ARA : Adaptive Representation Architecture

조선대학교 3학년
컴퓨터공학과
배준호

0. 요약Abstract

ARA(Adaptive Representation Architecture)는 기존 고정 임베딩 방식의 한계를 극복하기 위해 설계된, 고유값 기반의 동적 사전 확장 구조와 희박 행렬 기반 특징 추출 신경망이다. 본 논문에서는 Cosine 유사도의 한계를 보완하기 위해 Jaccard 및 LCS를 포함한 새로운 유사도 함수 ARAS(ARA Simility)를 정의하고, 고정 벡터가 아닌 입력 항목 자체를 벡터로 변환할 수 있는 인코딩 수식을 설계하였다.
입력된 자연어는 철자/단어/문장/문단 단위로 분해되어 동적으로 사전에 배치되며, 고윳값을 기준으로 희박 행렬로 정규화된 후 MLP에 입력된다. 이 과정은 오토인코더 기반으로 학습되며, 의미적 분류 없이도 형태 기반 특징을 정제하여 출력할 수 있다.
ARA는 생물학적 신경 구조에서 착안한 7계층 구조를 가지며, 기존 임베딩 방식과 병렬로 통합될 수 있는 유연성을 갖춘다.
실험 결과, 소규모 데이터 셋 만으로도 유사 형태의 단어들이 고윳값을 기준으로 자연스럽게 클러스터링됨을 확인하였다.
결론적으로 ARA는 언어에 종속되지 않으며, 자유로운 자연어 입력을 구조화된 의미 강조 벡터로 변환함으로써, 자연어와 신경망 사이를 효과적으로 연결하는 새로운 표현 방식의 가능성을 제시한다.

목차

1. 서론 (Introduction)

1.1 연구 배경

자연어 임베딩 기술은 단어의 의미를 벡터 공간상에 효과적으로 표현하기 위한 방법으로 비약적인 발전을 이루어왔다. Word2Vec, GloVe, FastText, BERT 등의 시스템은 대규모 말뭉치를 기반으로 단어의 주변 문맥 정보를 학습하고, 이를 통해 의미 유사성을 반영한 벡터 표현을 생성한다. 이러한 접근은 일반적인 문맥 기반 의미 해석에는 매우 유용하며, 다양한 자연어 처리 응용에서 탁월한 성능을 보여주고 있다.

하지만 실제 언어는 단어의 의미뿐만 아니라 형태와 용법, 문법적 역할 등 다양한 요소들이 복합적으로 작용한다. 예컨대 ‘arrange’, ‘arranging’, ‘arranged’와 같은 시제 변화나, ‘quick’, ‘quickly’, ‘quicken’과 같은 파생어들은 의미적으로 연관되어 있지만, 그 문장 내 역할이나 형태적 구조는 서로 다르다. 기존 임베딩 방식에서는 이러한 변형들이 대부분 동일한 벡터 근처에 위치하게 되어, 형태적 차이나 문법적 기능 변화에 대한 구분이 뚜렷하지 않은 경우가 많다.

또한, 벡터 유사도 기반 선택 과정에서는 ‘of’, ‘the’, ‘in’과 같은 기능어들이 자주 높은 유사도를 보이며 우선 선택되는 경향이 있다. 이는 문장 구성상 중요한 실질어보다 문법적 역할에 집중된 단어들이 강조되는 결과로 이어질 수 있다. 철자 구조나 시제, 접두사/접미사, 활용형과 같은 형태 기반 정보는 인간 인지에 있어 핵심적인 단서임에도 불구하고, 기존 임베딩 구조에서는 상대적으로 반영 비중이 낮은 편이다.

이러한 맥락에서 본 연구는 기존 문맥 기반 임베딩의 강점을 유지하면서, 형태 구조를 중심으로 한 보완적 유사도 판단 및 의미 강조 방식을 제안하고자 한다. 제안하는 ARA 시스템은 단어의 외형적 구조를 중심으로 유사도를 계산하고, 이를 기반으로 고윳값을 통해 의미 클러스터를 구성한 뒤, 구조적 강조를 적용하여 출력 벡터를 생성한다. 이를 통해 형태 및 용법에 따른 미세한 의미 차이까지 반영할 수 있는 가능성을 실험하고자 한다.

1.2 기존 임베딩 방식의 한계

자연어 처리 분야에서 발전된 임베딩 기법들은 대부분 고정된 차원의 벡터 공간 내에서 단어의 의미를 표현한다. Word2Vec, GloVe, FastText와 같은 모델들은 주변 문맥 정보를 기반으로 단어 벡터를 학습하고, BERT나 GPT 계열의 트랜스포머 모델들은 어텐션 메커니즘을 통해 문맥에 따른 유동적인 벡터 표현을 제공한다.

이러한 트랜스포머 기반 임베딩은 자연어 처리 분야에서 획기적인 진보를 이끌어낸 기술로, 문맥에 따라 의미가 달라지는 단어들을 유동적으로 표현할 수 있게 한 점에서 그 성과는 매우 크다고 할 수 있다. 특히 BERT나 GPT 계열 모델은 어텐션 메커니즘을 통해 문맥 전반을 고려한 정교한 표현을 생성할 수 있으며, 다양한 자연어 응용에서 탁월한 성능을 보이고 있다.

그럼에도 불구하고, 이들 시스템 역시 결국은 미리 정의된 유한한 차원의 벡터 공간 내에서 모든 의미를 표현해야 한다는 설계적 한계를 갖는다. 예컨대, BERT는 768차원, GPT-4는 1280차원으로 벡터 차원이 고정되어 있으며, 이 공간 안에 시제 변화, 복합 구조, 드문 조합 표현까지 모두 포함시키려면 일반화 혹은 희소성 문제를 피하기 어렵다.

즉, 아무리 뛰어난 트랜스포머라 하더라도, 그 임베딩 공간 자체는 구조적으로 유한하며,

표현력은 그 차원의 제약을 받는다.
또한, 현재의 트랜스포머 모델들은 고정된 어휘 집합(vocabulary)을 기반으로 작동하며, 새로운 단어나 조어가 등장했을 경우에는 일반적으로 서브워드 단위로 분해하여 처리한다. 이 방식은 어휘 수를 효과적으로 줄이고 학습 안정성을 높이는 데에 유리하지만, 단어 전체의 철자 구조나 시제, 활용형 등의 형태적 특성이 분해 과정에서 희석될 수 있다는 한계도 함께 가진다.
특히, 단어의 외형적 특성이나 철자 순서 기반의 구조적 차이를 명시적으로 반영하려는 경우에는, 이러한 부분 단위 표현 방식이 형태 간 구분을 충분히 표현하지 못할 수 있다.
예컨대, 특이한 조어, 복합어, 혹은 형태적 변형을 가진 단어들은 분해되거나 일반적인 단위로 치환되면서, 그 단어가 지니는 전체적인 외형이나 시제·활용에 따른 문법적 차이가 충분히 반영되지 못할 가능성이 있다. 특히 철자 구조의 순서나 변형 패턴처럼 형태 기반의 차이를 민감하게 구분하는 데는 일정한 한계가 존재한다.
이에 반해 ARA는 입력 단어의 철자 구조와 순서를 기반으로 한 고유코드(encode)를 생성한다. 이는 각 철자의 고유 인코딩 값에 위치 기반 감쇄 계수를 곱해 합산한 값으로, 단어의 외형적 구조를 수치화한 것이다. 예컨대 "running"과 "nugnrni"는 동일한 철자를 포함하더라도 순서에 따라 전혀 다른 encode 값을 갖게 된다.
이 encode는 단어가 기존 사전에 존재하는지를 판단하는 기준값으로 사용되며, 해당 encode를 사용하여 계산 된 유사도가 일정 이상이면 해당 구조와 유사한 항목의 고윳값(eigen)을 부여받는다. 고윳값은 구조적으로 유사한 단어들을 하나의 의미군 또는 형태군으로 묶는 데 사용되며, 이후 강조 연산(sigmoid-softmax)의 기준점으로 활용된다.
이 encode는 단어가 기존 사전에 존재하는지를 판단하는 기준값으로 사용되며, 유사도가 일정 이상이면 해당 구조와 유사한 항목의 고윳값(eigen)을 부여받는다. 고윳값은 구조적으로 유사한 단어들을 하나의 의미군 또는 형태군으로 묶는 데 사용되며, 이후 강조 연산(sigmoid-softmax)의 기준점으로 활용된다.
만약 encode 값이 기존 항목들과 충분히 다르다고 판단되면, 해당 단어는 완전히 새로운 고윳값을 부여받고 사전에 동적으로 추가된다. 즉, ARA의 사전은 고정된 벡터 테이블이 아니라 입력 구조에 따라 무한히 확장 가능한 동적 사전이다.
이때 고윳값의 총 개수는 이후 단계에서 특징을 분류하는 MLP 구조에 따라 사후적으로 제한되며, MLP의 출력 노드 수가 커질수록 ARA가 표현할 수 있는 고윳값의 범위도 넓어진다. 이는 ARA가 입력 구조 기반의 무한한 표현 가능성과, 후단 MLP에 의해 조절되는 실질적 분해능을 동시에 갖춘 체계임을 의미한다.
결과적으로 ARA는 단어의 의미를 벡터 공간 내의 위치로 고정하는 기존 방식과는 달리, 입력 구조 자체로부터 의미 클러스터를 유도하고, 이를 기반으로 사전을 점진적으로 재구성하는 체계를 구현한다. 이는 트랜스포머 계열 모델이 정해진 벡터 공간 내에서 의미를 조율하는 방식과 비교할 때, 형태 및 구조 기반의 보완적 접근으로 볼 수 있으며, 특히 형태 변화나 희귀 어휘에 대한 유연한 대응이 필요한 환경에서 유용할 수 있다.

