국민대학교
전자정보통신대학
컴퓨터공학부

캡스톤 디자인 I

종합설계 프로젝트

프로젝트 명	asi(a security insight)
팀 명	assist(a security safety important special team)
문서 제목	중간보고서

Version	1.0
Date	2020-04-19

팀원	손 현기 (조장)
	김 주환
	김 호준
	오 예린
	이 동윤
	RUSLAN
지도교수	윤 명근 교수

문서 정보 / 수정 내역

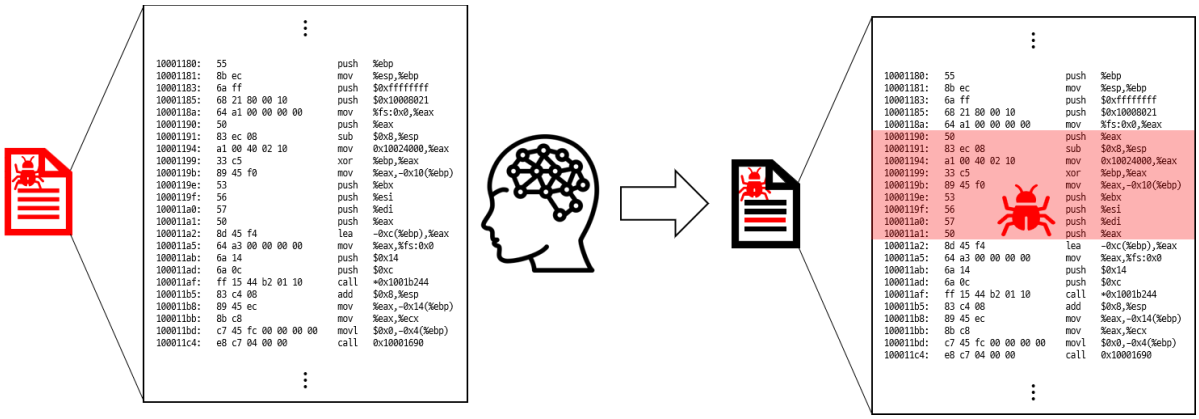
Filename	중간보고서-asi(a security insight).doc
원안작성자	김주환
수정작업자	

수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2020-04-19	김주환	1.0	최초 작성	전문에 대한 초안 작성

