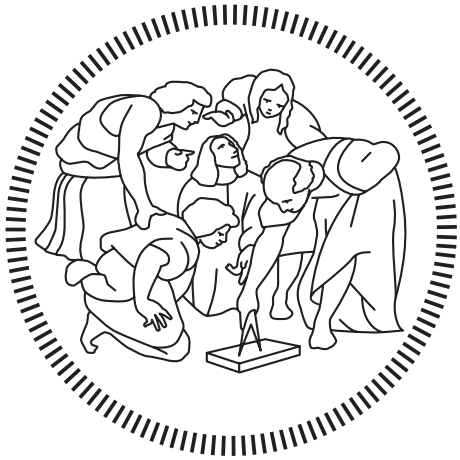
Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering
Department of Electronics, Information and Bioengineering



Transformers for Question Difficulty Estimation from Text

Supervisor
Prof. Paolo Cremonesi

Co-Supervisor
Dott. Ing. Luca Benedetto

Candidate
Giovanni Aradelli – 920885

Academic Year 2019 – 2020

Sommario

La calibrazione delle domande, ovvero la stima della loro difficoltà, è una componente molto importante dell'educazione. Infatti, il livello di conoscenza degli studenti può essere stimato dalla correttezza delle loro risposte alle domande dell'esame e dalla loro difficoltà. Una stima accurata della difficoltà delle domande può anche essere sfruttata per fornire agli studenti esercizi adatti al loro livello di abilità. Gli approcci tradizionali alla calibrazione delle domande sono la calibrazione manuale e il pre-test. Nella calibrazione manuale, uno o più esperti assegnano a ciascuna domanda un valore numerico che ne rappresenta la difficoltà, e questo è intrinsecamente soggettivo. Nel pre-test, le domande vengono somministrate agli studenti in un vero esame e successivamente la difficoltà è stimata a partire dalla correttezza delle loro risposte. Il pre-test introduce un lungo ritardo tra il momento della generazione della domanda e il momento in cui può essere utilizzata per valutare gli studenti. Ricerche recenti hanno cercato di risolvere questo problema stimando la difficoltà delle domande usando solo le loro informazioni testuali, sfruttando tecniche di *Natural Language Processing (NLP)* come modelli neurali o *bag of words*. L'idea alla base di ciò è ridurre (o eliminare) la necessità di calibrazione manuale e di pre-test, stimando la difficoltà delle domande dal loro testo, che è immediatamente disponibile dopo la creazione della domanda. I modelli linguistici pre-addestrati, in particolare i Transformers, hanno portato a notevoli miglioramenti in diverse aree del NLP, ma finora nessuno studio ha esplorato il loro utilizzo per la calibrazione delle domande. In questo lavoro, eseguiamo uno studio su come i modelli Transformers (in particolare, BERT e DistilBERT) si confrontano con lo stato dell'arte attuale nella stima della difficoltà dal testo e proponiamo un modello che è in grado di migliorare il precedente stato dell'arte. Il nostro modello è addestrato utilizzando il testo e la difficoltà delle domande, ma può opzionalmente sfruttare un corpus aggiuntivo di documenti per migliorare le prestazioni. Test effettuati su due diversi set di dati, uno pubblico e uno privato, mostrano che il nostro modello riduce la radice del valore quadratico medio (in inglese *Root Mean Square Error*, RMSE) degli studi precedenti fino al 6,5% e conferma la nostra intuizione sull'efficacia dei modelli basati su Transformers per stimare la difficoltà dal testo delle domande. Inoltre, analizziamo quali caratteristiche delle domande possono influire sull'errore di predizione del modello.

Abstract

Question Difficulty Estimation (QDE), a process which is also referred to as question calibration, is a very important task in education. Indeed, the knowledge level of students, also called skill, can be estimated from the correctness of their answers to exam questions and their difficulty. An accurate estimation of question difficulty can also be leveraged to provide students with exercises suitable for their skill level. Conventional approaches to question calibration are manual calibration and pretesting. In manual calibration, one or more domain experts assign to each question a numerical value representing the difficulty, and this is intrinsically subjective. In pretesting, questions are administered to students in a real test scenario, and then the difficulty is estimated from the correctness of their answers. Pretesting introduces a long delay between the time of question generation and when the question can be used to score students. Recent research tried to overcome this issue by estimating the difficulty of questions using only their textual information, exploiting Natural Language Processing (NLP) techniques such as neural models or bag of words. The idea behind this is to reduce (or eliminate) the need for manual calibration and pretesting by estimating the difficulty of questions from their text, which is immediately available at the moment of question creation. Pre-trained language models, especially Transformers, have led to impressive gains on several NLP tasks, but no previous work has explored their use for question calibration. In this work, we perform a study of how Transformer models (specifically, BERT and DistilBERT) compare with the current state of the art in the task of QDE from text, and propose a model which is capable of outperforming previous research. Our model is trained on the text of questions and their difficulty, but can optionally take advantage of an additional corpus of domain-related documents to improve performance. Tests on two different real-world datasets, one public and one private, show that our model reduces the Root Mean Square Error (RMSE) of previous baselines by up to 6.5% and confirms our intuition about the effectiveness of Transformer-based models for QDE from text. Furthermore, we carry out an analysis on which characteristics of the questions (such as length of the text and presence of numbers) can influence the prediction error.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The task of modeling student knowledge over time is defined as Knowledge Tracing (KT). An accurate estimation of students' skill levels can be leveraged by teachers to provide tailored content and understand if there are students struggling and, therefore, in need of further support. Also, it can be leveraged to perform personalized exercise recommendations. The growth of online education platforms has led to the availability of a large number of exercises. Rather than wasting their time on exercises that are too easy or too hard, students could practice more with exercises appropriate to their skill level. Student's skill is usually estimated using the correctness of their answers and the difficulty of each question they answered to. Questions, also called items, need to be calibrated before being used in a real exam. Calibration consists in estimating some latent (i.e., non-observable) characteristics of questions, such as their difficulty. Question Difficulty Estimation (QDE) is essential to give students questions that are not too difficult or too easy so that they can accurately identify the student's skill level. All examinees would answer wrongly to an item that is too difficult, giving no information about their knowledge. The same happens with too easy items, which would be answered correctly by everyone regardless of their skill level.

The standard methodologies used to estimate the difficulty of newly created questions are manual calibration and pre-testing with real students. In manual calibration, it is necessary the intervention of an expert who manually selects numerical values representing the difficulty of the questions; thus, the evaluation may be biased and is intrinsically subjective [31]. Also, it is time-consuming and non-scalable. In pre-testing, questions are administered to a group of students in a real test scenario. The difficulty of the questions under pre-testing is then estimated from the correctness of students' answers. Questions under pre-testing are not used for scoring, and they should be indistinguishable from the others. This method leads to an accurate and reliable estimation but introduces a long delay before the newly generated questions can be used in a real exam.

In the literature, there is a recent interest in developing a model able to automatically perform QDE from text. The growth of this interest is due to the fact that such a model might eliminate or at least reduce the need for pre-testing and manual labeling. Automatic difficulty assessment can support the creation of new questions, especially with the increasing availability of algorithms for automatic question generation, helping in real-time to discard too easy or too difficult ones. Recent models for QDE from text explored the use of Natural Language Processing (NLP) techniques

such as TF-IDF [6] or ELMo embeddings [72], but none of them makes use of the latest NLP approaches (e.g., Transformer models). Moreover, most previous models are trained on questions' text, but some models also require additional resources (such as lecture notes) to extract other useful information. This limits the utilization in those contexts where such additional resources are not available.

We explored the extent to which using Transformers language models can be beneficial for QDE from text and propose a model that outperforms previous studies. Our research focuses on two pre-trained models: BERT and DistilBERT. We fine-tuned the two models on the task of difficulty estimation from text, using only the text of the questions, but also experimented with the possibility of leveraging an additional corpus of domain-specific texts to improve their performance. We experimented on two different datasets, one private from Cloud Academy[1] and one public from ASSISTments[2]. Results show that the proposed model outperforms all previous baselines. Our model reduces the Root Mean Square Error (RMSE) by up to 4.7% with respect to previous approaches when trained only on questions' text. The improvement is even more significant, up to 6.5%, if our model is further pre-trained on an additional corpus of text related to the same domain of the questions. Furthermore, we propose an analysis of which characteristics of the question may influence the model performance. The code is publicly available for future research[3].

The rest of this document is organized as follows:

- **Chapter 2** Background, it provides fundamental theoretical notions.
- **Chapter 3** Related Works, it collects various research works present in the literature related to our work.
- **Chapter 4** Models, it describes our approach to perform Question Difficulty Estimation (QDE), giving an high-level view of the model architecture and training.
- **Chapter 5** Experimental Datasets, it presents the datasets and the pre-processing.
- **Chapter 6** Experimental Setup, it provides information about the setup used to perform the various experiments.
- **Chapter 7** Results, it shows and analyses the results of the experiments performed on the datasets.
- **Chapter 8** Conclusion, it gives the final considerations about the experiments and proposes a direction for future works.

[1] `https://cloudacademy.com/`
[2] `https://new.assistments.org/`
[3] `https://github.com/aradelli/transformers-for-qde`

Chapter 2

Background

This chapter has the purpose of establishing some common ground by introducing the theoretical foundations used for the development of the proposed model. Section 2.1 introduces the concept of difficulty of a question, presenting two popular psychometry frameworks. Then, Section 2.2 introduces a classic approach to text mining and Section 2.3 presents the most modern NLP techniques based on Transformers models.

2.1 Questions Calibration

2.1.1 Classical Test Theory

Classical Test Theory (CTT) is a theory that predicts outcomes of psychological testing, such as the difficulty of items or the ability of test-takers. The goal of CTT, also called "true score theory", is to improve the reliability and validity of tests. The term "classical" refers to the time when this framework was developed, and it is also in contrast with modern psychometric theories such as Item Response Theory. CTT has been the most used in the last century and has the advantage of being simple to compute and understand compared to Item Response Theory (IRT).

Classical Test Theory assumes that each individual has associated a true score T, which would be possible to observe if there was no error in the estimate. For a person taking a test, his true score is the expected value of the observed scores over an infinitely long run of repeated independent administrations of the same test [7]. Doing an infinite number of observations is impossible; therefore, the observed score X is used. The observed score X is made up of the sum of the true score T and an Error E; errors are assumed to be normally distributed with mean equal to zero. The (linear) model describing CTT is the following:

$$X = T + E \tag{2.1}$$

Where T and E are two unobservable (or latent) variables.
The major assumptions [24] underline the CTT are:

- T, E are not correlated.
- E is normally distributed with zero mean.
- The errors of different tests are not correlated.

A reliability coefficient can provide an estimate of the level of concordance between observed and true scores. The reliability of test scores ρ^2_{XT} is defined as follows:

$$\rho^2_{XT} = \frac{\sigma^2_T}{\sigma^2_X} = \frac{\sigma^2_T}{\sigma^2_T + \sigma^2_E} \tag{2.2}$$

Where σ^2_T is the true score variance, σ^2_X is the observed score variance, and σ^2_E is the error variance.

The reliability is zero when all variation in the observed scores is due to measurement error. The maximum value (i.e., $\rho^2_{XT} = 1$) is when there is no measurement error. However, the true scores of test-takers are not observable, thus the reliability is estimated indirectly using parallel-tests. Two tests are considered parallel if they have the same observed variance in the population of examinees, and each student has the same true score on both tests.

The concept of difficulty of an item in CTT is expressed by the *p-value*. The p refers to probability and is the fraction of correct responses in the considered population. Usually, the *p-value* is typically referred to as item difficulty or *correctness*. It should be noted that the higher the *p-value*, the easier the item is. In the same way, we can also define the *wrongness* as $1-$*p-value*.

The property of an item to discriminate between high ability examinees and low ability examinees is called *discrimination*. If an item is dichotomously scored (i.e., the score is 0 or 1), the discrimination estimate is computed as a point-biserial correlation.

CTT has several shortcomings that have led to the development of other models. The major limitation of CTT can be summarized as a circular dependency: (a) the person statistic is (item) sample dependent, and (b) the item statistics are (examinee) sample dependent [18]. On the other hand, CTT has weak theoretical assumptions, which make it easy to apply in many testing situations.

2.1.2 Item Response Theory

Item Response Theory (IRT)—also named latent traits theory—is a family of models used to perform abilities assessments. Important educational tests, such as the Graduate Management Admission Test (GMAT), use IRT to improve measurement accuracy and reliability [51]. However, IRT is not used only in the educational context: the latent trait can be a behavioral characteristic, such as customer satisfaction or others. This theory was developed around the mid-1900s but was not widely used until the end of the century. A property that has made IRT widely used is the

invariance property: items' latent traits do not depend on the ability distribution of test-takers [25].

In IRT, each student is supposed to have associated one or more latent traits, also called skills or abilities. The same is true for questions (also called items) that have one or more latent traits associated with them.

IRT models can be unidimensional or multidimensional (MIRT): in the first case, each student is modeled as having only one skill, while the second one introduces multiple skills referring to different topics (e.g., mathematics and science) and multiple difficulties for each question.

IRT models can be grouped in three main categories depending in the number of questions' parameters which are defined:

- One-Parameter Logistic (1PL), also known as Rasch Model [50], which defines a difficulty for each item.
- Two-Parameter Logistic (2PL), which defines a difficulty and a discrimination for each item.
- Three-Parameter Logistic (3PL), which defines a difficulty, a discrimination, and a guess factor for each item.

Different IRT models can be classified based on the score that can be assigned to students' answers. In the case the score of an item is only correct or wrong, i.e. item dichotomous, we have Dichotomous IRT. These models are very common since they can model Multiple Choice Questions (MCQ) or true-false questions.

Given a question i with difficulty b_i, discrimination a_i and a guess factor c_i, IRT defines the Item Response Function (IRF) (also know as Item Characteristic Curve (ICC)), which represents the probability that a student with a skill level theta (θ) correctly answers the question. The formula of the item response function is as follows:

$$P_i(\theta) = c_i + (1 - c_i)\frac{1}{1 + e^{a_i(\theta - b_i)}} \tag{2.3}$$

The difficulty parameter affects the location of the logistic curve along the horizontal axis: as the difficulty increases, a student with a given skill level is less likely to answer the question correctly. Figure 2.1 shows three items with different difficulty.

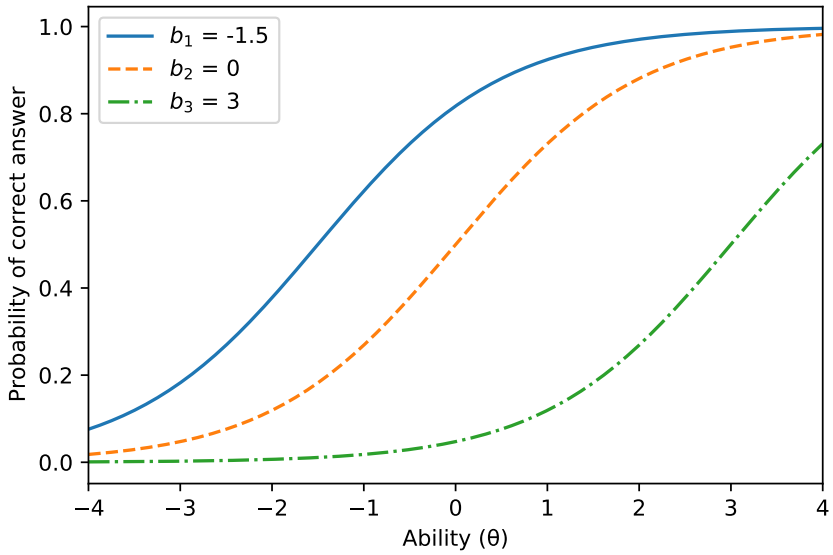


Figure 2.1. Effects of the difficulty on the Item Response Function (IRF).

The discrimination affects the slope of the logistic curve: as the discrimination increases, the steepness increases. Figure 2.2 shows three items with different discrimination. The question with discrimination $a_3 = 3$, manages to discriminate with good precision a student with an ability level (θ) lower than -1 or greater than 1, while the question with discrimination $a_1 = 0.5$ does not provide much information because students with different skill levels have a similar probability of correctly answering the question. A question with a large discrimination value can differentiate better between different skills level of students, and this is the reason why the discrimination can be considered as a proxy of the "quality" of a question.

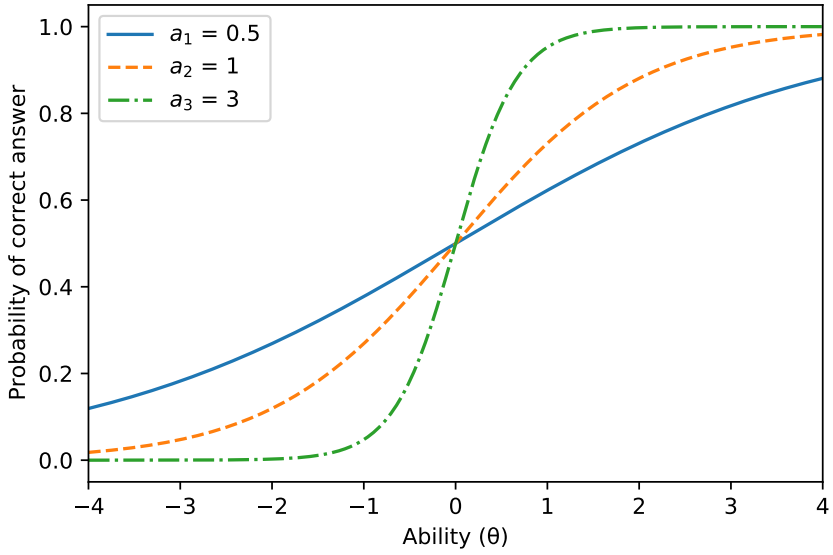


Figure 2.2. Effects of the discrimination on the Item Response Function (IRF).

The guess factor takes into consideration the fact that an examinee could pick the correct answer by chance, even without having the required knowledge level. An example where this may happen is in Multiple Choice Questions (MCQ).

Parameter estimation methods

In the literature, we can find four techniques for the estimation of item response models: joint maximum likelihood, conditional maximum likelihood, marginal maximum likelihood, and Bayesian estimation with Markov chain Monte Carlo. All of them rely heavily on the assumption that individuals are independent from each other and that the item responses of a given individual are independent, given that an individual's skill level [33]. The applicability of the aforementioned methods depends on the number of parameters to fit and whether we need to estimate both students' and items' latent traits or only one of the two.

It is important to point out that the quality of the estimated parameters, regardless of the method, is influenced by two factors [52]: i) test length (i.e., how many items are in the test); ii) sample size (i.e., how many different interactions for each item).

2.2 Traditional approaches to Natural Language Processing

Natural Language Processing (NLP) is a field of artificial intelligence that deals with the interaction between computers and humans using natural language. It is an interdisciplinary field that began around 1950, combining linguistic, computer science, and artificial intelligence. With the advent of big data, we have seen an exponential increase of data in the form of text and the need of processing them in an automated way. Dealing with language is very tricky: it is inherently ambiguous, words can assume various meanings depending on the context, and they can acquire new meanings over time; these and many other things make it a challenging task. Tasks in NLP frequently involve speech recognition, natural language understanding, and natural language generation.

2.2.1 Text Preprocessing

Text preprocessing is traditionally a necessary step for NLP tasks. It changes text into a more digestible form, below are some of the most common steps of this phase [62].

Tokenization

Tokenization consists in splitting a text into words, phrases, or other meaningful parts, namely "tokens". Tokens can be either words, characters, or subwords. Typically, the segmentation is carried out considering only alphabetic or alphanumeric characters that are delimited by non-alphanumeric characters (e.g., punctuations, whitespace).

As a simple example of word-level tokenization, we can consider the sentence:

This is an example of tokenization

The result is a list of tokens, where each token is a word:

[*This*, *is*, *an*, *example*, *of*, *tokenization*]

While it is the most straightforward way to separate texts in smaller chunks, it can cause problems when you have a vast corpus: it usually yields a huge vocabulary (the set of all unique tokens used), especially in morphologically rich languages. Another issue in tokenization is also how to deal with unknown words; usually, a special token is reserved for these words.

A better way to perform tokenization is Byte Pair Encoding (BPE) [55]. The sentence of before tokenized using BPE:

[*This*, *is*, *an*, *example*, *of*, *token*, *ization*]

WordPiece [54] is another subword tokenization algorithm very similar to BPE.

Stop word removal

Stop words are common words (e.g., prepositions, articles) that do not contribute much from a semantic perspective to the content or meaning of a document; for this reason, they should be eliminated from a text. Moreover, removing stop words reduces the dimensionality of the input space. The most frequent words in text documents are articles, prepositions, and pronouns that do not give the meaning of the documents. Very frequent words in the considered corpus, called corpus specific words, can be eliminated. These words are not necessarily very common in the language under consideration.

Stemming

This method is used to identify the root of a word. The purpose of this method is to remove various suffixes, to reduce the number of words, to have accurately matching stems, to save time and memory space. It is used as an approximate way for grouping words with a similar basic meaning together. For example, the words correlate, correlated, correlating, correlations, all can be stemmed to the stem "correlate". An algorithm that performs this step is called stemmer. One common implementation is the Porter stemmer [66].

Others

Other steps that can be done are converting all letters to lower cases, converting numbers into words or removing them, removing punctuations, removing white spaces, and expanding abbreviations. All this is done to prepare the text in a better manageable format for the machine, also reducing the dimensionality.

2.2.2 Encoding Text

Once the text has been tokenized, the tokens obtained must, however, be transformed into numbers to be feed as input to machine learning models. A naive way to do this is to assign a different number for each token. For example, using this approach on the sentence "encoding text is a serious thing", the encoding would be:

$$[\textit{encoding}, \textit{text}, \textit{is}, \textit{a}, \textit{serious}, \textit{thing}\ ] => [5, 2, 3, 1, 6]$$

Another very similar approach is to use a one-hot encoded representation for all the words in our vocabulary.

$$\begin{array}{lll} \textit{encoding} & => & [1, 0, 0, 0, 0, 0] \\ \textit{text} & => & [0, 1, 0, 0, 0, 0] \\ \textit{is} & => & [0, 0, 1, 0, 0, 0] \\ \textit{a} & => & [0, 0, 0, 1, 0, 0] \\ \textit{serious} & => & [0, 0, 0, 0, 1, 0] \\ \textit{thing} & => & [0, 0, 0, 0, 0, 1] \end{array}$$

As we can observe, this representation is sparse since most of the elements are zero. The two techniques presented before are not good encoders for several reasons: they do not capture meaning, semantic relationships, similarities between words, and do not take into consideration the context in which a word appears.

Word embedding is the common name for a set of language modeling and feature learning techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers. Embedding refers to any technique mapping a word (or phrase) from its original high-dimensional input space (the body of all words) to a lower-dimensional numerical vector space. The approaches used for mapping can be divided into two groups:

1. Frequency-based Embedding;
2. Prediction-based Embedding.

In the first category, there are techniques that exploit the frequency of a word's occurrence to extract embeddings. One of these is called TF-IDF, which will be covered in more detail later.

