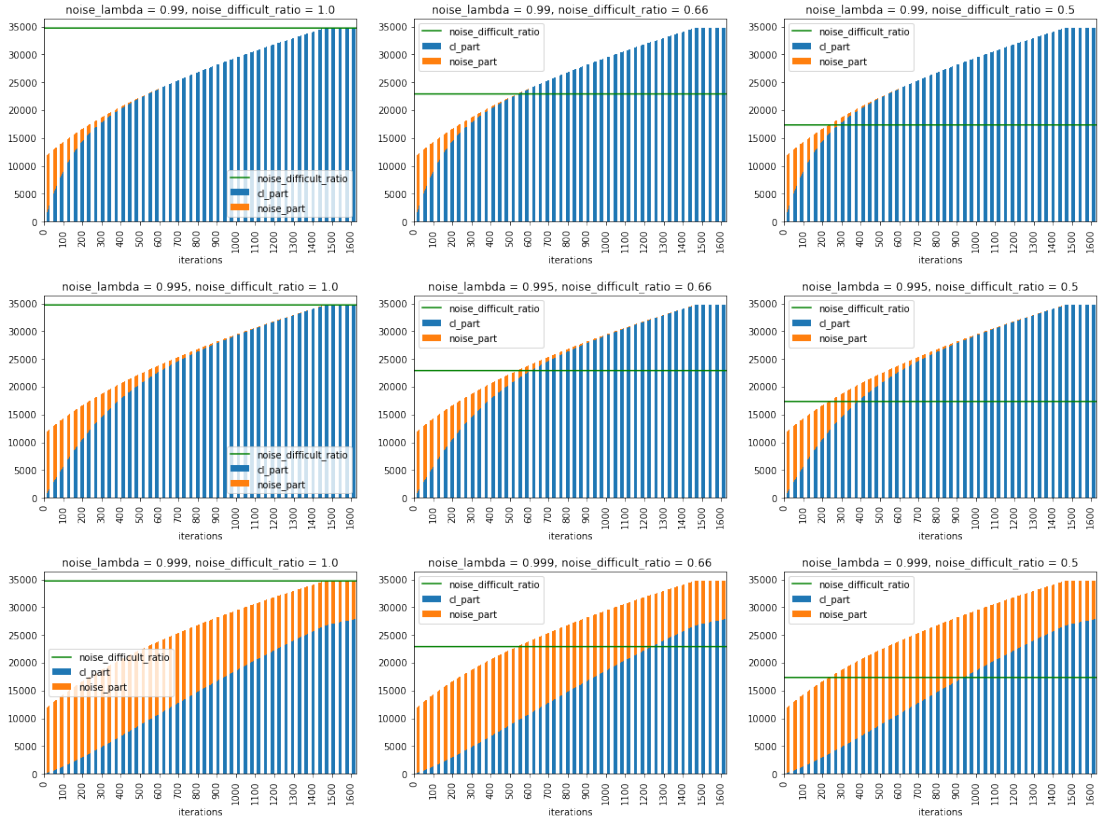


Figure 4.2: Visualization of the noise method with different noise_lambda and noise_difficult_ratio.

**Algorithm 2** Noise method.

$p \leftarrow \textbf{Min}(pacing\_function(s),\ 1)$
$ns \leftarrow \lambda^s \times p$
$cls \leftarrow p - ns$
$cl\_data \leftarrow \textbf{RandomSample}(D_{train\_ordered}[:cls],\ cls)$
$noise\_data \leftarrow \textbf{RandomSample}(D_{train\_ordered}[1-q:],\ ns)$
$sampling\_pool = \textbf{Concat}(cl\_data,\ noise\_data)$
$batch = \textbf{RandomSample}(sampling\_pool,\ bs)$

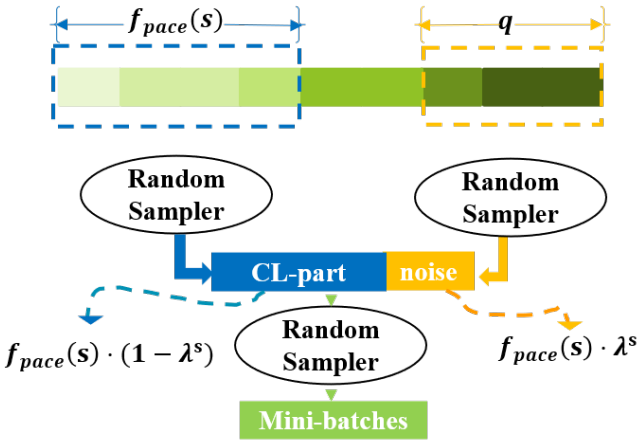


Figure 4.3: A visualization of the noise method.

# Chapter 5

# Experiments and Results

This chapter consists of two sections. The first section introduces two large-scale conversation response ranking (CRR) datasets, MSDialog and MANtIS, and the state-of-the-art language model BERT. Then we present the experimental implementations, including data pre-processing, training setting, and evaluation metric. In the second section, we report the results of the baseline CL framework with new pacing functions, the dynamic rescoring method, and the noise method, with the validating MAP presented in figures and the testing MAP listed in tables. We also give our discussions of the results. All the experiments were run on the Google Colab GPU. The source code and results are available in the link `https://github.com/ChenSQian/master-thesis`.

## 5.1 Experimental Setup

### 5.1.1 Conversation Response Dataset

A general format of data in the CRR dataset is in table 5.1. There could be one or more relevant responses and several irrelevant responses for one conversation. The task is to rank the relevant ones to the front and the irrelevant ones to the end. In this work, we use MSDialog and MANtIS, which are further introduced in the following sections.

Table 5.1: Format of CRR data.

| Label | Conversation | Response |
|---|---|---|
| 1 | utterance_1 **[SEP]** utterance_2 **[SEP]** ... **[SEP]** | relevant_response |
| 1 | utterance_1 **[SEP]** utterance_2 **[SEP]** ... **[SEP]** | relevant_response |
| ... | ... | ... |
| 1 | utterance_1 **[SEP]** utterance_2 **[SEP]** ... **[SEP]** | relevant_response |
| 0 | utterance_1 **[SEP]** utterance_2 **[SEP]** ... **[SEP]** | irrelevant_response |
| 0 | utterance_1 **[SEP]** utterance_2 **[SEP]** ... **[SEP]** | irrelevant_response |
| ... | ... | ... |
| 0 | utterance_1 **[SEP]** utterance_2 **[SEP]** ... **[SEP]** | irrelevant_response |

#### MSDialog

The MSDialog [56][39][38] dataset contains more than 35k dialogues of question answering interactions between information seekers and answer providers on Microsoft products from the forum Microsoft Community. The forum provides high-quality technical support from Microsoft staff and other experienced users to the questions posted by information seekers. MSDialog has several versions, and we use the MSDialog-ResponseRank version for the CRR task. The MSDialog-ResponseRank dataset is constituted by first filtering in the dialogues with the number of turns

ranges from 3 to 90, and then splitting the dataset into training/validating/testing according to the time of the question, and finally using the true answer from the agent as the relevant response and negative sampling the irrelevant responses. An example of MSDialog is in table 5.2 and a statistical descriptions of MSDialog is in table 5.3.

Table 5.2: An example of MSDialog.

| Label | Conversation | Response |
|---|---|---|
| 1 | Every time I open my outlook web app, the same e-mail from 9/10/2014 is displayed in the preview panel. It's not caused any problems yet, ... I've tried changing the preview settings back and forth from ""Show the first message automatically"" to ""Show a message only after I select it"" ... **[TAB]** Hi PERSON_PLACEHOLDER, We appreciate your time and effort for getting back to us and I apologize for the delay in responding. **[TAB]** Hi PERSON_PLACEHOLDER, Thanks for checking up! Yes the issue is still there. Any suggestions you have will be appreciated! | <<<AGENT>>>: Hi PERSON_PLACEHOLDER, We appreciate your time and effort for getting back to us and I apologize for the delay in responding. I understand that even after deleting certain e-mails, it is still showing in the preview Window whenever you open the Mail app. If you have configured your e-mail account in multiple devices... |
| 0 | Every time I open my outlook web app, the same email from 9/10/2014 is displayed in the preview panel. It's not caused any problems yet, ... I've tried changing the preview settings back and forth from ""Show the first message automatically"" to ""Show a message only after I select it"" ... **[TAB]** Hi PERSON_PLACEHOLDER, We appreciate your time and effort for getting back to us and I apologize for the delay in responding. **[TAB]** Hi PERSON_PLACEHOLDER, Thanks for checking up! Yes the issue is still there. Any suggestions you have will be appreciated! | <<<AGENT>>>: Thank you for posting your response. We would like to know the version of Windows Photo Gallery you are using. I suggest you check the Region and Language setting on your computer, if it is set to your current location. As Order Prints option in Windows Photo Gallery ... |

Table 5.3: Statistical descriptions of MSDialog.

| Type | Train | Valid | Test |
|---|---|---|---|
| Time of questions | 2005/11-2017/08 | 2017/08-09 | 2017/09-10 |
| Original Number of $(C, r)$ pairs | 174k | 37k | 35k |
| Number of $(C, r)$ pairs we use | 35k | 37k | 35k |
| Avg number of words per $C$ | 55.8 | 55.8 | 52.7 |
| Avg number of words per $r$ | 67.3 | 68.8 | 67.7 |
| Number of $r^+$ per $C$ | 1 | 1 | 1 |
| Number of $r^-$ per $C$ | 1 | 9 | 9 |

**MANtIS**

The MANtIS [34] is a multi-domain information-seeking dataset containing more than 80k dialogues collected from the question answering forum Stack Exchange. The MANtIS also has a conversation response ranking version that is generated similarly to MSDialog. However, due to a problem with the URLs of the dataset with irrelevant candidates, we can only access the one

with no irrelevant responses. So, we have used the negative sampling method from [36] and data transformation to add irrelevant responses. An example of MANtIS is in table 5.4 and a statistical descriptions of MANtIS is in table 5.5.