목 차

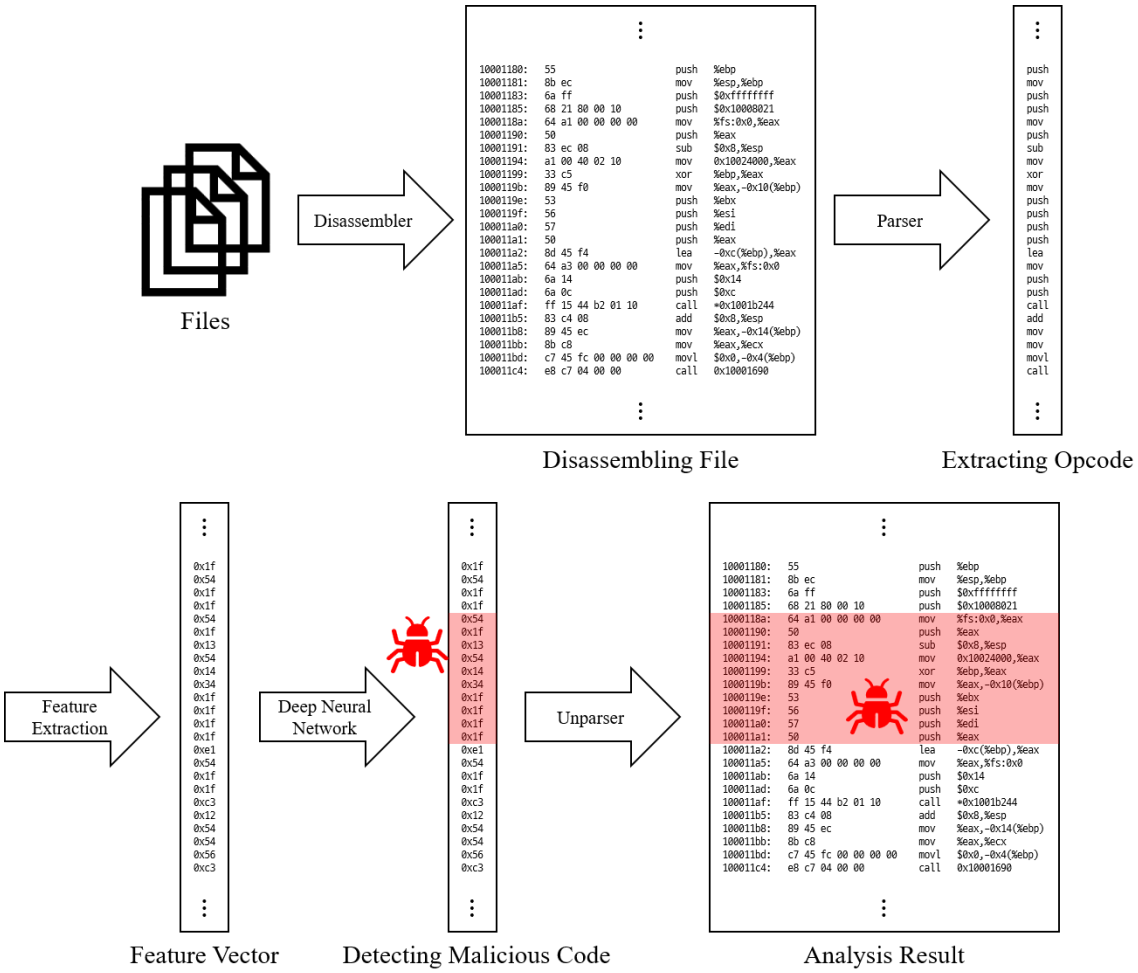
1 프로젝트 목표

본 프로젝트의 목표는 [그림 1]과 같이 악성코드 실행파일을 입력으로 받아 악성 행위를 수행할 것으로 예상되는 어셈블리어 코드를 추출하는 딥러닝 기반 소프트웨어를 개발함으로써 전문가의 악성코드 분석시간을 줄이는 것이다.



[그림 1] 제안하는 소프트웨어의 목표

구현하고자 하는 소프트웨어의 구체적인 흐름은 [그림 2]와 같다. 분석하고자 하는 파일이 입력되면 역어셈블러(disassembler)를 이용하여 어셈블리어를 추출하고, 추출한 어셈블리어 중 니모닉(mnemonic)을 파서를 이용해 추출한다. 신경망을 이용해 학습 및 분류를 하기 위해서는 입력을 특징벡터(feature vector)로 변환하는 과정이 필요하므로, 단어 임베딩(word embedding)과 같은 알고리즘을 이용하여 추출한 니모닉을 특징 벡터로 변환한다. 신경망은 특징 벡터를 입력 받아 악성 행위를 수행할 것으로 예측되는 니모닉을 추출한다. 마지막으로 원본 어셈블리어 파일 중 악성 행위를 수행하는 부분을 요약한 파일을 출력한다.



[그림 2] 제안하는 소프트웨어의 분석 방법 개요

2 수행 내용 및 중간결과

2.1 계획서 상의 연구내용

본 프로젝트의 세부 목표는 [표 1]과 같다. 1 단계는 신경망 학습을 위한 데이터를 수집 및 전처리하는 단계이다. 2 단계는 순환신경망(Recurrent Neural Network, RNN) 학습을 위해 니모닉을 특징 벡터로 변환하는 단어 임베딩 방법론을 개발하는 단계이다. 3 단계는 본 프로젝트의 목표인 악성 행위 추출을 위한 신경망을 구현 및 학습시키고, 이를 사용자가 편리하게 사용하기 위해 웹 서비스를 구축하는 단계이다. [표 2]는 이상의 목표에 대응하는 결과물 목록을 나타낸 것이다.