The second includes various techniques, for the most part, probabilistic models and neural-based models. They learn embeddings by predicting a word based on the words present in the context. One of the most relevant neural-based embedding technique is Word2Vec, developed by a team of researchers at Google [42] in 2013 and has become the de-facto standard for developing pre-trained word embedding.

2.2.3 Term Frequency–Inverse Document Frequency (TF-IDF)

TF-IDF is a statistical measure that is meant to indicate how relevant a word is to a document in a collection or corpus. It is used as a weighting factor in searches of information retrieval and text mining applications. The TF-IDF value increases proportionally to the number of times a word appears in the document and is counterbalanced by the number of documents in the corpus that contain the considered word, which helps to compensate for the fact that some words appear more frequently in general (such as stop words). TF-IDF is calculated with the following formula:

$$TF - IDF(t, d, D) = tf(t, d) \times idf(t, D) \tag{2.4}$$

Where t indicates the terms, d indicates each document, and D indicates the collection of documents.

The first term of the formula 2.4, $tf(t,d)$, is called term frequency and represents the number of times each word appears in each document. The second part $idf(t,D)$ is called inverse document frequency and represents how frequent or unfrequent a word is in the entire document set: The closer it is to zero, the more common a word is. It is calculated as indicated below:

$$idf(t, D) = log \frac{\mid D \mid}{1 + \mid \{d \in D : t \in d\} \mid} \tag{2.5}$$

Using TF-IDF, we can extract features from text and use them for a task as classification or regression. However, the dimensionality (size of the feature set) in TF-IDF for textual data is the size of the vocabulary across the entire dataset, leading to a huge computation on weighting all these terms [78]. Usually, not all the features are used, but it is built a vocabulary that only considers the top features ordered by term frequency across the corpus.

2.2.4 Machine Learning Models

Once the text has been transformed into vectors, these can be used for statistical analysis or as input to machine learning models. Machine learning is used in NLP in both unsupervised and supervised ways. An example of an unsupervised application is Latent Dirichlet Allocation (LDA) topic modeling, whose goal is to identify topics in a set of documents. Supervised learning is widely applied in several tasks, such as document classification and regression, using different models, including neural network, random forest, and Support-Vector Machines (SVM). Random forest is a supervised learning algorithm that can perform both classification and regression. It was proposed in 1995 [29], and in the following years, it has been improved in several ways. It is composed of an ensemble of decision trees. The prediction of a decision tree follows several branches of "if-then" decision splits similar to the branches of a tree. The endpoint is called a leaf, and it represents the final result: a predicted class (classification) or a value (regression). At each branch, the feature threshold that best split the samples locally is found. A single decision tree usually tends to overfit as its depth increases; random forests reduce the variance and bias by combining various decision tree models. Random forests train several decision trees on various subsamples of the dataset and various subsamples of the available features.

The principal hyperparameters in random forests are:

- The number of decision trees in the forest usually called estimators.
- The max depth of a tree: as the longest path between the root node and the leaf node.
- The minimum number of samples required to split each node.
- The minimum number of samples required for each leaf.
- The number of features to consider when looking for the best split.

2.3 Deep Learning in Natural Language Processing

The classical machine learning pipeline is typically done using a set of hand-crafted features that are then fed to a trainable model. A priori knowledge of the data is required to decide how and which features to extract. One of this approach's issues is that the selection of the features may require the need for an expert or may be difficult to perform and hence suboptimal. Moreover, these extracted features are usually applicable only to one specific problem and not transferable. Deep Learning is a category of machine learning algorithms in which the feature extractor is trained too, to learn the best representation of the data so that the features are tailored to the specific problem it is required to solve. It uses multiple hidden layers of neurons to extract low, mid, and high-level features. Deep Learning assumes that it is possible to learn a hierarchy of descriptors with increasing abstraction. Neural-based models, especially deep learning ones, have achieved superior results on various language-related tasks as compared to traditional machine learning models.

2.3.1 Feed Forward Neural Networks

Artificial neural networks are systems inspired by the biological neurons which allow humans to learn. The brain's computational model is distributed among simple units called neurons; it is intrinsically parallel and redundant and thus fault-tolerant. The neuron has a main connection with the rest of the network, called the axon, and other minor connections called dendrites. Neurons are connected through the synapses, which are the terminations of the dendrites, and neurons exchange charge through them. The impact on the receiver is different between synapses, so they have different weights. A neuron cumulates charges, and when it exceeds a threshold level for the membrane potential, it gets released through the axon (firing). Hence, the computation is a highly nonlinear phenomenon because the neuron is almost inactive until it suddenly spikes.

The basic component of an artificial neural network, called perceptron, was created in 1958 by Frank Rosenblatt. The perceptron is a binary linear classifier which applies a threshold function on the linear combination of the input features. Figure 2.3 describes the components of a perceptron: inputs $X_n = [x_1, x_2, ..., x_n]$, weights $W_n = [w_1, w_2, ..., w_n]$, the bias b (the threshold) and the activation function f.

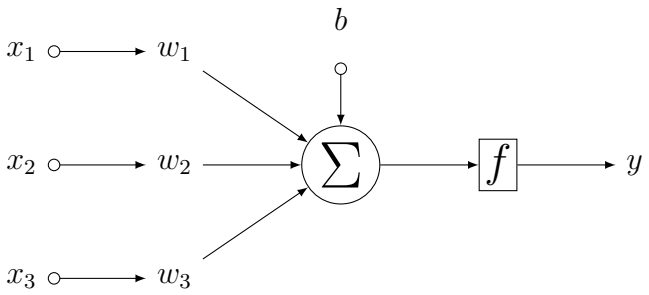


Figure 2.3. Perceptron.

The function describing the percepetron is the following:

$$y = f(\sum_{i=1}^{n} w_i x_i - b) = f(\sum_{i=0}^{n} w_i x_i) \tag{2.6}$$

This formula describes the output of the perceptron; the bias b can be inglobed into the summation, also becoming a parameter to be learned.

The training process of a single perceptron occurs through hebbian learning, introduced by Donald Hebb in his 1949 book *The Organization of Behavior* [28]. Hebbian learning is summarized in the following rules:

$$w_i^{k+1} = w_i^k + \Delta w_i \tag{2.7}$$

$$\Delta w_i^k = \eta t^k x_i^k \tag{2.8}$$

Where:

- η: learning rate
- x_i: i^{th} input at time k
- t^k: desidered output at time k

It is easily demonstrated that a single perceptron cannot learn a nonlinear separable space; for example, it cannot learn a XOR function. Hence the idea of combining several perceptrons together.

The combination of perceptrons in connected layers resulted in a Multi Layer Perceptron (MLP), which belongs to the class of Feed Forward Neural Network (FFNN). In FFNN, the flow of information moves only forward from the input nodes to the output nodes. There are no cycles or loops in the network. A general architecture consists of at least three layers of parallel neurons: an input layer, a hidden layer with a nonlinear activation function at each neuron, and an output layer. The dimensionality of the input and output layer depends on the problem, while the shape of the hidden layer, in terms of the number of neurons per hidden layer and the number of hidden layers, is a design parameter.

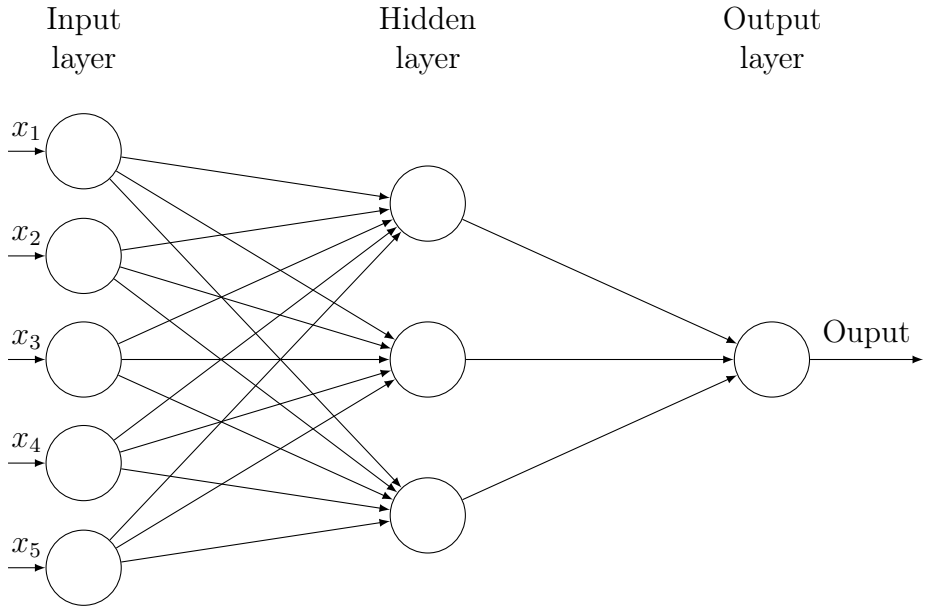


Figure 2.4. A FFNN with 5 neurons in the input layer, 3 neurons in the hidden layer and 1 neuron in the output layer.

Hebbian learning is not applicable to train this kind of network since we have more than one neuron. The most commonly used algorithm for FFNN training is called backpropagation. It computes the gradient of the loss function with respect to the network's weights for a sample or a group of samples (batch). Then, it updates the weights to minimize loss using gradient descent or other variants such as stochastic gradient descent.

2.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are derived from feedforward neural networks and, differently from FFNN, are able to handle dynamic data, using an internal state as a memory to process variable-length sequences of inputs. The reason why RNN are called recurrent neural networks is that the previous outputs are used as inputs for the current sequence; in this way, computation takes into account historical information. RNN models are commonly used in the fields of natural language processing and speech recognition.

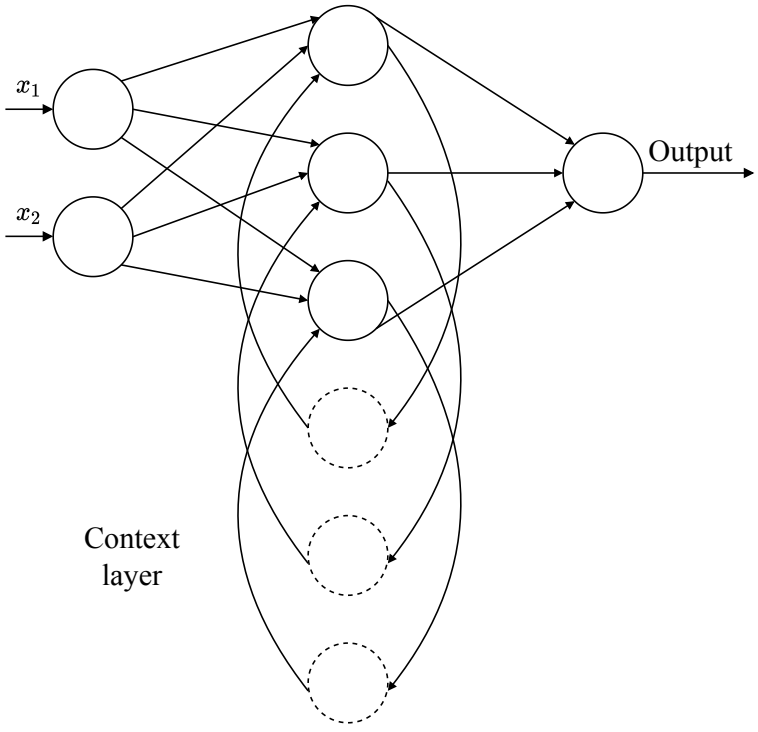


Figure 2.5. Elman neural network architecture.

In Figure 2.5 is represented the structure of an Elman network, a category of RNN. The main difference over the FFNN is the context layer, which receives input from, and returns values to, the hidden layer: the output produced at time t affects the parameter available at time $t + 1$.

The training of these kinds of networks is done using an algorithm called back-propagation through time, which is an extension of the standard backpropagation algorithm. It performs: i) Unfolding of the RNN for U time steps, obtaining a FFNN; ii) backpropagation is applied to the new network.

Vanishing gradient

The vanishing and exploding gradient phenomena, where the backpropagated error decreases or increases exponentially, are often encountered in the context of RNNs. The reason why it happens is the multiplicative gradient that can be exponentially decreasing/increasing to the number of layers. This problem is relevant, especially in RNN, because they have a large number of layers after the unfolding: the more the learning process goes into the past, the more the gradient will be close to zero, independently of the error (vanishing gradient). This is caused by some activation functions, such as sigmoid or hyperbolic tangent, which have a derivative smaller than 1. By repeatedly multiplying them during the training process, the gradient will go to zero.

One possible solution to solve the vanishing problem is to use an activation function called ReLU defined as follows:

$$f(x) = ReLU(x) = max(0, x) \tag{2.9}$$

It has a derivative equal to 1 for $x > 0$ and equal to 0 for $x < 0$. In the former case, the gradient is backpropagated as it is, while in the latter, no gradient is backpropagated from that point backward.

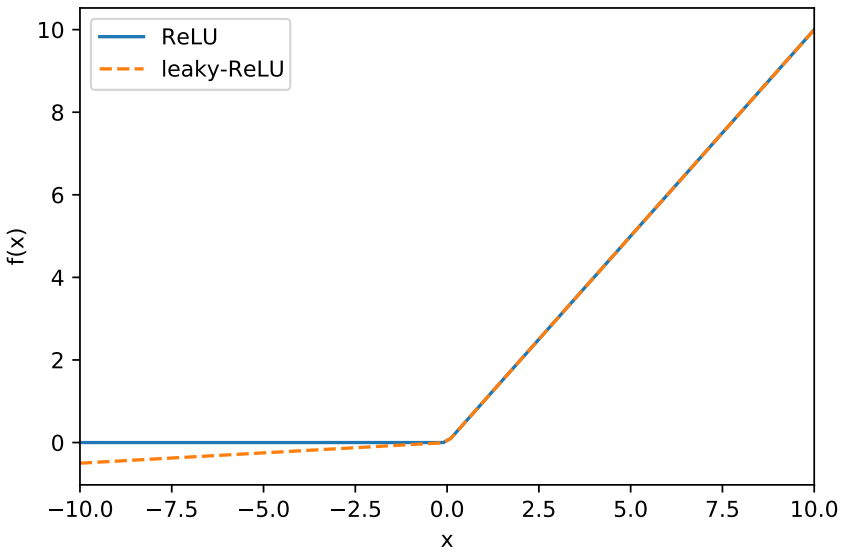


Figure 2.6. Differences between ReLU and leaky-ReLU.

The use of ReLU has a disadvantage: if the weights learned are such that the x is negative for the entire domain of inputs, the neuron never learns. This is known as the dying ReLU problem. The dying ReLU problem can be tackled by using a modified version of the activation function, called leaky ReLU defined as follows:

$$f(x) = max(0.01x, x) \tag{2.10}$$

LSTM

Another solution to deal with vanishing gradient is to use Long short-term memory (LSTM) networks [30]: particular types of RNN that can learn very long sequences. They have a cell state c_{t-1}, which allows the information to go through it unchanged. The LSTM can remove or add information to the cell state if needed through structures called gates. Gates allow to let information through if needed and are composed of a sigmoid neural net layer and a pointwise multiplication operation. There are three different gates: the Forget gate decides what information should be thrown away or kept from the previous step; the Input gate decides what information is relevant to add from the current step; the Output gate determines what the next hidden state should be.

The same problem that usually happens to RNN happens with LSTM as well, that is when sentences are too long LSTM do not work well. The reason is that the probability of keeping the context from a word that is distant from the current word being processed decreases exponentially with the distance from it. Another issue with RNN, and LSTM, is that it is difficult to parallelize the processing of sequences since you have to process word by word.

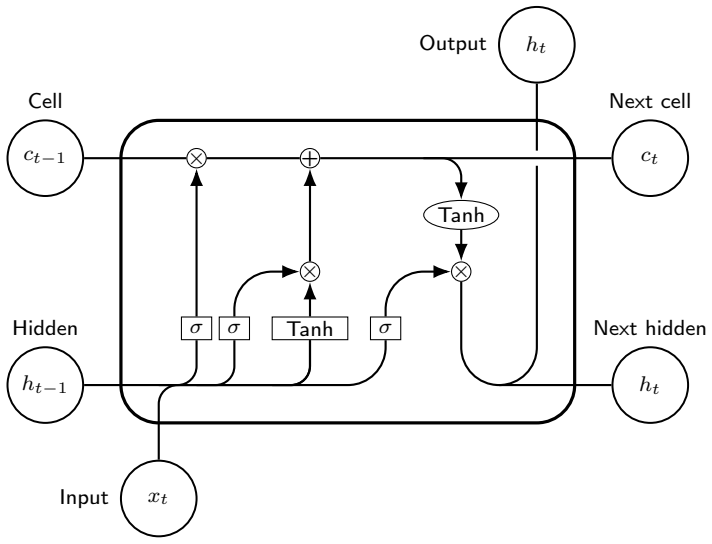


Figure 2.7. LSTM structure.

2.3.3 Preventing Neural Networks from overfitting

The training of neural network models is based on a finite set of training data. During the training, the model's objective is to perform well in predicting the training set's observations. In general, however, a machine learning scheme's goal is to produce a model that generalizes, that is, that predicts previously unseen samples. Overfitting occurs when a model fits the training data well while having low performance on the testing data. Some of the most used techniques to reduce overfitting in neural networks, also referred to as regularization techniques, are described below.

Dropout

Dropout is based on a simple idea. During training, some neurons are randomly switched off or "dropped out" with a certain probability preventing neurons from co-adapting too much. Co-adaptation is when some connections have more predictive capability than the others. While training, dropout trains different subnetworks, similar to an ensemble. Dropout can be applied to any hidden layers in the network, as well to the input layer, but it is not used on the output layer. The dropout is then not used during the testing (or inference).

Early stopping

The model tries to minimize the loss function on the training data by tuning the parameters. Overfitting networks show a monotone training error trend, but at some point, they loose generalization. At each epoch, the loss is calculated on a different data split, called validation set. When the loss of the validation set stops decreasing, the training is stopped. However, when the validation loss starts to rise, the training is not stopped immediately but continues for a number (the "patience") of epochs to avoid local minima. This simple strategy of stopping early based on the validation set loss is called Early Stopping.

L1 & L2 regularization

A way to avoid overfitting is to regularize the cost function of the neural network. A complex model is prone to overfitting, so to penalize complexity is possible to add to the loss function another term known as the regularization term.

L1 regularization, also known as Lasso Regularization is defined as follows:

$$\lambda \sum_{n=1}^{N} |w_i| \tag{2.11}$$

Where λ is a hyperparameter that determines the regularization's importance compared to the loss, and w are the weights. L1 regularization pushes weights to be

zero, producing sparse models, i.e., models where unnecessary features don't contribute to predictions.

L2 regularization, also known as Lasso Regularization is defined as follows:

$$\lambda \sum_{n=1}^{N} w_i^2 \tag{2.12}$$

L2 doesn't push the values to be exactly zero, but only close to zero.

2.3.4 Encoder-Decoder architecture

The encoder-decoder architecture is a very general design. The encoder is a neural network that maps an input space, for example, a sentence, to a latent space. The decoder is the complementary function that creates a map from the (encoder's) latent space to another target space. A task like machine translation is done by mapping the input space (e.g., a sequence of tokens in English) to another output space (e.g., a sequence of tokens in French) and linking them through a shared latent space.

An autoencoder is a particular case of an encoder-decoder where the target is equal to the input. The encoder and the decoder are trained together. The loss function is based on computing the delta between the actual and reconstructed input. The optimizer will try to train both encoder and decoder to lower this reconstruction loss. This process is completely unsupervised, and no label is needed. Once trained, we can keep only the encoder part and extract meaningful information from the input.

The encoder-decoder approach was successfully used in machine-translation, and it is called Sequence to Sequence Learning (seq2seq) [59]. Originally based on multiple layers of LSTM to map the input sequence to a fixed vector, and then others LSTM layers to decode the target sequence from the latent vector. However, a single fixed-size hidden state becomes an information bottleneck, resulting in performance degradation when dealing with long sentences. The idea to solve this problem is to introduce a mechanism called attention, which allows using all the encoder's hidden states as a source of information.

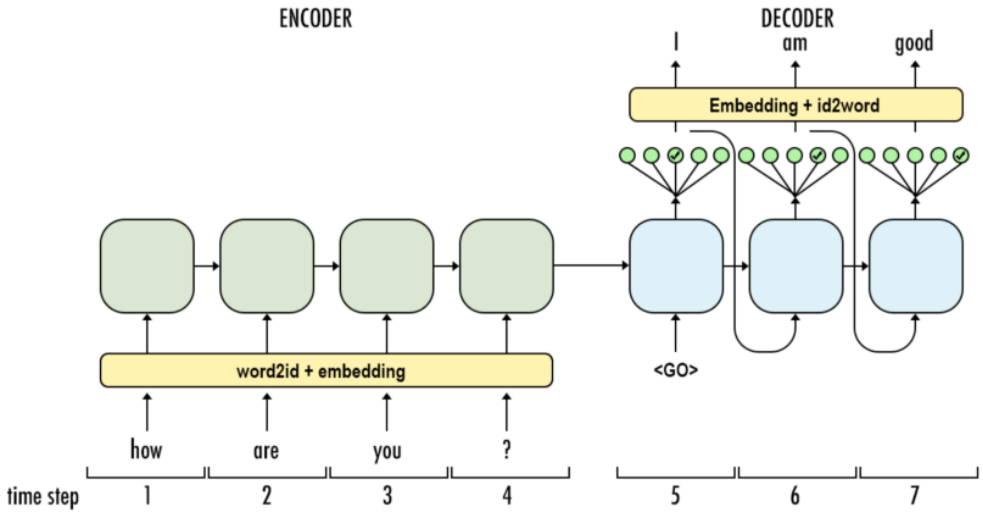


Figure 2.8. seq2seq training process.[1]

2.3.5 Attention

Attention is a concept that refers to how we, as humans, actively process certain information in the environment. An example is when we focus on different regions of an image or when we correlate words in a sentence. Researchers have tried to reproduce this mechanism to better memorize long source sentences in neural machine translation (NMT), and have become a fundamental part of sequence modeling, allowing modeling of dependencies without concern about their distance in the input or output sequences. It is usually presented combined with seq2seq models to explain how it works. This is because attention has initially been introduced as a solution to address the problem of capturing long dependence in seq2seq models. Attention is successfully applied in NLP, but also in computer vision: a visual attention mechanism was proposed in 2015 by Xu et al. [68].

Different models of attention exist, and can be grouped into three main categories:

- Self-Attention (also called intra-attention).
- Global/Local attention.
- Hard/Soft attention.