1.3 본 논문의 기여

본 연구는 기존 자연어 임베딩 방식의 강점을 존중하면서도, 단어의 형태 구조나 문법적 변형을 보다 정교하게 반영할 수 있는 구조 기반 보완 방식을 제안한다. ARA 시스템은 형태

중심의 인코딩, 구조적 유사도 기반 사전 선택, 의미 강조 출력을 위한 MLP 구조로 구성되며, 그 주요 기여는 다음과 같다.
첫째, 단어의 철자 단위 인코딩과 위치 기반 감쇄 계수를 결합한 형태 중심 유사도 판단 구조를 설계하였다. 이 구조는 시제 변화, 파생어, 복합어 등에서 나타나는 형태 차이를 민감하게 포착할 수 있으며, 기존 임베딩이 동일 벡터 근처에 위치시켰던 다양한 활용형들을 구조적으로 분리된 클러스터링 할 수 있다.
본 인코딩에 사용된 감쇄 함수는 철자의 위치에 따라 인코딩 기여도를 점진적으로 감소시키며, 해당 수식 및 시각화는 부록의 Append.1에 제시하고, 수학적 유도는 후반부에서 설명한다.
둘째, 이러한 형태 인코딩 결과로부터 고유 인코딩값(Encode)을 구성하고, 이를 기준으로 기존 사전 항목들과의 구조 유사도를 판단하는 ARAS (ARA Structural Similarity) 함수를 도입하였다. 이 함수는 세 가지 형태 기반 유사도 지표인
(1) 코사인 유사도, (2) 자카드 유사도, (3) 최장 공통 부분 수열(LCS)
을 조합하여 설계되었으며, 해당 유사도에 따라 사전의 동적 확장 여부를 결정하도록 하였다. 이를 통해 ARA는 고정된 어휘 테이블이 아닌, 입력 구조에 따라 확장되고 재구성되는 동적 사전 체계를 구현하였다.
셋째, ARA의 다층 사전 구조를 기반으로 입력 표현을 구성하고, 이를 후속 태스크에 연결하기 위한 신경망 기반 출력기(MLP)를 설계하였다. 이 MLP는 철자, 단어, 문장, 문단 사전에서 생성된 인코딩 정보를 종합하여 구조화된 입력 벡터를 구성하며, 출력 단계에서는 사용자가 정의한 분류군 또는 태스크 목적에 따라 sigmoid와 softmax를 조합한 강조 수식을 적용한다.
이렇게 생성된 의미 강조된 출력 특징 맵(feature map)은 downstream 분류, 태깅, 회귀 등의 다양한 작업에 활용될 수 있으며, 기존의 정적 임베딩 방식과도 상호 보완적으로 결합이 가능하다.

2. 관련 연구 (Related Work)

2.1 Word2Vec, BERT, FastText 등 기존 벡터 기반 시스템

자연어 임베딩 기법은 단어의 의미를 수치적으로 표현하는 데 있어 핵심적인 기술로 자리잡아왔다. 가장 널리 사용된 초기 방법 중 하나는 Word2Vec이다. Word2Vec은 단어의 주변 문맥을 예측하거나(context-based prediction), 주어진 문맥에서 중심 단어를 추정하는 방식(skip-gram, CBOW)을 통해 단어 벡터를 학습한다【Mikolov et al., 2013】. 이 방식은 단어를 고정된 차원의 벡터로 임베딩하며, 의미 유사성을 벡터 간 유클리드 거리나 코사인 유사도로 측정할 수 있다.
FastText는 Word2Vec의 한계를 보완하기 위해 제안되었으며, 단어 자체를 문자 n-gram의 조합으로 표현한다【Bojanowski et al., 2017】. 이를 통해 형태적으로 유사한 단어들(예: "run", "running", "runner") 간 의미 근접성을 어느 정도 확보할 수 있다. 그러나 이 구조에서도 여전히 형태의 순서나 문법적 역할 변화는 명시적으로 표현되지 않으며, 전체 벡터 공간은 정적으로 구성된다.

이후 트랜스포머(Transformer) 아키텍처【Vaswani et al., 2017】의 등장과 함께 BERT(Bidirectional Encoder Representations from Transformers)가 제안되면서, 문맥 기반 임베딩의 새로운 전환점을 맞게 되었다【Devlin et al., 2018】. BERT는 전체 문장을 양방향으로 인코딩하고, 단어별로 문맥에 따라 변화하는 벡터 표현을 생성함으로써, 다의어(disambiguation)와 문맥적 의미 파악에 뛰어난 성능을 보인다. 그러나 BERT 또한 고정된 차원의 벡터 공간 내에서 모든 의미를 표현해야 하며, 단어 단위가 아닌 subword 단위로 분해되어 처리되기 때문에 형태 정보의 정밀한 보존에는 한계가 있었다.

2.2 의미 기반 vs 형태 기반 임베딩 비교

기존 임베딩 기법은 대부분 의미 유사성 기반으로 학습되며, 의미가 유사한 단어들은 벡터 공간상에서 가까운 위치에 배치되도록 설계된다. 이는 분류, 질의 응답, 문서 유사도 분석 등 다양한 자연어 처리 과제에서 높은 성능을 보장해 왔다. 그러나 이러한 의미 기반 임베딩은 형태 변화(morphological variation)에 취약한 경우가 많다.

예컨대, "arrange", "arranging", "arranged"는 Word2Vec, FastText, BERT 모두 같은 근처의 벡터로 매핑된다. 이는 의미적 유사성 측면에서는 타당하지만, 실제 문장 내 용법(시제, 문법 역할)이나 철자 구조 분석을 필요로 하는 작업에서는 정보 손실을 초래할 수 있다. 반면 형태 기반 임베딩은 단어 외형 자체의 차이를 보다 정밀하게 포착하는 데 유리하다.

형태 기반 접근의 대표 사례로는 FastText의 n-gram 방식이나, 일부 언어에서의 morphological analyzer를 통한 전처리 방식이 있으며, 최근에는 이를 학습에 직접 반영하는 구조도 연구되고 있으나, 트랜스포머 계열 모델의 임베딩은 여전히 의미 중심 설계에 머무르는 경우가 대부분이다.

2.3 기존 유사도 함수(코사인, Top-K 등)의 한계

단어 간 유사도를 측정하기 위해 가장 널리 사용되는 방식은 코사인 유사도(cosine similarity)이다. 이는 두 벡터의 방향(정규화 내적)을 비교하여, 의미적 근접도를 계산한다. Word2Vec, FastText, BERT 기반의 유사도 판단은 대부분 코사인 유사도를 기반으로 한다. 그러나 코사인 유사도는 벡터의 크기나 형태 구조에는 민감하지 않기 때문에, 동일한 벡터 방향을 가지는 서로 다른 형태의 단어들을 정확히 구분하지 못한다.

또한 응용 시스템에서는 코사인 유사도 기반으로 Top-K 방식으로 유사 단어를 추출하거나 선택하는 전략이 자주 사용된다. 하지만 이 방식은 문법적으로 단순하고 빈도 높은 기능어(function words)가 과도하게 선택되는 문제를 야기할 수 있으며, 이는 문장의 의미를 왜곡하거나 태스크에 불필요한 정보를 더하는 결과로 이어진다.

이러한 이유로 최근에는 단순 벡터 유사도 외에도 문자 수준 유사도(LCS, Jaccard 등), 형태 기반 거리(MinEdit Distance, Levenshtein 등), 정렬 기반 스코어(BLEU, ROUGE의 서브토큰 변형) 등을 보조 지표로 사용하는 경우도 늘고 있다. 그러나 이들 또한 단일 지표로는 복합적인 언어 구조를 반영하기 어려우며, 특정 구조나 태스크에 과도하게 최적화되는 문제가 있다.

3. 시스템 개요 (System Overview)

3.1 ARA 구조의 전체 흐름

ARA의 큰 흐름은 다음과 같다.

1) ARAS 유사도 판단 -> 2) 사전 구축 -> 3) 사전-MLP간 값 사상 -> 4) MLP 연산

1)과 2)에 대한 상세한 설명은 4장에서, 3)과 4)에 대한 상세한 설명은 5장에서 진행하며, 각 단계에서의 흐름은 다음과 같다.

1) ARAS 유사도 판단. 해당 단계에서는 주어진 글을 입력으로 받아 유사도를 계산한다. 입력은 주어진 글을 문단/문장/단어/철자 단위로 파싱한 형태이며, 입력은 각 유형에 대응되는 사전과 연산 된다.

2) 사전 구축. 해당 단계에서는 주어진 입력과 이미 내재된 사전을 전체 비교하여 판단된 유사도에 따라 사전의 동적 확장 여부를 결정한다. 만일 역치값_1보다 높은 유사도를 가지는 값이 있을 경우 해당 항목을 발화시키며, 그 이하일 경우 입력값을 이용하여 항목을 추가해 사전을 확장시킨다. 이에 따라 다음 입력에서는 해당 항목이 발화되게 한다. 이 때, 발화의 과정에서 망각/발화에 따른 변수를 연산하여 가중치를 만들어 출력한다.

3) 사전-MLP간 값 사상. 해당 단계에서는 연산 된 출력값 목록을 이용하여 신경망에 넣을 입력값을 구축한다. 1)과 2) 단계를 거쳐 출력 된 리스트는 그저 문장에 따라 발화 된 단어들의 나열일 뿐이기에, MLP의 입력 형태와 직접적으로 호환되지 않으므로 중간 사상(Mapping)처리가 필요하며, 때문에 출력의 각 항목들을 지표로 하여 리스트를 MLP의 입력에 맞게 변환한다.

4) MLP 연산. 각 입력 노드의 위치에 맞게 사상된 입력 값을 이용하여 MLP를 연산한다. 이 때의 MLP는 오토인코더의 형태로 학습을 하며, 디코더의 라벨로는 3)의 사상 된 입력을 그대로 사용한다. 학습의 후에는 디코더를 분리한 다음, 인코더에 별도의 출력층을 연결하여 사용하며, 해당 출력층은 Sigmoid와 Softmax의 복합 연산이다.