Table 5.4: An example of MANtIS.

| Label | Conversation | Response |
|---|---|---|
| 1 | Im using windows 8 my system is :İntel core2duo t6400 2.0 ghz 2mb cache6 gb ddr3 1033 mhz ramNvidia 9600mGS Gpuand i have 500 gb hdd i love to play games so i prefer windows 8 for games but i would lıke to use ubuntu for my daily use i am thinking to use it alongside with windows 8 it is possible right? ... [**UTTERANCE_SEP**] Ubuntu 12.04 is a Long Term Support Release (LTS) while 13.04 is a Regular Release.If you want the latest and cutting edge technologies that Ubuntu has to offer, you may use 13.04 (or 13.10) but if you are looking for a tried and proven stable environment with continuous updates in the near future, I will stick with 12.04.... [**TURN_SEP**] first of all thank you for your answer i will use ubuntu for only daily use watch movies and internet surf so i am downloading 13.04 now but is my system enough for ubuntu ? beacuse ı was tried 12.04 before and it was feeling a bit laggy ... [**UTTERANCE_SEP**] [**UTTERANCE_SEP**] | Ubuntu's system requirements are quite modest according to the [System Requirements] Page (https://help.ubuntu.com). The lag that you experienced before might be due to the fact that your video card drivers are not installed. |
| 0 | Im using windows 8 my system is :İntel core2duo t6400 2.0 ghz 2mb cache6 gb ddr3 1033 mhz ramNvidia 9600mGS Gpuand i have 500 gb hdd i love to play games so i prefer windows 8 for games but i would lıke to use ubuntu for my daily use i am thinking to use it alongside with windows 8 it is possible right? ... [**UTTERANCE_SEP**] Ubuntu 12.04 is a Long Term Support Release (LTS) while 13.04 is a Regular Release.If you want the latest and cutting edge technologies that Ubuntu has to offer, you may use 13.04 (or 13.10) but if you are looking for a tried and proven stable environment with continuous updates in the near future, I will stick with 12.04.... [**TURN_SEP**] first of all thank you for your answer i will use ubuntu for only daily use watch movies and internet surf so i am downloading 13.04 now but is my system enough for ubuntu ? beacuse ı was tried 12.04 before and it was feeling a bit laggy ... [**UTTERANCE_SEP**] [**UTTERANCE_SEP**] | I get the impression you are new to Ubuntu. Drivers are not installed by coaxing the operating system to look in a certain place and find them, the way windows has always been. In Ubuntu, and all versions of Linux I've ever used, there's a procedure and that procedure is usually spelled out in detailed instructions that were probably included inside a text file which came in the compressed file you have already downloaded. If you give me the name of the file you downloaded I will see if I can download it and read the instructions for you.CORRECTION!!!!... |

Table 5.5: Statistical descriptions of MANtIS.

| Type | Train | Valid | Test |
|---|---|---|---|
| Time of questions | the oldest 70% | the middle 15% | the last 15% |
| Original number of $(C, r)$ pairs | 90k | 18k | 18k |
| Number of $(C, r)$ pairs we use | 165k | 181k | 180k |
| Avg number of words per $C$ | 98.2 | 107.2 | 110.4 |
| Avg number of words per $r$ | 91.0 | 100.1 | 94.6 |
| Number of $r^+$ per $C$ | 1 | 1 | 1 |
| Number of $r^-$ per $C$ | 1 | 9 | 9 |

### 5.1.2 BERT

The BERT model was proposed by Devlin et al.[12], which stands for ***B***idirectional ***E***ncoder ***R***epresentations from ***T***ransformers. BERT is a deep pre-trained language model with multiple Transformer[49] encoder layers. The procedures for general pre-training and question answering task fine-tuning with BERT is shown in figure 5.1. Unlike many standard unidirectional language models, the BERT is pre-trained by using a "masked language model" (MLM) pre-training objective, which enables it to get deep bidirectional representations from the unlabeled text by jointly conditioning on both its left and right context. MLM means that the BERT randomly puts masks on some words and force itself to predict those masked words according to the context on both left and right side. Another special design of BERT is that the model adopts next-sentence-prediction tasks for training. As shown in figure 5.1, pairs of sentences are taken as input, and the BERT is trained to discriminate whether sentence B is the next sentence after A or not. This design enables the BERT to learn relationships between sentences and have a better understanding of the context. This sentence pair classification setting in BERT has been proved to solve multiple retrieval and ranking tasks [10][7][32]. The outstanding competence of BERT motivates us to use it as a strong non-curriculum learning baseline. We use the uncased BERT-base (12 layers, hidden size of 768, 110M parameters) as the neural ranking model in this work.

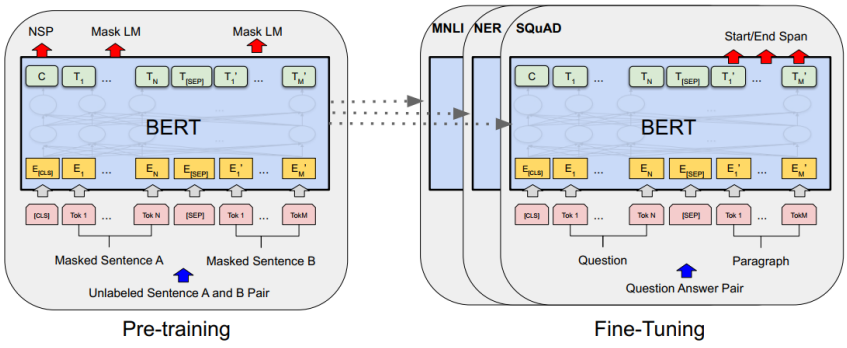


Figure 5.1: Overall pre-training and fine-tuning procedures for BERT [12].

### 5.1.3 Implementations

#### Pre-process MANtIS

The MANtIS train/valid/test dataset available online [33] is an incomplete one with no negative responses. The available format only has two columns, conversation and relevant response. Thus we have set up a pre-processing pipeline to transform the data into the form as desired. The first step is to add a label column with the value 1 to the dataset as originally, all the responses are

relevant. Then we use the negative sampling methods from the transformers_ranker library [36] to sample irrelevant responses. Among three sampling methods, we choose the BM25 [40] method to comply with the baseline curriculum learning framework [35].

The BM25 ranks a set of documents by considering the query terms appearing in each document, regardless of their proximity within the document [52]. The similarity score of BM25 method is calculated as presented in equation 5.1, where $q_i$ denotes the word in the query, $\text{IDF}(q_i)$ denotes the inverse document frequency weight of the query term $q_i$, $f(q_i, D)$ denotes the term frequency of $q_i$ in document $D$, $|D|$ denote the number of words in document $D$, avgdl is the average document length, and $k_i$ and $b$ are two hyper-parameters that can be chosen freely. In our case, for each conversation, the BM25 sampling method uses its relevant response as the query, and retrieves the most similar irrelevant responses from the sampling candidates pool. We use all the responses from itself and the training dataset as the sampling candidates pool for both validating and testing datasets. We have used the training dataset to reduce duplicated irrelevant responses as we need a huge number of negative samples, nine irrelevant responses for each conversation.

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)} \tag{5.1}$$

After the negative sampling step, the retrieved irrelevant responses still contain unwanted symbols that can be easily predicted as negative responses by BERT (with a validating MAP of 0.99). Thus we remove the unwanted symbols from the irrelevant responses. Finally, we sort the dataset according to the conversation and put the relevant response at the beginning, followed by nine irrelevant responses. A visualization of the pre-processing pipeline is shown in figure 5.2.

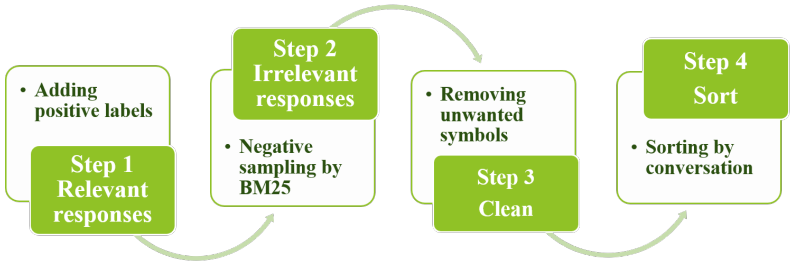


Figure 5.2: Pre-processing Pipeline of MANtIS.