Self-attention relates to different positions of a single sequence to compute a representation of the same sequence.

Xu et al. [68] have proposed the concept of Hard and soft attention; they present an attention-based model that automatically learns to describe the content of images based on this attention. In hard attention, some part of the image is selectively ignored, while in soft attention, all the image is used, but in an aggregated and reweighted form.

[1] `https://medium.com/@Aj.Cheng/seq2seq-18a0730d1d77`

Global and local attention were introduced in [40]. It is a concept very similar to hard and soft attention; the global one always attends to all sources of information, and local uses only a subset at a time. An improvement is that local attention is differentiable everywhere, while this is not true in hard attention.

In the beginning, the attention mechanism was applied to RNN, but this does not allow for processing inputs in parallel.

2.3.6 Transformers

The Transformer is a deep learning model introduced the first time in 2017 in the paper *Attention is all you need* [61] and mostly used in NLP tasks. Transformers are created to handle sequential data, such as natural language, without requiring the sequential data to be processed in a given order, unlike RNNs. This fact allows parallelization, reducing training time compared to RNNs, making easier the training on a large corpus of data. Moreover, Transformers models easily manage long-range dependencies, thanks to the attention mechanism.

The Transformer is based on an encoder-decoder architecture, as shows Figure 2.9 where the encoder is the block on the left and the decoder on the right. There are multiple identical encoder-decoder blocks stacked on top of each other. Both the encoder stack and the decoder stack have the same number of units. The Transformer is composed of:

- Scaled Dot-Product Attention.
- Multi-Head Attention.
- Position-wise Feed-Forward Networks.
- Embeddings and Softmax.
- Positional Encoding.

The Transformer relies on self-attention mechanism, more precisely on Scaled Dot-Product Attention. The encoded representation of the input is a set of key-value pairs, (K, V), both of dimension n (the input sequence length), where the keys and values are the encoder hidden states. In the decoder, the previous output is compressed into a query (Q of dimension m) and the next output is produced by mapping this query and the set of keys and values. The output is a weighted sum of the values, where the weight assigned to each value is calculated as the dot-product of the query with all the keys:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{n}})V \tag{2.13}$$

The attention is calculated multiple times, therefore referred to as Multi-head attention: *"Multi-head attention allows the model to jointly attend to information from*

different representation subspaces at different positions" [61]. Then each Multi-head attention layer is followed by a fully connected layer.

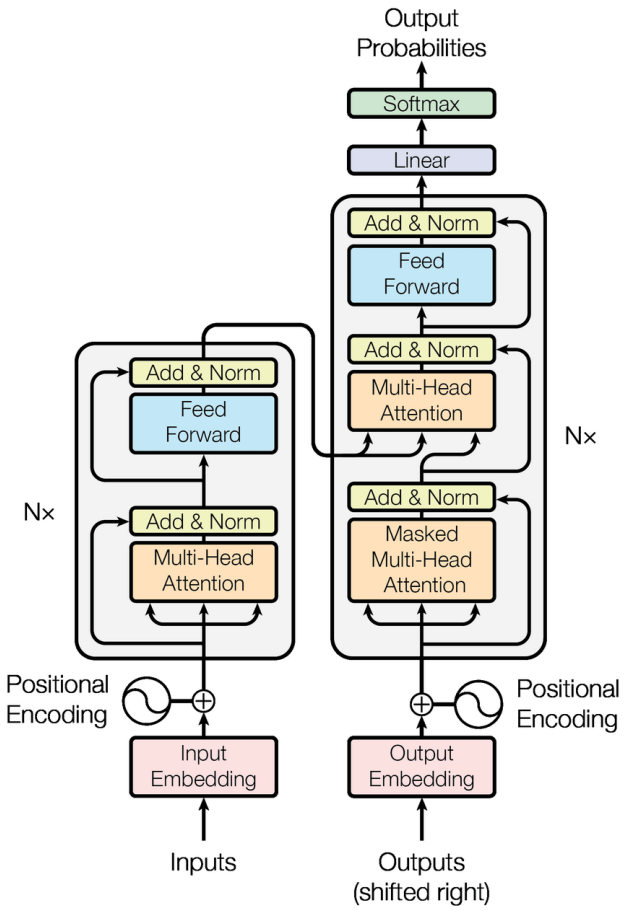


Figure 2.9. The Transformer - model architecture [61].

Another aspect to clarify is how to give information about the order of tokens in the input sequence since it different compared to RNNs. A position-dependent vector is added to each word-embedding, called positional encoding, that in the original paper uses sinusoidal functions.

2.3.7 Pre-trained Models & Transfer Learning

Transfer learning refers to a technique that allows the storing of knowledge obtained while solving a task and utilizing it for a different but related task. In some domain, it is challenging to construct a large-scale well-annotated dataset, or to train a model on a large amount of data could be very expensive. Transfer learning relaxes the hypothesis that training data must be independent and identically distributed with the test data [60]. It addresses the problem of lack of training data in a new domain,

transferring knowledge from a pre-existing data pool, and prevents the training of the model from scratch.

Transfer learning is common in the real world: learning to ride a motorbike might help to ride a car, or learning a new language might help you learn another. In deep learning, common transfer learning cases are in image vision and NLP fields.

An example of a pre-trained deep neural network in computer vision is the ResNet [27]. It is trained to classify more than a million images. One common source used to do the pre-training is the ImageNet dataset [14], a large dataset consisting of 1.4M images and 1000 classes. Next, it is possible to exploit the model to do a more specific task.

Transfer learning is also being widely used in the field of NLP, in the form of pre-trained language models. These models are, in most cases, trained on a large corpus of text in an unsupervised manner.

Bidirectional Encoder Representations from Transformers (BERT)

BERT is a deep learning model, developed by Google in 2018, that has given state-of-the-art results on eleven NLP tasks such as GLUE [63] and Stanford Q/A dataset (SQuAD) [49].

BERT is a multi-layer bidirectional Transformer. Two versions are presented in the paper [15]; their size is summarized in the following table.

Model name	# parameters (millions)	# layers	hidden size	attention heads
BERT-large	180	24	1024	16
BERT-base	110	12	768	12

Table 2.1. BERT models comparison.

BERT is pre-trained on two tasks:

1. MLM: 15% of the words in each input sequence is replaced with a `[MASK]` token. The goal is to train the model to predict the original value of the masked words, given the context provided by the other, non-masked, words in the sequence. Of these 15% tokens: 80% are replaced with a `[MASK]` token, 10% with a random word, and 10% use the original word.

2. Next Sentence Prediction (NSP): The goal here is to teach the model to understand the relationship between two sentences. The model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the following sentence or not in the original document. 50% of the inputs are a pair in which the second sentence is the following sentence in the original document, while in the other 50%, a random sentence from the corpus is chosen as the second sentence.

There are two steps in BERT: pre-training and fine-tuning. During the pre-training part, the model is trained in an unsupervised way on a large corpus. In fine-tuning, the pre-trained parameters are used as initialization, and all of the parameters are fine-tuned using labeled data from the downstream tasks.

The pre-training of the model is based on a large corpus constituted of the BooksCorpus (800M words) and English Wikipedia (2,500M words). The tokenizer used is WordPiece with a 30,000 token vocabulary. BERT relies on randomly masking and predicting tokens. The original BERT implementation performed masking once during data preprocessing, also called static masking.

For a given token, its input representation is constructed by three embeddings. The token embeddings are the vocabulary IDs for each of the tokens, the segment embeddings is a binary class to distinguish between sentence A and B (0 or 1), and position embeddings represents the position of each token in the sequence.

The special tokens present are:

- `[CLS]`: the first token of the sequence, it is used when doing sequence classification since it provides a representation of all the sentence.
- `[SEP]`: the separator token, which is used when building a sequence from multiple sequences.
- `[PAD]`: the token used for padding, in the case of sequences of different lengths.
- `[MASK]`: the token used when training this model on the task of masked language modeling.

BERT model has an excellent performance in all natural language understanding tasks, but it lacks in language generation. Another limitation is the maximum sequence length is limited to 512. To deal with longer sequences, a model called Transformer-XL has been developed [13].

DistilBERT

DistilBERT is a pre-trained model based on BERT [53], developed by Hugginface[2]. As claimed by the authors, it has 40% fewer parameters than BERT-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE benchmark. The technique used to obtain this lighter model is called distillation. Knowledge distillation or teacher-student learning is a compression technique in which a small model is trained to replicate a larger model's behavior.

The number of layers is reduced by taking one layer out of two, leveraging the common hidden size between student and teacher. Compared to the original BERT, in DistilBERT have been removed the token-type embeddings and the pooler (used for the next sentence classification task) since the next sentence prediction objective is eliminated. Unlike what was done in the implementation of BERT, A dynamic masking has been used: each time a sequence is fed to the Transformer will have different `[MASK]` tokens.

[2] `https://huggingface.co/`

Model name	# parameters (millions)	inference time (seconds)
ELMo	180	895
BERT-base	110	668
distilBERT	66	410

Table 2.2. Inference time of a full pass of GLUE task STS-B (sentimental analysis) on CPU with a batch size of 1 [53].

As we can see from Table 2.2, a lighter model like DistilBERT is cheaper to pre-train while keeping good capabilities, being useful for on-device computations in a proof-of-concept experiment.

Chapter 3

Related Works

This chapter provides an overview of the literature relevant to this work. Section 3.1 introduces research about Knowledge Tracing (KT), to motivate the choice of Item Response Theory (IRT) as the technique to model students' skills. Section 3.2 describes works focusing on questions difficulty estimation from text. Lastly, Section 3.3 illustrates various domain-specific pre-trained models, since our approach is also based on a pre-traning on Masked Language Modeling (MLM).

3.1 Knowledge Tracing (KT)

KT is the task of modeling students' knowledge over time, allowing us to predict whether students will give correct answers to questions. Estimating students' knowledge allows teachers to understand better their students' attainment levels, which can help improve the quality of learning material and provide personalized support. Recent surveys report [2, 45] that KT is most commonly performed using logistic models [8, 9, 65] (e.g., dynamic IRT) or neural networks. The primary information used for learner modeling is the history of the scores (i.e., correctness) of students' answers. Still, there are other potentially useful sources of information, such as the history of attempts, response times, question topics, and the use of hints.

Bayesian Knowledge Tracing (BKT)—introduced for the first time in [12]—models a learner's latent knowledge state as a set of binary variables (known/unknown), each of which represents understanding or non-understanding of a single concept (i.e., topic). The process of learning is modeled by a discrete transition from an unknown to a known state. Many assumptions adopted by the BKT model, as the binary knowledge state, are considered too simplistic. The skill mastery process is complex, and a simple binary variable might not be enough to model it.

Recent works tried to build KT models using neural networks, introduced for the first time in [47]. These models, often referred to as Deep Knowledge Tracing (DKT), seem to outperform logistic models in predicting the correctness of unobserved students' answers [1, 11, 76, 77], but not all works agree on this point [16, 41, 67, 75]. However, the quality of KT is not merely based on the prediction of students' performance. As it is often the case with neural models, DKT models lack explainability; some works have been done to give a better understanding of the model [37, 74]. The first work introduces a concept of probabilistic skill similarity, while the latter leverages a hybrid

system composed of neural network and IRT. However, they do not reach the same intuitiveness as logistic models.

The comparison of different models for the KT is of relevance to our work. In fact, through these models, we estimate the difficulty of the questions, that is our ground truth. Logistics models are still used in both literature and industry; plus, they are easy to interpret. For this reason, we decided to use a 1PL IRT model, which associates a skill level for each student and a difficulty level to each question.

3.2 NLP for Difficulty Prediction

In the literature, there are works [17, 21, 73] related to the prediction of the readability of questions alone. Difficulty and readability are not the same concepts; however, the latter can be used as a feature for estimating the difficulty.

Huang et al. [31] propose a technique to predict the difficulty in Standard Tests such as TOELF or SAT, called *Test-aware Attention-based Convolutional Neural Network*. Text encoding is done through Word2Vec; then, there is a CNN layer, an attention layer, and finally, a prediction layer. In reading tests, the answers to questions can be inferred from the given passages. The task in these cases is different because the answer can be deduced from the text of the question, as the goal is to measure how difficult it is to infer the response from the passage.

Some works have been done to assess the difficulty of automatically generated questions, using bag of words and measuring the degree of similarity between the key (correct choices) and the distractors (incorrect choices) [3, 35, 72].

The estimate of the difficulty of a question also involved the area of Community Question Answering (CQA). Research in this area was carried out by Liu et al., which propose in [38] a competition-based model for estimating question difficulty by leveraging pairwise comparisons between questions and users. Their dataset comes from Stack Overflow[1] and includes questions with different topics (programming, mathematics, and English). The model proposed significantly outperforms the previous PageRank-based approach. Although their model does not use text as an input feature, they demonstrate that different words or tags in the question descriptions indicate question difficulty levels, opening the possibility of predicting question difficulty from text.

Online Judge (OJ) systems are designed for self-directed learning without interventions of teachers. These systems are widespread in the programming domain, offering a real-time automatic assessment of users' solutions. Platforms such as Codeforces[2] offer different programming problems divided by difficulty levels. Categorizing problems based on difficulty is crucial to help a novice user to approach programming and providing them recommendations. Research has been carried out in this field, both for topic modeling [32, 79] and prediction of difficulty [32], but the latter does not use text as a feature.

In the next lines, we describe models for question calibration in education. In order to compare our model with others present in the literature, we first need to

[1] `https://stackoverflow.com/`
[2] `https://codeforces.com/`

analyze the concept of difficulty, i.e., the ground truth, used in various works. Three types can be identified:

1. CTT difficulty [48, 70, 71];
2. IRT difficulty [6, 48];
3. Category-based difficulty [19].

In most of the papers, the difficulty of a question is estimated using CTT; therefore, the *correctness* (i.e., *p-value*) or the *wrongness* is used. Using the *p-value* as difficulty, while simple to calculate, is less accurate than IRT, as it assumes that all students have the same skill level. The difficulty is not the only latent traits that have been tried to predict; indeed, Benedetto et al. [6] proposed a model for the prediction of both difficulty and discrimination. In order to estimate the discrimination, it is necessary to fit a 2PL model, which requires more interactions than a 1PL model. However, requiring a large number of interactions per question leads to a reduction in the dataset for QDE from text, as questions with few interactions are discarded. A large dataset of questions usually allows for obtaining better performance and reliability.

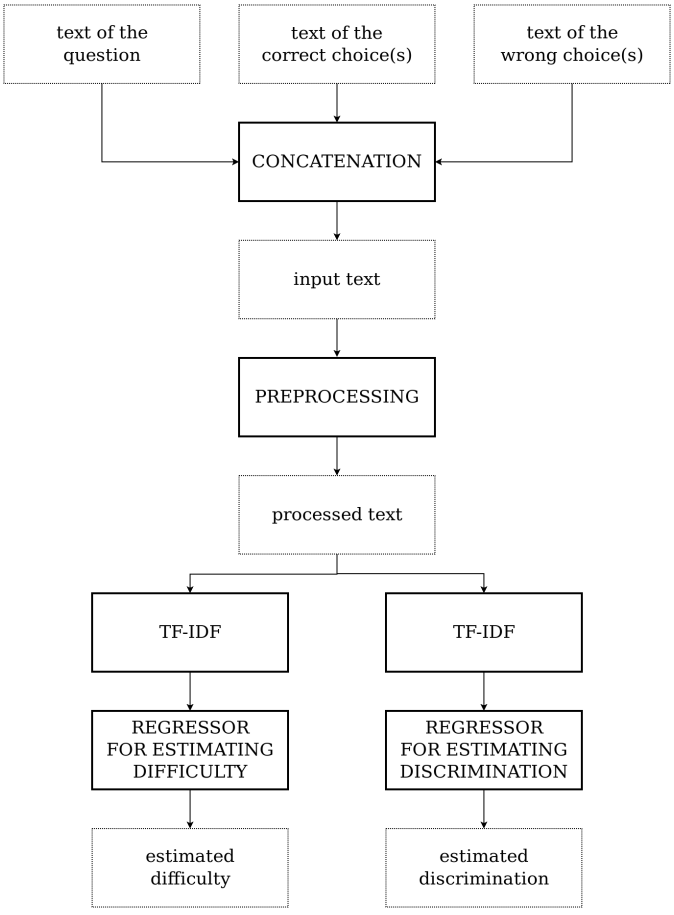


Figure 3.1. Structure of R2DE, from the input question to the estimated latent traits.

Benedetto et al. proposed R2DE [6], a regressor model that estimates IRT difficulty and discrimination of MCQ by looking at the text of the item and the text of the possible choices. It is trained and validated on a large scale dataset coming from an e-learning platform. As Figure 3.1 shows, the model uses TF-IDF to extract features from the text, a simple method but supported by the fact that keywords can indicate the difficulty of a question. The best model reported uses random forests as regressor and shows how the input question plus correct choice(s) and question plus all choices lead to comparable results, while the question stem alone leads to lower performance. The model was improved in a follow-up work [5], introducing new readability and linguistic features.

In the literature, there is a particular interest in difficulty prediction in the medical field. Qiu et al. in *Question Difficulty Prediction for Multiple Choice Problems in Medical Exams* [48] propose a "Document enhanced Attention based neural Network (DAN)" framework to predict the difficulty of multiple-choice problems in medical exams. The results are compared using two difficulties: one defined as in IRT and one described as the proportion of incorrect answers, i.e. the *wrongness* ($1-p\text{-}value$). One note is that they considered questions with at least ten interactions. This number of interactions might not be sufficient to obtain a valid ground of truth, especially for a 1PL model. The encoding is done using Bi-LSTM, and then there are three major steps:

1. the question stem and possible choices are enriched using relevant medical documents;
2. the question difficulty is divided into two components: the hardness for recalling the knowledge assessed by the question and the confusion degree to exclude distractors; for each part, there is an attention layer;
3. the two components are combined to predict the total difficulty.

The model shows excellent performance compared to the baselines used. Still, this solution relies on a large quantity of unstructured medical materials: around two million published papers and 500 textbooks. The additional corpus is used to enrich questions and to recall the knowledge assessed by the question.

Yaneva et al. in *Predicting the Difficulty of Multiple Choice Questions in a High-stakes Medical Exam* [71] propose a model to perform QDE from text in the medical field. They used a real-world dataset from a high-stakes medical licensing exam United States Medical Licensing Examination, utilizing *p-value* as ground truth difficulty. Different classes of features were developed: linguistic features (e.g., lexical, syntactic, semantic), Information Retrieval (IR) features, and two embeddings (ELMo and Word2Vec). These embeddings were obtained using around 20 million medical abstracts. Best results were achieved when combining all these features, showing a statistically significant improvement over the majority baseline.

Using a dataset of the same domain, Xue et al. [70] use transfer learning to predict the difficulty. In particular, an ELMo network was trained firstly for the prediction of response time, then used for difficulty prediction, and vice versa. They utilized three different ELMo configurations (small, middle, and original) and various combinations of inputs: stem only, options only, or a combination of both. The results indicate

that transfer learning can be applied to improve the prediction of questions difficulty when response time is used as pretraining but not vice-versa. Also, difficulty was best predicted using only the item stem, and the only baseline used is the majority baseline.

Fang et al. [19] propose a novel Bayesian inference-based Exercise Difficulty Prediction (BEDP) framework to predict the difficulty of visual-textual exercises. The representation of images is obtained from a Residual Network (ResNet), while texts are embedded using BERT. Then a Bayesian inference-based Softmax Regression classifier to predict the difficulty of the exercise. Two datasets were used, one containing mathematics exercises and the other medicine exercises. The medical dataset was manually labeled. In both cases, the difficulty is not a continuous value but is represented in categories.

The models used in the literature are validated using private datasets. Questions used in real tests are not publishable for obvious reasons, except when no longer used. Private datasets and the fact that the concept of difficulty is not unique makes it challenging to compare results between models. From this point of view, CQA provides a large pool of public questions. The interest of recent years in Question Answering has also increased the availability of MCQ datasets, such as the datasets offered by AI2[3]. These, however, do not have a label that represents the difficulty, but they could be used for a transfer learning or unsupervised training approach.

The model proposed in this thesis investigates the use of Transformers models to perform Question Difficulty Estimation (QDE) from text, using IRT as ground truth difficulty. Transformers-based models have achieved brilliant performances in many NLP tasks. For this reason, we decided to apply them for the estimation of the difficulty since the previous models use less modern techniques, such as TF-IDF or ELMo. Furthermore, we propose a solution that does not depend on additional textual material but can leverage it if available. Indeed we explored two different approaches: the first uses only the text of the questions, the second also uses external textual information to improve the performance. Our model is evaluated on two datasets coming from different sources, one public and one private.

3.3 Domain-specific Pre-training

Pre-trained models, such as BERT, are trained on a general domain corpus and can be then fine-tuned on different downstream tasks. However, if the downstream task concerns a different domain with a different vocabulary, the results might not be optimal. Two solutions to overcome this problem are: i) pre-training from scratch the Transformer model with a new vocabulary; ii) pre-training starting from the weights of the original model using the same vocabulary (referred to as continual pre-training [23]).

The scientific community is moving to share models pre-trained in different domains since these are costly to train. The following examples show domain-specific pre-trained models. SCIBERT [4], is pre-trained on a large multi-domain corpus of scientific publications (3.17B tokens) to improve performance on downstream scientific tasks. They did two experiments: i) training BERT starting from its original vocabulary and

[3] `https://allenai.org/data`

weights; ii) training BERT from scratch with a new vocabulary built on the new corpus. Both the models performed better than the original BERT, showing that while an in-domain vocabulary is helpful, their model benefits most from the scientific corpus pre-training. BioBERT [36] is another BERT-base model pre-trained on large-scale biomedical corpora. They show that pre-training BERT on biomedical corpora helps it to understand biomedical texts. Even if they used a huge corpus (18B words), they adopted the original vocabulary of BERT-base for two reasons: (i) compatibility of BioBERT with BERT; (ii) with the WordPiece tokenization, any new biomedical words can still be represented by subwords that are in the vocabulary of the original BERT. A novel work [23] from Microsoft research team shows that domain-specific pre-training from scratch can significantly outperform continual pre-training models like BioBERT. Sun et al. [58]. report that Within-task pre-training, i.e., when the pre-training corpus is the target task's training data, improves the performance on the tasked task.

Our additional corpus is 100 times less than the one used in BERT and DistilBERT. For this reason, we could not train the model from scratch. However, using a new vocabulary is not essential to have an increase in performance over the original.

Chapter 4

Models

This chapter describes how the two pre-trained Transformers models, BERT and DistilBERT, are trained with two different approaches. Section 4.1 illustrates the differences between the two pre-trained language models, Section 4.2 describes the additional pre-training on Masked Language Modeling (MLM), and Section 4.3 the fine-tuning on Question Difficulty Estimation (QDE) from text.

