KIPS Trans. Softw. and Data Eng.
Vol.11, No.8 pp.315~324
ISSN: 2287-5905 (Print), ISSN: 2734-0503 (Online)

https://doi.org/10.3745/KTSDE.2022.11.8.315

Analyzing Korean Math Word Problem Data Classification Difficulty Level Using the KoEPT Model

Rhim Sangkyu† · Ki Kyung Seo†† · Kim Bugeun††† · Gweon Gahgene††††

ABSTRACT

In this paper, we propose KoEPT, a Transformer-based generative model for automatic math word problems solving. A math word problem written in human language which describes everyday situations in a mathematical form. Math word problem solving requires an artificial intelligence model to understand the implied logic within the problem. Therefore, it is being studied variously across the world to improve the language understanding ability of artificial intelligence. In the case of the Korean language, studies so far have mainly attempted to solve problems by classifying them into templates, but there is a limitation in that these techniques are difficult to apply to datasets with high classification difficulty. To solve this problem, this paper used the KoEPT model which uses 'expression' tokens and pointer networks. To measure the performance of this model, the classification difficulty scores of IL, CC, and ALG514, which are existing Korean mathematical sentence problem datasets, were measured, and then the performance of KoEPT was evaluated using 5-fold cross-validation. For the Korean datasets used for evaluation, KoEPT obtained the state-of-the-art(SOTA) performance with 99.1% in CC, which is comparable to the existing SOTA performance, and 89.3% and 80.5% in IL and ALG514, respectively. In addition, as a result of evaluation, KoEPT showed a relatively improved performance for datasets with high classification difficulty. Through an ablation study, we uncovered that the use of the 'expression' tokens and pointer networks contributed to KoEPT's state of being less affected by classification difficulty while obtaining good performance.

Keywords : Math Word Problems, Generation Model, Transformer, Pointer Network, Classification Difficulty

KoEPT 기반 한국어 수학 문장제 문제 데이터 분류 난도 분석

임 상 규† · 기 경 서†† · 김 부 근††† · 권 가 진††††

요 약

이 논문에서는 자연어로 구성된 수학 문장제 문제 자동 풀이하기 위한 Transformer 기반의 생성 모델인 KoEPT를 제안한다. 수학 문장제 문제는 일상 상황을 수학적 형식으로 표현한 자연어 문제이다. 문장제 문제 풀이 기술은 함축된 논리를 인공지능이 파악해야 한다는 요구사항을 지녀 최근 인공지능의 언어 이해 능력을 증진하기 위해 국내외에서 다양하게 연구되고 있다. 한국어의 경우 문제를 유형으로 분류하여 풀이하는 기법들이 주로 시도되었으나, 이러한 기법은 다양한 수식을 포괄하여 분류 난도가 높은 데이터셋에 적용하기 어렵다는 한계가 있다. 본 논문은 이에 대해 '식' 토큰과 포인터 네트워크를 사용하는 KoEPT 모델을 사용했다. 이 모델의 성능을 측정하기 위해 현존하는 한국어 수학 문장제 문제 데이터셋인 IL, CC, ALG514의 분류 난도를 측정한 후 5겹 교차 검증 기법을 사용하여 KoEPT의 성능을 평가하였다. 평가에 사용된 한국어 데이터셋들에 대하여, KoEPT는 CC에서는 기존 최고 성능과 대등한 99.1%, IL과 ALG514에서 각각 89.3%, 80.5%로 새로운 최고 성능을 얻었다. 뿐만 아니라 평가 결과 KoEPT는 분류 난도가 높은 데이터셋에 대해 상대적으로 개선된 성능을 보였다. KoEPT가 분류 난도의 영향을 덜 받으며 좋은 성능을 얻게 된 이유를 '식' 토큰과 포인터 네트워크 때문이라는 것을 ablation study를 통해서 밝혔다.

키워드 : 수학 문장제 문제, 생성 모델, 트렌스포머, 포인터 네트워크, 분류 난도

1. 서 론

수학 문장제 문제(math word problem)는 수리적 문제 해결이 필요한 상황을 자연어로 기술한 수학 문제이다. 이런 문제들에서는 자연어 문장 내에 풀어야 하는 수리적 문제 상황과 연관된 수식의 내용이 함축되어 있다. 따라서 문제를 풀기 위해서는 주어진 문장들을 올바르게 이해하여 문장의 구성 요소들을 적절하게 추출하고 수식에 대응시킬 수 있어야 한다. 최근 자연어 이해 분야의 인공지능 연구자들은 수학 문장제 문제를 풀기 위해 이러한 요구조건에 주목하여, 인공지능이 수학 문장제 문제를 풀게 하는 데 많은 관심을 기울이고

※ 이 성과는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단 및 정보통신기획평가원의 지원을 받아 수행된 연구임(No.2020R1C1C1010162, No.2021-0-02146).
※ 이 논문은 2021년 한국정보처리학회 춘계학술발표대회에서 "KoEPT: Transformer 기반 생성 모델을 사용한 한국어 수학 문장제 문제 자동 풀이"의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 서울대학교 지능정보융합학과 석사과정
†† 비 회 원 : 서울대학교 지능정보융합학과 박사과정
††† 비 회 원 : 서울대학교 인공지능혁신인재양성교육연구단 연수연구원
†††† 종신회원 : 서울대학교 지능정보융합학과 교수
Manuscript Received : June 29, 2021
First Revision : January 17, 2022
Second Revision : April 26, 2022
Accepted : April 26, 2022
* Corresponding Author : Gweon Gahgene(ggweon@snu.ac.kr)

있다. 그 이유는 이렇게 자연어 문장을 이해하여 올바른 수식을 구성하는 과정이 인간의 추론(reasoning) 능력과 밀접한 관련이 있기 때문이다. 즉 인공지능이 수학 문장제 문제를 이해하여 풀 수 있게 된다는 것은, 모델이 자연어로 기술된 임의의 문제 상황을 추상화하여 해결할 수 있는 일종의 추론 능력을 보유하게 된다는 것을 의미한다.

이때 수학 문장제 문제에서 인공지능 모델이 문장 속에 함축된 내용을 이해했다고 말하기 위해서는 모델이 문제에 대응되는 올바른 방정식을 제시할 수 있어야 한다. 일반적으로 대수 영역을 다루는 수학 문장제 문제들은 피연산자와 연산자로 구성된 방정식으로 기술된다. 이 방정식을 올바르게 세우기 위해, 현재까지 보고된 국외의 선행 연구들은 크게 분류(classification)모델과 생성(generation)모델이라는 방법들을 사용하였다.

먼저 분류 모델이란, 데이터셋의 각 데이터들을 미리 설정된 템플릿들 중 한 범주에 속하도록 인공지능 모델이 분류하는 법을 학습하는 모델이다. 이때 템플릿이란, 문제들로부터 도출된 방정식들 중 연산자의 종류 및 피연산자들의 연산 순서는 동일하지만 문제의 맥락에 맞도록 숫자만 서로 다르게 나타나는 사례들을 한데 모아 개별 숫자들을 슬롯으로 바꿔 하나의 '유형'으로 추상화한 것을 가리킨다. 그러나 현재 공개된 데이터셋들을 살펴보면 유형이 과도하게 많거나, 유형별 분포가 유사하거나 균일하지 않아 분류가 어려운 데이터셋이 많다. 분류 모델은 이렇게 분류 난도가 높은 데이터셋에서 학습하기 어려울 수 있다. 또한 학습과정에서 제공된 유형 이외의 유형이 나타났을 때 이를 예측할 수 없다는 한계를 가진다.