### Training and Evaluating Setting

Following the framework of Penha et al.[35], we use the uncased base BERT with the Hugging Face library [53]. For the input, we concatenate the conversation and response with the SEP token. We take the conversation as segment A and the response as segment B. As we have set the maximum sequence length to 128, we truncate the sequence with the heuristic to keep a length balance between A and B whenever the concatenated sequence is excessively long. On the other hand, if the sequence is too short, we add padding tokens. The processed input format is illustrated in figure 5.3, consisting of four parts. The sequence is first tokenized by the BERT tokenizer, after which the tokens are converted into ids. The input ids, together with the input mask indicating if it is real tokens (1 for real tokens and 0 for padding tokens), the segment ids indicating the segment (0 for segment A and 1 for segment B), and the label ids constitute the input for BERT. We then fine-tune the BERT for sentence classification with the CLS token.

During training, we only select one irrelevant response to make the number of positive samples equal to the negative samples for each conversation in the training dataset. For optimization, the default cross-entropy loss and Adam optimizer [23] are used with a learning rate of 2e-5 and

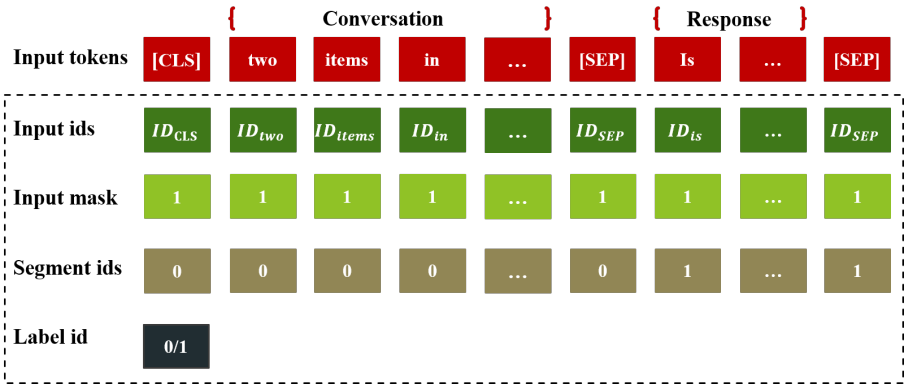


Figure 5.3: Convert data into valid input for BERT.

$\epsilon = 1e-8$. We train the model three epochs on MSDialog and one epoch on MANtIS with a batch size of 64. Because the MANtIS is around four times bigger than MSDialog, the model converges at different times. We evaluate the model performance on the validating dataset with the Mean Average Precision (MAP) metric every a fixed number of iterations to check if the model converges. However, for the MANtIS, with a limitation in training time with Google Colab GPU, we randomly sample 1/3 each time to validate instead of using the entire validating dataset. To eliminate the influence by initialization, we run each experiment with several runs (five seeds for MSDialog and three seeds for MANtIS again considering the data size) and use the average.

### Evaluation Metric

Mean average precision (MAP) [46] is a popular metric used to measure the performance of models in the field of information retrieval (IR). MAP is the mean of the average precision (AP) of every conversation in the dataset according to its name. Compared with the widely known precision that measures the ratio of true positive predictions, the average precision also considers the position of true positive predictions to judge the model's ranking ability. Before clarifying how to calculate the MAP, we need to define the precision at $k$ (P@k) shown in equation 5.2, which means the ratio of true positives until the $k^{th}$ position. Then equation 5.3 shows a general formula for calculating AP. Under the scenario of IR, $n$ refers to the number of retrieved documents for a query, GTP is the number of ground truth positives, $k$ denotes the position ranging from 1 to n, rel@k is an indicator function defined in equation 5.4 which is 1 and 0 for the relevant document at $k$ and irrelevant document at $k$, respectively. Finally, MAP is the mean of APs of all the queries in the dataset, as shown in equation 5.5.

$$\text{P@k} = \frac{|\{\text{ relevant documents@}k\} \cap \{\text{ retrieved documents@}k\}|}{|\{\text{ retrieved documents@}k\}|} \tag{5.2}$$

$$\text{AP@n} = \frac{1}{\text{GTP}} \sum_{k=1}^{n} \text{P@}k \times \text{rel@}k \tag{5.3}$$

$$\text{rel@k} = \begin{cases} 1 & \text{if document@k} \in \{\text{ relevant documents }\} \\ 0 & \text{if document@k} \in \{\text{ irrelevant documents }\} \end{cases} \tag{5.4}$$

$$\text{MAP@n} = \frac{\sum_{q=1}^{Q} \text{AP@n(q)}}{Q} \tag{5.5}$$

A visualization of two examples of calculating AP can be seen in figure 5.4. In these examples, the green ones with a checkmark and the orange ones with an X mark are relevant documents and

irrelevant documents, respectively. So the number of ground truth positives (GTP) is 3. Intuitively, the above example is worse than the below example because two irrelevant documents are ranked ahead of the relevant documents. In addition, the below example is a particular example that the ranking is entirely correct, and it gets an AP of 1. And the MAP of these two samples is the mean of their APs.

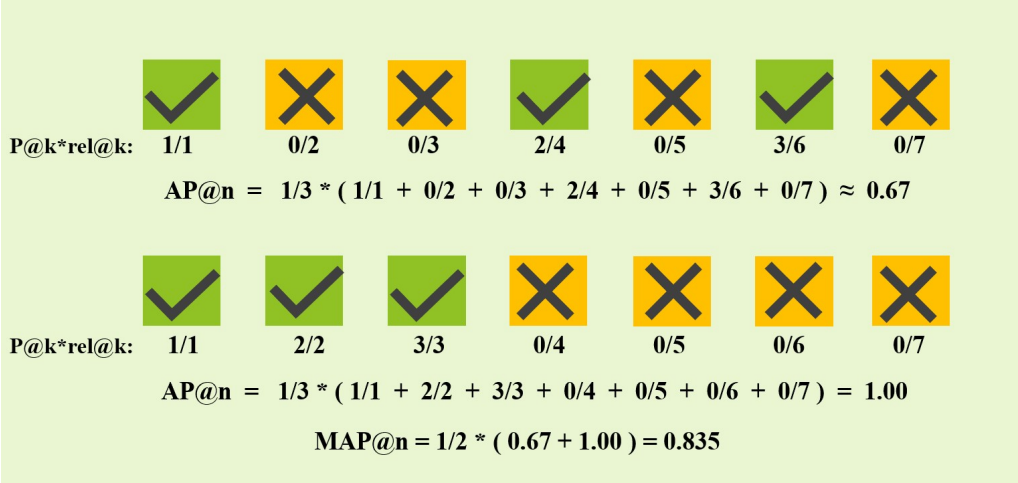


Figure 5.4: The calculation of MAP of two samples.

In the experimental setup of this work, the training dataset contains conversations with one relevant response and one irrelevant response for each. In comparison, both the validating and testing dataset consist of conversations with one relevant response and nine irrelevant responses. The different setting in the training dataset is to generate a scoring file for curriculum learning which has been elaborated on in chapter 4. MAP is calculated in the same way on both validating dataset and testing dataset. We take one arbitrary conversation $C_i$ from the validating dataset for example, $C_i$ has a response list $\{r_0, r_1, r_2, \ldots, r_9\}$ of length ten, where only $r_0$ is a relevant response. We get the corresponding model predictions $\{pred_0, pred_1, pred_2, \ldots, pred_9\}$ with every $pred_j \in [0, 1]$, after which we sort the predictions to a descending order. And then, together with the true label indicating the position of the relevant response $r_0$, we calculate the AP on the sorted list of predictions, as well as MAP both in the same way as the examples in figure 5.4.

## 5.2 Results

### 5.2.1 Basic Experiments with CL Framework

We first use the CL framework of Penha et al.[35] to do some basic experiments on the relatively smaller dataset MSDialog to check if their CL framework is beneficial for improving model performance. The scoring file we use for these experiments is generated by fine-tuning the baseline BERT three epochs with the training batch size of 8. Then, we compare CL and inverse CL results with six different pacing functions by the values of the mean average precision (MAP) of the BERT (the best till the moment) evaluated on the validating dataset and the MAP on the testing dataset. For the testing, we also use Student's t-test to check the statistical significance over baselines. In addition, we implement two more pacing functions called sigmoid and scurve to compare with the original ones. Likewise, for the model performance of the other experiments, we judge all by the same metric MAP. Besides, any of the single lines in the following figures is the average of five or three runs. Furthermore, the testing MAP results are listed in the following tables with an order of MAP values from high to low.

### CL and Inverse CL

The baseline non-CL is denoted as standard_training in the following figures and tables. The pacing function for the non-CL is always 1, so the complete training dataset is always accessible during the entire training process. As shown in the left part of figure 5.5, we can see all the CL training outperforms the baseline (black) from the beginning iterations. We also do the inverse CL experiments by sorting the data in inverse order from difficult to easy, the results of which are shown in the right part of figure 5.5. In inverse CL, the baseline beats all the others, with a pronounced distance, especially at the beginning between the baseline and several pacing functions (e.g., geom_progression, step, linear). Furthermore, for any pacing function, the validating MAP of CL is at least 3% higher than inverse CL with the same pacing function. We argue the results indicate that difficult examples can easily confuse the model, especially at early training stages when the model has little knowledge of the training dataset. The results comply with the hypothesis that CL can smooth the objective function with more straightforward examples. Overall, by comparing the CL and inverse CL results, we could see CL can improve the MAP of BERT on the validating dataset.