[표 1] 프로젝트의 세부 목표

1 단계 연구 목표
– 정상파일 및 악성코드 파일 데이터셋 확보 – 니모닉 추출 알고리즘 개발
2 단계 연구 목표
– 딥러닝 기반 특징 벡터 추출 도구 개발
3 단계 연구 목표
– 딥러닝 기반 이상탐지 알고리즘 확보 – 딥러닝 기반 자동 악성행위 코드 추출 도구 개발 – 악성코드 분석 보조도구 제공을 위한 서버 구축

[표 2] 프로젝트의 세부 결과물

1 단계 결과물
- 정상파일을 수집하는 웹 크롤러 - 어셈블리 코드로부터 니모닉을 추출하는 파서 - 정상파일 및 악성파일 데이터셋
2 단계 결과물
- 단어 임베딩 소프트웨어 (Word2Vec, FastText, Doc2Vec) - 1 단계의 데이터셋에 대한 특징 벡터
3 단계 결과물
- 각 하이퍼 파라미터별 실험 결과 - 최적화된 신경망 모델 - 자동 악성행위 코드 추출 도구 - 분석 파일 업로드 및 결과 시각화를 위한 웹

2.1.1 데이터 수집

신경망을 적절히 학습하기 위해서는 양질의 데이터가 필요하다. 우리는 학습에 필요한 정상 파일과 악성 파일을 얻기 위해 Microsoft Kaggle 프로젝트 malware prediction의 데이터셋, 한국인터넷진흥원의 정보보호 R&D 데이터 챌린지의 데이터셋 등을 이용한다. 추가로 웹 크롤러(web crawler)를 직접 개발하여 충분히 많은 정상파일을 확보한다.