반면 생성 모델은 학습 데이터의 템플릿(template)에 의존하지 않고 문제에 있는 피연산자와 연산자들을 모델이 직접 찾아내서 수식을 생성하는 방식이다. 이러한 특성 때문에 분류 난도가 높아 분류 모델로 학습하기 어려운 데이터에 대해 적절히 대처할 수 있다. 또한 생성 모델은 훈련 데이터에 등장하지 않는 데이터의 수식을 예측해야 하는 상황에서도 유연하게 대처할 수 있다. 분류 모델은 기존에 학습한 템플릿들 사이에서만 수식을 예측할 수밖에 없지만 생성 모델은 새로운 수식을 작성을 할 수가 있다. 따라서 기존 데이터셋들에 대해서는 템플릿 기반의 분류 모델을 사용하는 대신 생성(generation) 방식, 즉, 방정식을 직접 세울 수 있는 방식을 쓰는 것이 더 적합하다. 하지만 생성 모델은 토큰 단위로 수식 자체를 생성해야 한다는 더 어려운 과업을 부여받게 되므로, 분류 모델에 비해 학습이 어렵다는 난점이 있다.

하지만 현재까지 보고된 국내의 선행 연구들은 분류 모델에 관한 것이었다[1, 2]. 즉 한국어에서는 생성 모델에 대한 연구가 전무한 실정이다. 중국어권에서는 이미 중국어 수학 문장제 문제 생성 모델들로 다양한 시도를 활발하게 하고 있다[3, 4]. 또한, 중국어권에서는 이런 생성모델들을 통합하여 사용할 수 있는 툴킷(toolkit)도 개발하고 있다[5]. 더 나아가 영어권에서는 영어 생성 모델들이 생성한 풀이 자체의 품질을 측정하는 시도를 하고 있다[6]. 이에 이 연구에서는 최근 영어에서의 연구 성과를 기반으로 분류 모델 기반의 기존 한국어 모델들의 성능을 능가하는 새로운 생성 모델로서 KoEPT를 제안하고 이 모델에 대한 실험을 통해 그 성능을 검증하고자 한다. 나아가, 이 연구에서는 KoEPT의 구조가 어떤 방식으로 성능에 기여하는지에 대해서도 확인해 볼 것이다. 이를 위해 본 연구에서는 EPT에 대한 추가적인 실험으로 Ablation study를 수행하며, 본 실험 결과와 데이터셋의 특성과 비교한 추가적인 분석을 통하여 모델의 구조에 대한 이해를 얻고자 한다.

2. 선행 연구

본 절에서는 그동안 시도되었던 수학 문장제 문제 풀이 연구에 대한 맥락을 (1)초기와 (2)후기로 나눠서 소개한다. 그 후, (3)분류 모델과 생성 모델을 비교하여 설명하고 (4)생성 모델의 최신 연구 사례인 EPT에 대해 개괄한다.

먼저 최근 연구 사례들은 크게는 초기 연구, 후기 연구로 분류할 수 있다. 초기의 연구들은 문제 풀이에 도움이 될 자질을 직접 사람이 추출하는 방법으로 모델을 학습시켰다[1, 7-10]. 한국어의 경우 대표적으로, 사람이 직접 추출한 자질을 이용하여 학습을 시도했던 모델은 [1]에서 제시됐다. [1]의 연구는 의존 구문 분석과 언어 유형적 특성이 고려된 문맥 자질들을 인위적으로 미리 추출하고자 하였다. [1]은 대표적인 벤치마크 데이터셋인 ALG514[7]를 한국어로 번역한 후 Log-linear 기반의 통계 모델로 학습을 수행하였다. Table 1은 ALG514에 등장하는 문제, 수식, 템플릿에 대한 예시다.

하지만 이런 방법을 채택한 모델들은 사람이 추출할 자질을 사전에 정해줘야 모델이 학습할 수 있다는 것이 단점이다. 이러한 단점을 극복하고자 후기 연구들은 순수 신경망을 사용하는 연구들을 제안하였다.

최근 연구의 순수 신경망 모델들은 자질을 자동으로 추출할 수 있도록 설계됐다. 대표적으로 [2]에서 BERT[11]라는 사전 학습 언어 모델(Pre-Trained Language Model)을 이용하여 문제들을 방정식 템플릿으로 분류한 순수 신경망 기반의 분류 모델인 KoTAB을 만들었다. 이 모델은 BERT의 세계 지식(world knowledge)을 사용함으로써 수동으로 추

Table 1. Example of Math Word Problem

Question	A cosmetologist has a bottle of 7% hydrogen peroxide solution and a bottle of 4% hydrogen peroxide solution. The cosmetologist needs 300 milliliters of a 5% hydrogen peroxide solution. Find how many milliliters of each solution the cosmetologist needs to mix together.
Equation	$0.07\times x+0.04\times y=0.05\times 300$ $x+y=300$
Template	$a\times x+b\times y=c\times d$ $x+y=c$
Answer	$x=100,\ y=200$

출한 자질 없이도 기존 연구에 비해 더 나은 성능을 얻었다.

KoTAB과 같은 분류 모델(classification model)은 기존에 학습된 템플릿 중에서만 모델의 출력을 선택할 수 있다. 따라서 새로운 유형이 등장할 경우, 동일한 수학 도메인의 문제라 하더라도 해당 문제에 적합한 새로운 템플릿을 만들 수 없다. KoTAB 역시 분류 모델 방식을 차용하고 있으므로 템플릿의 개수가 많고 각 템플릿별 데이터의 수가 적을 경우 각 템플릿에 대한 충분한 학습이 어렵다.

이러한 분류 모델의 단점을 보완하는 방안은 생성 모델(generation model)을 사용하는 것이다. 생성 모델은 기존 템플릿 하나를 선택하는 대신 추론 과정마다 적합하다고 판단하는 연산자 혹은 피연산자를 선택해 템플릿을 직접 생성해나간다. 이렇게 되면 모델의 학습 방향이 기존에 학습한 템플릿과의 유사성을 찾는 것이 아니라, 문장에 등장하는 토큰들을 이용하여 적절한 수식의 구성 요소(연산자, 피연산자)를 생성하는 쪽으로 달라진다. 그 결과 모델은 새로운 유형의 데이터에 대해서도 대응하는 출력을 생성할 수 있게 된다. 이에 따라 새로운 템플릿을 생성할 수 없는 분류 모델의 단점을 보완한다.

생성 기반의 신경망 모델에 대한 연구 중에서 영어 수학 문장제 문제를 다룬 최고 성능 모델은 Expression Pointer Transformer(EPT)[12]다. EPT는 Transformer [13]를 기반으로 설계된 인코더-디코더 (Encoder- Decoder) 모델이다. EPT의 인코더는 ALBERT[14]라는 사전 학습 언어 모델을 통해 수학 문장제 문제의 언어적 특질(feature)들을 자동으로 학습할 수 있다. 한편 디코더는 문제에 대응되는 수식을 인코더를 통해 습득한 언어 정보들을 바탕으로 생성한다. 본 연구는 영어에서 최고 성능을 보고한 바 있는 EPT 모델을 한국어에 적용한 KoEPT를 제안함으로써 한국어에서도 영어와 유사한 수준의 수학 문장제 문제 풀이의 성능을 확보하고자 한다.