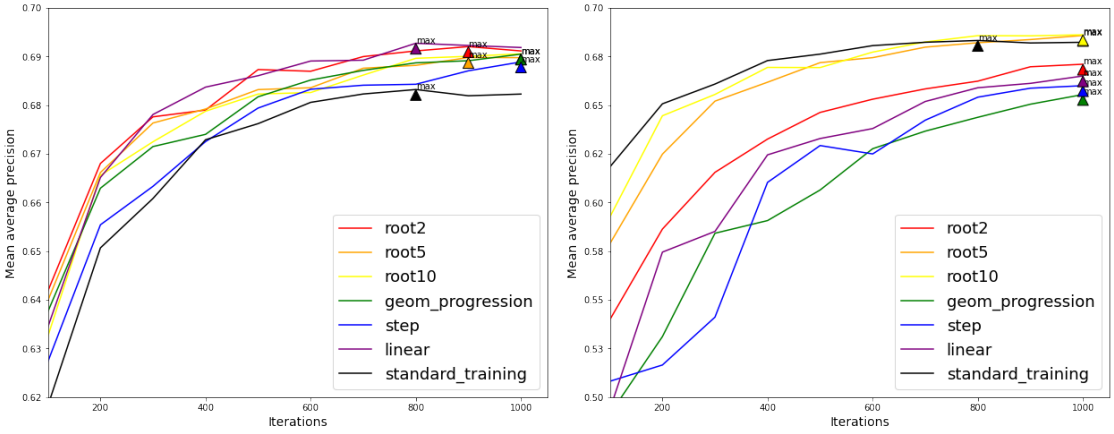


Figure 5.5: Validating MAP with CL (left) and inverse CL (right) training.

### New Pacing Functions: Sigmoid and Scurve

Then we implement two s-shape curve functions, sigmoid and scurve, to compare with the original pacing functions. As the figure 2.3 shows in chapter 2, the original seven pacing functions can be categorized into four types including convex (geom_progression), concave (root_n), straight line (linear) and stepped (step). Convex type starts slow and becomes faster later, whereas concave type starts fast and slows down later. Linear keeps a stable speed from the beginning to the end, whereas step splits the process into several stages and has a speed jumping from one stage to the next stage.

For a broader exploration, we adopt another type of different shape, where the pacing is from slow to fast and back to slow. This idea also comes from one idealized general form of the human learning curves [54], where people make progress slowly at first and gradually accelerate in the middle and slow down again when the learning activity reaches its limit. Unfortunately, the results of the sigmoid and scurve in figure 5.6 are worse than the other pacing functions. Especially the validating MAP with scurve is lower than the baseline, which on the other hand tells us that a proper pacing function is essential. It is worth noting that because the pacing function should start at 1/3 and reach 1 after 90% of iterations because of the experimental setting, the shape of the sigmoid function is an incomplete s-shape but is close to a convex curve. This incomplete s-shape can explain why sigmoid is better than scurve. Regarding the failure of the scurve, we

think it increases the pace too slowly at the first half training part (e.g., only the first 40% training data is accessible before the 30% of training iterations). Thus the model sees easier examples more times than difficult ones and assigns very uneven weights to the easy and difficult ones.

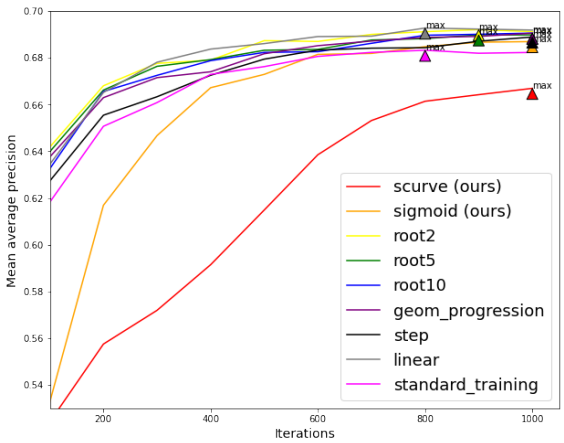


Figure 5.6: Validating MAP with sigmoid and scurve pacing function.

The testing results are shown in table 5.6, where they are sorted by testing MAP from high to low. Based on the validating results, we select the best CL pacing function root_2 and the worst inverse CL pacing function geom_progression. We also test on the new pacing functions sigmoid and scurve, as well as the baseline standard_training. Overall the testing results comply with the validating results that CL training outperforms inverse CL training. Moreover, the most outstanding difference is between CL and inverse CL with geom_progression, where CL is 4% better than CL. The t-test shows root_2 trained models with seeds 1 and 4 outperform the baseline non-CL models, while the other seeds have no statistically significant improvement. Nevertheless, we also see many inverse CL trained models, including the inverse_geom_progression models with all five seeds, are defeated by the baseline non-CL models at a 99% confidence interval.

Table 5.6: MSDialog testing MAP results of baseline non-CL (standard_training), baseline CL, inverse CL, and new pacing functions: sigmoid and scurve with five runs. The best testing MAP of each run is in bold. Superscripts($^{**/*}$) indicate statistically significant improvements over the baseline non-CL (standard_training), where * means the Student's t-test p-value is less than 0.05, and ** means p-value is less than 0.01.

| Seed | 1 | 2 | 3 | 4 | 5 | Average | SD |
|---|---|---|---|---|---|---|---|
| root_2 | **0.729**$^{**}$ | 0.713 | **0.728** | 0.722$^{**}$ | **0.732** | **0.725** | 0.007 |
| geom_progression | 0.728$^{*}$ | **0.714** | 0.727 | **0.723**$^{**}$ | 0.730 | 0.724 | 0.006 |
| sigmoid (ours) | 0.719 | 0.704 | 0.722 | 0.722$^{**}$ | 0.725 | 0.718 | 0.007 |
| **baseline non-CL** | 0.718 | 0.707 | 0.726 | 0.703 | 0.727 | 0.716 | 0.010 |
| inverse_root_2 | 0.713 | 0.692 | 0.713 | 0.703 | 0.703 | 0.705 | 0.008 |
| scurve (ours) | 0.707 | 0.681 | 0.705 | 0.709 | 0.691 | 0.699 | 0.011 |
| inverse_geom_progression | 0.701 | 0.664 | 0.696 | 0.665 | 0.696 | 0.684 | 0.016 |

### 5.2.2 Dynamic Rescoring Method

In this section, we discuss the exploration of the dynamic rescoring method designed by us. We apply this method only on the smaller dataset MSDialog as it requires more time to rescore several

times than the original static CL approach. This rescoring method takes more time because each time to generate a new scoring file, the model needs to evaluate the entire training dataset for an epoch with a batch size of 2 (a conversation with one relevant and irrelevant response). In this exploration, we manipulate two hyper-parameters, one is the starting scoring file, and the other is the frequency of rescoring. They are manipulated separately. Namely, the frequency is fixed when we compare different scoring files and vice versa.

For the first hyper-parameter, we generate five different scoring files by fine-tuning the baseline BERT for zero (use the BERT directly) to four epochs with a batch size of 64. The corresponding results are denoted as **preds_dif_dynamic_n_root_2** in figure 5.8 and table 5.8 with $n$ indicating the number of fine-tuning epochs of the starting scoring file. We set this hyper-parameter because we want to know how much the quality of starting scoring file matters.

We judge the quality of starting scoring file by their MAP on the training dataset after the fine-tuning, as shown in table 6.2. More intuitively, we visualize their distribution of scores on the training samples in figure 5.7. The score is the difference between BERT's prediction of the relevant and the irrelevant response as stated in chapter 2. Thus, negative score means wrong prediction (orange in figure 5.7), while positive score means correct prediction (green in figure 5.7). We can see that the more epochs the BERT is fine-tuned, the higher training MAP and the more correct predictions it gets. Moreover, we infer BERT has reached its highest training MAP before the fourth epoch because the sub-figures of three and four epochs are almost the same, and the training MAP decreases. Another outstanding observation is that the zero-epoch fine-tuned BERT has much more wrong predictions than the others, shown by the first sub-figure in figure 5.7.

Table 5.7: Training MAP of different starting scoring files.

| **Number of fine-tuning epochs** | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **Training MAP** | 0.728 | 0.946 | 0.965 | 0.978 | 0.977 |

The fine-tuning batch size used before is 8, which is time-consuming. Thus we use 64 for this part. Furthermore, we pick root_2 for this part as it is one of the best pacing functions on MSDialog from the previous CL experiments. Besides, we fix the rescoring frequency to every 200 iterations, which means the model rescores eight times during the three-epoch training.