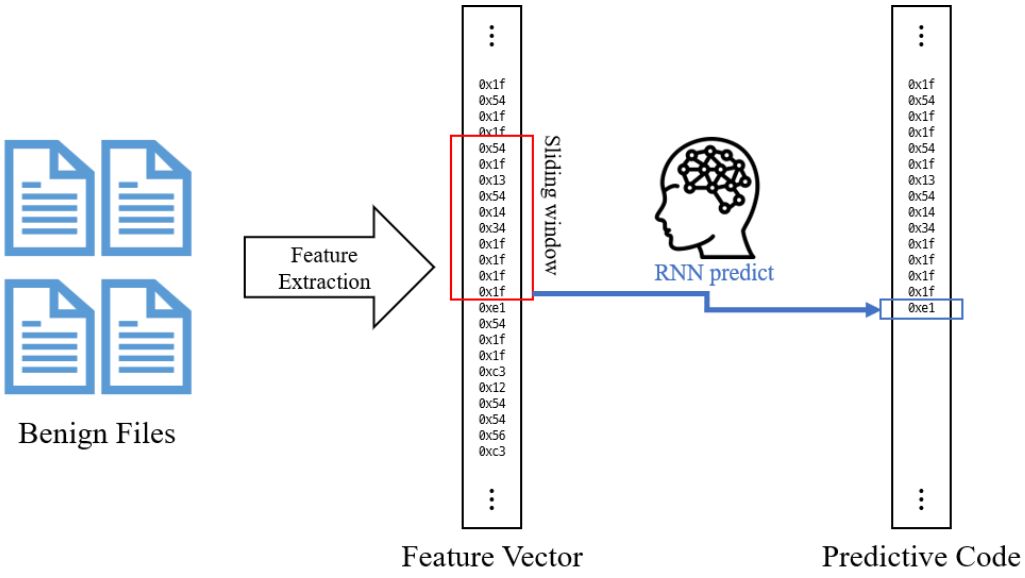
2.1.2 딥러닝 기반 특징 벡터 추출 도구

학습을 위해서는 수집한 로우 데이터(raw data)로부터 어셈블리어를 추출하고, 어셈블리어를 적합한 특징 공간에 사상시키는 단어 임베딩 방법을 개발해야 한다. 우리는 수집한 실행 파일로부터 어셈블리어를 추출하기 위해 상용 역어셈블러 프로그램인 IDA Pro를 사용한다. 단어 임베딩은 Word2Vec, Doc2Vec, FastText와 같은 딥러닝 기반 알고리즘을 적용한다. 본 목의 목적은 여러 알고리즘에 대해 단어 임베딩을 수행하고, 결과를 시각화 함으로써 적합한 임베딩 방법론을 선정하

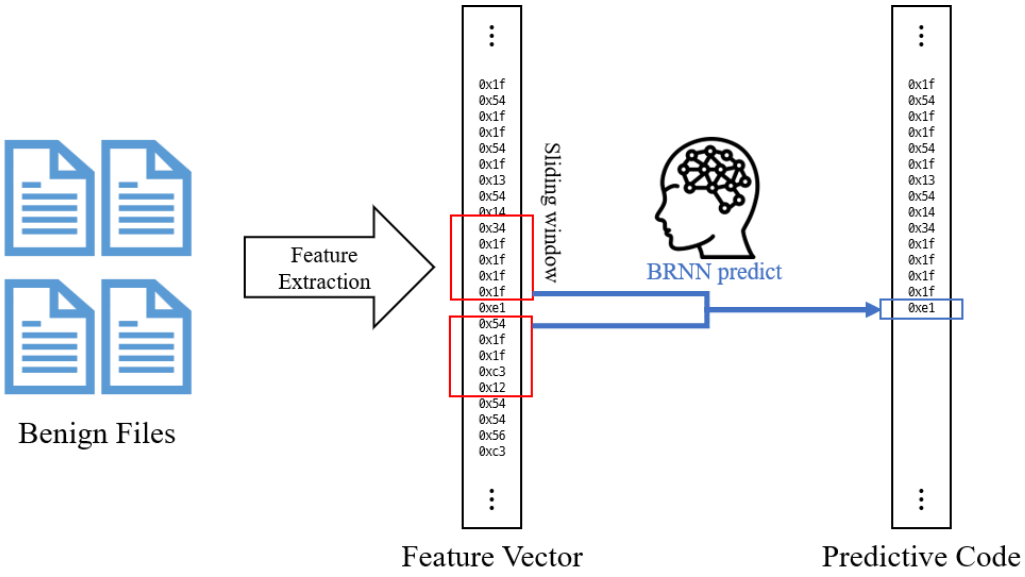
는 것이다.

2.1.3 딥러닝 기반 악성 행위 추출 도구

우리는 RNN을 기반으로 정상파일의 명령어의 언어모델을 학습시킴으로써 악성파일 중 악성 행위를 수행하는 부분을 추출한다. 학습단계에서는 정상파일을 이용하여, 이전 니모닉으로 다음 니모닉을 예측하는 방식과 주변 니모닉으로 중심 니모닉을 추측하는 방식, 두 가지 방식을 실험하고, 최적의 신경망을 채택한다. 이때 사용하는 RNN의 구조로는 Vanilla RNN, LSTM(long short-term memory models), GRU(gated recurrent unit)을 사용한다.