4. ARAS: 사전 선택 유사도 함수 및 동적 사전 확장

4.1 인코딩 : 철자 및 순서, 감쇄 부여

기존의 임베딩 기법의 경우 철자 및 단어에 대해 독립적인 벡터를 태깅하여 유사도를 연산하였다. 그러나 그 벡터의 크기는 유한하며, 사용자가 임의로 정해야 하거나 트랜스포머 등을 이용하여 거대한 데이터를 통한 벡터 구축이 필요하였다.

ARA 및 ARAS에서는 그러한 단계를 회피하고자 철자 및 자연어 마디 자체에 고유한 자체적인 벡터를 자동으로 생성하여 부여하는 방법을 고안하였다. 이는 철자 및 순서, 그리고 순서에 따른 감쇄를 통한 연산이다. 각 철자에 고유한 실수 값이 있으며, 해당 철자가 단어 혹은 문장의 어떤 순서에 위치하는지에 따라 그 값이 달라진다.

기존의 철자 기반 벡터가 가졌던 문제는 다음과 같다. 1) 대체로 기준으로 많이 쓰이는 아스키코드 혹은 유니코드의 값이 너무 큰 정수이다. 2) 순수히 철자기반으로 할 경우 그 철자들의 조합의 순서를 잘 반영하지 못하였다. 3) 단순히 철자*순서의 경우 그 단어가 길어질수록 벡터의 원소 수치가 급증하는 문제가 있다.

이에 따라 ARA/ARAS는 각 문제를 해결하기 위해 다음과 같은 방안을 제시하였다. 1) 아스키

코드 혹은 유니코드 값을 기반으로 하되, 그 값에 수식을 적용하여 두 자릿수 실수 범위 내로 정보를 압축시켰다. 2) 압축된 각 철자의 수치에 각 순서를 곱하여 그 위치 정보를 적용하였다. 3) 또한 그것과 동시에 단어의 길이에 따른 감쇄 함수를 적용하여 단어의 앞 부분이 강조되고 긴 단어 혹은 문장의 후반일수록 그 수치를 점진적 감쇄하도록 하였다.
1)의 수식은 다음과 같다.

return np.log(ord(ch) + 1) ** (2 - (2 ** (-ord(ch) / 128)))

즉, encodeSpell = log(ord(x) + 1)^(2-2^(-ord(x) / U))

각 항목의 의미는 다음과 같다. log(ord(x) + 1)는 128 혹은 65335까지의 거대한 정수 나열을 log 스케일로 압축한다. 그러나 log의 특성상 그 값은 수열의 후반으로 갈수록 점차적으로 과도한 압축이 발생한다. 이에 그 값에 (2-2^(-ord(x) / U))를 승수 계산하여, 전방 부분과 후방 부분의 압축량 차이를 조절하였다.
이 때의 U는 각 코드의 유형을 의미한다. 아스키코드의 경우에는 128을 대입하고, 유니코드 한글까지의 경우에는 65335를 입력한다. 즉, 해당 인코딩은 영어 뿐만 아니라 한글 영역까지 ARA를 적용할 수 있게 해 주는 수식이다.
이렇게 철자 당 정보 압축을 한 후, 각 단어에서의 그 철자의 순서를 곱하여 단어벡터의 원소에 위치 정보를 주입한다. 그 후 각 원소의 순서에 따른 감쇄 함수를 다시 곱하여 그 크기를 조절한다. 해당 감쇄 함수의 수식은 다음과 같다.

w = (np.exp(-0.125 * idx)) + 0.125 * np.log(5 * idx)

즉, decay = exp(-0.125x) + 0.125*log(5x)

각 항목의 의미는 다음과 같다. exp(-0.125x)는 입력된 인덱스 값에 대한 지수함수적 감쇄를 적용한다. 이 때의 0.125는 그 인덱스 증가에 따른 감쇄의 정도를 정하는 하이퍼 파라메터이며, 실험적으로 얻어낸 결과이다. 그러나 exp는 0에 가깝게 수렴하는 성질이 있기에 그 정도를 조절하고 하한을 정할 필요가 있다. 때문에 0.125*log(5x)를 더하여 그 값을 조절한다. 이 때의 0.125와 5는 하이퍼 파라메터이며, 실험적으로 얻어낸 결과이다. 해당 수식이 더해진 상한선은 (0.46, 0.99)이고, 하한선은 약 55 길이에서 0.3에 근접하며, 그 이후로는 다시 점진적으로 증가하는 형태를 가지나 그 기울기가 매우 작아 무시할 정도에 가깝다. 때문에 위 수식은 1~0.3의 범위 내에서 인덱스 증가에 따른 점진적 감쇄를 가장 잘 드러내는 수식이라고 볼 수 있다.
최종적인 인코딩 수식은 다음과 같다

Encode = encodeSpell * idx * decay(idx)

각 철자는 그 위치에 따라 해당 수식을 사용하여 각각 독립적인 값을 가지고, 단어는 곧 그 값의 집합인 벡터가 된다. 이에 따라 "running"과 "nugnrni" 등 철자가 같고 위치만 다른 경우에도 모두 다른 벡터를 가지게 되고, 또한 길이가 얼마나 긴 단어 혹은 문장이 입력되어도 모두 다른 고유의 인코딩 벡터 값을 가지게 된다.

4.2 유사도 수식 설계

위에 따라 계산 된 인코딩 벡터 값을 사전의 항목과 입력 값에 적용하여 고유벡터를 얻어낸 후 그 벡터의 유사도를 계산한다. 그러나 기존의 Cosine 유사도의 경우 Black/White와 같은 반의어나 make/makes와 같은 변형 형태를 잘 감지하지 못하는 경우가 있다. 이에 따라 본 논문에서는 Cosine 유사도에 더해 다른 유사도 둘을 더 적용하여 더욱 상세한 유사도를 도

출할 수 있는 수식을 제안하고, 그것을 ARAS 라고 지칭한다.
ARAS의 수식은 다음과 같다.

$$ARASim = a*CosSim(v1,v2) + b*Jaccard(A, B) + c*LCS(A, B)$$

이 때, v1과 v2는 인코딩 된 벡터이고, A와 B는 철자 그대로의 단어 혹은 문장이다. a, b, c의 값은 하이퍼 파라메터이며, 본 논문에서는 가장 기본적인 평균값 0.333...을 사용한다. 해당 수식의 의미는 다음과 같다. 1) 고전적 CosSim을 이용하여 두 단어 벡터의 유사도를 검출하여 ARAS의 기저로 한다. 2) Jaccard를 사용하여 두 단어의 공통된 철자의 개수를 확인한다. 3) LCS를 활용하여 그 단어가 얼마나 비슷한 순서인지 확인한다.
1)은 가장 기반이 되는 유사도로서, 기존에도 거의 대부분의 기법에 사용되었던 유사도이다. 그러나 앞서 말했듯 Black/White 혹은 Cold/Hot과 같은 완벽한 반의어에도 높은 유사도를 보이는 문제점이 있었다. 그것을 보정하고자 2)를 적용한다. 해당 유사도를 이용하여 앞선 반의어의 사례의 유사도를 대폭 낮추고, make/makes와 같은 변형 형태의 유사도를 대폭 높인다. 그러나 보통의 Jaccard의 경우 make/smake와 같은 단어 순서의 변형을 인지하지 못하는 문제가 있다. 이것을 보정하기 위해 3)을 적용한다. LCS는 두 단어 상에서 공통된 철자의 길이를 탐색하고 두 단어 중 가장 긴 전체 길이로 나눈 값을 반환한다. 이를 이용하여 Jaccard의 순서 취약 문제를 해결하고, 앞선 두 유사도에 대한 신뢰도를 높인다.
해당 유사도를 활용하여 도출된 기존 취약 단어들의 유사도는 다음과 같다.
--예시들 만들어서 넣을 것
이를 통해, ARAS 유사도는 기존의 Cos 유사도의 단점을 보완하고 더욱 정밀한 유사도 탐색이 가능하게 만들었음을 알 수 있다.
4.3 유사도 값에 따른 구간 분기와 동적 사전 생성 및 확장
입력된 단어 혹은 문장과 사전에 저장 된 단어 혹은 문장의 유사도 판단이 끝난 경우, 해당 유사도를 이용하여 사전의 확장을 결정한다. 이 때 두 개의 역치값이 사용되며, th1, th2라고 칭한다. 해당 값에 따른 결정은 다음과 같다.

$$State = 1)(\text{해당 항목 발화/Sim} > th1)$$
$$2)(\text{사전 확장 및 실수 고윳값 적용/}th1 > Sim > th2)$$
$$3)(\text{사전 확장 및 독립 고윳값 적용/}th2 > Sim)$$

각 상태에 대한 의미는 다음과 같다. 1) 만일 유사도가 높은 값이 있을 경우, 입력된 값과 해당 사전의 항목이 일치한다는 뜻이므로 입력된 값이 이미 사전에 있다는 뜻이며 그 항목을 발화한다. 2) 만일 중간 즈음의 유사도를 가질 경우, 입력된 값과 유사한 형태의 항목은 있으나 입력된 값에 적절히 맞는 항목은 없다는 뜻이므로, 해당 유사한 형태의 항목의 근방에 고윳값을 지정하여 사전을 추가한다. 3) 만일 사전 내부에 위치한 항목들의 모든 유사도가 일정 이하일 경우, 해당 입력은 사전에 대해 완전히 새로운 항목이라고 판단하고 독립된 유사도를 가진 별도의 항목으로 사전에 추가한다.
이 때 고윳값Eigen이라는 항목을 사용한다. 이 고윳값은 각 항목에 대한 유사성의 지표이다. 고윳값은 정수 혹은 실수로 이루어져 있으며, 3)의 경우 정수를, 2)의 경우 실수를 부여받는다. 또한 2)의 경우 사전에 있는 각 정수 고윳값마다의 유사도를 측정하여 가장 높은 유사도를 가진 정수 고윳값의 근방에 배치하고, 이 때의 간격은 해당 유사도와 가우시안을 활용하여 높은 유사도를 가질수록 기준이 되는 정수 고윳값과 가까운 곳에 위치하게 만든다. 이에 따라, 사전을 유사도 기준으로 정렬할 경우 유사한 형태를 지닌 단어들이 자연스레 가까운 고윳값을 가지게 되는 클러스터링 효과를 갖게 된다.