From the validating MAP, we can see that the dynamic rescoring method fails to outperform the baseline non-CL (black) in the left part of figure 5.8, but work equally well as the baseline CL (root_2) method. Another observation is that the scoring file generated by fine-tuning with more epochs is slightly more informative and beneficial on the validating dataset. This observation complies with the intuition that fine-tuning facilitates the BERT with more knowledge of the training data. However, on the testing dataset, the dynamic methods work almost equally well with each other except for the one with the starting scoring file generated by zero-epoch fine-tuned BERT. The dynamic_0_root_2 is outperformed by root_2 with all five seeds at the confidence interval of 95% or 99%. This extreme case shows that a starting scoring file generated by a completely "inexperienced" model can be misleading. On the other hand, for generating the starting scoring file, one-epoch fine-tuning can be good enough. Thus we can save time in this step. Also, the dynamically trained models show statistically significant improvement over baseline non-CL and the baseline CL trained models averagely in one seed out of five seeds. This observation might indicate that the dynamic method has no exciting performance.

For the second hyper-parameter, we fix the starting file to the three-epoch fine-tuned one and compare three different values of rescoring frequency, namely 200, 400, and 800 iterations. The results of 400, 800 are denoted as **preds_dif_dynamic_34_root_2** and **preds_dif_dynamic_38_root_2**, respectively. The results in the right part of figure 5.8 show that rescoring every 800 iterations

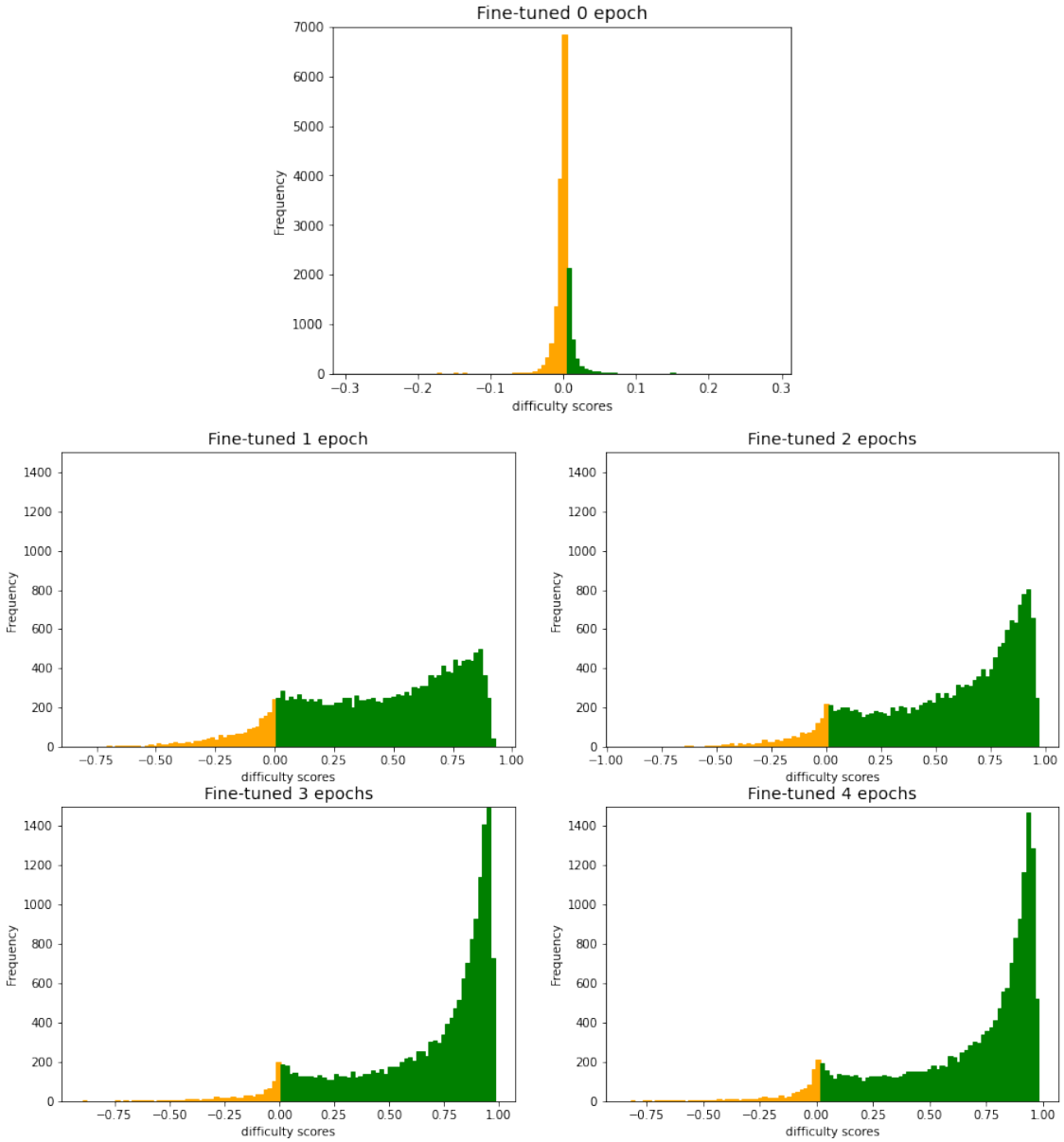


Figure 5.7: The distribution of scores on the training dataset, where the number of fine-tuning epochs ranges from 0 to 4. Orange indicates negative score, while green indicates positive score.

works best, and both frequencies of 400 and 800 iterations outperform the baseline CL, but they still cannot outperform the baseline non-CL. Moreover, the testing results only show a trivial difference between the CL methods and baseline non-CL. To conclude, our dynamic rescoring method is not beneficial compared with the baseline non-CL, while it works slightly better than the static baseline CL method.

Besides, because we spot that the original root_2 can not outperform the baseline non-CL in figure 5.8, which is contradictory to the result from the basic CL experiments in figure 5.5. This finding motivates us to rerun the same basic CL experiment with the scoring file generated with the only difference that the fine-tuning batch size is changed from 8 to 64. Furthermore, as the result in figure 5.9 tells, almost no pacing function outperforms the baseline non-CL. This again means

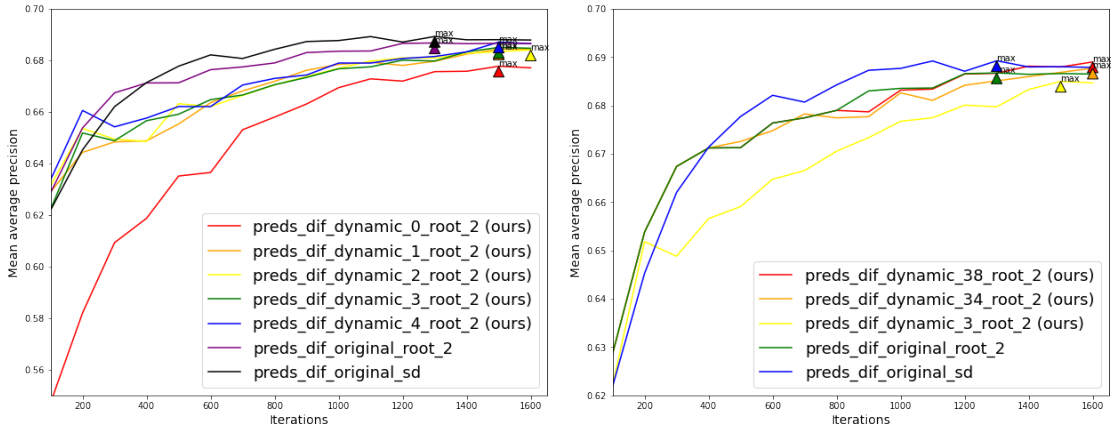


Figure 5.8: Validating MAP with dynamic rescoring: different starting scoring file (left), different frequencies of rescoring (right).

Table 5.8: MSDialog testing MAP results of dynamic method with five runs. Superscripts($^{**/*}$) indicate statistically significant improvements over the baseline non-CL (standard_training), while subscripts($_{**/*}$) indicate over the baseline CL (root_2), where * means the Student's t-test p-value is less than 0.05, and ** means p-value is less than 0.01.

| Seed | 1 | 2 | 3 | 4 | 5 | Average | SD |
|---|---|---|---|---|---|---|---|
| **baseline CL** | 0.729 | 0.712 | 0.726* | 0.710 | 0.726 | **0.721** | 0.008 |
| dynamic_38_root_2 (ours) | 0.729 | 0.705 | 0.726* | 0.716* | 0.727 | 0.720 | 0.009 |
| dynamic_1_root_2 (ours) | 0.721 | **0.714** | 0.728 | 0.703 | **0.733** | 0.720 | 0.011 |
| dynamic_34_root_2 (ours) | 0.728 | 0.703 | **0.736**$^{**}_{**}$ | 0.704 | 0.723 | 0.719 | 0.013 |
| dynamic_2_root_2 (ours) | 0.731* | 0.705 | 0.719 | **0.722**$^{**}_{**}$ | 0.716 | 0.718 | 0.008 |
| dynamic_3_root_2 (ours) | **0.732*** | 0.707 | 0.724 | 0.702 | 0.726 | 0.718 | 0.012 |
| dynamic_4_root_2 (ours) | 0.725 | 0.710 | 0.725 | 0.706 | 0.723 | 0.718 | 0.008 |
| **baseline non-CL** | 0.722 | 0.708 | 0.718 | 0.707 | 0.730 | 0.717 | 0.009 |
| dynamic_0_root_2 (ours) | 0.720 | 0.694 | 0.716 | 0.702 | 0.716 | 0.709 | 0.010 |

selecting a proper scoring file is nontrivial. Sometimes a tiny difference in setting can lead to an opposite conclusion. In other words, if the scoring file is misleading, the CL method would also be ineffective. Unfortunately, due to the limited time, we have not got the chance to run the dynamic method with a batch size of 8.