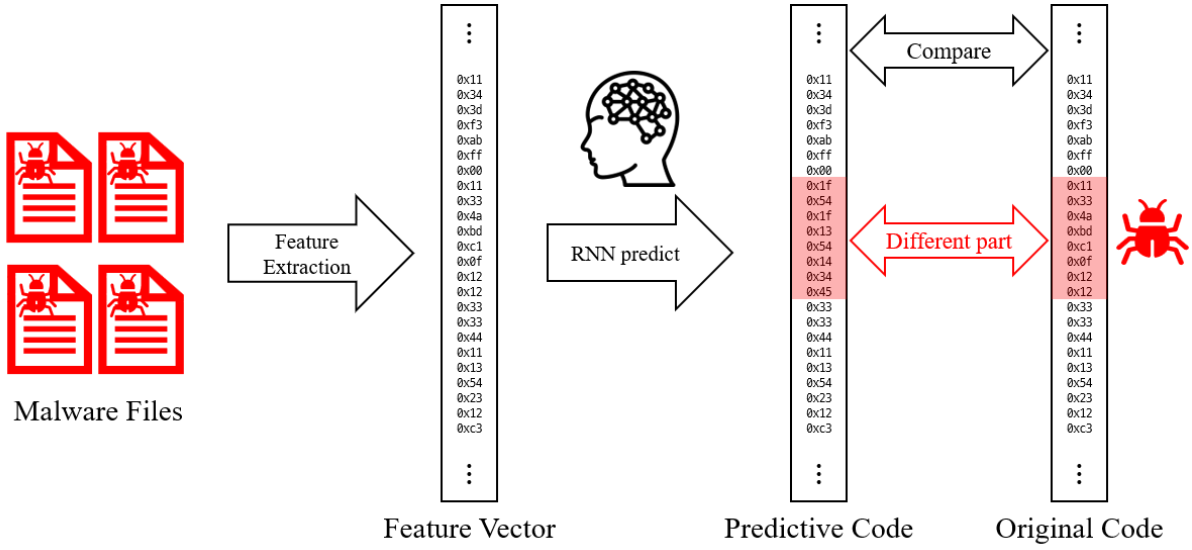


[그림 3] 이전 니모닉으로 다음 니모닉을 예측하는 학습 방안



[그림 4] 주변 니모닉으로 중심 니모닉을 예측하는 학습 방안

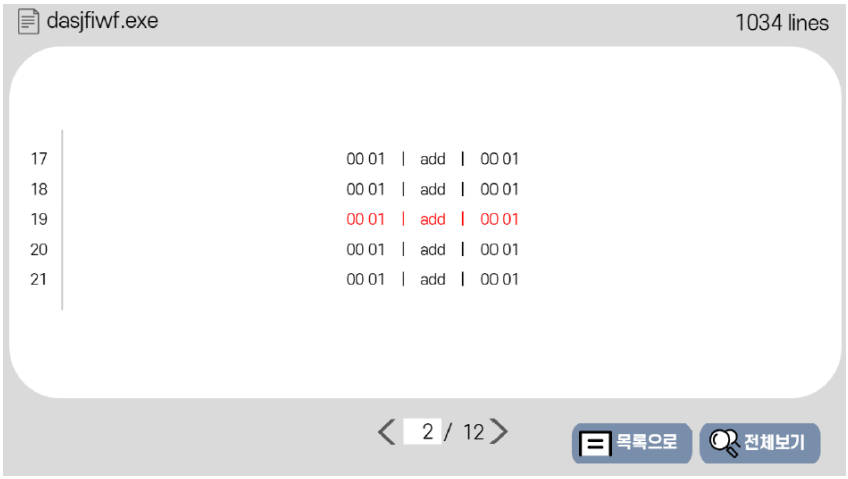
악성 행위 코드를 추출하는 방법은 [그림 5]와 같다. 학습된 신경망이 예측한 니모닉과 원본 니모닉을 비교했을 때 신경망이 제대로 예측하지 못하는 지점을 악성 행위를 수행하는 부분이라 추정한다.



[그림 5] 악성 행위 코드 추출 방안

2.1.4 웹 서비스

제안된 시스템을 사용자 친화적으로 제공하기 위하여 웹을 기반으로 서비스를 제공한다. 웹은 사용자가 악성으로 의심되는 파일을 업로드하면 [그림 6]과 같이 코드를 분석한 결과를 시각화하여 나타낸다.