3. 방법론

본 연구에서 제안하는 KoEPT는 한국어 수학 문장제 문제를 자동으로 풀기 위해 만들어진 모델이다. KoEPT는 EPT [8]에서 제안된 구조를 바탕으로 한국어에 맞게 개선된 생성 기반 모델이다. KoEPT는 EPT와 마찬가지로 Transformer의 구조처럼 인코더-디코더로 나뉘어 있으며, Figure 1에서 이 구조를 볼 수 있다. Fig. 1은 "한 숫자가 다른 수의 두 배보다 여덟이 더 크고 이 두 수의 합은 20이다. 두 수를 구하시오."라는 문제가 KoEPT를 거쳐 정답이 도출되는 과정을 표현한 예시다. KoEPT의 구조를 설명하기 위해서 크게 전처리, 인코더, 디코더, 그리고 출력단, 4개 부분으로 나누어서 순차적으로 다루겠다.

KoEPT에 한국어 수학 문장제 문장을 입력으로 받고 전처리 과정을 거친다. 전처리 과정에서는 문제의 문장들 안에서 등장하는 한국어 수사 표현을 아라비아 숫자로 변환한다. 이를 위해서 한국어 수사 표현과 그에 대응하는 숫자를 미리 사전 형태로 정의하였다. 예를 들어, '두'와 '여덟'처럼 문제에서 수량적의 의미를 담고 있는 글자들을 '2'와 '8'로 변환한다. 이 전처리 과정을 거치는 이유는 기존 데이터셋이 한국어로 번역되는 과정에서 특정 아라비아 숫자들이 고유 수사 표현으로 변화되어 문장의 뉘앙스가 달라지는 경우가 있었기 때문이다. 그래서 일관적인 학습을 위해 해당 수사 표현들을 아라비아 숫자로 대응시키는 사전을 정의하여 변환시켰다.

전처리 과정에서 숫자 변환 후, 문장은 인코더에서 토큰화 과정을 거친다. 토크나이저로는 BERT와 관련 모델들에서 사용하는 WordPiece 토크나이저를 사용한다. 이 토크나이저를 한국어에 적용하는 방법으로는 형태소 단위의 방법과 어절 단위 방법의 두 가지가 존재한다[15]. 그 중에서 본 연구에서는 어절 단위의 토크나이징을 선택했다. 그 이유는 문제를 구성하는 문장들이 하나의 맥락을 형성하고 있기에, 개별 단어 단위를 구성하는 요소들의 의미를 강조하는 형태소 단위보다 어절 단위가 더 적합하다고 판단하였기 때문이다. 또한 어절 단위 토크나이징을 사용하는 과정에서 숫자와 붙어있던 조사들은 독립적으로 분리가 된다. 한국어의 교착어적 특성을 고려했을 때, 이렇게 분리된 조사들은 문법적으로 주

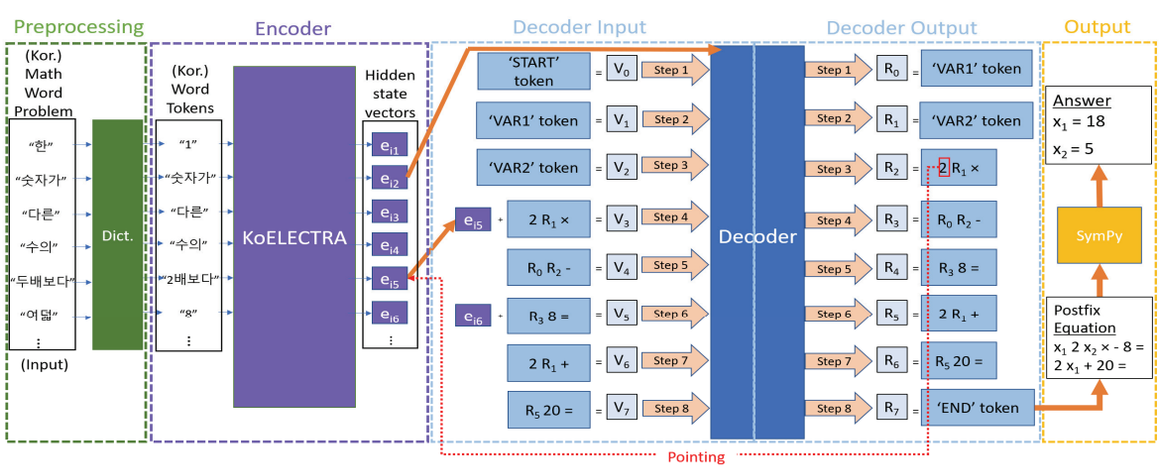

Fig. 1. Model Diagram of KoEPT

변 단어에 대한 정보를 제공해줄 수 있다.

다음으로 인코더 층에서 대규모 말뭉치로 구축된 사전 지식을 활용하기 위해 한국어 사전 학습 언어 모델을 사용한다. 언어 모델을 통해서 얻게 된 입력 문제에 대한 은닉 상태 벡터들은 디코더에서 메모리로 제공이 된다. EPT에서는 ALBERT를 사용하였으나, 본 연구에서는 KoELECTRA[16]를 사용한다. 그 이유는 두 가지가 있다. 첫째로는 최근 ELECTRA[17]는 기존의 EPT에서 사용됐던 ALBERT 등의 BERT 계열 사전 학습 언어 모델보다 여러 downstream task에 높은 성능을 낸다고 보고하였다. 둘째로 한국어로 개발된 ALBERT 계열 모델인 KB-ALBERT[18]가 적합하지 않았기 때문이다. 이 모델은 전문 영역에 해당하는 금융 관련 코퍼스로 학습되어 사전 지식을 활용하고자 하는 본 태스크의 의도에 부적합하다고 판단되었다.

디코더는 입력 부분과 출력 부분으로 구성된 여러 '단계(step)'의 연산을 거쳐 수식을 출력한다. 단계의 개수는 KoEPT에 입력된 수학 문장제 문제에 따라 결정된다. 예를 들자면 Fig. 1의 수학 문장제 문제 입력에 대해서는 총 여덟 단계를 거쳤다. 디코더 입력 부분은 항상 첫 번째 단계를 '방정식 시작'을 의미하는 토큰(START)을 받는 것으로 시작한다. 입력받은 문제의 변수 개수에 따라 두 번째 아니면 세 번째 단계까지는 '변수'를 나타내는 토큰(VAR)을 생성하게 된다. 이후 단계적으로 새로운 '식' 토큰을 생성한다. '식' 토큰은 주어진 문제를 방정식 형태로 나타낸 것이다. '식' 토큰 하나에는 덧셈, 뺄셈, 곱셈, 나눗셈, 등호 중 한 가지의 연산자와 연산자의 계산에 필요한 2개의 피연산자를 포함하고 있다. 각 단계에서는 직전 단계에서 디코더가 스스로 생성한 '식' 토큰들을 입력값으로 받으며, 마지막 단계에서 '방정식 끝'을 나타내는 토큰(END)을 디코더가 스스로 생성할 때까지 이 과정이 반복된다.