### 5.2.3 Noise Method

In this section, we discuss our experimental results of applying the noise method on both MSDialog and MANtIS, which are presented in the figure 5.11 and 5.12, respectively. We fix the corresponding best pacing function for each dataset, which is root_2 for MSDialog and root_5 for MANtIS. Before implementing the noise method introduced in chapter 4, we first tried a more straightforward method with a fixed noise of 1/2, 1/3, 1/4, and 1/8 on the MSDialog dataset. For example, if the noise is 1/2, for each iteration, we compose the sampling candidate pool by concatenating half data from the CL-scheduled data and the rest randomly from the entire dataset. As the left half of figure 5.11 shows, the simple noised root_2 outperforms the baseline non-CL but loses to the baseline CL.

Then we consider giving the training more flexibility. Thus we implement this noise method by adjusting the noise dynamically with the number of iterations as explained and visualized in chapter

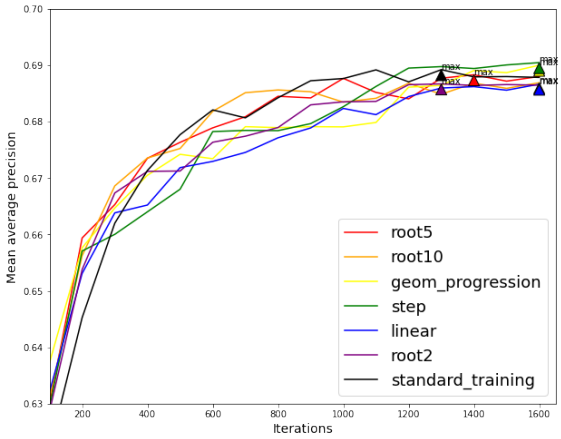


Figure 5.9: Validating MAP with the scoring file generated by BERT with a fine-tuning batch size of 64.

4. The influence of this noise method on the validating dataset can be seen in the right part of figure 5.11, where the corresponding results are denoted as **root_2_l_$\lambda$_r_$q$** with the **noise_lambda** $\lambda$ and the **noise_difficult_ratio** $q$. With the pink being the baseline CL and the light blue being the baseline non-CL, many trained with the noise method outperform them on the validating dataset. We suppose that the noise method resembles the human learning process, where sometimes students learning difficult concepts out of the school curriculum helps them perform better in school work. Also, this noise method resembles the human-like learning strategy that allows more exploration in the early learning stages and gradually back to focus on the learning goal.

We also apply the noise method on the MANtIS dataset, as shown in figure 5.12 with the same denotations. Again many trained with the noise method outperform the baseline non-CL but no explicit improvement over the its baseline CL (root_5). Besides, because the best pacing function for MANtIS is root_5, using **noise_lambda** $\lambda = 0.999$ during the training requires the noise candidate pool size to be bigger than 0.6 of the training dataset size as visualized in figure 5.10, thus **noise_lambda** $\lambda = 0.999$ and **noise_difficult_ratio** $q = 0.5$ is excluded on MANtIS.

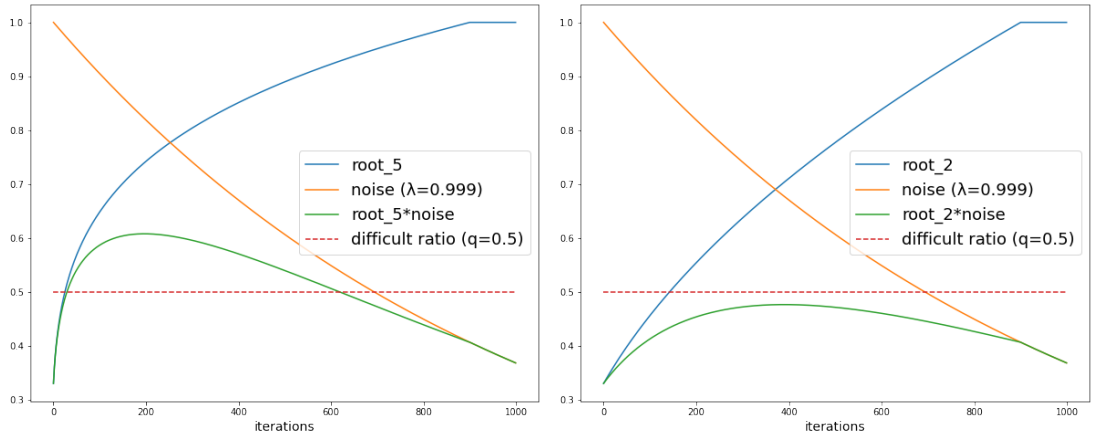


Figure 5.10: Setting with $\lambda = 0.999$ and root_5 exceeds $q = 0.5$ during the training(left), while the same setting is safe with root_2 (right).

The testing results of both are listed in the following table 5.9. For MSDialog, the number of superscripts ($^{**/*}$) indicates many of the noise method trained models outperform the baseline

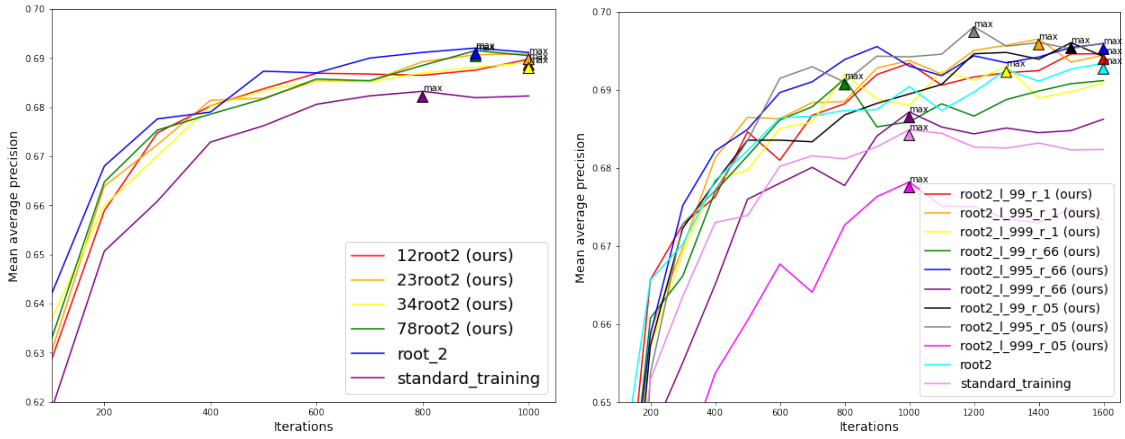


Figure 5.11: Validating MAP with the noise method on MSDialog.

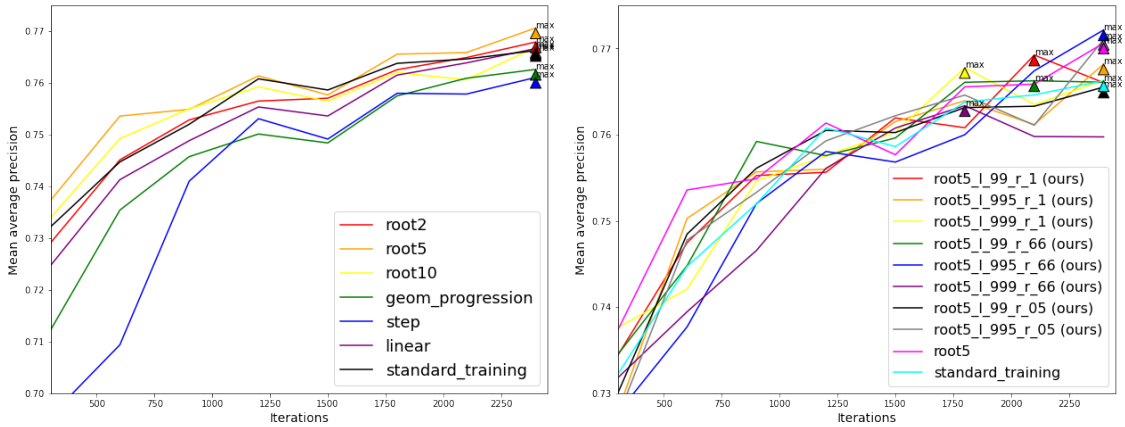


Figure 5.12: Validating MAP with the noise method on MANtIS.

non-CL statistically significantly. Although the best noise method with ($\lambda = 0.995,\ q = 0.5$) is 0.5% higher than the baseline CL on the validating dataset, it only works equally well as root_2 on the testing dataset. For MANtIS, although the validating results witness no explicit difference between the noise method and the original method, the best method with ($\lambda = 0.995,\ q = 0.66$) is 0.3% better than both baseline non-CL and root_5 on the testing dataset. Also, as the subscripts ($^{**}/_{*}$) tells, the other top noise methods also have statistically significant improvement over non-CL and root_5. Another interesting finding is that the baseline CL trained model cannot outperform the baseline non-CL on the testing dataset, which is contradictory to the results of Penha et al.[35]. We think this is because we have used an extra strategy to randomly sample a third of the validating data each time for validating while they use the entire dataset. As explained before, we use this extra strategy mainly due to the limitation of computation resources (google Colab) we have.