QDE consists in predicting the difficulty of a question from text. The task performed is a regression: the input is a sequence of words representing the question, and the target is a real number representing the difficulty. We apply pre-trained language models to this regression task by adding a linear regressor layer. Then, we fine-tune the parameters of the Transformers model and the regressor layer jointly.

In addition to that, we experiment with the possibility of further pre-training the pre-trained Language models on a corpus specific to the fine-tuning task domain. The pre-trained weights are then used to perform the fine-tuning on the regression task. Figure 4.1 shows our general approach; the dotted line is the solution that does not leverage the further pre-training, while the continuous line is the one that requires it.

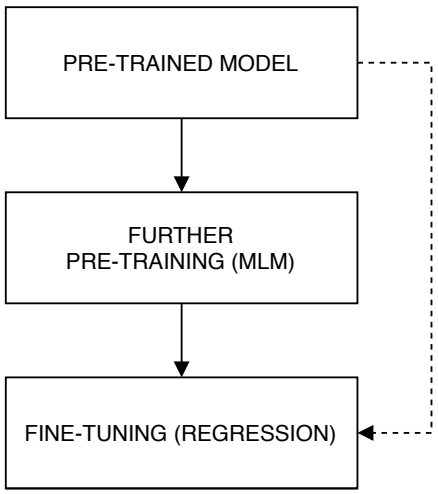


Figure 4.1. The two different approaches: the dotted line represents the approach which performs only fine-tuning for QDE from text, the continuous line is the approach with the additional pre-training on MLM.

4.1 BERT and distilBERT

BERT and DistilBERT are two very similar pre-trained models; in fact, DistilBERT is a Transformer model trained by distilling *BERT-base*, with which it shares almost all the architecture. Table 4.1 shows the size of the two models. DistilBERT has fewer parameters, which makes it lighter and faster to train. As we can see, the difference lies in the number of layers, which leads to a smaller number of parameters for DistilBERT.

Model name	# parameters (millions)	# layers	hidden size	attention heads
BERT-base	110	12	768	12
distilBERT	66	6	768	12

Table 4.1. BERT and distilBERT size comparison.

BERT and distilBERT are pre-trained on a general corpus of 3,300 million words: the Toronto BooksCorpus (800M words) and English Wikipedia (2,500M words). The Wikipedia texts are pre-processed to ignore lists, tables, and headers. Both the models address the problem of out-of-vocabulary words using a variant of Byte Pair Encoding (BPE) [55] named WordPiece [54]. The vocabulary of the two pre-trained models is identical and has the same pre-specified size of 30,522 tokens. Moreover, both models use the same special tokens (`[PAD]`, `[MASK]`, `[CLS]`, `[SEP]`). We consider the uncased version.

The maximum sequence length supported by the models is 512. For our experiments, we set it to 128 or to 256, depending on the input. Only the first 128 or 256 are considered for longer sequences, while shorter ones are padded with the special token `[PAD]`. This design choice is based on the analysis of our datasets, as the questions are short.

4.2 Further pre-training on MLM

The original BERT is trained on two objectives: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), while DistilBERT only on the first. We decide to pre-train both models only on MLM, for consistency and also because NSP improves only slightly downstream task performance [39].

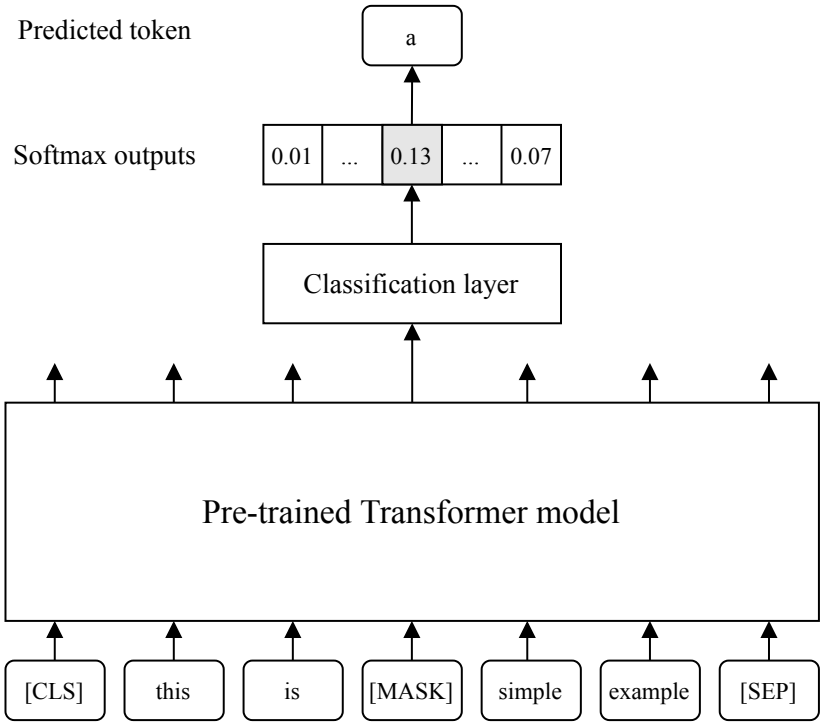


Figure 4.2. Additional pre-training on Masked Language Modeling (MLM).

In MLM pre-training, one of the input words is substituted by the special token `[MASK]`, and the model is asked to predict (using the contextual embedding of the surrounding tokens) the word that was masked. Figure 4.2 shows a high-level view of masked training. The `[MASK]` token is predicted by feeding the embedding output (the final hidden vectors corresponding to the mask tokens) into a fully connected layer. The number of output neurons is equal to the number of tokens in the vocabulary; each token's probability is calculated with softmax.

Our corpus is 100 times smaller than the one used in BERT. For this reason, we cannot train the model from scratch. We use the original weights as initialization for MLM pre-training and the same original vocabulary as BERT. Creating a new vocabulary based on our corpus would require training from scratch. Before training, we mask each sequence as in the original BERT implementation: 15% of the sequence's tokens are masked; of those, 80% is masked with `[MASK]`, 10% is set to a random token, and 10% is just left as it is. The special tokens are never masked nor used as random tokens.

The following shows an example of masking:

Input = *[CLS] this is an [MASK] of masking [SEP]*

Label = *example*

After passing through the dense layer, the final hidden vectors corresponding to the `[MASK]` tokens are fed into an output softmax over the vocabulary. The MLM

objective is a cross-entropy loss on predicting the masked tokens. The cross-entropy loss is defined as follows:

$$loss = \sum_{k=1}^{K} (y_k log(\hat{y}_k)) \tag{4.1}$$

Where $\hat{y}$ is the predicted probability for the class k, y binary indicator (0 or 1) if class label k is the correct classification. The number of classes is equal to the size of the vocabulary.

4.3 Fine-tuning on Question Difficulty Estimation

The fine-tuning of BERT and DistilBERT is straightforward since the self-attention mechanism in the Transformer allows modeling many downstream tasks. Figure 4.3 shows a high-level view of the fine-tuning process of a Transformer model.

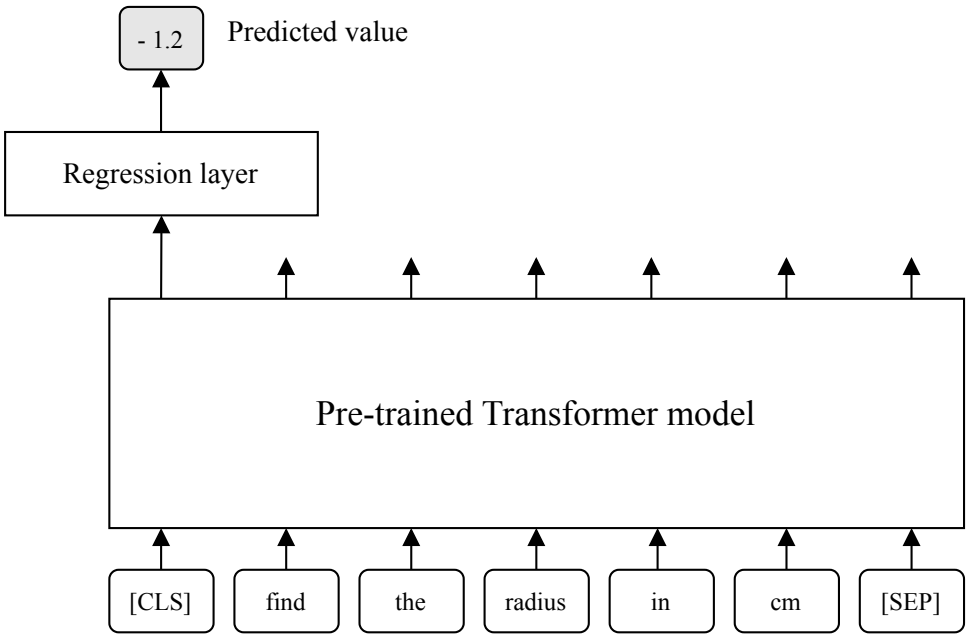


Figure 4.3. Fine-tuning.

It is done as follows. We obtain a fixed-dimensional representation of the input sequence using the final hidden state (i.e., the output of the Transformer) of the `[CLS]`

token, denoted as $C \in \mathbb{R}^H$, that represent the sequence. Then we add a linear layer, whose parameters matrix has dimension $\in \mathbb{R}^{K \times H}$, where K is the number of output neurons ($K = 1$, for us). Finally, the prediction is the output $O = linear(CW^T)$. In our case, since we are performing regression, the activation function is linear.

The pre-trained model and the linear layer are linked as described in the BERT paper: "The first token of every sequence is always a special classification token (`[CLS]`). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks." [15]. The `[CLS]` token is the best representation when we fine-tune all the parameters of the model, like in this case. If we wanted to use BERT as a feature-based extractor (i.e., when the Transformer model is frozen), this would not be valid anymore. In fact, the authors show that the best choice is to concatenate the token representations from the top four hidden layers. The same observations are valid for DistilBERT.

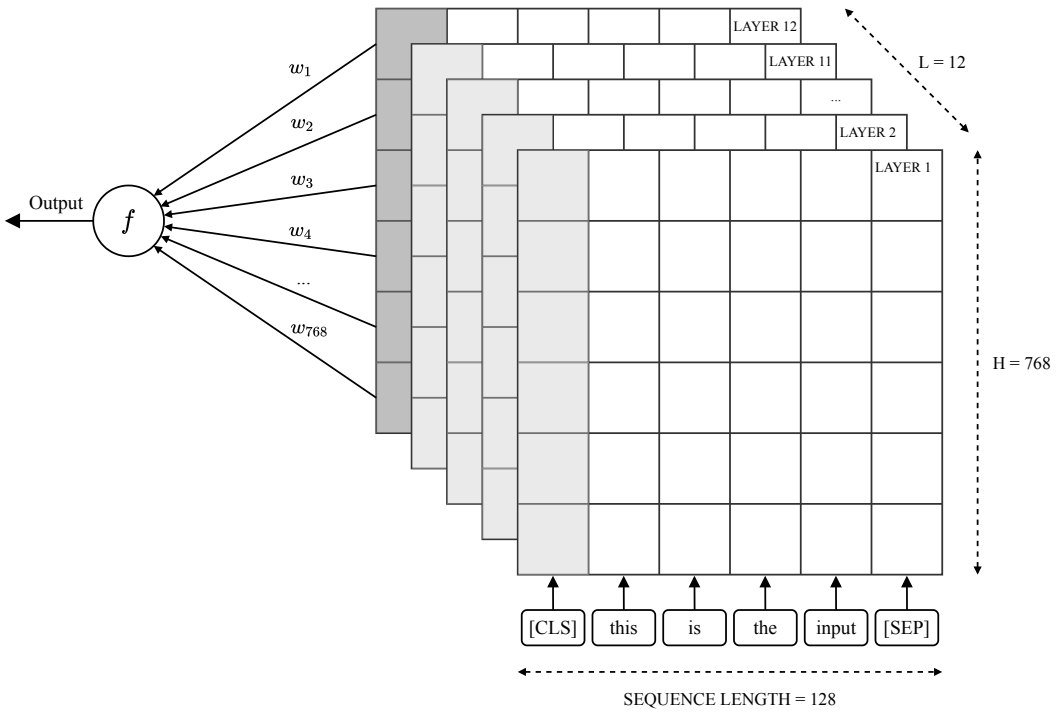


Figure 4.4. Fine-tuning model architecture of BERT in detail.

Figure 4.4 shows more in detail the fine-tuning of our model, specifically the BERT version. As we can see, the embedding of the last hidden layer of `[CLS]` is fed to a linear regressor layer. The only difference in the DistilBERT version is that the number of Transformer blocks (L) is 6 instead of 12.

For fine-tuning, we use Mean Squared Error (MSE) as loss function. MSE is a widely used loss in regression cases, and it is differentiable. The choice of MSE as loss function is justifiable from a probabilistic point of view as a Maximum Likelihood

Estimation (MLE) procedure (under the assumption data are sampled from a normal distribution) [22].

It is calculated as the average of the squared differences between the predicted $\hat{y}$ and observed y values:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.2}$$

Chapter 5

Experimental Datasets

This chapter introduces the two data collections used for the experiments as well all pre-processing done to create the final dataset. Section 5.1 presents the publicly available data collection provided by ASSISTments[1], while Section 5.2 the CloudAcademy[2] data collection.

Both data collections are composed of two datasets: i) the *Interactions (I)* dataset and ii) the *Questions (Q)* dataset. The *Interactions* dataset stores students' answers to questions. Each row provides (at least): the user id, the question id, the correctness of the answer, and the interaction's timestamp. The *Interactions* dataset is used to estimate (with IRT) the difficulty of each question, which will be used as ground truth while training the model that performs difficulty estimation from text. The *Questions* dataset contains the textual information about the items. For each question, it provides the question id, the text, and, for CloudAcademy data, the text of the possible choices.

An additional dataset referred to as *Lectures*, is provided by Cloud Academy. It contains transcripts of the IT technologies lectures related to the questions and can be leveraged by our model to estimate the difficulty of questions more accurately.

5.1 ASSISTments Dataset

ASSISTments is an online intelligent tutoring system, developed by the Worcester Polytechnic Institute, that assists students and teachers. The platform provides teachers and administrators with contents from open educational resources but also the possibility to build and share their own questions. Students can complete assignments using the ASSISTments web-platform, receiving instant feedback. Teachers also receive reports on students' progress and class performance. This platform is a generic system that can be applied to any subject, although it offers mostly math content.

5.1.1 Interactions data

The *Interactions* dataset was published in [20] and contains data related to the school year 2012-2013. This dataset has been used in several works, such as [56, 64, 69],

[1] `https://new.assistments.org/`
[2] `https://cloudacademy.com/`

mostly about KT.

This dataset contains 6,123,270 interactions between users and questions. Each interaction has 35 attributes; the most relevant to our work are described below:

- *problem_id*: indicates the id of the problem.
- *user_id*: indicates the unique identifier of the student.
- *start_time*: is a timestamp when a student start a problem.
- *start_time*: is a timestamp when a student submits the answer to a problem.
- *problem_type*: identifies the typology of the problem.
- *original*: identifies if a problem is the main problem or a scaffolding.
- *correct*: correctness of the answer.
- *template_id*: identifies a template; questions from the same template are very similar.
- *skill*: indicates the skill associated with the problem.

correct The *correct* attribute indicates the correctness of the student's answer. It is equal to 1 if the student answers correctly on the first attempt, otherwise, it is 0. Decimal values are calculated as a partial credit based on the number of hints and attempts needed to solve. Table 5.1 shows the distribution of correct values; the overall correctness is 67.64%, and there are few partial scores.

Correct	Count	Percentage
1	4,141,564	67.64%
0	1,976,383	32.28%
Decimal values	5,323	0.08%
Total	6,123,270	100%

Table 5.1. Distribution of scores.

problem_type As Table 5.2 shows, there are six different types of problems:

- *algebra*: math evaluated string (text box);
- *choose_1*: MCQ with one correct choice (radio buttons);
- *fill_in*: simple string-compared answer (text box);

- *open_response*: open response question (text box);
- *choose_n*: MCQ with multiple correct choices (radio buttons);
- *rank*: rank multiple objects.

99% of the interactions are originated by the first three categories.

Problem Type	Count	Percentage
algebra	3,500,688	57.17%
choose_1	1,847,657	30.17%
fill_in_1	742,960	12.13%
open_response	17,642	0.29%
choose_n	11,597	0.19%
rank	2,726	0.05%
Total	6,123,270	100%

Table 5.2. Types of problems.

original Some problems can be broken down into simpler subproblems called "scaffolding". When a student cannot solve the main problem, he can decide to solve the scaffoldings. The *original* attribute identifies a main problem (*original* = 1) from a scaffolding (*original* = 0). If a problem has no scaffolding, it is marked as a main problem. The main problem and its associated scaffolding have a different *problem_id*, but same *template_id*. As Figure 5.3 shows the interactions are mostly with main problems; looking at the number of problems: 84% of the problems are main, while 16% are scaffolding.

Original	Count	Percentage
main problem	5,819,737	95%
scaffolding problem	303,533	5%
Total	6,123,270	100%

Table 5.3. Main and scaffolding problems.

start_time* and *end_time Each student-problem interaction has two associated timestamps: *start_time* indicates the beginning, *end_time* indicates the end. The first interaction of the dataset is dated 2012-09-01, while the last one is dated 2013-08-31. The days passed from the first to the last interaction of each student were computed. The average is 73 days with a standard deviation of 99 days. Figure 5.1 shows the distribution of the days passed on the platform; as we can notice, there is a peak of students who have been on the platform only a few days.

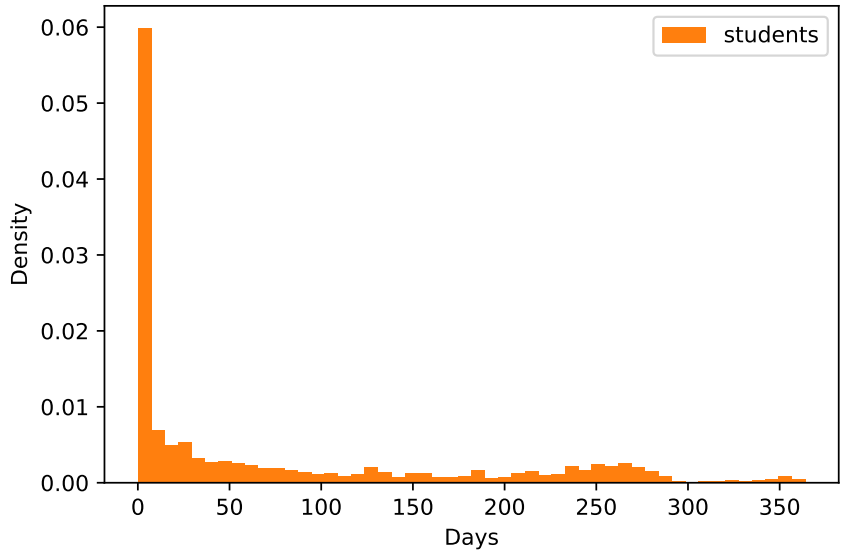


Figure 5.1. Distribution of days between the first and last interaction on ASSISTments.

skill The attribute *skill* refers to the skill of a problem, but in 72% of the cases, its value is *null*. In total, there are 179 different skills.

template_id Each *problem_id* has associated only one *template_id*. A *template_id*, on the other hand, can have several problems associated with it, which are very similar to each other. In order to create a large number of problems, it is common for content creators to make a few templates and substitute different numbers and keywords in the problems and answers. An example of two different problems generated from the same template is the following:

If a new jacket sells for $34, find the total cost if you were charged 5% sales tax.

If a new shirt sells for $36, find the total cost if you were charged 6% sales tax.

A note should be made about scaffolding problems: although they have the same template as the main problem, they are different. The choice of the identifier to use to distinguish the problems, *template_id* or *problem_id*, is important. Problems

generated by the same *template_id* have very similar text and are created to have the same level of difficulty.

We wanted to check if the difficulty between problems generated by the same template is similar, and we verified it by comparing the correctness as an approximation of the difficulty. First, we calculated the correctness of original problems with at least 50 interactions to get a more accurate estimate. Then we calculate the standard deviation of correctness for each template with at least 10 problems. Finally, we computed the mean of all the standard deviations resulting in 0.078, which represents how the correctness varies for problems of the same template. This result is small, considering a range of correctness that is between 0 and 1. Besides, we also compared it with the standard deviation between different templates. First, we kept original problems with at least 50 interactions. Then we calculated the correctness per template and then the standard deviation of correctness between different template, that is 0.194. As we can see, the standard deviation for problems of the same template is very small compared to the standard deviation between different templates. Supported by these reasons, it was decided to use the *template_id* as item identifier.

Pre-processing We used the *Interactions* dataset to obtain each item difficulty using IRT, which is the ground truth. It must be suitably pre-processed to get an accurate estimate, and we proceeded as follows. For each student-item pair, we considered only the first answer in chronological order (i.e., the first attempt) because if a student has already seen a question, he will be more advantaged to answer correctly, influencing the IRT estimate. We kept only items with at least 50 interactions. The more interactions an item has, the more accurate is the estimate of difficulty. However, considering a minimum threshold of interactions reduces the number of items. A low number of items means a small dataset that is used to train and test our model. This has some disadvantages, such as increased overfitting. Therefore, we tried to find a fair trade-off between the goodness of the estimate and the number of items.

The *correct* label also allows partial credits, which is why it has been converted, as suggested by ASSISTments, to a binary variable using the formula: $1 = correct$, $< 1 = incorrect$. Table 5.4 compares the dimensionality of the dataset before and after pre-processing.

	Raw dataset	After pre-processing
# interactions	6,123,270	2,820,051
# users	46,674	43,868
# problems	179,999	55,178
# templates	96,403	18,659

Table 5.4. *Interactions* dataset dimensionality.

# interactions i	Percentage of items
$50 <= i <= 100$	68.03%
$100 < i <= 200$	19.54%
$200 < i <= 500$	8.15%
$i > 500$	4.29%
Total	100%

Table 5.5. Interactions per item after pre-processing.

Table 5.5 shows the distribution of the number of interactions per question. On average, each question has 151 interactions (i.e., student responses) with a standard deviation of 303. On average, each student answered 64 different questions with a standard deviation of 113.

Figure 5.2 and 5.3 show the distribution of items and students per correctness after the pre-processing. In the first case, the distribution follows a log-normal shape with a negative skew. In the second case, the distribution follows a gaussian shape; there are peaks for values 0 and 1, which can be caused by students with few interactions.