각 단계에서 디코더 입력 부분은 다음과 같이 Equation (1)로 표현할 수 있다.

$$V_i = FeedFoward(Concat(a_{i1}, a_{i2}, f_i)) \quad (1)$$

여기서 'FeedFoward'는 선형 피드 포워드 연산(Linear Feed Foward)을 의미하고 Concat은 괄호 안에 있는 모든 벡터를 한 차원에 대하여 이어준다. 그리고 aij는 i번째 '식' 토큰의 피연산자 임베딩 값을 나타내고 fi는 i번째 '식' 토큰의 연산자의 임베딩 값을 의미한다. 피연산자 임베딩 값은 피연산자의 종류에 따라 결정된다. 피연산자의 종류는 3가지다. (1) 첫 번째 종류의 피연산자는 문제에서 직접 등장하는 숫자다. 이 종류의 피연산자를 가진 '식' 토큰은 해당 피연산자와 직접 대응될 것으로 예상되는 인코더 은닉 벡터(hidden state vectors)와 합하여 디코더의 출력 부분(Output)으로 보내진다. (2) 두 번째 종류의 피연산자는 과거 단계의 '식' 토큰들이다. 이는 Fig. 1에서 'R'로 시작하는 기호로 표현하였다. 이 종류의 피연산자를 가진 '식' 토큰은 과거 단계의 출력으로 생성된 은닉 벡터를 현 단계 '식' 토큰과 합하여 디코더의 출력 부분으로 보내진다. (3) 세 번째 종류의 피연산자는 문제에 숫자 형태로 등장하지 않지만, 수식에는 포함되어야 하는 숫자이다. 예를 들자면, 백분율을 구하는 문제를 구하는 경우 문제에서 직접 언급하지 않지만, 문제에 직접 등장하는 숫자에 1/100을 곱해야 한다. 문제에 직접 1/100이 등장하지 않지만, 연산에는 필수적이기 때문에 피연산자로 등장해야 한다. 이런 종류의 숫자들에 대응하기 위해서 사전에 테이블을 만들어서 이런 종류의 숫자들의 후보군을 만들어서 각각 고정된 벡터를 정의한다. 이 종류의 피연산자를 가진 '식' 토큰은 사전에 정의된 테이블에 해당 숫자를 탐색한 후 그 벡터와 합하여 디코더의 출력 부분으로 보내진다.

각 단계에서 디코더 출력 부분은 다음에 올 것으로 예상하는 연산자-피연산자 묶음을 동시에 출력한다. 우선 연산자의 경우 다음과 같은 식을 통해서 다음에 올 연산자를 예측하여 출력한다.

$$f_{i+1} = \arg\max_f \sigma(f \mid FeedFoward(h_i)) \quad (2)$$

여기서 $\sigma(f|x)$는 소프트맥스(softmax) 함수 $\sigma(x)$의 분포에 대하여 연산자 f를 선택하게 될 조건부확률을 나타내고 hi는 i번째(직전 단계) '식' 토큰의 은닉 벡터를 의미한다.

피연산자는 다음과 같은 방식으로 attention 값을 계산하여 다음에 올 피연산자들을 출력한다.

$$Q_{ij} = FeedFoward_Q(h_i) \quad (3)$$

$$K_{ij} = FeedFoward_K(A_{ij}) \quad (4)$$

$$a_{i+1,j} = \arg\max_a \sigma(a \mid \frac{Q_{ij}K_{ij}^{\top}}{\sqrt{Dim}}) \quad (5)$$

여기서 Qij는 attention의 query 벡터를 의미하고 Kij는 key 벡터이다. Dim은 차원을 나타낸다. Qij는 이전 단계의 '식' 토큰을 이용하여 계산한다. Kij를 계산하기 위해서 Aij가 사용되는데, 이는 다음 피연산자들의 후보를 담고 있는 행렬이다. 이 행렬은 예측되는 다음 '식' 토큰의 종류에 따라 결정된다. 이 예측도 디코더 출력 부분에서 함께 한다. 그리고 Equation (5)를 보면 일반적인 attention 구조와 달리 max 함수가 있다. 이는 문맥적 정보를 반영하기 위해 '포인터 네트워크(Pointer Network)'[19]를 사용하기 때문이다. 이를 사용하여 디코더는 현 단계에서 예측되는 피연산자의 종류에 따라 피연산자와 대응하는 것으로 예상하는 인코더에서 받은 은닉 상태 벡터 혹은 이전에 생성한 '식' 토큰들을 중에서 후보 벡터를 선택한다. 이렇게 선택된 벡터와 현 단계의 '식' 토큰을 합하여 다음 단계의 입력으로 사용될 수 있는 상태로 출력이 된다. Fig. 1에서 이런 '포인팅'을 이전에 출력된 토큰들과 문제의 단어를 가리키는 빨간 점선 화살표들로 나타냈다. Fig. 1의 예시에서는 세 번째 단계에서 디코더가 '2'라는 피연산자가 문제에서 직접 등장하는 숫자라고 예측하였다. 이에 따라 디코더의 출력 부분은 네 번째 단계에서 해당 피연산자

를 포함한 '식' 토큰과 합칠 인코더의 은닉 상태 벡터를 '포인팅'하게 된다. 이 은닉 상태 벡터는 '2배보다'를 토큰화한 것이다. 네 번째 단계의 디코더 입력 부분은 포인팅 이 은닉 상태 벡터를 가져와서 연산을 진행한다. 가독성을 위해 세 번째 단계에서 일어나는 포인팅까지만 Fig. 1에 나타나게 하였다.

마지막으로 출력단에서는 디코더에서 후위(postfix) 표현식으로 출력된 '식' 토큰들을 종합하여 수식을 파싱(parsing)한 후 정답 수식과 비교한다. 출력단에서 토큰들을 종합하는 방법은 '식' 토큰끼리 맞물리게 하여 풀어쓰는 것이다. Fig. 1의 예시를 들자면 일곱 번째 단계의 R6 '식' 토큰은 R5 '식' 토큰을 피연산자로 갖고 있기 때문에 R6의 '식' 토큰은 후위 표현식으로 '2 R1 + 20 ='으로 풀어쓸 수 있다. 나머지 '식' 토큰들도 같은 방법으로 수식을 풀어쓴다. Fig. 1처럼 일원연립방정식의 문제는 두 수식을 그대로 연결하여 최종적으로 출력한다. 이 종합된 수식을 가지고 파이썬 라이브러리인 SymPy[20]를 이용하여 정답이 도출된다.

4. 데이터셋

본 연구에서 KoEPT의 성능 측정 및 선행 연구 결과와 비교하기 위해, 공개된 영어 데이터셋인 IL, CC, ALG514를 [1]과 [2]에서 한국어로 번역한 것을 실험에 사용하였다. IL은 [21]에서 제작한 데이터셋으로 사칙연산으로 이루어진 562개의 일차방정식 문제를 포함하고 있다. CC는 [22]에서 commoncoresheets.com라는 웹 사이트로부터 600개의 문제를 수집 및 구축한 데이터셋이다. ALG514는 [7]에서 제작한 데이터셋이고 algebra.com라는 웹 사이트에서 수집한 문제 514개로 구성되어 있다. 모든 문제는 사칙연산으로 이루어진 일차방정식 혹은 이원일차연립방정식 문제들이다.

5. 분류 난도 척도

선행 연구들을 살펴보면 분류 및 생성 모델 사이의 성능 차이가 발생하는 것을 확인할 수 있다. 본 연구에서는 그 원인이 학습 데이터셋의 분류 난도 때문일 것이라고 추정한다. 이를 입증하기 위해서는 분류 난도를 측정할 수 있는 객관적인 척도가 필요하다. 기존 수학 문장제 문제 데이터셋들에 대해 지금까지 정량적인 난도 측정 결과를 보고한 사례가 없었으므로, 본 연구에서는 실험 결과를 심층적으로 분석하기 위하여 각 데이터셋에 대해 '분류 난도'를 추가적으로 정량적인 방법을 도입하여 측정하였다.

분류 난도 척도에 관한 연구 사례로는 [23]이다. [23]은 데이터셋의 분류 난도를 결정하는 요소들을 제시했다. 또한 분류 난도에 대한 종합적인 측정 척도를 제안했다. 이 척도는 Shannon 다양성 인덱스[24], Hellinger 유사도[25], Distinct 단어 빈도 비율, Imbalance 인덱스[23]를 모두 종합하여 도출되는데, 각 수치는 다음과 같은 의미가 있다.