그 클러스터링의 효율 역시 크다고 볼 수 있다. 실험 결과, 기존의 거대한 데이터를 활용하여 벡터를 연산하여 추론하는 트랜스포머/어텐션과 수동으로 벡터를 지정해야 했던 word2vec과는 달리, 2~3kb, 약 5천자 정도의 짧은 영어 텍스트(동화와 단편 소설) 15편을 각각 한 번씩 학습시킨 결과, ~ing와 ~ed, ~es 혹은 형태적으로 유사한 자모음을 가진 어휘들이 높은 정확도로 군집하는 것이 드러났다. 해당 실험에 대한 상세 과정과 수치는 6장에서 제시한다.

4.4 망각 및 발화, 강조에 따른 점수 부여

동적 사전 생성시 2)와 3)의 경우에는 완전히 새로운 항목이므로 높은 발화값을 부여하며, 이 때의 값은 1.4 +- 가우시안 값을 가진다. 이 때 1.4와 가우시안에 곱해지는 표준편차 역시 하이퍼 파라메터이며 실험을 통해 얻어낸 값이다.

그러나 기존에 존재하는 항목의 경우 일반적인 유사도를 직접적으로 활용하는 것은 알맞지 않다. ARAS 유사도의 상하한은 1.0~0.0으로 소수 범위이며, 반복 혹은 희소에 따른 변화가 고려되어 있지 않다. 이에 따라 ARA에서는 반복과 희소에 대한 변수를 유사도Sim에 적용하여 최종 점수Score를 얻게 설계하였고, 해당 수식은 다음과 같다.

$$r\_s = sigmoid(r),\ c\_s = sigmoid(c)$$

$$w = exp(m * r\_s) / log(1+n*c\_s)$$

$$score = Sim * w * V$$

각 수식의 의미는 다음과 같다. m과 n은 하이퍼 파라메터로, 각각의 항목에 대해 변화량을 나타낸다. r은 발화, c는 망각의 경우를 나타내는 변수이다. 해당 변수는 들어 온 입력에 대해 그 항목의 발화 여부에 따라 증감된다. r은 발화 횟수에 따라 증가하고, c는 발화 실패 시 증가하여 망각을 표현한다. 이 두 변수는 상호 경쟁적으로 작동하여, 발화된 항목은 점차 강화되고, 발화되지 않은 항목은 점차 약화된다

또한 해당 r과 c를 sigmoid로 변환하여 충분한 비선형성과 0.0~1.0의 정규화를 얻고, 그 값들을 다시 exp와 log에 넣는다. exp는 r의 증가에 따른 기억 강화를 의미하고, log는 c의 증가에 따른 망각의 정도를 의미한다. 발화가 결정될 시 r 증가 c 감소 -> exp가 증가 및 log 감소 -> w가 증가하는 형태가 되고, 발화가 되지 않은 경우는 그 반대 기전이 작동하여 w가 감소하는 결과를 가진다. 즉, 해당 w는 반복적인 발화에 따른 기억의 강화와 시간의 흐름에 따른 기억의 망각을 의미한다.

w와 Sim이 곱해진 값에 V를 곱하여 최종 Score를 얻게 된다. 이 때의 V는 강조 값으로, 항목 하나에 대해서는 스칼라 값이나 사전 전체의 값에 대해서는 스칼라가 이어진 1차원 벡터 값을 가진다. V는 해당 항목을 얼마나 강조할 것인가를 의미한다. 이 값은 1로 초기화되며, 사용자가 직접 조정할 수 있다. 즉, 만일 사용자가 컴퓨터 관련 자연어를 주로 처리할 경우, 해당 V 벡터를 1 이상으로 조정하여 컴퓨터 기술과 관련된 어휘가 더욱 잘 발화되도록 만들 수 있으며, 혹은 어떠한 주제를 약화 시키고 싶은 경우 V의 값을 0.0~1.0 사이로 조정하여 직접 해당 Score를 조정할 수 있다. 이 V는 ARA의 확장성을 보장하는 벡터로, 사용자는 V를 조정하는 것만으로 ARA의 '주목 대상'을 변경할 수 있게 되며, 사전에 미리 계산된 V를 적용하는 것으로 해당 ARA는 사용자가 원하는 범주에 특화된 특징 추출기로 동작할 수 있게 된다.

5. ARA: 의미 강조 및 오토인코더

5.1 사전 발화값 처리 및 Eigen, Score, Encode의 MLP 적용

4절의 내용을 통해 얻어진 사전의 발화 리스트는 Score의 내림차순으로 정렬 된 리스트의 형태를 가진다. 그러나 동적 사전의 특성상 해당 리스트의 길이는 가변이며, 항목의 위치 또한 자유롭기에 MLP의 입력으로 삼기에는 적절하지 못하다. 그렇기에 만들어진 사전의 발화 리스트를 MLP의 입력으로 적절히 사상하는 단계가 필요하다.

해당 사상 과정에서 사용되는 변수는 다음과 같다. 1)Score, 2)Eigen, 3)Encode. 모두 사전에 미리 계산 된 값이며, 사전의 발화 시 리스트로 함께 넘겨지는 값이다. 각 항목을 이용한 사상 과정은 다음과 같다. 1) Score의 값으로 정렬 된 리스트를 순회하며 round(Eigen) 값 순서대로 재정렬한다. 2) MLP의 입력 노드에 각각 할당 된 Eigen 영역의 위치에 3) Score와 Encode를 곱한 값을 넣는다.

각 순서의 의미는 다음과 같다. 1) 이미 사전의 동적 생성 규칙 상 Eigen은 n을 기준으로 +-0.5를 경계로 하여 그 의미가 변화하기에, round를 통해 유사한 고윳값(=유사한 형태 및 의미)을 가진 발화 값들끼리 정렬한다. 2) 각 고윳값에 따라 MLP의 입력 위치를 미리 지정해 놓는다. 이 때 발화된 값들 중 동일한 고윳값 영역이 다중 발화될 가능성이 있기에, 각 고윳값 당 4개의 노드를 가지는 입력 영역을 할당한다. 즉, MLP의 입력 크기는 각 사전에서의 최대 고윳값 * 4의 크기를 가지며, 이에 따라 사전의 고윳값 상한 역시 MLP의 연산 한계에 따라 조정해야 한다. 3) 해당 고윳값 영역대를 통해 입력의 위치를 정했으며, 그것에 입력할 실질적인 값으로 Sigmoid(Score * Encode)를 넣는다. 이 때, 유사한 고윳값과 별개로 각 단어마다 고유한 값을 가지는 Encode를 통해 해당 유사도 영역대에서 어떠한 어휘가 입력되었는지를 정하여 입력의 해상도를 높이며, 이 값에 Score를 부여하여 그 항목이 얼마나 강하게 발화되었는지에 대한 정보를 주입한다. 즉, 해당 값은 x*w로 볼 수 있으며, 이 값에 Sigmoid를 적용한 값을 입력으로 삼음으로서 입력의 정규화 및 비선형성 추가를 이루고, MLP의 1단계로 볼 수 있다.

이러한 사상 과정을 통해 동적 사전의 능동 확장에도 불구하고 MLP의 입력을 균일하게 유지할 수 있다. 또한 이 때의 입력은 희박 행렬로, 약 8192개의 거의 대부분의 영역이 0인 상태에서 오직 문장에 따라 발화 된 약 100개 이하의 항목들만 값을 가진 형태를 지닌다. 이것이 갖는 의미는 다음과 같다. 1) 동적 사전 확장에 따른 여유 공간을 마련할 수 있다. 2) 희박 행렬에 따른 그 값의 민감함을 인식할 수 있다. 3) 드롭아웃 등의 최적화 기법이 필요 없으며 입력 그 자체로 오버피팅을 방지한다. 4) 동적 사전 확장의 특성상 가장 먼저 고윳값이 생성된 노드에서부터 학습이 시작되고, 이후의 노드들은 역전파 과정에서 앞서 생성된 노드들의 학습 경로를 기반으로 학습이 진행되어 그 연관성을 부여받는다.

이상의 과정을 통해 발화 리스트가 희박 행렬로 사상되고, 그 희박 행렬이 MLP에 입력되며 특징 추출 단계가 시작된다.