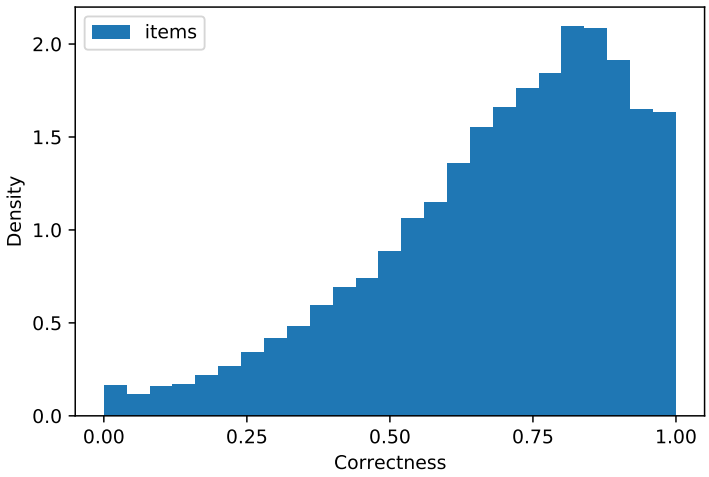


Figure 5.2. ASSISTments, distribution of items per correctness.

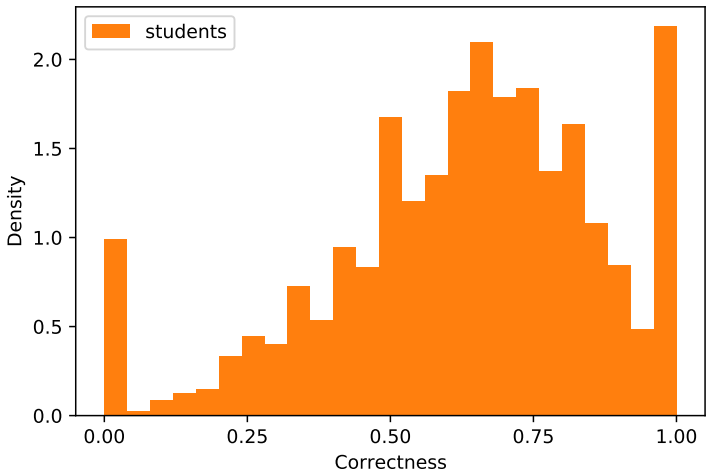


Figure 5.3. ASSISTments, distribution of students per correctness.

5.1.2 Questions data

The *Questions* dataset of ASSISTments contains the textual information of questions (also referred to as problems). Some research works [43, 44, 57] have used this dataset to predict the skill associated with each question. The attributes relevant to our work are:

- *problem_id*: indicates the id of the problem.
- *body*: contains the textual information of the problem.

The dataset is composed of 179,950 *problem_id*. Several texts are duplicates: 138,084 *body* over 179,950 are unique. Other information, such as the attributes of the Interaction data, can be obtained merging this dataset with the Interaction dataset on *problem_id*. Question texts do not always provide complete information about the request. In fact, they sometimes refer to images, tables, or graphics that are not present in the text: 9.2% of the items in the raw dataset refers to an external image. Furthermore, in the case of MCQ problem typology, only the stem is present and not the choices.

First of all, the texts of problems that are not in the *Interactions* dataset are excluded (i.e., all those with an insufficient number of interactions are excluded). Then we performed a specific data pre-processing divided into two steps: i) text cleaning, ii) text elimination. In the first step, the texts were only transformed and not completely removed. HTML tags, URLs, newlines have been removed. There are texts with references to the exercise number or external books (e.g., "Page 2", "Question #2"). Because they are not textual information and might increase the risk of overfitting, we tried to remove them. The following is an example of text before and after the cleaning step:

Raw string:
<p>Convert 0.2 into a <span

*style="color: #674ea7;">fraction. You must simplify your answer to lowest terms.<br id="yct810" /> *
*n   *

Pre-processed string:
Convert 0.2 into a fraction. You must simplify your answer to lowest terms.

Subsequently, we manually searched unsuitable problem patterns. We identified three categories of these problems: i) system messages (e.g., info on whether the question is correct or not, how many attempts are available); ii) problems referring to external books; iii) problems where all the text of the questions is in the image. Table 5.6 shows some examples of unusable problems.

Text	Problem type
"Sorry, that is incorrect. Let's go to the next question!"	*choose_1*
"Submit your answer from the textbook."	*algebra*
"Earth Science QUESTION 10"	*choose_1*
"Problem 6"	*fill_in_1*

Table 5.6. Examples of unusable problems.

We removed problems with less than two words and also problems with the same text but with a different template since most of the times are unusable. Each *template_id* has several very similar problem texts associated with it, as described above. We decided to keep a text for each *template_id*. This is done to increase generalization; the presence of very similar texts in training and in the test would falsify the results since the same template problems have a similar difficulty. Another option is to split test and train with different *template_id*. However, some *template_id* have several dozen texts associated, while others are one. This would have created problems during the training and evaluation of the results. Patikorn et al. in [44] used a dataset from the same platform ASSISTments, and they showed that keeping problems from the same template is the cause of overfitting in the skill-tagging problem. Their solution was to keep one problem per *template_id*.

	Raw	Merged	Cleaned	Final
# unique texts	138,084	58,320	52,156	11,393
# template_id	125,264	18,883	12,993	11,393
# problem_id	179,950	47,898	41,254	11,393

Table 5.7. *Questions* dataset dimensionality through pre-processing steps.

Table 5.7 shows the dimensionality of the dataset through the various steps: i) raw: the raw dataset; ii) merged: after keeping items with at least 50 interactions; iii)cleaned: after the text pre-processing; iv) final: after keeping only one text per template. The final dataset used for the prediction of difficulty from text is composed of 11,393 items.

Length (# words)	Percentage of items	After pre-processing
len $<$ 2	5.31%	0%
2 $<$ len $<=$ 10	44.54%	10.66%
10 $<$ len $<=$ 50	37.15%	74.38%
50 $<$ len $<=$ 100	9.67%	13.97%
len $>$ 100	3.33%	0.99%
Total	100%	100%

Table 5.8. Distribution of problems per length.

Table 5.8 shows the distribution of *Questions* dataset before and after pre-processing.

5.2 Cloud Academy Dataset

Cloud Academy is an e-learning provider offering online courses about IT technologies. The dataset used in our experiments is a sub-sample of their data collection, containing only questions about cloud technologies (e.g., AWS[3] , GCP[4], Azure[5]). Differently from ASSISTments, there is no concept of "template"—thus, all the questions are unique—and all the questions are MCQ. In addition to the text of the questions, the text of the possible choices is available as well, information that is not available in the ASSISTments dataset.

5.2.1 Interactions data

The *Interactions* dataset contains 7,323,502 interactions between users and questions, involving 24,696 users and 13,603 questions. Each interaction is characterized by a user id, an item id, a correct label, and a timestamp. Table 5.9 shows the distribution of correct values, the overall correctness in the raw dataset is 66.51%.

[3] `https://aws.amazon.com/`
[4] `https://cloud.google.com/`
[5] `https://azure.microsoft.com/`

Correct	Count	Percentage
1	4,870,727	66.51%
0	2,452,775	33.49%
Total	7,323,502	100%

Table 5.9. Distribution of scores.

As done with the ASSISTments dataset, for each student-item pair, only the first answer in chronological order is considered (i.e., the first attempt), and only items with at least 50 interactions are considered.

# interactions i	Percentage of items
$50 <= i <= 100$	36.75%
$100 < i <= 200$	20.99%
$200 < i <= 500$	23.60%
$i > 500$	18.66%
Total	100%

Table 5.10. Interactions per item after pre-processing.

Table 5.10 shows the distribution of the number of interactions per question. On average, each question has 304 interactions (i.e., student responses) with a standard deviation of 365. On average, each student answered 114 different questions with a standard deviation of 161.

Figure 5.4 and 5.5 show the distribution of items and students per correctness after pre-processing. In both cases, the distribution follows a gaussian shape, but in the case of students, there are peaks for values 0 and 1 which can be caused by students with few interactions.

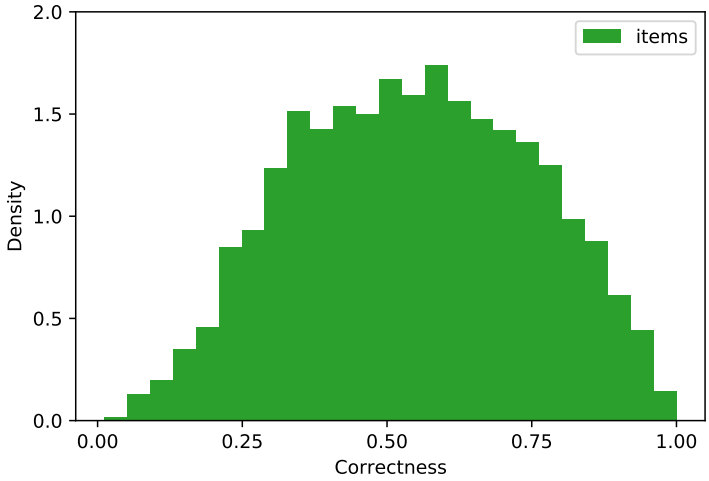


Figure 5.4. Cloud Academy, distribution of questions per correctness.

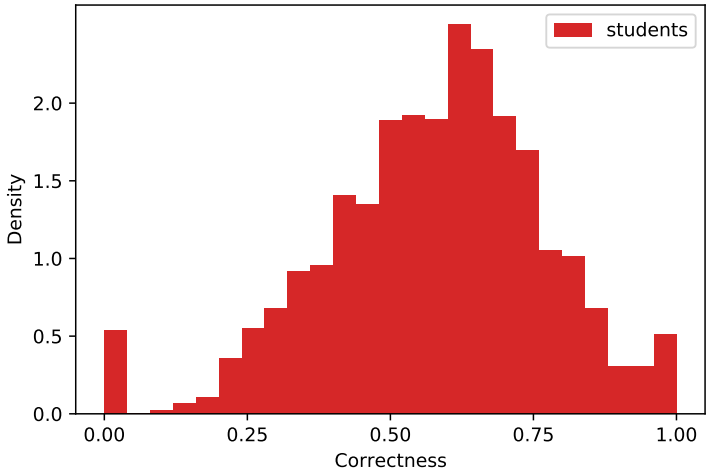


Figure 5.5. Cloud Academy, distribution of students per correctness.

5.2.2 Questions data

The question dataset required less pre-processing than that of ASSISTments. Questions texts with less than 50 interactions have been removed. Subsequently, the HTML tags were removed.

Length (# words)	Percentage of items
$2 < \text{len} <= 10$	10.66%
$10 < \text{len} <= 50$	74.38%
$50 < \text{len} <= 100$	13.97%
$\text{len} > 100$	0.99%
Total	100%

Table 5.11. Distribution of questions per length.

It is essential to analyze the number of words in the question stem and the possible choices. This is useful for sizing the model input. Table 5.11 shows the distribution of the number of words in the questions, where only the stem of the question is considered.

# of choices	Percentage of items
4	83.24%
5	13.00%
6	3.42%
> 6	0.34%
Total	100%

Table 5.12. Distribution of questions per number of possible choices.

All the questions in the dataset are of type MCQ; the number of possible choices is shown in Table 5.12. The texts of the possible answers are also available; on average, a choice contains 6.8 words.

5.2.3 Lectures data

Additionally, we also consider an additional dataset—referred to as *Lectures*—provided by Cloud Academy, which contains the transcripts of some of the online lectures about cloud technologies. Pre-trained models, such as BERT, are usually trained in a general domain. However, to give the model a better understanding of a specific domain, it is possible to do an additional pre-training. This pre-training is done in an unsupervised manner and allows to improve performance, as shown in [58]. The paper refers to "within-task pre-training" when BERT is further pre-trained on the same data used for the downstream task training, and to "in-domain pre-training" when the pre-training data is obtained from the same domain of the target task.

The lecture dataset is composed of 2,826,126 words. This dataset is merged with the *Question* dataset (containing all the questions, except those used for model testing) that is composed of 401,912 words. The total corpus has 3,228,038 words. Then the corpus is divided into sentences, based on punctuation (full stop, question mark, exclamation mark) for a total of 159,563 sequences, with an average of 20 words per sequence. Table 5.13 shows the distribution of sentences per length.

Length (# words)	Percentage of items
len <= 10	23.24%
10 < len <= 50	72.79%
50 < len <= 100	3.70%
len > 100	0.27%
Total	100%

Table 5.13. Distribution of sentences per length.

Chapter 6

Experimental Setup

This chapter introduces the experimental setup. Section 6.1 describes how we estimate our target variable (i.e., the difficulty of the questions). Section 6.2 describes the setup for the pre-training on Masked Language Modeling (MLM) and Section 6.3 illustrates the setup related to the fine-tuning on Question Difficulty Estimation.

As shown in Figure 6.1, training is performed in two steps: i) a 1PL IRT model is fitted to estimate the difficulty of each question; ii) the IRT estimated difficulty is used as target label to train the Transformers-based model, which gets the text as input.

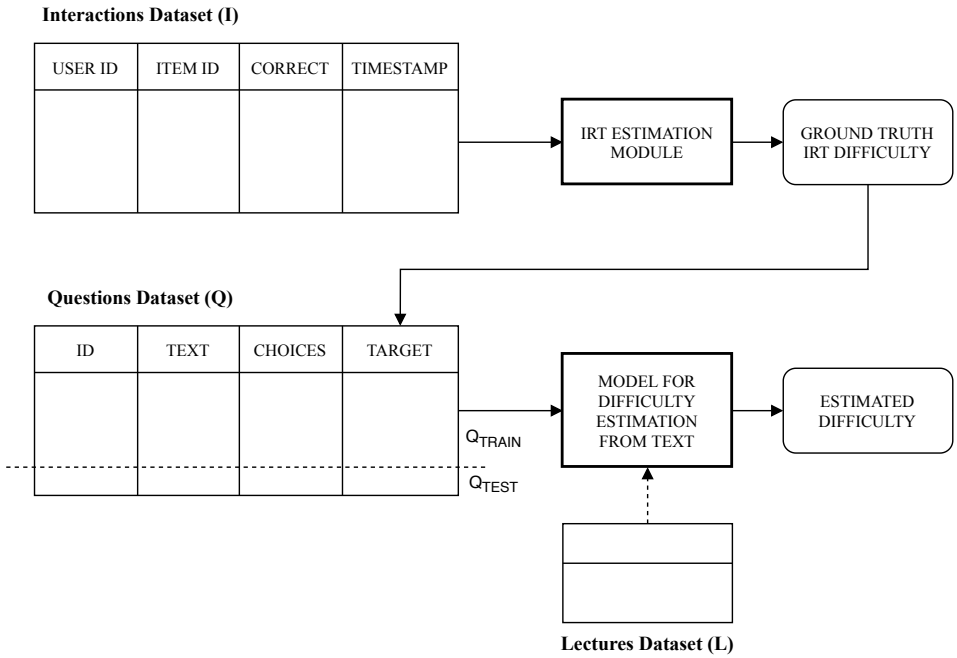


Figure 6.1. Experimental setup.

The first step involves the estimation with IRT of difficulty, starting from the Interactions dataset. Then the IRT difficulties, which are considered as ground truth, are associated with the Question dataset. It is split into two sets with a proportion 80:20 for training (Q_{TRAIN}) and testing (Q_{TEST}) our Transformers-based model. A portion of the training set (10%), called validation set, is reserved for doing the

hyperparameters tuning. Furthermore, the proposed model can optionally exploit an additional text dataset (i.e., the *Lectures* dataset) to further pre-train the pre-trained language models on the task of MLM and improve performance.

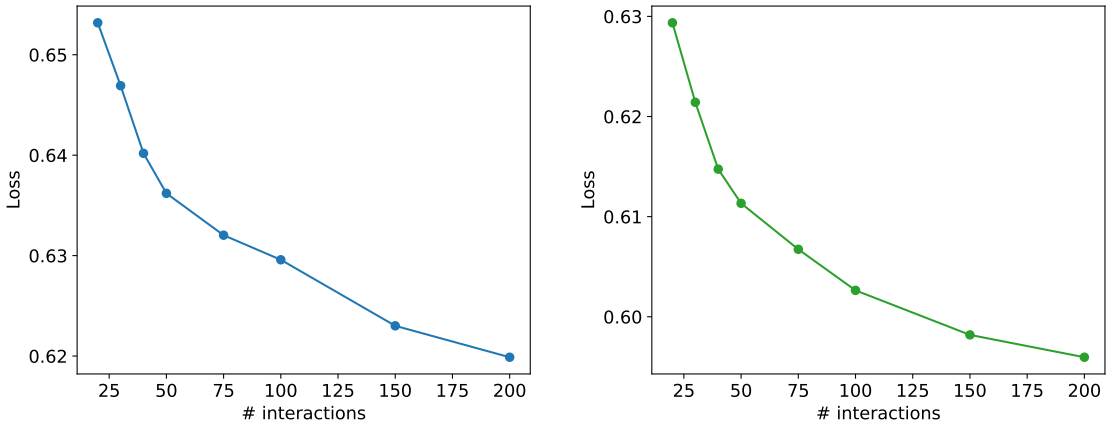
The final objective of our model is to estimate the difficulty of a question without using the Interaction dataset, but only the textual information. This allows calibrating questions without having any interactions. Additionally, the code is publicly available[1].

6.1 IRT Estimation

The estimation of ground truth difficulties is made using a 1PL IRT model. To do so, we use a *python* library called *pyirt*[2]. It implements the EM (Expectation-Maximization) algorithm for computing the maximum likelihood estimates of parameters in IRT models, that is described in [26]. The inputs it uses are a unique student identifier, a unique item identifier, a timestamp of the interaction, and a binary score value. The range of difficulty is set as $[-5; 5]$, while discrimination is fixed to 1.

The choice to use a one-parameter model is dictated by the fact that more complex models would have required many interactions. We set a minimum number of samples per question equal to 50. We are aware that a more accurate estimate would require over 50 interactions per item, but raising this threshold would reduce the number of questions at our disposal too much. However, we remind that the average number of interactions per item is much higher than the minimum threshold.

For each dataset, we assess the overall quality of the Interactions dataset in two ways: i) using the loss of *pyirt*; ii) comparing the estimated difficulty from different subsets of interactions. The *pyirt* library has an internal loss, based on likelihood, that indicates the goodness of fitting (the smaller, the better); we evaluate how that loss varies as the number of interactions increases for a group of items.



(a) ASSISTments dataset.

(b) Cloud Academy dataset.

Figure 6.2. Fitting loss per number of interactions.

[1] `https://github.com/aradelli/transformers-for-qde`

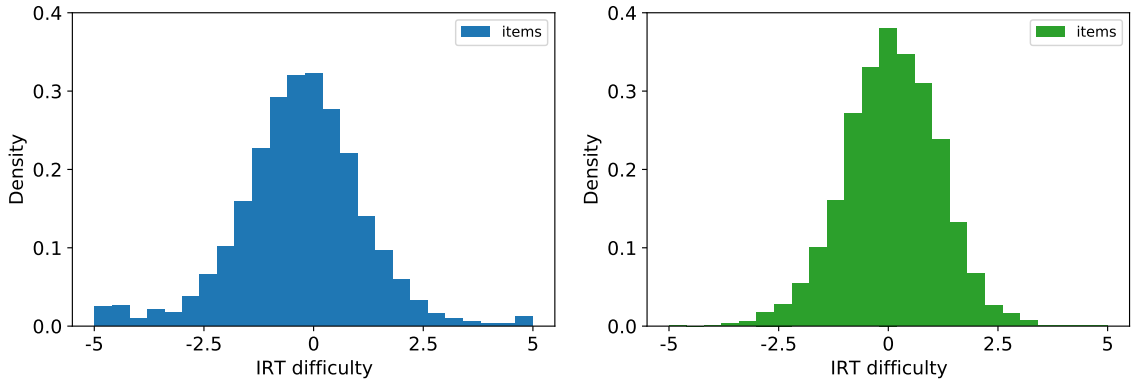
[2] `https://github.com/17zuoye/pyirt`

Figure 6.2 shows the trend of the loss as the number of interactions increases. We select items with 200 interactions and then calculate the difficulty using the first 20, 30, ..., 200 to see how the estimate improves. The behavior of the curves on both graphs is similar. As we can see, with more than 50 interactions, the steepness of the curves begins to decrease.

We also assess the stability of the difficulty of different samples of interactions. For each question, we create two sub-sets consisting of 50 different interactions. After estimating the difficulty using IRT with both subsets, we compare, on average, how much the estimated difficulty for each question has varied.

The ASSISTments Interactions dataset has an average number of interactions per item of 151, with a minimum of 50. Students interacted with 64 different items on average. The final loss that measures the goodness of fit of the model to our data, calculated by *pyirt* is 0.62. On average, the difficulty varied by 6.46% between the two different subsets.

The Cloud Academy Interactions dataset has an average number of interactions per item of 304, with a minimum of 50. Students interacted with 115 different items on average. The final loss that measures the goodness of fit of the model to our data, calculated by *pyirt* is 0.58. The difficulty varied by 5.29% between two different subsets.



(a) ASSISTments dataset.

(b) Cloud Academy dataset.

Figure 6.3. Distribution of items per IRT difficulty.

Figure 6.3 shows the distributions of the estimated difficulty for the ASSISTments and Cloud Academy datasets. We can observe in both cases a Gaussian distribution, with an average of zero. In the ASSISTments distribution, we notice outliers, i.e., items with difficulty equal to 5 or -5. Such outliers are due to questions which are always answered correctly or wrongly by the students; we believe that there might be two reasons for that: i) some questions are probably too difficult or too easy for the students taking the exams, and ii) the dataset most likely contains some unusable problems (e.g., system logs), even after the data-cleaning we performed.

6.2 Pre-training on MLM

Pre-training is done in an unsupervised manner on Masked Language Modeling (MLM). As a starting point we use the weights and vocabulary of *bert-base-uncased* and *distilbert-base-uncased* models that are available in the *transformers* library from Hugging Face[3]. For both models, the experimental setup in this phase is the same. This pre-training requires an additional corpus, i.e., the Lectures dataset described in chapter 5, which is available only for Cloud Academy. All data is used to train the model. Each sequence is statically masked before the training, as described in Chapter 4

We implement our model using Python. We use in particular these libraries: i) *Transformers* by Hugging Face[4] for the availability of DistilBERT and BERT architectures, weights, and tokenizers; ii) *TensorFlow*[5] and *Keras*[6] are used to wrap the model provided by Hugging Face and train it for our task. As concerning the hardware, the training is done on the Google Cloud Colab[7] platform using a Tensor Processing Unit (TPU). TPU is an accelerator, developed by Google, specialized in training deep neural networks.

We train several models with a different number of epochs: 4, 12, 24, 36. The original BERT has been trained from scratch, with a number of epochs equal to 40. In our case, we are not starting from scratch, but we use for the initialization the weights of the pre-trained models, thus we assume an ideal number of epochs lower than 40. As optimizer, we use Adam [34]: an adaptive learning rate optimization algorithm designed for training deep neural networks.

We experiment with the following hyperparameters:

- Sequence length = 128;
- Batch size = 256;
- Adam learning rate = 1e-5;
- Number of epochs = 4, 12, 24, 36;
- Dropout = 0.1.