1) Shannon 다양성 (Shannon Diversity)

Shannon 다양성 S는 데이터셋 전체의 다양성을 나타내는 지표이다. 이 지표가 높을수록 특정 분류에서 데이터가 발생했는지를 예측하기 어렵다는 것을 의미한다.

$$S = -\frac{1}{\ln(m)}\sum_{i}^{m} P_i \ln(P_i) \tag{6}$$

Pi는 데이터셋에 존재하는 총 m개의 분류 중에서 i번째 분류에 데이터가 포함될 확률을 의미한다. 1/ln(m)을 곱한 이유는 해당 지표를 정규화시키기 위함이다.

2) Hellinger 유사도 (Hellinger Similarity)

Hellinger 유사도 H는 데이터셋에 존재하는 분류들의 유사성 하방(lower bound)을 측정하는 척도이다. 여기서 유사도는 두 분류의 L2 거리를 이용한다. 가장 유사하지 않은 두 분류의 유사도가 높은 데이터셋은 대부분의 분류가 유사하다고 해석할 수 있다. 따라서 이 지표가 높을수록 모델이 분류 간 구분이 어렵다고 해석할 수 있다.

$$H = \min_{i,j}\left(1 - \frac{1}{\sqrt{2}} \left| \sqrt{P_i} - \sqrt{P_j} \right|_2\right) \tag{7}$$

Pi는 데이터셋의 m개의 분류 중에서 i번째 분류에 데이터가 포함될 확률을 의미한다. 두 분류의 확률이 쌍이 되어 유사도를 구한 후에 모든 쌍의 유사도 중에서 가장 낮은 것을 데이터셋의 유사도 H로 정한다.

3) Distinct 단어의 빈도 비율 (Distinct word Ratio)

Distinct 단어의 빈도 비율 D는 전체 단어 수 대비 고유 단어의 수를 나타낸다. 데이터셋 내에서 쓰이는 단어의 종류가 다양할수록 모델이 데이터의 분류를 예측할 때 쓰일 수 있는 개별 단어들의 정보량이 감소한다. 따라서 이 지표가 높을수록 모델이 단어들로부터 얻을 수 있는 분류에 대한 정보가 적어져 분류하기 어려운 데이터셋이라고 해석할 수 있다.

$$D = \frac{(No.\ of\ distinct\ words)}{(No.\ of} \tag{8}$$

4) Imbalance 인덱스 (Imbalance Index)

Imbalance 인덱스 I는 데이터셋의 균일하지 않은 정도를 나타낸다.

$$I = \sum_{i}^{m} \left| \frac{1}{m} - \frac{n_i}{T_{Data}} \right| \tag{9}$$

여기서 m은 총 분류의 개수를 의미하고 ni는 i번째 분류에 해당하는 데이터의 개수이다. TData는 데이터 내에 존재하는 모든 데이터의 개수를 의미한다. 만약 데이터셋의 m개의 분류가 모두 같은 양의 데이터를 가지고 있다면 데이터 내에 존재하는 모든 데이터 수 TData는 분류의 개수 m과 각

분류에 있는 데이터 수 ni의 곱으로 나타낼 수 있다. 따라서 이처럼 완벽히 균일한 데이터셋의 I는 0의 값을 가진다. 균일하지 않은 데이터셋일수록 분류를 구분하기 어렵기 때문에 이 지표가 높으면 모델이 데이터에 대한 분류를 결정하기 어렵다는 것을 의미한다.

[23]에서는 이 네 가지 수치를 합산(S+H+D+I)하여 데이터셋의 분류 난도의 측정값을 얻을 수 있다고 보고했다. 수학 문장제 문제 데이터셋에도 각 데이터셋의 템플릿 개수나 각 템플릿당 문제 개수의 편차가 난도 차이를 유발할 것이므로, 우리는 [23]의 방식을 이용하여 IL, CC, ALG514의 분류 난도를 측정하였다. 이 종합 수치를 데이터셋 별로 계산하면 데이터셋 사이의 상대적인 분류 난도를 비교할 수 있게 된다. 해당 측정 기준에 따른 난도는 Table 2와 같다.

Table 2를 통해 ALG514가 데이터셋 중에서 분류 난도가 가장 높다는 것을 확인할 수 있다. 이렇게 난도 차이가 발생하는 이유는 세 데이터셋의 총 데이터 수는 IL는 562개, CC는 600개, ALG514는 514개이지만 IL과 CC는 템플릿의 개수가 12개이고 ALG514는 25개로 데이터 수 대비 템플릿의 개수의 비율이 ALG514가 높기 때문이다. 한국어 데이터셋 분류 난도의 차이를 고려한다면 모델의 성능이 이에 영향을 받을 수 있기 때문에 각 데이터셋 별 실험 결과를 해석할 필요가 있다.

6. 실 험

본 연구에서 제안한 모델인 KoEPT에 대한 분석을 위해 여러 실험을 하였다. 6.1에서는 KoEPT의 성능을 확인하는 실험을 소개한다. 6.2에서 KoEPT의 주요 부분인 '식' 토큰과 포인터 네트워크가 각각 성능에 미치는 영향을 확인하는 실험을 제시한다.

6.1 실험 1: 성능 비교 실험

KoEPT의 성능이 선행 연구의 수준을 능가할 수 있는지를 검증하기 위해 성능 비교 실험을 수행하였다. 이에 대해 본 절에서는 1) 해당 실험의 수행 과정을 제시할 것이며, 2) 이 실험을 통해 얻은 성능 결과를 제시할 것이다.

Table 2. Difficulty of Korean Math Word Problem Datasets

	CC	IL	ALG514
Difficulty	2.60	3.32	4.26
Shannon Diversity	1.41	1.56	2.71
Hellinger Similarity	0.92	0.74	0.60
Distinct word Ratio	0.01	0.04	0.19
Imbalance Index	0.26	0.98	0.76

1) 성능 비교 실험 방법

본 연구에서는 기존의 한국어 수학 문장제 문제 풀이 모델들과 제안하는 모델인 KoEPT의 성능 비교를 하고자 한다. 이를 위해 공개된 한국어 데이터셋인 IL, CC, ALG514를 가지고 실험을 수행하였다. 하지만 이 데이터셋들이 train/valid/test split을 제공하지 않는 관계로, 각 데이터셋별로 5겹 교차 검증(5-fold cross-validation)을 이용하여 KoEPT의 학습을 진행하였다.

학습 후 선행 연구들과의 성능 비교를 위해서는 기존의 보고 방식과 동일하게 KoEPT가 출력한 식을 통해 도출한 답을 실제 답과 비교하여 산출한 정답률을 평가 척도로 사용한다. KoEPT가 사용한 하이퍼파라미터(hyperparameter)들은 학습 에폭(epoch), 배치 크기(batch size), 사전 준비 에폭(warm-up epoch), 학습률(learning rate), 디코더층 개수를 제외하고는 기본적으로 KoELECTRA와 동일하다. 모든 데이터셋에 대하여 100 학습 에폭을 사용하였고, 배치 크기는 2048로 고정했다. 사전 준비 에폭과 학습률은 그리드 탐색(grid search) 알고리즘을 이용하여 탐색했다. 사전 준비 에폭은 5, 10, 15, 20, 25 중에서 최적의 성능을 돌려주는 것을 사용했다. 학습률은 0.00125, 0.00176, 0.0025 중에서 최적을 선택했다. 디코더층의 개수도 마찬가지로 2, 3, 4, 5 중에서 최적을 사용하였다. 그리드 탐색은 각 데이터셋의 첫 번째 겹(fold 0)의 훈련 집합(training set)과 검증 집합(validation set)에만 사용하여 탐색을 진행했다. 모든 실험은 6개의 32GB RAM과 2개의 Quadro RTX 8000 GPU가 탑재된 서버에서 실험되었다.