5.2 MLP 구조 및 역할, 출력층 설계

ARA는 문단/문장/단어/철자로 구분 된 4개의 사전을 사용하고, 해당 사전은 각각의 MLP를 가지며 그 MLP의 입력 노드와 고윳값 영역은 모두 독립이다. 해당 MLP의 입력은 각각 5.1에서 이루어진 사상 과정을 거치며, 이에 따라 문단/문장/단어/철자가 각기 다른 MLP를 통해

1차적으로 특징 추출이 이루어진다. 이 MLP의 계층은 사상 과정을 포함하여 3계층으로, 모든 층이 Sigmoid를 사용하여 충분한 비선형성 및 생물학적인 특징을 부여받는다.
4개의 MLP의 끝단은 모두 512개의 노드를 가지고, 해당 층에서 출력 된 4 * 512개의 입력을 Concat하여 2048개의 벡터를 얻는다. 그 후, 해당 2048개의 벡터를 다시 종합 MLP의 입력으로 삼는다. 해당 종합 MLP는 각 사전에서 충분히 추출된 각각의 특징들을 종합하여 문장 전체에 대한 특징을 추출하는 역할을 가진다. 이 과정에서 단어 혹은 철자 단위에서는 볼 수 없는 문장 단위의 특징을 고려할 수 있고, 특히 'take care of' 등의 여러 어휘가 모인 구절을 상세히 감지하는 것이 가능하며, 문단 단위의 고려를 통해 전체적인 글의 주제와 특징을 고려할 수 있게 된다.
해당 종합 MLP는 3개의 층을 가지며, ReLU를 사용하여 Gradient Descent 문제를 해결함과 동시에 과도한 비선형성 대신 명확한 특징 값만을 추출하는 과정을 거치게 된다. 또한 마지막 단은 16*16의 256개의 노드로 이루어진 맵으로 볼 수 있으며, 그 출력값에 1) Sigmoid를 거친 후 2) 각 16개의 열마다 Softmax를 적용하는 출력층을 적용시켜 최종 값으로 삼는다. 각 과정의 의미는 다음과 같다. 1) Sigmoid를 통해 해당 값들 각각에 대한 독립적인 확률분포를 만든다. 이는 입력된 글에 대해 어떠한 특징이 얼마나 부여되어 있는지에 대한 정량적인 값을 제공한다. 2) 16개의 열마다 Softmax를 부여한다. 이 때 앞선 16개는 특징에 따른 분류로, 시간, 공간, 감정 등의 대분류적인 특징 나열을 나타내며, 그 분류당 할당 된 Softmax가 적용되는 16개는 어제, 오늘, 내일 / 위, 아래, 오른쪽, 왼쪽 등 각 분류의 하위에 있는 상세한 특징 분류를 나타낸다. 즉, 1)의 과정을 통해 전체적인 확률 분포를 잡고, 2)의 과정을 통해 그 분류에 해당하는 특징의 값을 더욱 상세히 강조시킨다. 이 때의 16 등의 수치는 하이퍼 파라메터로, 사용자가 직접 원하는 특징의 개수를 부여할 수 있으며, 해당 논문에서는 테스트의 효율을 위해 16*16/256개의 출력을 지정하였다.

5.3 오토인코더를 통한 MLP 학습

5.1과 5.2의 과정을 통해 얻은 출력 값은 명확히 라벨이라고 할 수 있는 것이 없다. 동적 사전 확장의 장점으로 입력되는 거의 모든 자연어를 수용할 수 있다는 것이 있지만, 그 자연어에 대한 라벨의 수동 태깅이 거의 불가능하다는 단점도 존재한다. 때문에 ARA에서는 MLP의 학습을 위해 오토인코더 구조를 사용한다. 5.2에서 구성한 MLP의 구조를 인코더로 삼고, 그 구조를 반전하여 역순으로 배치해 디코더를 만든다. 인코더에서 4개의 MLP가 하나로 합쳐진 것 역시, 디코더의 구조로 하나의 신경망의 출력을 4개로 나누어 각기 다른 MLP에 입력하는 형태로 재현한다. 활성화 함수의 유형 역시 각 MLP에 따라 유지된다. 이때 인코더의 출력층은 학습 도중 분리하고, 디코더의 출력으로는 각 4개로 분할 된 인코더의 입력 크기를 그대로 사용한다. 이에 따라 인코더에 입력되는 벡터, 즉 사전에서 발화되어 사상 된 벡터를 그대로 사용할 수 있으며, 그것이 곧 라벨이 된다. 이렇게 구성된 오토인코더는 입력으로 들어온 사전의 발화 여부를 그대로 디코더를 통해 재현하고, 8192 - 256 - 8192라는 구조를 통해 가운데에 위치한 256개의 노드에 대해 특징화를 진행할 수 있게 된다.
이렇게 ARA는 4장부터 5장까지 이어진 구조를 통해 제한 없는 자연어 입력에 대한 동적 사전 확장 및 특징화 구조를 가지게 된다. 또한 V를 통해 ARA의 특화 영역을 사용자가 간단히 변경할 수 있다. ARA를 통해 추출된 자연어의 특징은 그대로 트랜스포머 혹은 어텐션 등의 기존 신경망의 입력으로 삼을 수 있으며, 이는 자연어의 동적인 입력과 신경망의 정적인 구

조 사이를 잇고 그 특징화를 통해 신경망 학습의 효율을 높일 수 있는 가교 역할을 의미한다.

6. 실험 및 분석 (Experiments & Analysis)

6.0 실험 조건

실험 결과 확인의 기준은 단어사전. 실험에 사용한 텍스트는 약 5000자 내외의 단편 동화 10편, 1500~30000자 단위의 대사 포함 단편 소설 6편, 총 16편의 영어 문학 텍스트. 총 5에폭, 역치값은 th1=0.9, th2=0.55, 고윳값 상한은 1408개.

6.1 단어의 형태에 따른 클러스터링 분포

	어휘	고윳값	어휘	고윳값	어휘	고윳값
1	setting-hen	74.852	beaming	75.022	singing	75.197
2	morning	74.867	waiting	75.022	emotion	75.203
3	killing	74.886	forming	75.023	heading	75.204

Append.2 일부 발췌.

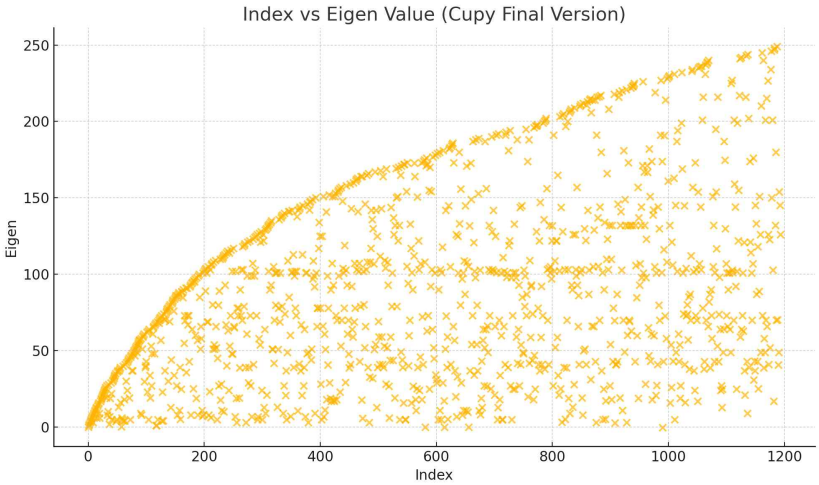
해당 클러스터링은 고윳값 75의 영역 중 일부를 가져 온 것이다.
도표에서 보다시피, 각 텍스트에서 출현한 단어들 중 -ing와 연관된 단어들이 높은 군집률로 모인 것을 볼 수 있다. 이는 에폭 1부터 나타난 현상이며, 이후의 에폭은 사전의 확장이 아닌 MLP의 학습에 사용되었다. 때문에 사실상 클러스터링 자체는 에폭 1에 다 완료되었다고 볼 수 있다.
이러한 현상은 -ing 형태에만 국한된 것이 아닌, 다양한 형태의 어휘와 각 변형에도 적용되었다. 다음은 그 결과 중 일부이다.

	어휘	고윳값	어휘	고윳값	어휘	고윳값
1	really	49	meow	56.943	washed	100.980
2	family	49.004	show	56.947	landed	100.982
3	kindly	49.011	Meow	57	wicked	100.988
4	ready	49.027	knew	57.000	locked	100.988
5	hardly	49.038	know	57.059	Indeed	100.993
	A		B		C	

A 영역대는 고윳값 49 영역대로, -ly 혹은 -y로 끝나는 어휘들이 군집 된 것을 확인할 수 있다. B 영역대는 -ow 혹은 -w 로 끝나는 어휘들이 군집 된 것을 확인할 수 있으며, 특히 knew-know의 형태 변화 역시 인식된 것을 알 수 있다. C 영역대는 -ed 형태의 어휘들이 상당수 군집 되었으며, 주로 동사의 과거형이 군집 된 것을 알 수 있다.
본문 이외에도 단어 사전에서 상당히 많은 형태의 군집이 관측되었다. 특히나 -ing, -ed,

-ly와 같은 특이 접미사의 어휘들은 약 3~5개의 별도 군집이 확인되었고, 이러한 단어 변형을 제외한 명사와 형용사 역시 포함된 자모음의 형태와 순서에 따라 유사한 어휘들이 군집된 것을 확인할 수 있었다.
특히 가장 많이 군집 된 형태는 -ed, -d 형태로, 해당 고윳값 대역폭의 단어들은 대부분 -ed, -d 형태를 지니었거나 철자 내부에 포함된 형태로서, 과거형 혹은 수동태의 형태를 가진다. 해당 군집은 특히나 큰 형태임과 동시에, 이 군집을 제외하고도 약 3개의 -d 군집이 더 발생하였다. 이에 따라 해당 군집들은 -ed, d 형태의 어휘들을 충분히 수용한다고 볼 수 있다.
특히나 이 군집은 단 1 에폭 만으로 100.5~103.5의 구간 사이에 167개의 단어가 압축된 특이 사례로서, ARA의 유사도 기반 사전 확장의 성능을 증명한다. 이에 대한 결과는 **Append.2**에 첨부한다.