6.3 Fine-tuning on Question Difficulty Estimation

The fine-tuning task receives as input the texts of the questions and targets the difficulty estimated by IRT. For both datasets, we use the original configuration of *bert-base-uncased* and *distilbert-base-uncased*. Also, in some of the experiments, we use the pre-trained weights on MLM described in the previous section for the Cloud Academy dataset.

[3] `https://huggingface.co/transformers/pretrained_models.html`
[4] `https://huggingface.co/transformers/`
[5] `https://www.tensorflow.org/`
[6] `https://keras.io/`
[7] `https://colab.research.google.com`

The ASSISTments dataset provides only the stem of MCQ. The Cloud Academy dataset provides both the stem and the possible answers allowing us to test several input combinations:

- *question only*: stem;
- *question + correct*: stem [SEP] correct choice;
- *question + all*: stem [SEP] choice_1 [SEP] ... [SEP] choice_n.

Both datasets are split in the following way: 80% training and 20% testing. Then, the training set is further divided into 90% training and 10% validation. The validation set is used for the choice of hyperparameters and for early stopping. Table 6.1 shows the dimension of the splits of the datasets used for fine-tuning on QDE.

Dataset	# train	# validation	# test
ASSISTments	8109	901	2253
Cloud Academy	4530	504	1259

Table 6.1. Datasets size.

The training is done on the Google Cloud Colab platform using a NVIDIA® Tesla® V100 with 16GB of memory.

To reduce overfitting, we use dropout and early stopping. The dropout is applied to the regressor layer that we add on top of the Transformer. Moreover, the dropout is applied inside the Transformer for all fully connected layers in the embeddings and encoder. With early stopping, the loss on the validation set is used to determine when overfitting begins and to select the best epoch. The optimizer is Adam, to which we only tune the learning rate.

We experiment with the following hyperparameters:

- Sequence length = 128, 256;
- Batch size = 16, 32, 64;
- Learning rate = 1e-5, 2e-5, 3-5;
- Patience early stopping = 10 epochs;
- Dropout regressor layer = 0.1, 0.2, 0.3, 0.4, 0.5;
- Dropout internal = 0.1, 0.2, 0.3, 0.4, 0.5.

Chapter 7

Results

This chapter presents the results of the experiments on Question Difficulty Estimation (QDE) from text on the two experimental datasets. Section 7.1 introduces the metrics that have been used. Section 7.2 briefly presents the literature models used as baselines. Lastly, Section 7.3 presents a quantitative and qualitative analysis of the results on the ASSISTments dataset (in Section 7.3.1) and on the Cloud Academy dataset (in Section 7.3.2).

7.1 Metrics

The QDE task consists in estimating a continuous numerical value that, in our case, is between -5 and $+5$. Being a regression task, we evaluate the models using two standard regression metrics: the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE).

The MAE is the average over the samples of the absolute values of the differences between the predictions $\hat{b}_i$ and the corresponding observations b_i (in our case, the difficulties predicted by the model and the IRT difficulties). The MAE is a linear score metric, which means that all the single differences are weighted equally in the average. It is defined as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|b_i - \hat{b}_i|) \tag{7.1}$$

The RMSE is a quadratic scoring metric that measures the average magnitude of the error. It is defined as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(b_i - \hat{b}_i)^2} \tag{7.2}$$

The errors are squared before being averaged; consequently, the RMSE gives a relatively high weight to large errors (e.g., an error of 10 is 100 times worse than an error of 1). The RMSE is always larger than or equal to the MAE; the greater the difference between the two metrics, the greater the variance in the sample's errors. If the RMSE is equal to the MAE, it means that all the errors are of the same magnitude. Both metrics are always non-negative, and smaller values indicate better performance.

7.2 Baselines

This section introduces the models used as baseline while experimenting with the Transformer models: the majority baseline (Section 7.2.1), R2DE (Section 7.2.2), and ELMo (Section 7.2.3). We choose R2DE and ELMo because i) they are the only models that do not necessarily require an additional dataset of domain-related documents (in addition to the texts of the questions) and ii) they are the only two models for which we had access to the code.

7.2.1 Majority

It is important to start by comparing the results with a very simple model. We use the majority baseline, which predicts the same difficulty for each test question, regardless of its text. The difficulty is obtained as the average difficulty of the training questions.

7.2.2 R2DE

R2DE, which stands for Regressor for Difficulty and Discrimination Estimation, is a model proposed in [6], and the code is available[1]. The model can estimate both the difficulty and the discrimination of questions from text. Since we do not focus on discrimination estimation in this work, we consider only the part of the model that performs difficulty estimation. This does not affect the performance of the model, as it is made of two parallel components that are trained separately for difficulty and discrimination.

The first steps of text preprocessing are the following: i) stop words removal, ii) punctuation removal, and iii) stemming. The features are then extracted from the input text of each item using TF-IDF, a frequency-based technique. Only the top N features are considered; this is done by sorting the features according to their number of occurrences in the corpus and keeping only the N most frequent ones. The number of features N is a hyperparameter. The features are fed into random forests regressor. The best hyperparameters are chosen with five-fold cross-validation to find the best configuration. A grid-search is performed on the following hyperparameters:

- Random forests $n\_estimators$ = [50, 100, 150, 200, 250, 300]

[1] `https://github.com/lucabenedetto/r2de-nlp-to-estimating-irt-parameters`

- Random forests max_depth = [15, 25, 50, 75, 100]
- TF-IDF $max\_features$ = [1000, 1200, ..., 4000]

7.2.3 ELMo

Xue et al. [70] proposed an ELMo based model to predict the difficulty and the response time of MCQ. The text is preprocessed with tokenization, lemmatization, and stopwords removal. The model is based on ELMo [46], pre-trained on the One Billion Word Benchmark [10]. An encoding layer is added to learn the sequential information from the ELMo embedding output. The encoding layer is made of a Bidirectional LSTM network. This layer allows the extraction of encoding features, which captures more abstract information than the embeddings alone. A dense layer then follows the encoding layer to convert the feature vectors to the targets through a non-linear combination of the feature vectors' elements. Figure 7.1 shows the general structure of the model.

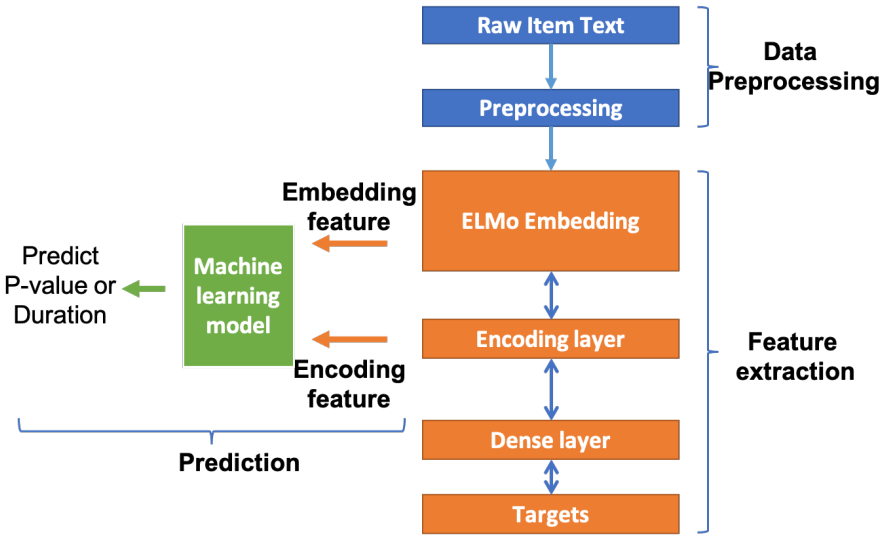


Figure 7.1. Structure of the ELMo-based model presented in [70].

We reproduce the best configuration for predicting the difficulty only, referred to as "Method 1"; that is, we use the encoding feature and ELMo original. The model is originally meant to estimate the *p-value* of the questions, but, in our case, we use it to estimate the difficulty defined as in IRT. This does not affect the accuracy of the model since the *p-value* is a floating-point number (as the IRT difficulty we are estimating).

7.3 Evaluation

The models are all trained on the same dataset (Q_{TRAIN}) and compared using the same test set (Q_{TEST}). We do three different runs for each model, using three different random seeds. Then, we calculate the average and the standard deviation (SD) of the results. We also calculate the performance of the models for questions that are very easy or very hard (i.e., with difficulty b greater than 2 or less than -2) referred to as "extreme" questions. This is done to see how the models perform on different types of questions and understand whether the accuracy of the estimate depends on the difficulty of the question. We use the different input configurations presented in Chapter 6: i) question only (*Q only*); ii) question and correct choice(s)(*Q + correct*); iii) question and all choices (*Q + all*). Furthermore, the Transformers models are evaluated with and without the additional pre-training on MLM, as explained in Chapter 4.

7.3.1 ASSISTments

The ASSISTments Q_{TEST} is made up of 2253 questions. For MCQ, only the stem is available, so the only input configuration that can be tested is *Q only*. Also, since we do not have any domain-related documents, such as the *Lectures* dataset, pre-training on MLM cannot be performed.

Comparison with the state of the art

Table 7.1 shows the results of the experiments on the ASSISTments dataset. It indicates (in this order): the model name, the input configuration, if pre-training on MLM is performed, the MAE and RMSE for all the questions, and the same metrics for "extreme" questions.

Model	Input	MLM	MAE	MAE, |b|>2
Majority	-	-	1.066 ± 0.000	2.882 ± 0.000
R2DE [6]	Q only	-	0.966 ± 0.001	2.408 ± 0.005
ELMo [70]	Q only	-	0.933 ± 0.013	2.025 ± 0.052
DistilBERT	Q only	N	0.919 ± 0.009	1.864 ± 0.059
BERT	Q only	N	**0.911** ± 0.003	**1.849** ± 0.074
			RMSE	RMSE, |b|>2
Majority	-	-	1.423 ± 0.000	3.033 ± 0.000
R2DE [6]	Q only	-	1.304 ± 0.001	2.700 ± 0.003
ELMo [70]	Q only	-	1.255 ± 0.017	2.375 ± 0.036
DistilBERT	Q only	N	1.239 ± 0.010	2.259 ± 0.037
BERT	Q only	N	**1.228** ± 0.003	**2.243** ± 0.038

Table 7.1. ASSISTments results.

The BERT-based model has the best performance on all the metrics, reducing RMSE by 13.76% and MAE by 14.53% compared to the majority baseline. Moreover, it reduces the MAE by 2.33% and the RMSE by 2.17% compared to the best performing baseline that is ELMo. This difference between BERT and ELMo appears even more evident if we consider extreme questions (i.e., with $|b|>2$): the BERT model reduces by 8.7% the MAE and 5.9% the RMSE. We observe a large standard deviation on "extreme" questions. We think it is due to the fact that these questions are only 13% of the total and to an instability of the models.

The DistilBERT authors claim to maintain 97% of BERT's performance, which in our case is confirmed. Indeed, the performance of BERT and DistilBERT is very similar, the difference is less than 1%. The best configuration of the hyperparameters for both BERT and DistilBERT is the following:

- Sequence length = 128;
- Batch size = 64;
- Learning rate = 1e-5;
- Patience early stopping = 10 epochs;
- Dropout regressor layer = 0.5;
- Dropout internal = 0.25.

Analysis of the best performing model

We perform an analysis of the best performing model, BERT, and compare it with the other models. First, we look at the difference between the train and test performance. Then, we analyze the distribution of the predicted difficulties. Lastly, we look at what characteristics of the question text may have influenced the model error.

To better understand the models, it is important to compare the performances on the test set and those on the training set. Table 7.2 shows the performance of the various models, indicating the difference between the metrics in training and in the test (averaged over three runs). R2DE has a similar error on training and test set. However, this also leads to poor results on the test set compared to BERT. This, and the fact that R2DE then tends to predict always difficulties around 0 (as we will show later on), may suggest an underfitting phenomenon. We can see that the Transformer models, BERT and DistilBERT, have a much lower error on the training set than the test set. This occurs even though we use higher dropout values (0.5) than those usually used for BERT fine-tuning (0.1) and early-stopping to try to reduce overfitting.

	Train	Test	
Model	MAE	MAE	Δ MAE
R2DE [6]	0.916 ± 0.001	0.966 ± 0.001	0.050
ELMo [70]	0.805 ± 0.008	0.933 ± 0.013	0.128
DistilBERT	0.683 ± 0.062	0.919 ± 0.009	0.236
BERT	0.608 ± 0.100	0.911 ± 0.003	0.303
	RMSE	RMSE	Δ RMSE
R2DE [6]	1.216 ± 0.001	1.304 ± 0.001	0.088
ELMo [70]	1.061 ± 0.007	1.255 ± 0.017	0.194
DistilBERT	0.897 ± 0.087	1.239 ± 0.010	0.342
BERT	0.798 ± 0.124	1.228 ± 0.003	0.439

Table 7.2. ASSISTments, train and test errors.

Figure 7.2 plots the loss on the training and the validation sets over training epochs. We can see how the two curves decrease up to the 16th epoch, after which the error on the validation starts to rise (i.e., the model begins to lose generalization). The 16th epoch is chosen by early stopping as it has the smallest error on the validation set; the training goes on for 10 more epochs (patience).

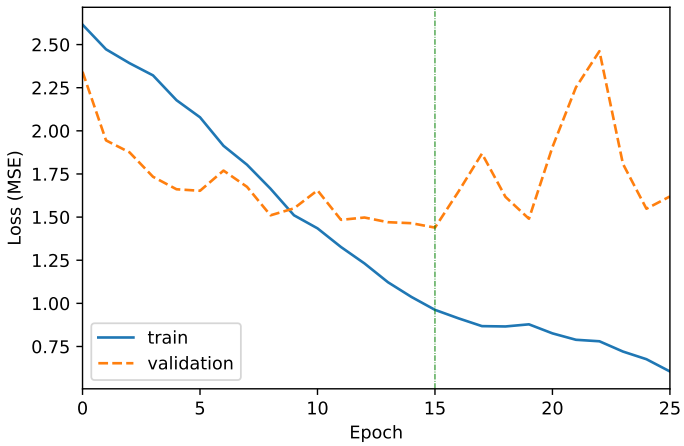
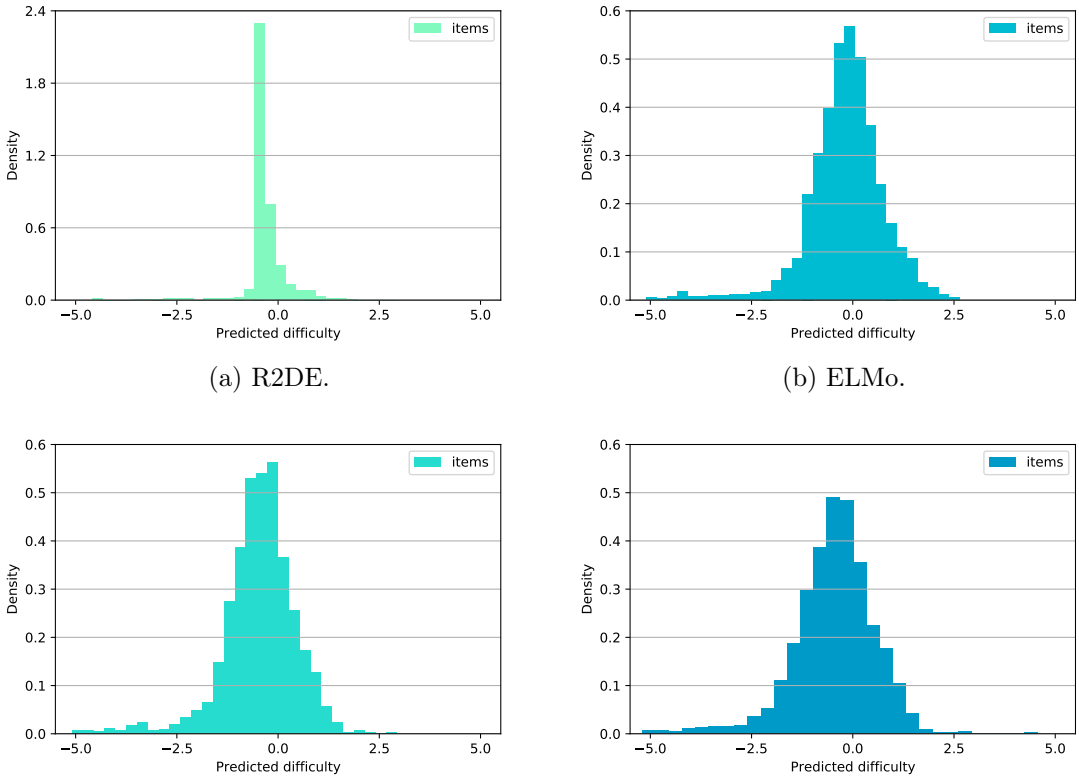


Figure 7.2. ASSISTments, model loss over training epochs.

Figure 7.3 shows the distribution of test difficulties predicted by R2DE, ELMo, DistilBERT, and BERT (the target difficulty of the test set is shown in Figure 7.4). We can immediately notice how R2DE tends to predict difficulties around 0 (note that the plot of R2DE has a different scale on the y-axis). The other models, on the other hand, have a Gaussian distribution. BERT, compared to other models, has a lower density around 0.



(a) R2DE.

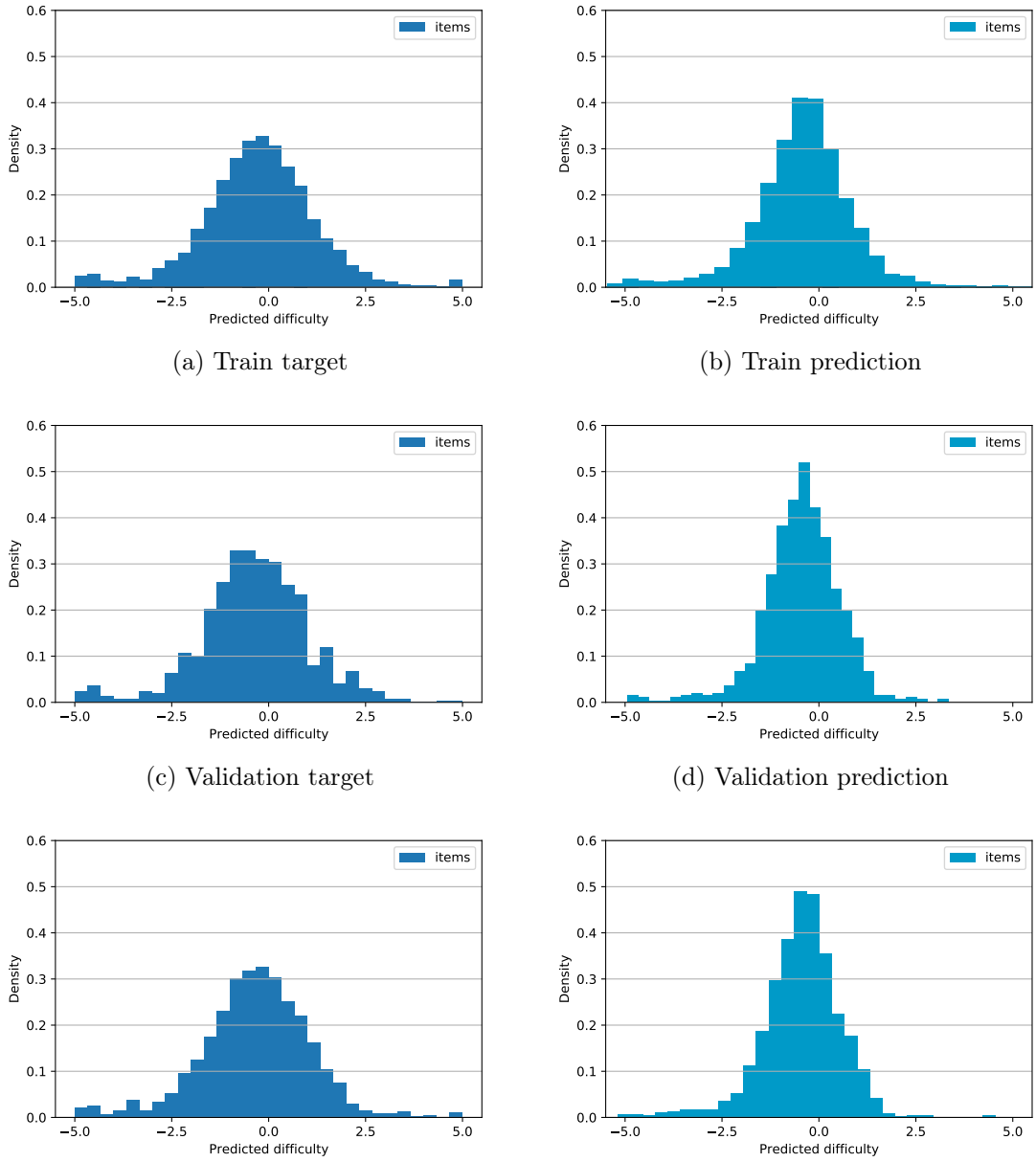
(b) ELMo.

(c) DistilBERT.

(d) BERT.

Figure 7.3. Test set, distribution of predicted difficulties.

Figure 7.4 shows the distribution of target difficulties and the ones predicted by BERT across all splits of the dataset. We see that the target difficulties in the various splits are distributed similarly. We also note that BERT maintains the Gaussian distribution of the target but tends to predict difficulties closer to 0 compared to the target.