2) 성능 비교 실험 결과

Table 3는 본 연구에서 진행한 실험 결과와 선행 연구에서 보고된 성능을 비교한 표이다. Table 3을 보면, 본 연구에서 제안하는 생성 모델인 KoEPT가 기존에 제안된 분류 모델 기반의 선행 연구들과 비교했을 때 CC에서는 대등한 수준의 성능을, IL과 ALG514에서는 선행 연구들에 비해 더 높은 성능을 보였다는 것을 확인할 수 있다.

특히 분류 난도를 고려하였을 때, KoEPT는 분류 난도가 높은 데이터셋일수록 분류 모델보다 더 나은 성능을 보였다. CC의 경우 분류 난도가 가장 낮은 데이터셋이었기에 소폭 낮았지만 오차 범위 내에서 거의 대등한 성능을 낸 반면, CC에 비해 더 난도가 높은 것으로 측정된 IL과 ALG514에서는 KoEPT가 더 나은 성능을 내는 것이 확인되었다.

Table 3. Experiment Results

	IL	CC	ALG514
Log-linear	-	-	78.6 (0.5)
KoTAB	85.2	99.3	-
KoEPT (Proposed)	89.3 (0.7)	99.1 (0.3)	80.5 (1.2)

* Numbers within the parentheses () are standard deviations

이 실험을 통해서 생성 모델인 KoEPT가 높은 성능을 가지고 있다는 것을 확인할 수 있다. 하지만 데이터셋별로 KoEPT의 성능 향상 폭이 상이하므로, 향후 개선을 위해서는 KoEPT의 어떤 부분이 성능 향상에 기여하는지를 구체적으로 분석할 필요가 있다. 이를 확인하기 위해 본 연구에서는 실험 2의 ablation study를 통해 KoEPT의 각 부분별 성능 향상의 기여도를 확인하고 그 결과를 보고할 것이다.

6.2 실험 2: Ablation study

KoEPT의 방정식 생성 능력에 영향을 주는 두 부분은 연산자와 피연산자를 동시에 생성하는 '식' 토큰과 문제 텍스트에 등장하는 숫자를 직접 지칭해줄 수 있도록 하는 포인터 네트워크다. 이 두 요소가 중요한 이유는 모델이 연산자와 피연산자를 예측하여 수식을 생성하는 과정에 이 두 요소들이 직접 관여하기 때문이다. 따라서 이 두 부분의 성능 기여도를 확인하기 위해서는, 이들이 없을 때 모델의 성능을 측정하는 Ablation study를 진행해야 한다. 또한 데이터셋의 분류 난도별로 '식' 토큰과 포인터 네트워크이 성능에 다르게 영향을 미칠 수 있으므로 이 부분에 대해서도 분석해야 한다. 이에 대해 본 절에서는 1) 해당 Ablation study를 진행한 방법에 대해서 서술할 것이며, 2) 그 결과를 확인해볼 것이다.

1) Ablation study 실험 방법

우리는 Ablation study를 하기 위해 진행한 추가 실험에서는 '식' 토큰과 포인터 네트워크를 둘 다 제외하거나 포인터 네트워크만 제외한 하위 모델을 제작했다. 첫 번째 하위 모델은 순수 Transformer로 피연산자와 연산자를 쌍으로 묶어서 학습하게 도와주는 '식' 토큰 대신 연산자와 피연산자를 분리하여 학습한다. 그리고 디코더에서 포인터 네트워크 대신 일반 attention 방식으로 다음 연산자 혹은 피연산자를 예측한다. 두 번째 하위 모델은 '식' 토큰을 쓰되 포인터 네트워크는 여전히 안 쓰는 모델이다. 이 모델은 연산자와 피연산자를 '식' 토큰으로 묶어서 학습하지만, 여전히 디코더에서는 일반 attention을 쓴다.

6.1과 마찬가지로 KoEPT를 포함한 세 모델의 성능을 IL, CC, ALG514 모든 데이터셋에서 측정하였다. 이 세 데이터셋에 대해 KoEPT와 그 하위 모델의 성능을 측정하여 데이터셋의 분류 난도와 KoEPT의 '식' 토큰과 포인터 네트워크와의 관계에 대해서 분석 할 수 있게 되었다. 이 실험에서 쓰이는 모델들의 하이퍼파라미터와 실험 환경은 6.1 실험과 동일한 조건과 탐색 알고리즘을 사용하였다.

2) Ablation study 실험 결과

Ablation study의 실험 결과는 Table 4를 통해 확인할 수 있다.

Table 4를 통해 확인할 수 있듯이, '식' 토큰과 포인터 네트워크의 존재 여부에 따라 성능 차이가 발생했다. 특히 데이터셋의 분류 난도에 따라 포인터 네트워크의 성능에 대한 기여도가 달라지는 양상이 나타났다. 상대적으로 쉬운 데이터셋인 CC에서는 포인터 네트워크를 추가할 시의 성능 상승 폭이 약 3% 정도였다. 또한 IL에서는 약 10%, 그리고 가장 어려운 데이터셋인 ALG514에서는 약 33%의 성능 향상이 있었다.

Table 4. Ablation Experiment Results

	CC	IL	ALG514
Vanilla Transformer	80.3 (15.3)	77.2 (2.8)	42.0 (1.4)
+ Expression	96.8 (1.0)	79.3 (1.7)	47.9 (1.1)
+ Pointer Net.	99.1 (0.3)	89.3 (0.7)	80.5 (1.2)

* Numbers within the parentheses () are standard deviations

7. 토 의

실험 결과, KoEPT는 예상과 같이 선행 연구의 결과들과 대등하거나 더 나은 성능을 보여주었다는 점에서 더 우수한 모델이라고 볼 수 있다. KoEPT의 성능이 검증됨에 따라, 본 연구에서는 KoEPT의 각 부분의 기여도를 심층적으로 검토하고자 한다. 이를 확인하기 위해, 우리는 선행 연구에서 제안한 모델과 KoEPT가 각각 출력한 결과를 IL과 ALG514 데이터셋에서 뽑아내어 추가적인 사례 분석을 수행하여 ablation study의 결과와 함께 논의할 것이다. 이때 사례 분석 대상으로 삼은 각각의 사례들 중에서 CC 데이터셋의 경우는 기존 모델과 KoEPT의 학습 성능이 모두 100% 가까이 나왔기에 분석 대상에서 제외했다.

7.1 '식' 토큰

먼저 KoEPT의 '식' 토큰은 연산자와 피연산자가 동시에 생성되는 하나의 단위로 기능하므로, 피연산자와 연산자의 관계를 디코더의 추론 과정 동안 유지할 수 있게 해 준다는 특성이 있다. 이러한 특성은 다른 모델들에 비해 KoEPT에서 새로운 숫자가 입력되었을 때, 이 숫자를 모델이 다른 연산자의 피연산자로 오인할 확률이 줄어들도록 하는 것으로 보인다. 이에 대한 예시는 Table 5에서 확인할 수 있다. Table 5는 다음과 같은 문제에 대한 예시다: '어느 회사에서는 극세사 소파를 빌릴 때보다 가죽 소파를 빌릴 때 월 30원이 더 든다. 수정이는 8개월 동안 가죽 소파를 빌린 후 극세사 소파로 교환하여 12개월 동안 빌렸다. 수정이는 소파 임대료로 총 1620원을 지불했다. 가죽 소파와 극세사 소파의 임대료를 각각 구하여라.' 이 예시에서, 피연산자와 연산자를 묶는 '식' 토큰의 효과를 확인해 볼 수 있다.