6.2 사전 동적 확장에 따른 고윳값 분포 변화



해당 그래프는 단어사전의 고윳값-인덱스 분포이다.
인덱스가 증가할수록 최근에 사전에 추가 된 사전이며, Eigen은 그 확장에 따라 부여 된 고윳값을 의미한다.
위 그래프를 볼 때, Index가 약 200의 이전 까지는 y = x에 가까운 기울기로 급격한 학습 및 고윳값이 부여되는 것을 확인할 수 있다. 이는 비어 있던 사전이 입력되는 단어들을 흡수하며 급격히 확장되는 것을 의미한다.
또한 약 500 이후의 학습 추세는 그 이전보다 기울기가 감소하며, 이는 곧 새로운 고윳값이 부여되기보다는 이미 부여 된 고윳값을 기반으로 실수 고윳값이 부여되어 단어가 압축되고 있다는 것을 의미한다.
즉, 해당 그래프는 학습에 따른 독립 고윳값의 팽창과 점진적인 의미 압축을 의미하며, 특히 고윳값 100대의 단어들이 일렬로 늘어진 것은 그 부분의 고윳값 대역에 많은 량의 어휘 압축이 일어났다는 것을 의미한다. 해당 고윳값 영역대의 어휘는 앞서 6.1에서 언급하였던 -ed 형태로, **Append.2**에 첨부한 자료이다.

7. 인체 커넥톰 기반 설계 철학 (Biological Motif)

7.0 배경

ARA/ARAS 구조를 구축하는 데에는 실제 인체의 기전에서 수많은 영감을 받았다. 인간의 신경학적 기억 기전, 성장 발달에 따른 인지능력 향상 과정, 시냅스의 연결과 단절 등이 그것이다. 이 7장에서는 실제 인체 기전과 어떠한 유사점이 어떤 항목과 연결되는지를 말하고자 한다.

7.1 신경망 7층 구조

사전을 제외한 MLP의 구조는 다음과 같다. 1) 사전-MLP 사상, 2) 각 사전 당 MLP_1, 3) 각 사전 당 MLP_2, 3) 사전 간 벡터 Concat, 4) 종합 MLP_1, 5) 종합 MLP 2, 6) 종합 MLP 3, 7) 출력층. 이러한 7층 구조는 실제 인체 대뇌피질의 구조를 차용하였다. 실제 인체의 대뇌피질 구조는 다음과 같다. 1) 상위피질 연결 및 정보 조절, 2) 영역 내 뉴런 연합, 3) 영역 간 수평 연결, 4) 감각 입력 수용, 5) 운동 출력 조정, 6) 하위 구조 연결, 7) 심층 통합. 해당 구조 간 순서는 다르나, 그 전체적인 층위는 유사하다. 특히 MLP의 7)과 대뇌피질의 5), MLP의 3)과 대뇌피질의 3)이 그것이다. 이러한 유사성을 통해, ARA는 그 MLP의 크기를 7층으로 결정하였다.

7.2 동적 사전 확장 및 학습

동적 사전이 유사도와 고윳값 기준으로 확장되는 것 역시 인체의 기전에서 차용하였다. 특히나 영아-유아기의 성장 과정에서 발달되는 인지능력이 그것이다. 사람의 인지능력 역시 이미 존재하는 항목을 중심으로 다른 항목들을 해석하거나 연결시켜 인지 영역을 확장하며, 유사한 항목들을 묶어 생각하거나 새로운 것을 배우는 것이 바로 그것이다.
또한 그렇기 때문에 ARA의 학습 커리큘럼 역시 인간의 학습 과정에서 차용할 수 있었다. 논문에서 테스트용 학습으로 사용한 '동화'가 바로 그것이다. 영아-유아기에 주로 접하는 자연어인 동화와 노래는 그 뜻이 명확하고 다양한 기본 어휘를 사용하며 기초적이고 정확한 문법을 가졌다는 특징이 있다. 이는 ARA의 동적 사전 확장 구조에 더없이 알맞은 특징으로, 다채로운 어휘를 통해 독립적이고 다양한 고윳값 특징을 만들 수 있었으며, 동화를 통해 얻은 기초적인 문법과 어휘를 기반으로 단편 소설과 같은 다양한 텍스트를 효과적으로 학습시킬 수 있었다.

7.3 고윳값 클러스터

7.2에서 언급했듯이 인간의 인지능력은 이미 존재하는 항목을 중심으로 다른 항목들을 재해석 혹은 학습할 수 있다. 이러한 특징은 ARA에 고윳값이라는 항목을 추가할 수 있는 모티브가 되어주었다. 고윳값 도입 이전의 ARA는 Sim과 Encode라는 두 항목만으로 단어의 정량적인 평가를 진행하려고 하였으나 실패하였다. Sim은 매 순간마다 변하는 변수였고, Encode는 단어 자체에 대한 고윳값일 뿐 그 단어의 형태에 대해서는 알 수 없었다. 때문에 그런 구조에 고윳값이라는 항목을 추가하였고, 그 의미를 단어의 형태에 대한 지표로 삼아 사전 확장 및 클러스터링에 대한 지표로 삼았다. 그 결과 동화 10편이라는 매우 적은 데이터에도 불구

하고 훌륭한 성능으로 클러스터링을 이룰 수 있었다.

7.4 반복과 강화, 망각 구조

인간의 인지 능력에 대한 모티브는 반복과 망각 역시 마찬가지이다. 인간의 뇌는 뉴런과 시냅스로 이루어져 있고, 그 시냅스의 연결에 따라 어떠한 기억이 결정된다. 그 시냅스는 반복된 자극이 이루어질 시 연결이 강화되고, 자극이 없이 시간이 흐르면 점차 약해져 소멸한다. 이에 따라 인간은 필요한 기억과 필요하지 않은 기억을 분류할 수 있으며 보다 효율적인 인지능력을 가질 수 있다. ARA는 그것을 차용하여, 각 항목에 대한 발화와 망각의 지표를 추가하였다. 이는 반복된 발화 시 그 출력이 증가하고, 망각이 계속될 경우 그 출력이 점차적으로 감소하는 것으로 구현하였다.

그러나 실제 ARA에서는 아무리 발화가 되지 않더라도 아예 항목을 소멸시키는 것은 허용하지 않았다. 그것은 곧 데이터 셋의 낭비로 이어지기 때문이었다. 때문에 사전의 항목에 반복과 망각에 대한 상하한을 지정하고 gap이라는 변수를 추가하였다. 이러한 조치로 인해 해당 항목이 아무리 발화되지 않더라도 아예 소멸하는 것을 막으며, 오버플로우와 같은 지수적 값 폭발을 막음과 동시에, 계속된 망각의 도중 발화될 경우 그 값을 튀게 만들어 더욱 기억을 하게 되는, 인간의 망각주기와 같은 효과를 가지게 되었다. 해당 부각의 기준은 gap으로, pass가 지속될 경우 gap이 증가하고, fire의 과정에서 gap이 일정 값 이상일 경우, 즉, 해당 항목이 오랫동안 잊혀졌다가 발화되었을 경우 r과 c의 값을 대폭 조정하여 그 값을 부각시키는 것으로 구현하였다.

7.5 인지 기반 신경 표현 모델로서의 확장성

위의 모든 구조는 단순한 생물학적 은유에 머무르지 않고, 실제 ARA 시스템의 구조와 기능 설계 전반에 직접 반영되었다.

대뇌피질의 7층 구조를 반영한 계층적 MLP, 시냅스 연결에 기반한 발화/망각 수식, 유아기의 언어 습득과 유사한 동적 사전 확장, 고윳값 기반의 자율 클러스터링 등은 모두 인간 인지 체계의 핵심 기전을 연산적으로 표현한 결과물이다.

이처럼 ARA는 생물학적 커넥톰 구조를 모티브로 하여 자연어 처리 시스템을 설계하고자 한 시도로,기존 임베딩 기법이 갖지 못했던 설계 철학과 적응성을 구조적 수준에서 실현하였다.

8. 결론 및 향후 과제 (Conclusion & Future Work)

8.1 ARA/ARAS의 정리 및 의의

ARAS는 ARA 시스템을 위해 설계된 새로운 유사도 함수이지만, 그 구조적 특징과 정밀도 측면에서 기존 의미 기반 임베딩에서도 독립적으로 활용이 가능한 수준의 성능을 지닌다. 기존 Cosine 유사도만으로는 구분하기 어려운 반의어 구분, 형태 변형(makes/make, runs/run) 인식, 그리고 거의 쓰이지 않던 Jaccard와 LCS 기반의 형태 유사도 요소 도입은 ARAS의 가

장 큰 특징이다. 특히 이는 ARA의 자체 인코딩 방식(Encode)과 결합될 경우, 단어별 미리 지정 된 고정 벡터가 아닌 입력 항목 자체로부터 실시간으로 고유 벡터를 생성할 수 있다는 점에서 강력한 장점을 갖는다.
ARA는 이러한 ARAS를 바탕으로 동작하며, 입력 자연어에 따라 동적으로 사전을 확장하고, 고윳값 기반으로 특징을 정렬하며, 그 출력을 희박 행렬로 변환하여 MLP로 정규화시켜 입력하는 구조를 가진다. ARA에서 부여되는 고윳값은 정수 사이에 무한한 실수 값을 허용하기 때문에, 실제 정수 고윳값이 MLP의 입력 노드 수에 제한된다 하더라도 사전의 확장성과 유사도 기반 구조화는 거의 무한에 가까운 자유도를 가진다. 예를 들어 학계 연구에 따르면, 10대 후반 원어민 기준 실제 사용 가능한 의미적 독립 어휘 수는 영어의 경우 약 20,000~30,000개(Nation & Waring, 1997), 한국어의 경우 약 25,000~35,000개(국립국어원, 2011; 김정숙 외, 2009)로 추정된다. 이는 ARA의 고윳값 기반 입력 설계가 실질적인 자연어 처리 범위에 대해 충분한 표현력을 갖추고 있음을 뒷받침한다. 결국 ARA는 언어에 종속되지 않으며, 거의 모든 형태의 동적인 자연어 입력을 정적인 MLP 구조로 변환하는 구조를 가짐으로써, 기존의 고정 벡터 기반 임베딩 방식과 달리 자체적인 기억 구조, 확장성, 정규화, 의미 강조를 동시에 실현할 수 있는 임베딩 기법이라 할 수 있다.