The difference between the results of MSDialog and MANtIS can be explained by two reasons. The first is that the validating and testing dataset are almost of the same size for MSDialog, while the testing is two times bigger than the validating dataset for MANtIS. The second potential reason is that MANtIS datasets are around four times bigger than MSDialog. To summarize, the noise CL methods have improved over the baseline non-CL and CL on the smaller MSDialog validating dataset but not on the testing dataset, whereas the situation is the opposite on the much bigger MANtIS dataset. These different observations on two different datasets indicate the unsteadiness

of our noise method. As stated before, we save the best model of the highest validating MAP to do testing. The MSDialog results indicate that the models can not generalize well on the testing. On the other hand, the noise methods have good generalization performance on the larger dataset MANtIS. We tend to believe that a larger dataset is more convincing and trustworthy than a smaller dataset. Besides, either from validating or testing results of both datasets, we can see that the noise methods with $\lambda = 0.999$ generally have poor performance. We think this $\lambda$ is excessively big that it has introduced too much noise, which is harmful to the training. Whereas for the other parameter $q$, no explicit preference can be concluded from the results.

Table 5.9: MSDialog and MANtIS testing MAP results of noise method with three runs. Superscripts($^{**/*}$) indicate statistically significant improvements over the baseline non-CL (standard_training), while subscripts($_{**/*}$) indicate over the baseline CL (root_2 for MSDialog, root_5 for MANtIS), where * means the Student's t-test p-value is less than 0.05, and ** means p-value is less than 0.01.

| **MSDialog** | | | | | |
|---|---|---|---|---|---|
| **Seed** | **1** | **2** | **3** | **Average** | **SD** |
| **baseline CL** | **0.734**$^{**}$ | 0.712$^{*}$ | 0.732$^{**}$ | **0.726** | 0.010 |
| root_2_l_995_r_05 (ours) | 0.726 | **0.714**$^{**}$ | **0.734**$^{**}$ | 0.725 | 0.008 |
| root_2_l_995_r_1 (ours) | 0.731$^{*}$ | 0.713$^{**}$ | 0.726$^{*}$ | 0.723 | 0.007 |
| root_2_l_99_r_1 (ours) | 0.729 | 0.711$^{*}$ | 0.729$^{**}$ | 0.723 | 0.008 |
| root_2_l_99_r_66 (ours) | 0.730$^{*}$ | 0.710$^{*}$ | 0.729$^{**}$ | 0.723 | 0.009 |
| root_2_l_995_r_66 (ours) | 0.731$^{*}$ | 0.711$^{*}$ | 0.726$^{*}$ | 0.723 | 0.009 |
| root_2_l_99_r_05 (ours) | 0.730$^{*}$ | 0.708 | 0.729$^{**}$ | 0.722 | 0.010 |
| root_2_l_999_r_1 (ours) | 0.724 | 0.704 | 0.734$^{**}$ | 0.721 | 0.012 |
| root_2_l_999_r_66 (ours) | 0.724 | 0.704 | 0.722 | 0.716 | 0.009 |
| **baseline non-CL** | 0.722 | 0.702 | 0.718 | 0.714 | 0.009 |
| root_2_l_999_r_05 (ours) | 0.715 | 0.700 | 0.711 | 0.709 | 0.006 |
| **MANtIS** | | | | | |
| **Seed** | **1** | **2** | **3** | **Average** | **SD** |
| root_5_l_995_r_66 (ours) | 0.720 | **0.715**$_{**}$ | **0.720**$^{**}$ | **0.719** | 0.002 |
| root_5_l_99_r_1 (ours) | 0.720 | 0.712$_{**}$ | 0.720$^{**}$ | 0.717 | 0.004 |
| root_5_l_99_r_05 (ours) | **0.721** | 0.710$_{*}$ | 0.718 | 0.716 | 0.004 |
| root_5_l_995_r_05 (ours) | 0.719 | 0.712$_{**}$ | 0.719 | 0.716 | 0.003 |
| **baseline non-CL** | 0.718 | 0.713$_{**}$ | 0.717 | 0.716 | 0.002 |
| **baseline CL** | 0.720 | 0.708 | 0.720$^{*}$ | 0.716 | 0.006 |
| root_5_l_999_r_1 (ours) | 0.719 | 0.707 | 0.717 | 0.714 | 0.005 |
| root_5_l_99_r_66 (ours) | 0.720 | 0.708 | 0.714 | 0.714 | 0.005 |
| root_5_l_995_r_1 (ours) | 0.706 | 0.713$_{**}$ | 0.717 | 0.712 | 0.004 |
| root_5_l_999_r_66 (ours) | 0.716 | 0.705 | 0.714 | 0.712 | 0.005 |

# Chapter 6

# Conclusions and Future Work

This chapter summarizes the conclusions of the experimental results presented in chapter 5, including our contributions and limitations. Finally, corresponding to the limitations, we give several directions for future work.

## 6.1 Conclusions

In this work, we explored whether and how Curriculum Learning (CL) can make the state-of-the-art neural language model BERT perform better in Information Retrieval (IR), taking two conversation response ranking (CRR) datasets MSDialog and MANtIS as examples. CL can improve model performance without requiring additional training data, which is beneficial for training, especially when a lack of training data is the bottleneck for many problems. For the CL, we focused on the vanilla CL [3], which is to design a curriculum for the training dataset. The general procedure of the vanilla CL in this work consists of two steps, the first is to sort the data from easy to difficult with a scoring file, and the second is to train the model on the ordered dataset in the speed determined by a pacing function.

### 6.1.1 Contributions

On the CL framework of Penha et al.[35], we first compared the results of some basic experiments with CL and inverse CL. We found easy samples first (CL) more beneficial than difficult ones first (inverse CL). We argue that difficult samples can confuse the model, especially at early training stages, while easy samples can smooth the objective function for the model. On the other hand, we found that CL methods fail to outperform the non-CL baseline when the scoring file is generated with a slight difference in the setting. We also implemented two more pacing functions with the s-shape curve to imitate a human-like learning speed, which accelerates at early stages and slows down later. Unfortunately, the s-shape functions failed to outperform the original pacing functions, and we think it is because it allocates excessive weights to easy examples. We conclude from the above observations that it is nontrivial to find a proper scoring file and a practical pacing function. On the other hand, once they are found, CL can outperform the baseline above 1%.

We also implemented two new methods for more comprehensive exploration: the dynamic rescoring method for the first step and the noise method for the second step. The results of the dynamic rescoring method have witnessed no performance improvement with our dynamic method over the baseline but some slight improvement over the original pacing function. More importantly, we found that scoring files generated with more fine-tuning steps are more informative and beneficial, and a comparatively lower rescoring frequency is better. The noise method works much more effectively than the dynamic rescoring method, according to the experiment results. Especially on the larger dataset MANtIS which is more convincing and trustworthy, our best noise method

has a 0.3% higher testing MAP than the baseline CL and non-CL. We assume this effectiveness comes from the noise method resembling the human learning process, where some difficult concepts picked up occasionally by students out of school curriculum can help them perform better in school work. Moreover, this noise method allows the model more exploration in the early learning stages and gradually back to focus on the learning goal. Another finding is that the amount of noise should not be excessive. Otherwise, it harms the performance.

### 6.1.2 Limitations

According to our experimental results, the CL methods are not stably effective. For example, with the MSDialog dataset, the noise CL methods show exciting improvement over baselines on the validating dataset. However, they show no difference with the baseline CL when testing. Also, statistically significant improvement of testing MAP is not seen in every run. Even the baseline CL only outperforms the baseline non-CL for two out of five seeds with a confidence interval of 99%. Another problem is that informative scoring files and suitable pacing functions are essential for CL strategies to be effective. However, it is hard to find proper scoring files and pacing functions. Also, practical experiments seem to be preferred to theoretical analysis because doing experiments is faster and more effective. Still, lacking theory support can be misleading.

We have made some compromises with the limitations of the computation resource (Google Colab), which can influence the results. For example, for validating on the large dataset MANtIS, we randomly sample a third instead of the complete validating dataset for computing validating MAP each time. This extra validating sampling strategy can lead to different observations from MSDialog and MANtIS because the validating and testing size are almost the same for the former, while the validating size is only a third of the testing size for the latter.

Another limitation is that the reasons for performance improvement by CL strategies over baseline non-CL are not analyzed clearly. We mainly have summarized the experimental observations and tried explaining the results by proposing hypotheses. However, the details of what the models have learned remain unknown.

## 6.2 Future Work

We leave the above-stated remaining limitations to the following future work. Besides, we are also interested in the direction of dynamic sampling as our dynamic rescoring method has no exciting performance over the baselines.