예시를 보면 숫자 '30'은 기존의 Log-Linear 모델에서 올바르지 않게 배치되고 있다. '30'은 '가죽 소파의 임대료'를 의미하는 x1과 '극세사 소파의 임대료'를 의미하는 x2의 차이를 나타내는 숫자로 쓰여야 하나, Log-linear 모델에서는 덧셈 연산자의 피연산자로 오인되었다. 이로 인해 두 수식이 하나의 수식으로 예측되어 모델은 오답을 계산하였다. 하지

Table 5. Example 1: Effect of Expression Token

Log-linear vs. KoEPT	Dataset: ALG514
	Expected Equation: $x_1 - x_2 = 30$ $12 \times x_1 + 8 \times x_2 = 1620$
	Log-linear's Equation: (Incorrect) $12 \times x_1 + 30 = 1620$
	KoEPT's Equation: (Correct) $x_1 - x_2 = 30$ $12 \times x_1 + 8 \times x_2 = 1620$

Table 6. Example 2-1: Effect of Pointer Network

Log-linear vs. KoEPT	Dataset: ALG514
	Expected Equation: $x_1 + x_2 = 27$ $0.5 \times x_1 + 0.3333 \times x_2 = 11$
	Log-linear's Equation: (Incorrect) $x_1 + x_2 = 27$ $0.5 \times x_1 + 2 = 11 \times x_2$
	KoEPT's Equation: (Correct) $x_1 + x_2 = 27$ $0.5 \times x_1 + 0.3333 \times x_2 = 11$

Table 7. Example 2-2: Effect of Pointer Network

KoTAB vs. KoEPT	Dataset: IL
	Expected Equation: $49 - 44 = x_1$
	KoTAB's Equation: (Incorrect) $\square - 44 = x_1$
	KoEPT's Equation: (Correct) $49 - 44 = x_1$

만 '식' 토큰을 이용한 KoEPT에서는 연산자와 피연산자의 관계가 유지되기 때문에, 원래 두 식의 모양이 유지되어 올바른 답이 도출되었다.

7.2 포인터 네트워크

또한 KoEPT가 채용한 포인터 네트워크는 모델이 피연산자에 대해서 학습을 할 때 포인터 네트워크를 통해서 문제의 원문 텍스트에 등장하는 숫자와 직접 대응하는 피연산자를 선택하도록 학습이 된다. 이러한 방식을 채용하면 피연산자를 탐색할 때 발생하는 오류가 적어질 수 있다는 이점이 있다. 실제로도 실험에 따른 출력 결과를 분석한 결과, KoEPT는 피연산자를 혼동하거나 찾지 못하는 경우가 기존 모델에 비해 적다는 것이 확인되었다. 특히 이러한 포인터 네트워크의 특성은 상대적으로 난도가 높은 데이터셋인 IL, ALG514에서 KoEPT의 성능 향상에 크게 기여한 것으로 보인다. Table 6과 Table 7의 사례를 통해 이런 모습을 확인할 수 있다.

Table 6는 다음과 같은 문제에 대한 예시다: '두 수의 합은 27이다. 첫 번째 수의 0.5배에 두 번째 수의 0.3333배를 더한 값은 11이다. 작은 수와 큰 수를 각각 구하여라.' 이 예시는 기존 모델인 Log-Linear에서 피연산자와 연산자를 잘못 묶어서 수식을 작성한 사례이다. 두 번째 곱셈 연산자와는 숫자 '0.3333'이 묶여 있어야 하지만 등호 연산자와 묶여 있어야 할 '11'이 대신 그 자리에 있다. 또한, 등호 연산자의 피연산자로 문제에 등장하지만, 수식 작성에는 불필요한 '2'가 사용되고 있다. 이는 문제에 등장하는 '두'라는 글자 때문에 발생한 문제라고 볼 수 있다. 기존의 분류 모델이 사전 학습된 데이터에 존재하는 템플릿으로만 새로운 문제를 분류할 수 있다는 점을 고려한다면, 이러한 사례는 모델이 수식을 자신이 학습한 템플릿에 맞추기 위해서 덧셈의 두 번째 피연산자와 문제에서 '두'에 해당하는 숫자인 '2'를 대응시킨 사례일 수 있다.

Table 7는 다음과 같은 문제에 대한 예시다: '수정이의 몸무게는 49 kg이다. 정욱이의 몸무게는 44 kg이다. 수정이는 정욱이보다 얼마나 무거운지 구하여라.' 이 예시는 기존 모델인 KoTAB이 피연산자를 아예 찾지 못한 대표 사례이다. KoTAB은 이 사례에서 뺄셈에 대한 첫 번째 피연산자로 '49'를 선택하지 못하고 첫 번째 피연산자 없이 계산을 수행했다. Table 7에서 이를 사각형 모양 빈 칸으로 표현하였다. 특히 KoTAB의 경우에는 데이터셋 중 IL의 학습 과정에서 피연산자 대응과 관련된 오류가 자주 발생했는데, KoEPT는 포인터 네트워크를 사용함으로써 이런 오류들이 줄어들어 더 두드러진 성능 향상을 나타낸 것으로 판단된다.

또한 ablation study의 실험 결과에서도 확인할 수 있듯이, 어려운 데이터셋일수록 포인터 네트워크의 성능 기여도가 컸는데, 이런 추세의 원인을 파악하기 위해 본 연구에서는 분류 난도를 구성하는 지표 중 분류 난도의 가장 큰 비중을 차지하는 Shannon 다양성 수치와 비교해보고자 하였다. Table 8을 통해 데이터셋별로 분류 난도를 구성하는 각 지표의 결과값을 확인할 수 있다. Table 8에서 확인할 수 있듯이, 포인터 네트워크가 크게 기여하는 IL과 ALG514는 Shannon 다양성이 CC에 비해 상대적으로 높게 나타나는 데이터셋이다. 이 값이 높다는 것은 데이터셋 안에서 문제의 구성 방식이 더 다채롭고 피연산자들이 등장하는 위치의 편차가 크다는 것을 의미한다. 따라서 이를 통해 포인터 네트워크가 예측하기 어려운 문제들에서 피연산자들의 위치가 어떻게 파악되어야 하는지를 학습하면서 이에 해당되는 숫자들을 대응시켜줘 성능 향상에 현저하게 기여했다는 것을 추측할 수 있다.

Table 8. Details of Dataset Difficulty

	CC	IL	ALG514
Shannon Diversity	1.41	1.56	2.71
Vanilla Transformer	80.3 (15.3)	77.2 (2.8)	42.0 (1.4)
+ Expression	96.8 (1.0)	79.3 (1.7)	47.9 (1.1)
+ Pointer Net.	99.1 (0.3)	89.3 (0.7)	80.5 (1.2)

8. 결론 및 한계점

본 논문은 순수 신경망 모델로 한국어 수학 문장제 문제를 해결하기 위해서, 분류 난도가 높은 데이터셋을 학습할 수 있는 KoEPT를 제안했다. KoEPT는 인코더-디코더 구조로 되어 있는 생성 모델이다. KoEPT의 성능을 평가하기 위해서 본 논문에서는 한국어 데이터셋 IL, CC, ALG514를 이용하여 KoEPT의 성능을 확인했다. 또한 각 데이터셋의 분류 난도에 따라 발생하는 KoEPT의 성능 변화도 함께 확인했다. 확인 결과, KoEPT는 IL과 ALG514에서는 최고 성능을 냈고, CC에서 최고 성능과 비슷한 성능을 냈다. 이를 통해 KoEPT는 상대적으로 분류 난도에 영향을 적게 받는다는 것을 확인했다.