8.2 기존 구조와의 통합 가능성 (기존 임베딩 결합 등)

ARAS와 동적 사전 시스템은 분명 구조적 정교함과 사전 구축에 필요한 데이터의 총량 면에서 우수한 성능을 가지고 있으나, 그 절대적인 정답률과 실제 통계적인 성과와 같은 지표에서는 아직 부족하다고 할 수 있다. 때문에 이를 보조하기 위해 기존의 고전적인 임베딩 기법을 병행해서 사용하는 것을 제안한다. 이는 각 사전 별 MLP에서 종합 MLP로 통합하는 과정이 있기 때문인데, 바로 이 과정에서 연결되는 벡터로 독립적으로 시행 된 word2vec 등의 결과를 추가하여 MLP에 입력하는 것이 가능하다. 이는 곧 word2Vec 특유의 의미 공간 기반 연산 시스템이 MLP에 부여된 효과를 얻으며, 더욱 정량적이고 명확한 특징 추출이 가능하다는 장점을 얻을 수 있다. word2Vec 뿐만 아니라 다른 임베딩 기법도 결합이 가능하며, 이는 기존 임베딩 방식과의 병렬 사용을 통해 ARA가 가진 구조적 장점은 유지하면서도, 통계적 성능과 범용성을 함께 확보할 수 있는 실용적인 접속점이 될 수 있음을 보여준다.

8.3 하이퍼파라메터 튜닝

ARA 구축 과정에서 특히 민감하게 반응했던, 중요한 하이퍼 파라메터는 다음과 같다.
1) 각 사전 별 역치값 th1, th2 기준, 2) 고윳값 상한, 3) r과 c의 상수.
1)의 경우는 매우 중요하였다. th1은 모든 사전에 대해 약 0.9~0.85로 고정이었으나, th2는 그 값에 따라 클러스터링의 강도가 민감하게 변화하는 결과를 얻을 수 있었다. th2가 너무 높으면 거의 대부분의 어휘들이 독립적인 클러스터를 가져 고윳값 고갈 현상이 일어났고, th1이 너무 낮으면 반대로 거의 대부분의 어휘가 연관이 거의 없는 어휘에 엮이게 되어 클러스터링에 실패하는 현상이 일어났다. 사전 별 유사도의 평균이 모두 달랐기에, th2의 경우는 실험을 통해 정하였고, 이후로도 더 좋은 값을 찾을 수 있을 것이라 생각하는 바이다.
2)의 경우는 MLP의 연산량 한계와 엮인 상수였다. 실제 MLP의 연산이 적절한 시간 내에 종료되는 크기를 가지고 있으면서도, 그 어휘를 충분히 담을 수 있는 용량을 선택하여야 했

다. 가장 우선적으로 정해진 고윳값 상한은 철자 사전으로, 해당 사전의 고윳값 상한은 아스키코드의 상한인 128로 지정하였으며 유니코드의 경우 추후 조정이 필요하다. 이는 8.4절에서 다룬다.
가장 많은 고윳값이 필요했던 사전은 단어 사전이었으며, 실제로도 문장/문단 사전에 비해 가장 많은 약 1500개의 고윳값을 가지고 있다. 그러나 실제로 테스트 중 저장된 어휘는 약 400개의 고윳값에 약 2000개의 단어들로, 고윳값 하나 당 평균 5개의 단어가 압축되었다는 결론을 얻을 수 있다. 다만 이것은 평균이며, 실제로는 하나의 고윳값 클러스터로 30개에 달하는 단어들이 엮인 사례도 있었다.
3)의 경우 역시 중요한 상수값이다. r과 c의 스텝을 너무 크게 잡으면 실제 사전의 발화 Score가 폭주하거나 거의 없어지게 되었고, 지수함수의 특성상 그 값이 굉장히 거대하게 변하는 문제가 있었다. 때문에 r과 c의 스텝 값을 실험을 통해 약 0.02 정도로 매우 작게 잡았으며, 이는 실제 자연어 속에서 나타나는 일반적인 단어의 희소성과 the, to 와 같은 특정 단어의 반복성을 모두 만족할 수 있는 값이었다.

8.4 다른 언어로의 확장

ARA가 가진 Encode는 실제 유니코드에도 적용이 가능한 수식이며, 한국어의 끝인 65335의 변환된 값이 약 30에 달하는 압축률을 보인다. 또한 이 역시 승수 연산을 통해 다른 단어들 간의 압축 간격을 보장한 값으로, 이는 곧 해당 Encode 수식이 영어 뿐만 아니라 한국어, 그리고 그 사이에 위치한 타 외국어에도 적용이 가능하다는 장점을 가진다.
그러나 실질적인 한계 역시 존재한다. 대표적인 예시로는 철자 사전의 고윳값 상한이 있다. 철자 사전은 그 특성상 들어오는 모든 철자가 독립적인 고윳값으로 배치되기에, 한글을 지원한다고 가정하면 65335개의 고윳값이 필요하고, 또 65335개의 노드를 가진 MLP 입력층이 필요하다는 계산이 나오게 된다. 이는 실제 컴퓨팅 연산에서 매우 불리한 수치이다.
때문에 그 대안으로 중간의 타 언어의 영역만큼 한국어 코드의 값을 감산하여 유니코드 자체를 편집하는 방안, 혹은 거의 사용되지 않는 고어를 제외한 수치를 제시하는 바이지만, 이 역시 한글 세트의 11172개 라는 크기 때문에 실질적인 대안은 강구를 해야 할 따름이다.

8.5 캐싱을 통한 순차탐색 최적화

ARA의 동적 사전 확장은 입력에 따라 유사한 항목을 발화하거나 사전에 등록하여 확장할 수 있는, 매우 유연한 구조를 가지는 구조이다. 그러나 이러한 구조는 시간이 지남에 따라 성능 저하의 요인으로 작동할 수 있다. ARA의 연산 특성상 사전에 존재하는 모든 원소들을 순회하며 유사도를 비교해야 하는데, 학습 초반 작은 크기의 사전은 문제가 없으나 점차적으로 사전이 확장될수록 순차탐색의 시간이 길어져 연산의 효율이 떨어지는 증상이 발현하였다.
이를 극복하기 위한 방법으로 사전을 순회하는 매니저 클래스에 캐싱 기법을 도입하는 것을 제안하는 바이다. 이는 사전의 확장 시 정수 고윳값을 미리 매니저 클래스의 캐시 리스트로 저장해두는 것으로, 파싱되어 들어 온 입력을 사전 전체와 비교하는 것이 아니라 우선적으로 그 캐시 리스트와 비교하여 가장 유사도가 높은 Top-K를 추출하고, 해당 목록의 고윳값 대역들만 비교하는 방법이다. 이러한 방식은 전체 순회에 비해 획기적으로 순회의 크기를

줄일 수 있으며, 특히나 정수 고윳값 항목이 그 클러스터 대역을 대표한다는 전제가 있기에 가능한 방법이기도 하다.
다만 이러한 방식은 표본에 따른 캐시 기반 탐색이라는 특성상 전체 탐색에 비해 정확성이 떨어질 수 있으며, 중복 어휘 등록 혹은 누락, 혹은 클러스터의 분산 문제 등의 리스크를 수반할 수 있다. 또한, 사전 자체가 미리 고윳값 기준으로 정렬되어 있어야 가능한 방법이기에 실질적인 구현의 문제가 있기도 하다.
그럼에도 불구하고, 캐시 기반 순차탐색 최적화는 ARA의 최적화와 실시간성, 연산량 감소를 위한 유의미한 개선 방향이며, 후속 개선안으로 고려하기에 충분하다고 생각하는 바이다.

8.6 타 포맷 간 전이

ARA의 동적 사전 확장 시스템은 텍스트 입력에만 국한된 구조가 아니다. 그 작용 기전은 단순히 입력과 이미 학습된 항목 간의 유사도 계산과 그에 따른 사전 확장이라는, 형태에 무관한 연산 흐름으로 이루어지기 때문이다. 즉, 핵심은 입력이 어떤 종류든 간에, 기존에 축적된 항목들과의 유사도를 측정하고, 그 결과에 따라 새로운 고윳값을 부여하거나 기존 클러스터에 포함시키는 흐름이며, 이는 텍스트 외의 형태로도 쉽게 확장 가능하다.
예를 들어, 이미지의 경우에는 사전의 항목을 잘 학습 된 CNN의 커널 필터로 사용하여, 입력 이미지와의 합성곱, ReLU, 정규화 등의 전처리 연산을 통해 유사도 기반 사전 발화가 가능하다.
음성의 경우에도 FFT를 통한 주파수 스펙트럼 기반 비교를 통해 입력과 음소 간의 유사도를 계산할 수 있으며, 이는 곧 음향 사전의 확장 역시 같은 방식으로 구현 가능함을 의미한다.
이처럼 기존 입력에 대해 사용되던 전처리 연산(CNN, FFT 등)을 변형하여 그대로 유사도 추출 연산으로 활용한다면 ARA의 동적 사전 구조는 자연어뿐 아니라 이미지, 음성, 센서 신호 등 다양한 입력 포맷으로의 확장이 가능하다.
또한, 이러한 입력 포맷은 각각의 사전에 대해 독립적으로 작용할 수 있으며, 단 하나의 신경망으로 수렴될 수도 있다. ARA는 이전 8.2절에서 제안한 것과 같이 각 사전에 대한 MLP의 출력들을 병렬로 연결하여 종합 MLP에 입력하는 것이 가능한 구조이다. 즉, 다양한 형태의 자유로운 자연 입력들을 종합하여 정해진 특징 맵으로 추출이 가능한 멀티모달 임베딩 구조가 될 수 있다. 이는 시각, 청각, 촉각 등을 통해 얻은 대상의 기억을 엮어 공감각이라는 것을 만듦과 동시에 그 기억과 연관시키는 인체의 인지 특성을 모방하였다고 볼 수 있다.
이에 따라 ARA가 하나의 임베딩 구조에 그치지 않고, 입력-표현 연결 방식의 범용 아키텍처가 될 수 있음을 시사한다.