(a) Train target

(b) Train prediction

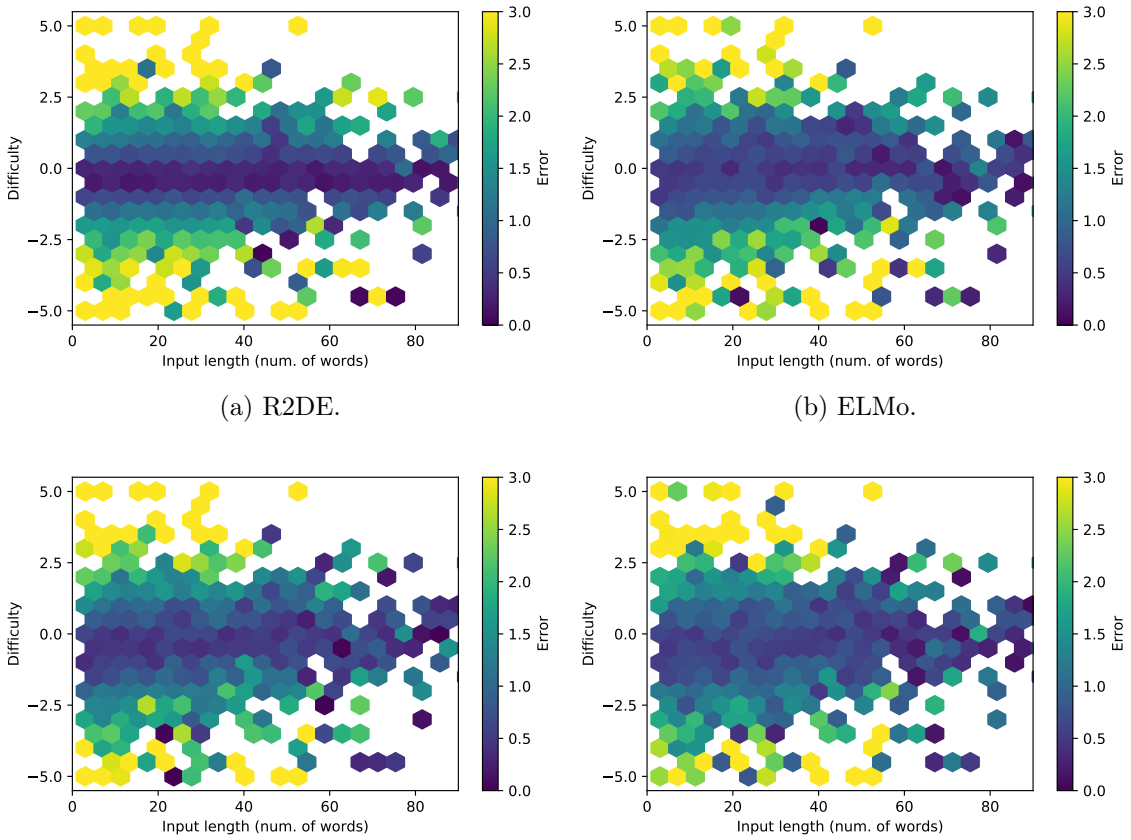
(c) Validation target

(d) Validation prediction

(e) Test target

(f) Test prediction

Figure 7.4. Distribution of the target difficulties and BERT predictions.



(a) R2DE.

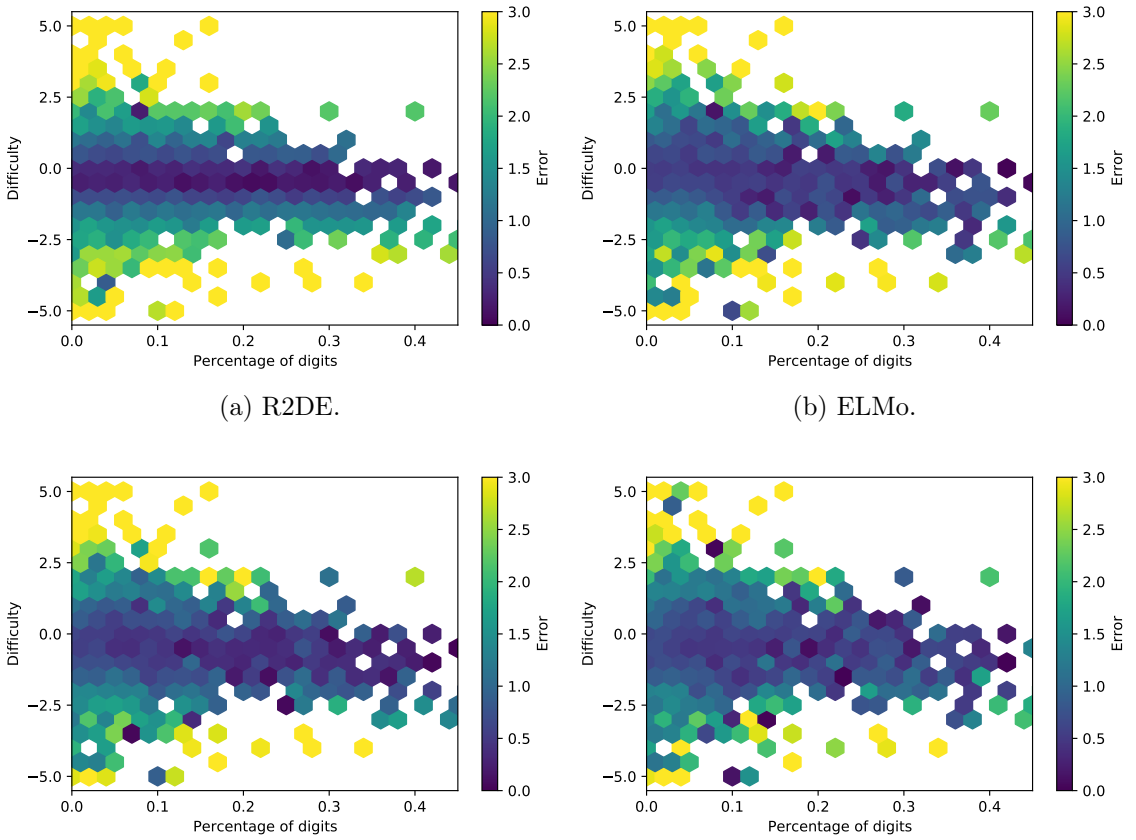
(b) ELMo.

(c) DistilBERT.

(d) BERT.

Figure 7.5. ASSISTments, error depending on the input length and true difficulty.

Figure 7.5 shows the dependence between true difficulty, the prediction error, and length of the question (in terms of the number of words) in the test set. We considered questions with less than 90 words for better visualization, representing 97.5% of the test set's questions, and we calculate the error of a single point as $|b - \hat{b}|$, where b is the true difficulty and $\hat{b}$ is the predicted difficulty. We can note that regardless of the length of the question, R2DE always tends to predict difficulties around the average, i.e., equal to 0. All the other models show different behavior from R2DE. Even in BERT, the best performing model, the error increases when the true difficulty is far from the average, but not as much as in R2DE. There is no clear relationship between question length and prediction error.



(a) R2DE.

(b) ELMo.

(c) DistilBERT.

(d) BERT.

Figure 7.6. ASSISTments, error depending on percentage of digits in the input text and true difficulty.

Many of the ASSISTments questions are related to mathematics; therefore, some questions contain numbers. To see the influence of the presence of numbers, we calculated the percentage of digits as the number of digits divided by the total number of characters in the question. Figure 7.6 shows us the dependence between true difficulty, the prediction error, and the percentage of digits in the test set. We see that digits in the questions are quite relevant; indeed, about 20% of the test questions have a percentage of digits greater than 0.1. Again, we can note that regardless of the percentage of digits, R2DE always tends to take difficulties around the average. In the other models for true difficulties around 0, the error seems to decrease as the digits' percentage increases. The presence of numbers does not seem to have caused problems for the models.

Digits	%	Avg. b	BERT MAE	R2DE MAE	DistilBERT MAE	ELMo MAE
Y	80%	-0.314	0.869	0.926	0.871	0.889
N	20%	-0.442	1.067	1.122	1.058	1.063
			RMSE	RMSE	RMSE	RMSE
Y	80%	-0.314	1.171	1.238	1.175	1.189
N	20%	-0.442	1.433	1.547	1.425	1.453

Table 7.3. Models performance per question with and without digits.

Continuing the previous analysis, Table 7.3 shows the models performance for questions with or without digits. As we can see, all models follow the same pattern, i.e., they perform better with questions containing digits. A possible explanation is that the questions with no digits are only 20%, and therefore the model did not have enough examples to learn. Furthermore, questions containing numbers can be related to the same domain, mathematics. In contrast, questions that do not contain numbers could have completely different domains and, therefore, might be more difficult to learn due to the lack of training samples.

Type	%	Avg. b	BERT MAE	R2DE MAE	DistilBERT MAE	ELMo MAE
algebra	46%	-0.182	0.852	0.902	0.853	0.872
choose_1	38%	-0.687	0.965	1.012	0.952	0.979
fill_in_1	14%	0.007	0.876	0.967	0.877	0.860
			RMSE	RMSE	RMSE	RMSE
algebra	46%	-0.182	1.156	1.205	1.156	1.148
choose_1	38%	-0.687	1.275	1.377	1.265	1.328
fill_in_1	14%	0.007	1.211	1.276	1.215	1.170

Table 7.4. Models performance per question type.

Unlike the Cloud Academy dataset, the ASSISTments dataset contains different types of questions, as described in Chapter 5 (e.g, *algebra*, *choose_1* and *fill_in_1*). Table 7.4 shows the performance of the models per question type. As we can notice, in all models, the performance on *choose_1* is lower than the others. This can be motivated by the fact that these problems are MCQ, of which we do not have complete textual information; indeed, the texts of the answers are not provided. Also, these

questions have a difficulty average b far from 0. R2DE tends to be "lazy" and makes predictions around 0 (i.e., the average of the difficulty), explaining why the error is more significant. We can see how ELMo performs slightly better for *fill_ in_ 1* than for *algebra*. While for the other models, the opposite happens. This fact could be due to ELMo's different text pre-processing.

			BERT	R2DE	DistilBERT	ELMo
Type	%	Avg. b	MAE	MAE	MAE	MAE
With "?"	43%	-0.312	0.894	0.937	0.911	0.926
Without "?"	57%	-0.359	0.918	0.986	0.906	0.921
			RMSE	RMSE	RMSE	RMSE
With "?"	43%	-0.312	1.182	1.234	1.194	1.242
Without "?"	57%	-0.359	1.260	1.356	1.253	1.249

Table 7.5. Models performance per question with and without "?".

Another feature that distinguishes the various items is the presence or absence of the question mark "?". Table 7.5 shows us the performance of question patterns with and without "?". BERT and R2DE predict more easily the difficulty of questions with "?", while DistilBERT and ELMo have almost the same performance regardless of the type. We also notice how R2DE performs much worse for questions without "?", even if they have similar average difficulty. A possible explanation for the problematic difficulty prediction with questions without "?" may derive from the fact that this type of question often refers to external images.

7.3.2 Cloud Academy

The Cloud Academy Q_{TEST} is made up of 1259 questions. All the questions are MCQ, and, differently from ASSISTments, the text of the possible choices is available. Thus, we can test the three input configurations described in Chapter 6. We also experiment with pre-training the Transformers-based models on MLM using the additional *Lectures* dataset.

Analysis of different configurations

Table 7.6 shows the results of DistilBERT and BERT with different configurations. The results with pre-training on MLM reported refer to the training of Transformer-based model for 24 epochs on the additional *Lectures* dataset.

Model	Input	MLM	MAE	RMSE
DistilBERT	Q only	N	0.805 ± 0.005	1.017 ± 0.005
DistilBERT	Q + cor.	N	0.799 ± 0.010	1.019± 0.017
DistilBERT	Q + all	N	0.794 ± 0.005	1.013 ± 0.007
BERT	Q only	N	0.807 ± 0.015	1.022 ± 0.020
BERT	Q + cor.	N	0.789 ± 0.010	0.999 ± 0.017
BERT	Q + all	N	0.811 ± 0.011	1.027 ± 0.013
DistilBERT	Q only	Y	0.802 ± 0.002	1.016 ± 0.002
DistilBERT	Q + cor.	Y	0.786 ± 0.006	0.994 ± 0.009
DistilBERT	Q + all	Y	0.794 ± 0.004	1.009 ± 0.005
BERT	Q only	Y	0.808 ± 0.010	1.020 ± 0.015
BERT	Q + cor.	Y	**0.773** ± 0.010	**0.978** ± 0.011
BERT	Q + all	Y	0.801 ± 0.015	1.014 ± 0.015

Table 7.6. Cloud Academy, results of BERT and DistilBERT.

The best performing model is BERT pre-trained on MLM using as input the question stem plus the text of the correct choice(s) (i.e., the Q + *cor.* configuration). The results show that the input configuration leading to better performance is Q + *correct* for all models except DistilBERT without pre-training, where Q + *all* leads to better performance. In general, we can say that the possible choices provide useful information that leads to an increase in performance. We can notice that the pre-training on MLM, under the same input configuration, increases the performance by up to 2% on the MAE and up to 1.9% on the RMSE.

The best configuration of the hyperparameters for both BERT and DistilBERT is the following:

- Sequence length = 256 for Q + *all*, 128 for others;
- Batch size = 16;
- Learning rate = 2e-5;
- Patience early stopping = 10 epochs;
- Dropout regressor layer = 0.5;
- Dropout internal = 0.5.

Comparison with the state of the art

Table 7.7 shows the results of the best configurations of the proposed Transformers-based models compared to the baselines. We can observe how the Transformers-based models perform better than the baselines, even without the additional pre-training

on MLM. Our best model—BERT with pre-training on MLM and *Q + correct* as input—reduces the MAE by 4.9% and the RMSE by 5.4% with respect to the best baseline, i.e., R2DE. We can see how all models have lower performance if only the question's stem is used. R2DE, unlike BERT and ELMo, performs better using the question and all the possible choices.

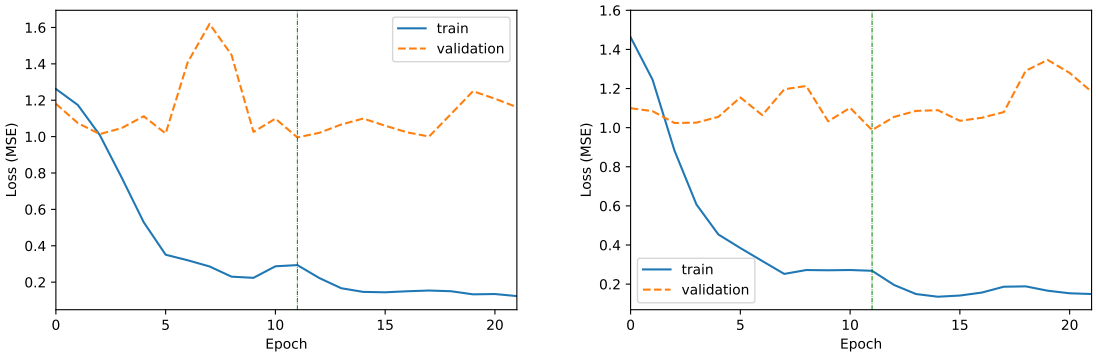
Model	Input	MLM	MAE	MAE, |b|>2
Majority	-	-	0.845 ± 0.000	2.527 ± 0.000
R2DE [6]	Q only	-	0.826 ± 0.001	2.397 ± 0.004
R2DE [6]	Q + cor.	-	0.819 ± 0.001	2.320 ± 0.005
R2DE [6]	Q + all	-	0.813 ± 0.001	2.331 ± 0.008
ELMo [70]	Q only	-	0.833 ± 0.002	2.286 ± 0.032
ELMo [70]	Q + cor.	-	0.831 ± 0.008	2.184 ± 0.033
ELMo [70]	Q + all	-	0.839 ± 0.004	2.213 ± 0.025
DistilBERT	Q + all	N	0.794 ± 0.005	2.203 ± 0.044
BERT	Q + cor.	N	0.789 ± 0.010	2.118 ± 0.130
DistilBERT	Q + cor.	Y	0.785 ± 0.007	2.078 ± 0.065
BERT	Q + cor.	Y	**0.773** ± 0.010	**2.044** ± 0.143
			RMSE	RMSE, |b|>2
Majority	-	-	1.069 ± 0.000	2.568 ± 0.000
R2DE [6]	Q only	-	1.051 ± 0.001	2.468 ± 0.005
R2DE [6]	Q + cor.	-	1.033 ± 0.002	2.391 ± 0.005
R2DE [6]	Q + all	-	1.034 ± 0.001	2.405 ± 0.008
ELMo [70]	Q only	-	1.053 ± 0.002	2.373 ± 0.025
ELMo [70]	Q + cor.	-	1.048 ± 0.010	2.276 ± 0.018
ELMo [70]	Q + all	-	1.057 ± 0.007	2.308 ± 0.015
DistilBERT	Q + all	N	1.013 ± 0.007	2.309 ± 0.036
BERT	Q + cor.	N	0.999 ± 0.017	2.222 ± 0.110
DistilBERT	Q + cor.	Y	0.994 ± 0.009	2.192 ± 0.058
BERT	Q + cor.	Y	**0.978** ± 0.011	**2.139** ± 0.121

Table 7.7. Cloud Academy results.

Analysis of the best performing model

We perform an analysis of the best performing model, BERT with pre-training on MLM, and compare it with the other models. First, we look at the difference between

the train and test performance. Then, we analyze the distribution of the predicted difficulties. Lastly, we look at what characteristics of the question text may have influenced the model error.



(a) Without further pre-training.

(b) With further pre-training.

Figure 7.7. Cloud Academy, model loss over training epochs.

Figure 7.7 plots the loss on the training and validation sets over training epochs of the BERT model and the BERT model with further pre-training respectively. As we can observe, the model with further pre-training shows a more stable validation loss, and also the training loss goes down faster. In these two training runs , the epoch with the lowest validation error is the same (but it's not guaranteed to always happen).

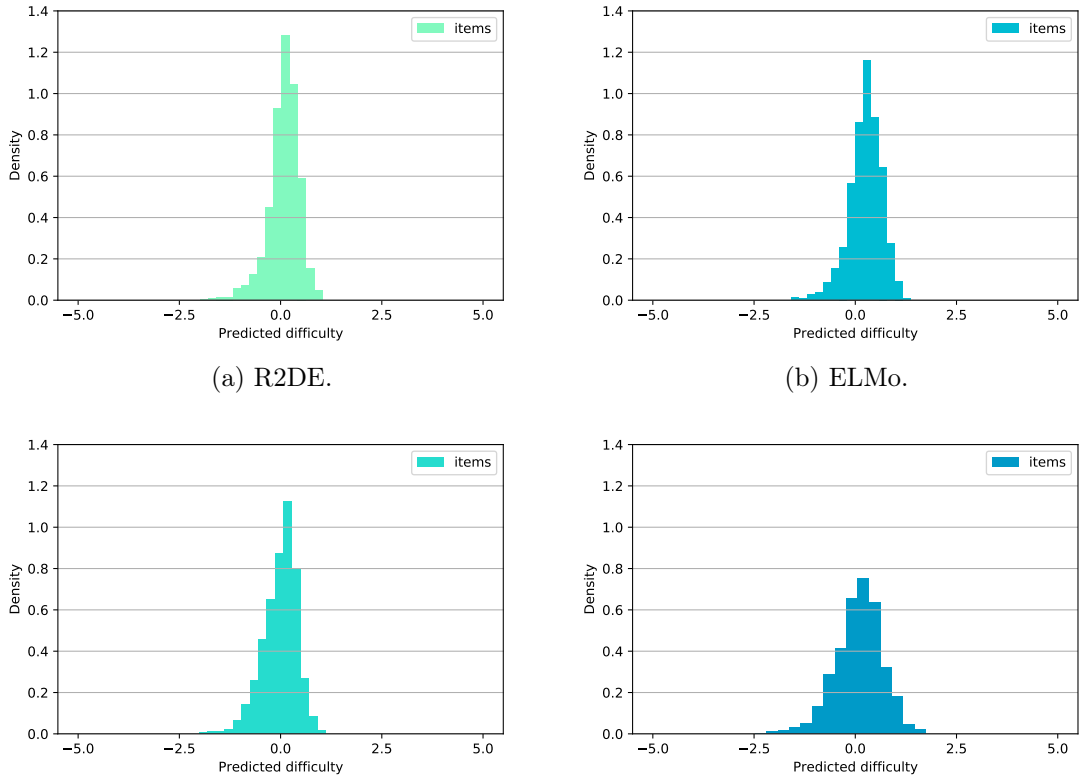
	Train	Test	
Model	MAE	MAE	Δ MAE
R2DE [6]	0.366 ± 0.002	0.813 ± 0.001	0.366
ELMo [70]	0.727 ± 0.013	0.831 ± 0.008	0.104
DistilBERT	0.643 ± 0.062	0.785 ± 0.007	0.142
BERT	0.364 ± 0.249	0.773 ± 0.010	0.409
	RMSE	RMSE	Δ RMSE
R2DE [6]	0.448 ± 0.002	1.034 ± 0.001	0.586
ELMo [70]	1.061 ± 0.007	1.048 ± 0.010	0.013
DistilBERT	0.897 ± 0.087	0.994 ± 0.009	0.097
BERT	0.547 ± 0.311	0.978 ± 0.011	0.521

Table 7.8. Train and test errors.

As we have already done for the ASSISTments dataset, we compare the models performance on the test set and on the training set (averaged over three runs). Table 7.8 shows the performance of the various models, indicating the difference between

the metrics on the training set and test set. We can see that BERT and R2DE have a much lower error on the training set than the test set. Furthermore, we can see a large standard deviation of BERT on the training set, as the early stopping has stopped the training at different epochs. This, however, led to similar results on the test set.

Figure 7.8 shows the distribution of test difficulties predicted by R2DE, ELMo, DistilBERT, and BERT (the target difficulty of the test set is shown in the Figure 7.9). All the predictions have a Gaussian distribution; however, we can notice how the distributions of R2DE, ELMo, and DistilBERT have a lower variance than that of BERT.



(a) R2DE.

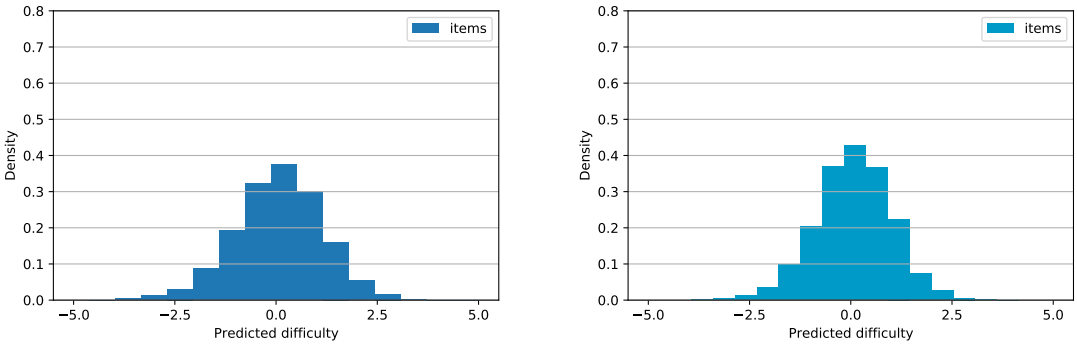
(b) ELMo.

(c) DistilBERT.

(d) BERT.

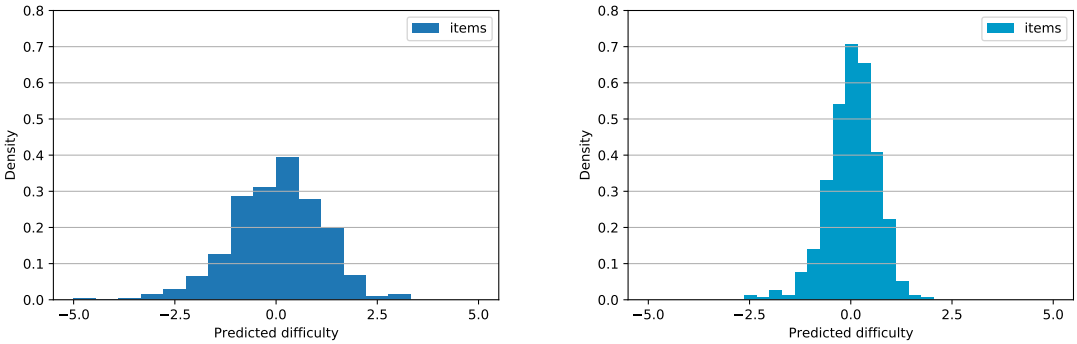
Figure 7.8. Test set, distribution of predicted difficulties.