본 논문의 기여점은 다음과 같다:

1. 한국어 수학 문장제 문제를 푸는 첫 생성 모델인 KoEPT를 개발했다.
2. KoEPT는 기존 한국어 데이터셋인 CC에서는 최고 성능에 대등한 99.1%, IL(85.2%에서 89.3%)과 ALG514(78.6%(±0.5))에서는 최고 성능을 보였다.
3. Ablation study를 통하여, KoEPT에서도 EPT와 마찬가지로 '식' 토큰과 포인터 네트워크가 효과적이라는 것이 확인되었다.
4. 본 연구에서는 성능과 더불어 데이터의 난도를 정량적으로 확인할 수 있도록, 정보 이론에 기반한 분류 난도의 측정 방식을 통해 분류 난도와 성능 비교를 통해 KoEPT의 특성을 심층적으로 파악할 수 있었다.

KoEPT가 분류 난도에 영향을 적게 받는 이유는 '식' 토큰과 포인터 네트워크 활용하기 때문이라는 것을 Ablation study를 통해 분석했다. 여기서 이 특징들 때문에 KoEPT는 난이도별로 상이하게 나타나는 데이터셋의 특성에 서로 다른 방식으로 대응할 수 있다는 것을 확인했다.

그러나 본 연구에서 실험했던 데이터셋들이 모두 사칙연산에 국한된 데이터셋이다. 따라서 KoEPT가 더 복잡한 문제를 학습할 수 있는지는 확인하기 어렵다. 또한 현재 한국어 문장제 문제 풀이 모델들의 종류가 다양하지 않아, 복잡한 문제들을 풀이할 수 있는 모델의 성능 비교 분석이 제한된다. 이런 점에서 KoEPT가 앞으로 더 다채로운 수식을 포괄하고 있는 수학 문장제 문제에도 활용될 수 있을 것으로 기대된다.

References

[1] C. Woo and G. Gweon, "solving automatically algebra math word problem in Korean," *Annual Conference on Human and Language Technology*, pp.310-315, 2018.

[2] K. Ki, D. Lee, and G. Gweon, "KoTAB: Korean template-based arithmetic solver with BERT," *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp.279-282, 2020.

[3] J. Zhang, L. Wang, R. K. Lee, Y. Bin, Y. Wang, J. Shao, and E. Lim, "Graph-to-Tree learning for solving math word problems," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp.3928-3937, 2020.

[4] J. Zhang, R. K. Lee, E. Lim, W. Qin, L. Wang, J. Shao, and Q. Sun, "Teacher-Student networks with multiple decoders for solving math word problem," *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI-20, pp.4011-4017, 2020.

[5] Y. Lan et al., "MWPToolkit: An open-source framework for deep learning-based math word problem solvers," [Internet], https://github.com/LYH-YF/MWPToolkit

[6] D. Hendrycks et al., "Measuring mathematical problem solving with the MATH dataset," *35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks*. 2021.

[7] N. Kushman, Y. Artzi, L. Zettlemoyer, and R. Barzilay, "Learning to automatically solve algebra word problems," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, Vol. 1: Long Papers, pp.271-281, 2014.

[8] D. Zhang, L. Wang, L. Zhang, B. T. Dai, and H. T. Shen, "The gap of semantic parsing: A survey on automatic math word problem solvers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.42, No.9, pp.2287-2305, 2019.

[9] S. Roy and D. Roth, "Mapping to declarative knowledge for word problem solving," *Transactions of the Association for Computational Linguistics*, Vol.6, pp.159-172, 2018.

[10] L. Zhou, S. Dai, and L. Chen, "Learn to solve algebra word problems using quadratic programming," *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp.817-822, 2015.

[11] J. D. Kenton, M. W. Chang, and L. K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of NAACL-HLT*, pp.4171-4186. 2019.

[12] B. Kim, K. Ki, D. Lee, and G. Gweon, "Point to the expression: Solving algebraic word problems using the expression-pointer transformer model," *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp.3768-3779, 2020.

[13] A. Vaswani et al., "Attention is all you need," *Advances in Neural Information Processing Systems*, pp.5998-6008, 2017.

[14] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," *International Conference on Learning Representations*, 2019.

[15] J. Lim, H. Kim, and Y. Kim, "Recent R&D trends for pre-trained language model," *Electronics and Telecommunications Trends*, Vol.35, No.3, pp.9-19, 2020.
[16] J. Park, Pretrained ELECTRA Model for Korean [Internet], https://github.com/monologg/KoELECTRA.
[17] K. Clark, M. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training text encoders as discriminators rather than generators," *International Conference on Learning Representations*, 2019.
[18] D. Lee, J. Park, and S. Oh, "KB-ALBERT" [Internet], https://github.com/KB-AI-Research/KB-ALBERT
[19] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in Neural Information Processing Systems*, Vol.28, pp.2692-2700, 2015.
[20] A. Meurer et al., "SymPy: Symbolic computing in Python," *PeerJ Computer Science*, Vol.3, 2017.
[21] S. Roy, T. Vieira, and D. Roth, "Reasoning about quantities in natural language," *Transactions of the Association for Computational Linguistics*, Vol.3, pp.1-13, 2015.
[22] S. Roy and D. Roth, "Solving General Arithmetic Word Problems," In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp.1743-1752, 2015.
[23] E. Collins, N. Rozanov, and B. Zhang, "Evolutionary data measures: Understanding the difficulty of text classification tasks," *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pp.380-391, 2018.
[24] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol.5, No.1, pp.3-55, 2001.
[25] L. Le Cam and G. L. Yang, "Asymptotics in statistics: Some basic concepts," Springer Science and Business Media, 2012.

임 상 규

https://orcid.org/0000-0003-3709-6950
e-mail : sk.rhim@snu.ac.kr
2020년 연세대학교 경제학과(학사)
2020년 ~ 현 재 서울대학교 지능정보융합학과 석사과정
관심분야 : Natural Language Processing, Human-Computer Interaction

기 경 서

https://orcid.org/0000-0002-9866-0052
e-mail : kskee88@snu.ac.kr
2013년 서울대학교 미학과(학사)
2016년 서울대학교 미학과(석사)
2017년 ~ 현 재 서울대학교 지능정보융합학과 박사과정
관심분야 : Natural Language Processing

김 부 근

https://orcid.org/0000-0002-7771-4103
e-mail : cd4209@snu.ac.kr
2013년 서울대학교 수학교육과(학사)
2016년 KAIST 웹사이언스대학원(석사)
2022년 서울대학교 융합과학기술대학원 (박사)
2022년 ~ 현 재 서울대학교 인공지능혁신인재양성교육연구단 연수연구원
관심분야 : Natural Language Processing, Human-Computer Interaction

권 가 진

https://orcid.org/0000-0003-3268-477X
e-mail : ggweon@snu.ac.kr
2002년 University of California, Berkeley, Economics / Computer Science(B.A.)
2004년 Carnegie Mellon University, Human Computer Interaction (M.S.)
2012년 Carnegie Mellon University, Human Computer Interaction(Ph.D.)
2012년 ~ 2016년 KAIST 지식서비스공학과 조교수
2016년 ~ 현 재 서울대학교 지능정보융합학과 부교수
관심분야 : Human-Computer Interaction, Learning Science, Multimedia Educational Technology, Natural Language Processing