- Finding a good way to measure the difficulty of data is still one of the biggest problems for applying curriculum learning in many fields. In this work, we followed the best scoring method from Penha et al.[35] by first pairing each training conversation with one relevant response and one irrelevant response, and then using the difference of BERT's predictions of them as the difficulty score. Nevertheless, this scoring method has made the training dataset different from the validating or testing dataset. Because the training data has an even number of relevant and irrelevant responses, while the validating or testing data has eight times more irrelevant responses than relevant responses. This data distribution difference can affect the model performance when validating and testing. So we could utilize all the nine irrelevant responses to generate the scoring file to counteract this data distribution difference.
- The vanilla CL method usually pre-defines a curriculum that is fixed and static during the training process. We have proposed the dynamic rescoring method to let the model re-evaluate the difficulty with newly learned knowledge. Unfortunately, our method has no exciting improvement over baselines. As there are other related learning methods with dynamic sampling strategies, for example, self-paced learning [27][58] and active learning [9]. We can explore more in this direction.

- Finally, a more thorough analysis of the differences between models trained with CL and non-CL is worth investigating. We have seen improvement in the validating and testing MAPs from CL methods to baseline non-CL, yet we are unaware of what happens in the inner side of the model. For example, we could analyze the knowledge learned by each model layer and investigate where the model has put most of its attention on. More analyses and explanations can give more instructions to the subsequent work.

# Chapter 7

# Acknowledgement

I feel thrilled to have finished my graduation project. Meanwhile, I appreciate all the help and support from those critical people and the Eindhoven University of Technology. First of all, I would like to thank my supervisor for his supervision, including sharing ideas, giving suggestions, and discussing with me in the weekly meetings. Also, I appreciate the suggestions and help on my thesis and final presentation from the graduation committee. Then I have to say thanks to my university TU/e. During these two years, I have learned much new knowledge and picked up many skills by completing those practical works one by one. I love the library, which is a perfect place for studying. Besides, I also love the student sports center, where I enjoy my free time with various exciting sports (especially climbing) and become more energetic to study. Very importantly, I would like to thank my friends and family sincerely because, without their support, it is impossible for me to start and enjoy the two years of doing the master's degree that I treasure so much. I feel so lucky to get a lot of helpful feedback on the practice presentation from my friends. Finally, I want to thank myself for all my effort along the way.

# Bibliography


[1] Eugene L. Allgower and Kurt Georg. *Introduction to Numerical Continuation Methods*. Society for Industrial and Applied Mathematics. 8

[2] Y. Bengio. Neural net language models. *Scholarpedia*, 3(1):3881, 2008. revision #140963. 5

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8. ACM Press. 2, 8, 9, 10, 11, 12, 18, 39

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 12

[5] Antoine Caubrière, Natalia Tomashenko, Antoine Laurent, Emmanuel Morin, Nathalie Camelin, and Yannick Estève. Curriculum-based transfer learning for an effective end-to-end spoken language understanding and domain portability. 2, 12, 18

[6] Xinlei Chen and Abhinav Gupta. Webly supervised learning of convolutional networks. 2

[7] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D. Davison. Table search using a deep contextualized language model. pages 589–598. 2, 17, 27

[8] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. 12

[9] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. 40

[10] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. 2, 17, 27

[11] James H. Martin Dan Jurafsky. Speech and language processing. 4

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. 2, 12, 17, 27

[13] Jeffrey L. Elman. Learning and development in neural networks: the importance of starting small. 48(1):71–99. 2, 9

[14] Meng Fang, Tianyi Zhou, Yali Du, Lei Han, and Zhengyou Zhang. Curriculum-guided hindsight experience replay. page 12. 3, 19, 21

[15] Nicola Ferro, Claudio Lucchese, Maria Maistro, and Raffaele Perego. Continuation methods and curriculum learning for learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1523–1526. ACM. 12

[16] Tommaso Furlanello, Zachary C. Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. 14

[17] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. pages 55–64. 17

[18] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management - CIKM '13*, pages 2333–2338. ACM Press. 17

[19] Nitin Indurkhya and Fred J Damerau. Natural language processing. page 676. 3

[20] 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. 2

[21] Prateek Jain and Purushottam Kar. Non-convex optimization for machine learning. 7, 8

[22] Tae-Hoon Kim and Jonghyun Choi. ScreenerNet: Learning self-paced curriculum for deep neural networks. 11

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 28

[24] Tom Kocmi and Ondrej Bojar. Curriculum learning and minibatch bucketing in neural machine translation. pages 379–386. 2, 12, 18

[25] Kai A. Krueger and Peter Dayan. Flexible shaping: How learning in small steps helps. 110(3):380–394. 2, 9

[26] Gaurav Kumar, George Foster, Colin Cherry, and Maxim Krikun. Reinforcement learning based curriculum optimization for neural machine translation. In *Proceedings of the 2019 Conference of the North*, pages 2054–2061. Association for Computational Linguistics. 12

[27] M Pawan Kumar, Ben Packer, and Daphne Koller. Self-paced learning for latent variable models. page 9. 40

[28] Cao Liu, Shizhu He, Kang Liu, and Jun Zhao. Curriculum learning for natural answer generation. page 7. 18

[29] Christopher Manning, Prabhakar Raghavan, and Hinrich Schuetze. Introduction to information retrieval. page 581. 5

[30] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. 17

[31] Pietro Morerio, Jacopo Cavazza, Riccardo Volpi, Rene Vidal, and Vittorio Murino. Curriculum dropout. 11

[32] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. 2, 17, 27

[33] Gustavo Penha. MANtIS - a multi-domain information seeking dialogues dataset. 27

[34] Gustavo Penha, Alexandru Balan, and Claudia Hauff. Introducing MANtIS: a novel multi-domain information seeking dialogues dataset. 25

[35] Gustavo Penha and Claudia Hauff. Curriculum learning strategies for IR: An empirical study on conversation response ranking. 2, 12, 13, 14, 15, 18, 20, 28, 30, 37, 39, 40

[36] Gustavo Penha and Claudia Hauff. On the calibration and uncertainty of neural learning to rank models for conversational search. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 160–170. Association for Computational Linguistics. 26, 28

[37] Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M. Mitchell. Competence-based curriculum learning for neural machine translation. 18

[38] C. Qu, L. Yang, W. B. Croft, J. Trippas, Y. Zhang, and M. Qiu. Analyzing and characterizing user intent in information-seeking conversations. In *SIGIR '18*, 2018. 24

[39] C. Qu, L. Yang, W. B. Croft, Y. Zhang, J. Trippas, and M. Qiu. User intent prediction in information-seeking conversations. In *CHIIR '19*, 2019. 24

[40] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In Bruce W. Croft and C. J. van Rijsbergen, editors, *SIGIR '94*, pages 232–241. Springer London. 17, 28

[41] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. 3(4):333–389. 14

[42] Douglas L.T Rohde and David C Plaut. Language acquisition in the absence of explicit negative evidence: how important is starting small? 72(1):67–109. 2, 9

[43] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 373–374. ACM. 17

[44] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. 10, 11

[45] Jonathan Talmor, Alon, Berant. MultiQA: An empirical investigation of generalization and transfer in reading comprehension. 12

[46] Ren Jie Tan. Breaking down mean average precision (mAP). 29

[47] Yi Tay, Shuohang Wang, Luu Anh Tuan, Jie Fu, Minh C. Phan, Xingdi Yuan, Jinfeng Rao, Siu Cheung Hui, and Aston Zhang. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. 18

[48] Mohamed Trabelsi, Zhiyu Chen, Brian D. Davison, and Jeff Heflin. Neural ranking models for document retrieval. 17

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. NeurIPS 2017 conf. 5, 6, 7, 17, 27

[50] Chengyi Wang, Yu Wu, Shujie Liu, Ming Zhou, and Zhenglu Yang. Curriculum pre-training for end-to-end speech translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3728–3738. Association for Computational Linguistics. 12, 18

[51] Wikipedia contributors. Natural language processing. Page Version ID: 1037947074. 3

[52] Wikipedia contributors. Okapi BM25. Page Version ID: 1008742667. 28

[53] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. 28

[54] Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Lai Jian-Huang, and Tie-Yan Liu. Learning to teach with dynamic loss functions. page 12. 31

[55] 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*, pages 6095–6104. Association for Computational Linguistics. 2, 18

[56] L. Yang, M. Qiu, C. Qu, J. Guo, Y. Zhang, W. B. Croft, J. Huang, and H. Chen. Response ranking with deep matching networks and external knowledge in information-seeking conversation systems. In *SIGIR '18*, 2018. 24

[57] 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. 12, 18

[58] Qian Zhao, Deyu Meng, Lu Jiang, Qi Xie, Zongben Xu, and Alexander G Hauptmann. Self-paced learning for matrix factorization. page 7. 40

[59] Siqi Zheng, Gang Liu, Hongbin Suo, and Yun Lei. Autoencoder-based semi-supervised curriculum learning for out-of-domain speaker verification. In *Interspeech 2019*, pages 4360–4364. ISCA. 12, 18

[60] Yikai Zhou, Baosong Yang, Derek F. Wong, Yu Wan, and Lidia S. Chao. Uncertainty-aware curriculum learning for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6934–6944. Association for Computational Linguistics. 18