Figure 7.9 shows the distribution of the target difficulties and the ones predicted by BERT across all splits of the dataset. We see that the target difficulties in the various splits are distributed similarly. We also note that BERT maintains the Gaussian distribution of the target but tends to predict difficulties closer to 0 compared to the target in the validation and test sets.



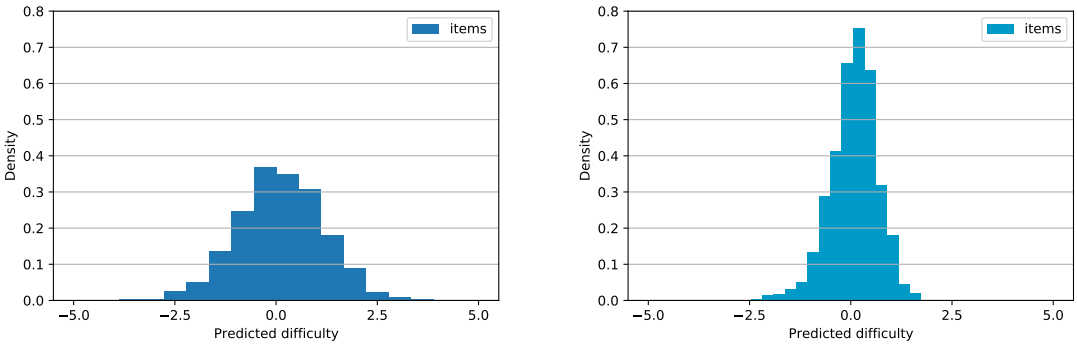
(a) Train target

(b) Train prediction



(c) Validation target

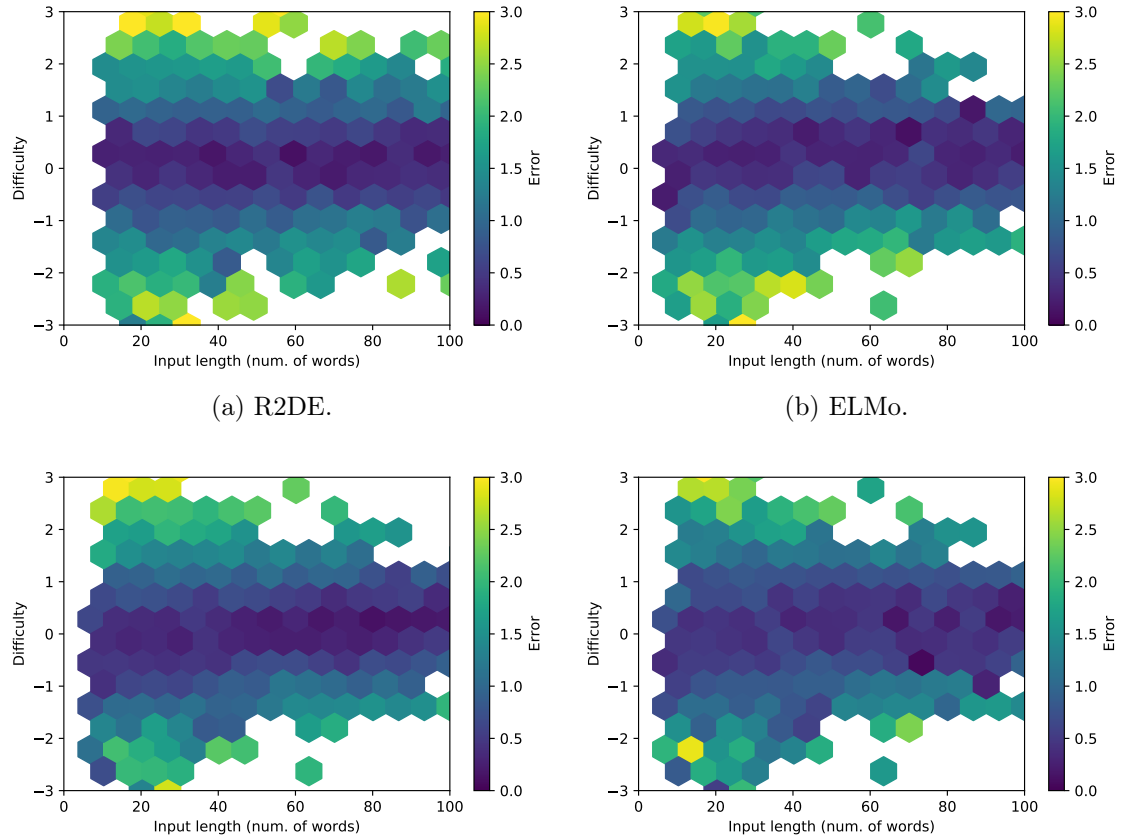
(d) Validation prediction



(e) Test target

(f) Test prediction

Figure 7.9. Distribution of the target difficulties and BERT predictions.



(a) R2DE.

(b) ELMo.

(c) DistilBERT.

(d) BERT.

Figure 7.10. Cloud Academy, error depending on the input length and true difficulty.

Figure 7.10 shows the dependence between true difficulty, the prediction error, and length of the questions (in terms of the number of words) in the test set. We consider questions with less than 100 words and target difficulty between -3 and 3 for better visualization (1067 samples over 1259 in the test set). For calculating the number of words, we use the best input configuration for each model: Q + *all* for R2DE and Q + *correct* for the others. For all models, the error increases as the target difficulty moves away from zero, showing how models tend to make predictions around zero. The models have at their disposal many examples of questions with difficulty around zero, and few for extreme difficulties. The graphs do not show an influence of the text length on the prediction error.

Cloze	%	Avg. b	BERT MAE	R2DE MAE	DistilBERT MAE	ELMo MAE
Y	18%	-0.144	0.809	0.892	0.836	0.891
N	82%	0.166	0.756	0.795	0.772	0.819
			RMSE	RMSE	RMSE	RMSE
Y	18%	-0.144	1.034	1.149	1.076	1.142
N	82%	0.166	0.958	1.007	0.976	1.019

Table 7.9. Models performance per question type.

In the Cloud Academy dataset, we have two types of questions: i) cloze questions, where the correct choice goes in place of an underscore in the stem of the question, ii) questions with a question mark at the end. The difficulty of cloze questions on average is lower than those with a question mark. 18% of the questions in the test set are cloze questions. Table 7.9 shows the performance of the models by type of question (cloze, or not cloze). It can be seen that all models show a more significant error on cloze-type questions. This can be due to several factors, including the lower number of samples. Another explanation could be the fact that these items are not close to the concept of "question" in natural language.

Digits	%	Avg. b	BERT MAE	R2DE MAE	DistilBERT MAE	ELMo MAE
Y	50%	0.191	0.758	0.796	0.783	0.830
N	50%	0.032	0.770	0.828	0.784	0.834
			RMSE	RMSE	RMSE	RMSE
Y	50%	0.191	0.976	1.027	1.004	1.051
N	50%	0.032	0.967	1.040	0.986	1.033

Table 7.10. Models performance per digits.

Table 7.10 shows the performance of the models based on the presence or absence of digits in the question text. Unlike the ASSISTments dataset, about half of the questions do not contain any numbers (for this reason, we report only the binary table and not the scatter plot with the percentage of digits per question). There is no clear pattern if the presence of numbers affects the prediction error.

# correct	%	Avg. b	BERT MAE	R2DE MAE	DistilBERT MAE	ELMo MAE
1	88%	0.099	0.773	0.823	0.788	0.836
> 1	12%	0.200	0.705	0.739	0.751	0.801
			RMSE	RMSE	RMSE	RMSE
1	88%	0.099	0.984	1.049	1.003	1.051
> 1	12%	0.200	0.879	0.920	0.927	0.980

Table 7.11. Models performance per number of correct choices.

The Cloud Academy dataset consists of MCQ with one or more correct choices. Table 7.11 shows us the performance of the models on the test set based on the number of correct choices to the question, only one or more than one. We can see that the models more easily predict questions with more than one choice. BERT and DistilBERT have good performance with questions with multiple choices, suggesting that the encoding used is not wrong. The encoding of the possible choices certainly deserves further attention. For instance, to better understand why R2DE obtains better performance by exploiting all possible choices.

Chapter 8

Conclusion

In this work, we have conducted a study about how pre-trained Transformers models, specifically BERT and DistilBERT, perform in the task of Question Difficulty Estimation (QDE) from text, and we have proposed a model that outperforms previous approaches by up to 6.5% on RMSE. We evaluated the models on two real-world datasets (one public and one private), using as ground truth Item Response Theory (IRT) difficulties. Transformers have become the de facto standard in several Natural Language Processing (NLP) tasks and confirmed their effectiveness in QDE.

We explored two approaches to train our model: with and without an additional pre-training on Masked Language Modeling (MLM). This further pre-training can increase the performance of the model by utilizing an additional dataset (e.g., books or lecture notes) related to the task domain. Previous approaches are either totally dependent on additional documents (i.e., they cannot work with the text of the questions only) or cannot leverage such information. Differently from them, our approach can work with or without a supplementary corpus of documents. The proposed model can outperform the state of the art approaches being trained only on the questions text and can be further improved if such an additional dataset is available. Specifically, we experimented with two pre-trained language models: BERT and DistilBERT. Both models proved to be better than the baselines, and the best performing model turned out to be the one based on BERT.

As an outcome of our study, we can say that: i) if an additional dataset if available, the pre-training on MLM improves the performance of Transformers model; ii) if the only available data is the text of the questions, DistilBERT might also be a good option, as it retains almost the same performance of BERT but at a fraction of the computational cost; iii) the possible choices help to estimate the difficulties as all the models show a larger error using only the stem of the questions.

We have analyzed which characteristics of the questions affect the prediction error of the models in our experiments. We have noticed that the error naturally increases as the magnitude of the difficulty increases and that R2DE, in particular, tends to predict more frequently difficulty close to zero. This is also due to the Gaussian distribution with zero mean of the training data as the models cannot see many questions that are very easy or very difficult. The length of the question does not seem to affect the prediction error of BERT and the other models. It has also been noticed that BERT has worse performance on cloze questions. One reason could be that this type of question is further away from natural language. Plus, we had fewer

samples available for these questions than those ending with a question mark.

Our contribution opens up several directions of research that can be explored in the future. A possible idea might be to improve the performance of BERT on cloze questions by using a different encoding. In fact, during the MLM training, what BERT tries to do is to predict the hidden word, which is the same objective of the cloze questions. The `[MASK]` token has the same function as the underscore. Therefore, the token could be reused in the fine-tuning on QDE instead of the underscore.

The model interpretability is very important to provide support to question creators. Unlike other models, Transformers are based on the Attention mechanism, which might explain which part of the question affects the difficulty. However, we are aware that Transformers models interpretation is not immediate, and there is ongoing research on it.

Another future experiment could involve using Question Answering as an auxiliary task: a first fine-tuning of BERT to answer the questions and then a second to perform QDE. In the literature, an attempt has been made to use response time prediction as an auxiliary task [70], but no one has used Question Answering. This approach requires no additional data apart from the question and the correct choice and could lead to increased performance.

In our case, we used an additional dataset—containing the transcripts of lessons related to the questions—to further pre-train the model on MLM. This further pre-training has led to an increase in performance, but it would be interesting to examine if it really is the best way to exploit such a dataset. Other approaches might use the additional dataset to see how many times the topic required by the question appears in the corpus in order to extract useful Information Retrieval (IR) features. We could compare which method between the two is the most effective way to leverage this additional information.

Acronyms

CTT	Classical Test Theory
IRT	Item Response Theory
1PL	One-Parameter Logistic
2PL	Two-Parameter Logistic
3PL	Three-Parameter Logistic
GMAT	Graduate Management Admission Test
ICC	Item Characteristic Curve
NLP	Natural Language Processing
TF-IDF	Term Frequency–Inverse Document Frequency
MLE	Maximum Likelihood Estimation
FFNN	Feed Forward Neural Network
MLP	Multi Layer Perceptron
RNN	Recurrent Neural Network
seq2seq	Sequence to Sequence Learning
LSTM	Long short-term memory
ReLU	Rectified Linear Unit
BPE	Byte Pair Encoding
SVM	Support-Vector Machines
BERT	Bidirectional Encoder Representations from Transformers
GLUE	General Language Understanding Evaluation
SQuAD	Stanford Q/A dataset
SVM	Support-Vector Machines
CQA	Community Question Answering
IR	Information Retrieval
MCQ	Multiple Choice Questions
LDA	Latent Dirichlet Allocation
OJ	Online Judge
KT	Knowledge Tracing

BKT	Bayesian Knowledge Tracing
NSP	Next Sentence Prediction
MLM	Masked Language Modeling
MSE	Mean Squared Error
RMSE	Root Mean Square Error
MAE	Mean Absolute Error
IRF	Item Response Function
TPU	Tensor Processing Unit
QDE	Question Difficulty Estimation

Bibliography

[1] Ghodai Abdelrahman and Qing Wang. Knowledge tracing with sequential key-value memory networks. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 175–184, 2019.

[2] Abir Abyaa, Mohammed Khalidi Idrissi, and Samir Bennani. Learner modelling: systematic review of the literature from the last 5 years. *Educational Technology Research and Development*, 67(5):1105–1143, 2019.

[3] Tahani Alsubait, Bijan Parsia, and Ulrike Sattler. A similarity-based theory of controlling mcq difficulty. In *2013 second international conference on e-learning and e-technologies in education (ICEEE)*, pages 283–288. IEEE, 2013.

[4] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

[5] Luca Benedetto, Andrea Cappelli, Roberto Turrin, and Paolo Cremonesi. Introducing a framework to assess newly created questions with natural language processing. *arXiv preprint arXiv:2004.13530*, 2020.

[6] Luca Benedetto, Andrea Cappelli, Roberto Turrin, and Paolo Cremonesi. R2de: a nlp approach to estimating irt parameters of newly generated questions. In *Proceedings of the Tenth International Conference on Learning Analytics & Knowledge*, pages 412–421, 2020.

[7] Denny Borsboom and Gideon J Mellenbergh. True scores, latent variables, and constructs: A comment on schmidt and hunter. *Intelligence*, 30(6):505–514, 2002.

[8] Hao Cen, Kenneth Koedinger, and Brian Junker. Learning factors analysis–a general method for cognitive model evaluation and improvement. In *International Conference on Intelligent Tutoring Systems*, pages 164–175. Springer, 2006.

[9] Hao Cen, Kenneth Koedinger, and Brian Junker. Comparing two irt models for conjunctive skills. In *International Conference on Intelligent Tutoring Systems*, pages 796–798. Springer, 2008.

[10] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[11] P. Chen, Y. Lu, V. W. Zheng, and Y. Pian. Prerequisite-driven deep knowledge tracing. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 39–48, 2018.

[12] Albert T Corbett and John R Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

[13] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei Fei Li. Imagenet: a large-scale hierarchical image database. pages 248–255, 06 2009.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Xinyi Ding and Eric C Larson. Why deep knowledge tracing has less depth than anticipated. *International Educational Data Mining Society*, 2019.

[17] William H DuBay. The principles of readability. *Online Submission*, 2004.

[18] Xitao Fan. Item response theory and classical test theory: An empirical comparison of their item/person statistics. *Educational and psychological measurement*, 58(3):357–381, 1998.

[19] Jiansheng Fang, Wei Zhao, and Dongya Jia. Exercise difficulty prediction in online education systems. In *2019 International Conference on Data Mining Workshops (ICDMW)*, pages 311–317. IEEE, 2019.

[20] Mingyu Feng, Neil Heffernan, and Kenneth Koedinger. Addressing the assessment challenge with an online system that tutors as it assesses. *User Modeling and User-Adapted Interaction*, 19(3):243–266, 2009.

[21] Thomas François and Eleni Miltsakaki. Do nlp and machine learning improve traditional readability formulas? In *Proceedings of the First Workshop on Predicting and Improving Text Readability for target reader populations*, pages 49–57, 2012.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[23] Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. *arXiv preprint arXiv:2007.15779*, 2020.

[24] Ronald K Hambleton and Russell W Jones. Comparison of classical test theory and item response theory and their applications to test development. *Educational measurement: issues and practice*, 12(3):38–47, 1993.

[25] Ronald K Hambleton, Hariharan Swaminathan, and H Jane Rogers. *Fundamentals of item response theory*. Sage, 1991.

[26] BA Hanson. Irt parameter estimation using the em algorithm (tech. rep.), 2000.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949.

[29] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[31] Zhenya Huang, Qi Liu, Enhong Chen, Hongke Zhao, Mingyong Gao, Si Wei, Yu Su, and Guoping Hu. Question difficulty prediction for reading problems in standard tests. In *AAAI*, pages 1352–1359, 2017.

[32] Chowdhury Md Intisar and Yutaka Watanobe. Cluster analysis to estimate the difficulty of programming problems. In *Proceedings of the 3rd International Conference on Applications in Information Technology*, pages 23–28, 2018.

[33] Matthew S Johnson et al. Marginal maximum likelihood estimation of item response models in r. *Journal of Statistical Software*, 20(10):1–24, 2007.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Ghader Kurdi, Bijan Parsia, and Uli Sattler. An experimental evaluation of automatically generated multiple choice questions from ontologies. In *OWL: Experiences And directions–reasoner evaluation*, pages 24–39. Springer, 2016.

[36] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[37] Jinseok Lee and Dit-Yan Yeung. Knowledge query network for knowledge tracing: How knowledge interacts with skills. In *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*, pages 491–500, 2019.

[38] Jing Liu, Quan Wang, Chin-Yew Lin, and Hsiao-Wuen Hon. Question difficulty estimation in community question answering services. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 85–90, 2013.

[39] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[40] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[41] Ye Mao. Deep learning vs. bayesian knowledge tracing: Student models for interventions. *Journal of educational data mining*, 10(2), 2018.

[42] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[43] Zachary A Pardos and Anant Dadu. Imputing kcs with representations of problem content and context. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 148–155, 2017.

[44] Thanaporn Patikorn, David Deisadze, Leo Grande, Ziyang Yu, and Neil Heffernan. Generalizability of methods for imputing mathematical skills needed to solve problems from texts. In *International Conference on Artificial Intelligence in Education*, pages 396–405. Springer, 2019.

[45] Radek Pelánek. Bayesian knowledge tracing, logistic models, and beyond: an overview of learner modeling techniques. *User Modeling and User-Adapted Interaction*, 27(3-5):313–350, 2017.

[46] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[47] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In *Advances in neural information processing systems*, pages 505–513, 2015.

[48] Zhaopeng Qiu, Xian Wu, and Wei Fan. Question difficulty prediction for multiple choice problems in medical exams. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 139–148, 2019.

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

[50] Georg Rasch. Studies in mathematical psychology: I. probabilistic models for some intelligence and attainment tests. 1960.

[51] Lawrence Rudner. *Implementing the Graduate Management Admission Test Computerized Adaptive Test*, pages 151–165. 01 2010.

[52] Alper Sahin and Duygu Anil. The effects of test length and sample size on item parameters in item response theory. 2017.

[53] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[54] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[55] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[56] Thomas Sergent, François Bouchet, and Thibault Carron. Towards temporality-sensitive recurrent neural networks through enriched traces.

[57] Stefan Slater, Ryan Baker, Ma Victoria Almeda, Alex Bowers, and Neil Heffernan. Using correlational topic modeling for automated topic identification in intelligent tutoring systems. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*, pages 393–397, 2017.

[58] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

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

[60] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[62] S Vijayarani, Ms J Ilamathi, and Ms Nithya. Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, 5(1):7–16, 2015.

[63] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[64] Tianqi Wang, Fenglong Ma, and Jing Gao. Deep hierarchical knowledge tracing. In *Proceedings of the 12th International Conference on Educational Data Mining*, 2019.

[65] Xiaojing Wang, James O Berger, Donald S Burdick, et al. Bayesian analysis of dynamic item response models in educational testing. *The Annals of Applied Statistics*, 7(1):126–153, 2013.

[66] Peter Willett. The porter stemming algorithm: then and now. *Program*, 2006.

[67] Kevin H Wilson, Yan Karklin, Bojian Han, and Chaitanya Ekanadham. Back to the basics: Bayesian extensions of irt outperform neural networks for proficiency estimation. *arXiv preprint arXiv:1604.02336*, 2016.

[68] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[69] Liangbei Xu and Mark A Davenport. Dynamic knowledge embedding and tracing. *arXiv preprint arXiv:2005.09109*, 2020.

[70] Kang Xue, Victoria Yaneva, Christopher Runyon, and Peter Baldwin. Predicting the difficulty and response time of multiple choice questions using transfer learning. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 193–197, 2020.

[71] Victoria Yaneva, Peter Baldwin, Janet Mee, et al. Predicting the difficulty of multiple choice questions in a high-stakes medical exam. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 11–20, 2019.

[72] Victoria Yaneva et al. Automatic distractor suggestion for multiple-choice tests using concept embeddings and information retrieval. In *Proceedings of the thirteenth workshop on innovative use of NLP for building educational applications*, pages 389–398, 2018.

[73] Victoria Yaneva, Constantin Orăsan, Richard Evans, and Omid Rohanian. Combining multiple corpora for readability assessment for people with cognitive disabilities. Association for Computational Linguistics, 2017.

[74] Chun-Kit Yeung. Deep-irt: Make deep learning based knowledge tracing explainable using item response theory. *arXiv preprint arXiv:1904.11738*, 2019.

[75] Chun-Kit Yeung and Dit-Yan Yeung. Addressing two problems in deep knowledge tracing via prediction-consistent regularization. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, pages 1–10, 2018.

[76] Jiani Zhang, Xingjian Shi, Irwin King, and Dit-Yan Yeung. Dynamic key-value memory networks for knowledge tracing. In *Proceedings of the 26th international conference on World Wide Web*, pages 765–774, 2017.

[77] Liang Zhang, Xiaolu Xiong, Siyuan Zhao, Anthony Botelho, and Neil T Heffernan. Incorporating rich features into deep knowledge tracing. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*, pages 169–172, 2017.

[78] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. A comparative study of tf* idf, lsi and multi-words for text classification. *Expert Systems with Applications*, 38(3):2758–2765, 2011.

[79] Wayne Xin Zhao, Wenhui Zhang, Yulan He, Xing Xie, and Ji-Rong Wen. Automatically learning topics and difficulty levels of problems in online judge systems. *ACM Transactions on Information Systems (TOIS)*, 36(3):1–33, 2018.