8.7 결론 및 향후 과제

본 논문에서는 기존의 고정 벡터/빅데이터 기반 임베딩 기법의 한계를 극복하고자, 고유값(Eigen, Encode) 및 ARAS 유사도 기반 동적 사전 확장 구조와 희박 행렬 기반 특징 추출 신경망을 결합한 ARA를 제안하였다.
이를 위해 기존 Cos 유사도의 한계를 개선한 ARAS 유사도를 새롭게 정의하였고, 수동 벡터 지정이 아닌 입력 항목 자체가 고유한 벡터로 변환 가능한 구조를 설계하였다. 또한 유사도와 고윳값Eigen을 기준으로 한 동적 확장 사전 구조를 제안하여, 거의 무한에 가까운 자유

도의 자연어 입력을 가능하게 하였으며, 그 결과로는 지정된 크기로 추출 된 특징이 출력되도록 하였다.

ARA는 기존 자연어 임베딩의 패러다임인 의미 기반 분류와는 다른, 형태 기반 분류라는 방향으로 접근하여, 구조적으로는 생물학적 커넥톰에서 영감을 받은 7계층 구조, 처리 방식에서는 동적 확장 사전과 형태 및 순서 강조 기반의 벡터 추출기, 학습 구조에서는 레이블이 없는 상태에서도 오토인코더 방식으로 자율 학습 가능한 특징 추출기라는, 세 가지 측면에서 의미 기반 언어 처리 구조의 새로운 가능성을 제안하였다.

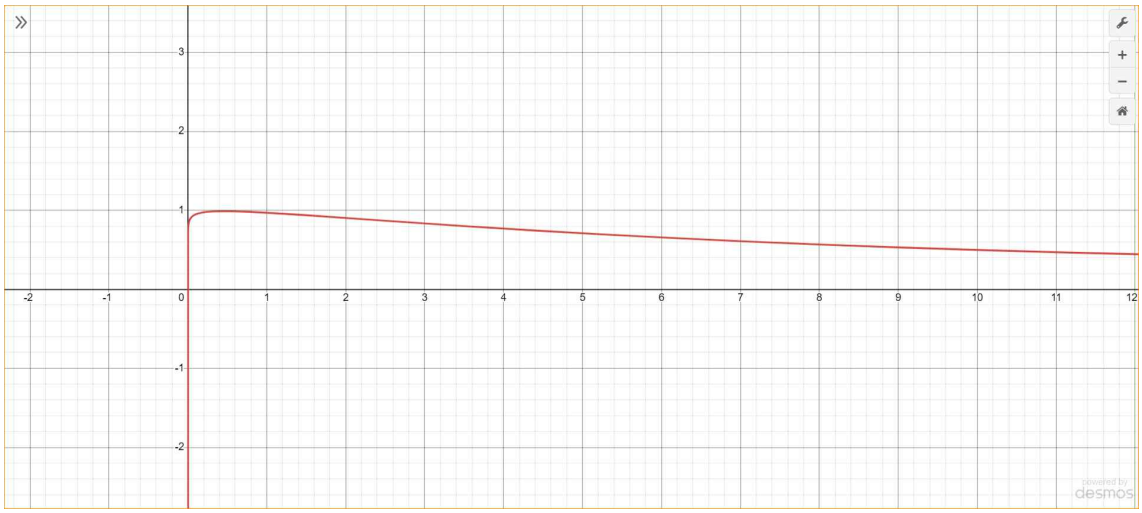
물론 현재의 ARA는 절대적인 분류 정확도, 계산 자원 부담, 고유 하이퍼 파라메터의 어려운 튜닝이라는 면에서 실용적 제약이 존재하며, 특히 다국어 지원 및 고급 어휘 학습 시 고윳값 고갈 문제는 향후 해결해야 할 중요한 과제 중 하나이다.

결론적으로 ARA는 단순한 임베딩 대체 도구가 아닌, 현실의 자연어와 정제된 인공신경망을 높은 자유도로 이어주는 가교이며, 트랜스포머와는 다른 방향에서 문단/문장/단어/철자를 분석하여 특징을 추출하는 새로운 방식이 될 수 있을 것이다.

부록 (Appendix)

Append.1 감쇄수식 그래프

$$Y = \exp(-0.125x) + 0.125*\log(5x)$$



상한 (0.46, 0.98934). x > 0 이상일 경우의 점근적 하한은 약 y = 0.3
일반적인 어휘의 경우 10글자 이내이므로 해당 함수는 약 1 ~ 0.5 이내에서 점진적 감쇄가 이루어진다

Append.2 고윳값 사전 클러스터링 일부 샘플

해당 항목에서는 단어 사전에서 클러스터링의 효과를 가장 잘 보였던 고윳값 대역폭들을 소개한다.

A. 고윳값 대역폭 75 +- 0.5

	어휘	고윳값	어휘	고윳값	어휘	고윳값
1	setting-hen	74.852	beaming	75.022	singing	75.197
2	morning	74.867	waiting	75.022	emotion	75.203
3	killing	74.886	forming	75.023	heading	75.204
4	sinking	74.888	resting	75.025		
5	against	74.893	shining	75.026		
6	dancing	74.907	standing	75.026		
7	highest	74.910	cursing	75.035		
8	melting	74.912	talking	75.048		
9	hanging	74.919	helping	75.060		
10	panting	74.923	barking	75.064		

11	getting	74.927	sicking	75.079		
12	leaving	74.938	feeding	75.082		
13	chasing	74.951	tracing	75.089		
14	walking	74.956	evening	75.103		
15	passing	74.977	running	75.118		
16	sitting	75.000	jumping	75.133		
17	burning	75.004	rooming	75.134		
18	knowing	75.007	playing	75.145		
19	feeling	75.009	smiling	75.166		
20	leaning	75.015	looking	75.179		

B. 고윳값 대역 폭 100.5~103.5

	어휘	고윳값	어휘	고윳값	어휘	고윳값	어휘	고윳값
1	marked	100.843	palace	101.019	head	102.034	pleased	103.061
2	boiled	100.851	wanted	101.021	shed	102.038	excited	103.069
3	shaded	100.852	mashed	101.022	wide	102.043	carriages	103.071
4	coldly	100.859	leaned	101.023	idea	102.052	stained	103.081
5	yelled	100.866	rushed	101.024	lied	102.057	replied	103.082
6	amazed	100.870	picked	101.030	need	102.057	climbed	103.087
7	catsup	100.871	barred	101.031	deal	102.066	trailed	103.088
8	talked	100.872	hugged	101.041	feel	102.067	riddled	103.094
9	damned	100.875	indeed	101.041	lie	102.070	midday	103.097
10	opened	100.875	crawled	101.043	Life	102.083	bloated	103.098
11	rolled	100.880	advice	101.045	view	102.122	cracked	103.114
12	passed	100.880	pushed	101.052	paid	102.125	invited	103.116
13	hopped	100.890	leaped	101.065	Field	102.126	carriage	103.116
14	gnawed	100.896	scared	101.070	dead	102.172	dropped	103.127
15	filled	100.899	amused	101.073	terrific	102.793	maidens	103.169
16	canvas	100.900	capable	101.080	granted	102.846	fancies	103.201
17	blazed	100.911	killed	101.089	enemies	102.851	Emerald	103.225
18	jumped	100.912	seemed	101.094	maiden	102.872		
19	coiled	100.917	looked	101.095	reached	102.878		
20	cellar	100.927	waddled	101.097	marched	102.899		
21	clever	100.933	peeled	101.101	treated	102.904		
22	missed	100.934	kissed	101.107	floated	102.913		
23	clearer	100.935	showed	101.114	worried	102.913		
24	cooked	100.937	cannot	101.117	extreme	102.917		
25	wilted	100.942	cables	101.118	marriage	102.920		
26	Looked	100.953	rubbed	101.119	bargain	102.941		
27	helped	100.956	played	101.120	dressed	102.965		
28	hatred	100.960	declare	101.140	corridor	102.969		
29	warned	100.962	matted	101.141	daytime	102.974		
30	tipped	100.968	walked	101.171	knocked	102.978		
31	gouged	100.968	worked	101.175	remained	102.981		
32	gloved	100.968	danced	101.180	pleaded	102.987		
33	melted	100.971	life	101.822	wrapped	102.992		
34	tapped	100.971	feet	101.916	visited	102.997		

35	candle	100.972	wife	101.929	merrily	102.997		
36	Wicked	100.974	laid	101.936	thanked	102.999		
37	needed	100.977	deed	101.941	married	103.000		
38	smiled	100.978	said	101.942	dripped	103.011		
39	washed	100.980	held	101.945	started	103.019		
40	landed	100.982	read	101.956	rebuild	103.027		
41	wicked	100.988	fled	101.969	carried	103.039		
42	locked	100.988	lead	101.979	smelled	103.043		
43	Indeed	100.993	ride	101.984	entered	103.044		
44	called	101.000	used	101.985	hundred	103.048		
45	nailed	101.001	died	102.000	snapped	103.049		
46	lifted	101.001	tied	102.000	decided	103.050		
47	candles	101.007	seed	102.005	angrier	103.060		
48	tucked	101.012	does	102.020	carrying	103.060		
49	pulled	101.014	spud	102.020	debased	103.060		
50	change	101.014	done	102.031	arrived	103.061		

Append.3 ARA 코드 구조 요약
[https://github.com/jun-Bridge/ARA.git]

Append.4 출처
--찾아서 넣을 것

250521_2200 논문 초안