[그림 6] 웹 기반 분석 결과 시각화 안

2.1.5 수행 계획

[표 3] 프로젝트 수행 계획

항목	세부내용	1 월	2 월	3 월	4 월	5 월	6 월
요구사항분석	요구 분석						
	SRS 작성						
관련분야연구	딥러닝 기술 연구						
	관련 논문 동향조사						
구현	웹 크롤러 제작 및 데이터 수집						
	파서 구현						
	단어 임베딩						
	신경망 구현						
	웹 서버 구축						

	웹 프론트엔드 개발						
테스트	시스템 테스트						

2.2 수행내용

2.2.1 데이터 수집

학습에 필요한 정상 및 악성 파일을 수집하기 위하여 Microsoft의 Kaggle 프로젝트인 Microsoft malware classification[1]의 데이터셋, 한국인터넷진흥원의 정보보호 R&D 데이터 챌린지의 데이터셋을 활용한다. Microsoft malware classification 프로젝트는 악성 파일의 패밀리를 분류하는 것을 목표로 한다. 데이터셋은 바이트 파일 10,868개와 어셈블리 파일 10,868개로 총 21,738개이며, 각 악성 파일의 패밀리가 제공된다. 한국인터넷진흥원 정보보호 R&D 데이터 챌린지[2]는 2017년부터 시행된 경진대회이며 악성코드를 분류하는 것을 목표로 한다. 데이터셋은 정상 파일 10,000개와 악성 파일 10,000개이며, 각 파일의 정상/악성 여부가 제공된다.

본 프로젝트에서는 학습을 위해 정상 파일을 사용하므로 양질의 정상 파일을 확보하는 것이 필수적이다. 이를 위해 우리는 정상 파일을 수집하기 위한 크롤러를 제작했다. 크롤링 대상은 높은 신뢰도로 정상이라 판정할 수 있는 시스템의 DLL(dynamic link library) 파일과 Steam 사의 게임 인스톨러를 수집했다. 각 크롤러는 GitHub의 crawler 폴더에서 확인할 수 있다.

얻은 파일은 실행파일과 DLL 파일이므로 신경망 학습을 위해서는 파일 중 어셈블리 코드를 추출하는 과정이 필요하다. 우리는 이를 위해 상용 역어셈블러 프로그램인 IDA Pro를 사용했으며, 역어셈블된 결과 중 니모닉을 추출하기 위해 파서를 Python으로 구현했다. 구현한 파서는 IDA Pro 상에서 동작하며, 역어셈블된 결과를 기본 블록(basic block) 단위로 나눈 것을 Python 피클(pickle)로 저장하도록 구현했다.

데이터 수집 및 파싱은 제안서 상의 개발 계획에 맞게 3월에 개발을 완료하였다.

[1] https://www.kaggle.com/c/malware-classification/overview

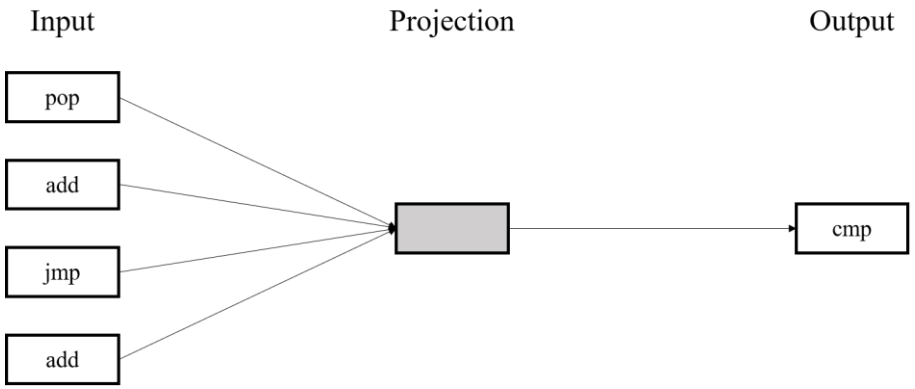
[2] http://datachallenge.kr/

2.2.2 딥러닝 기반 특징 벡터 추출 도구

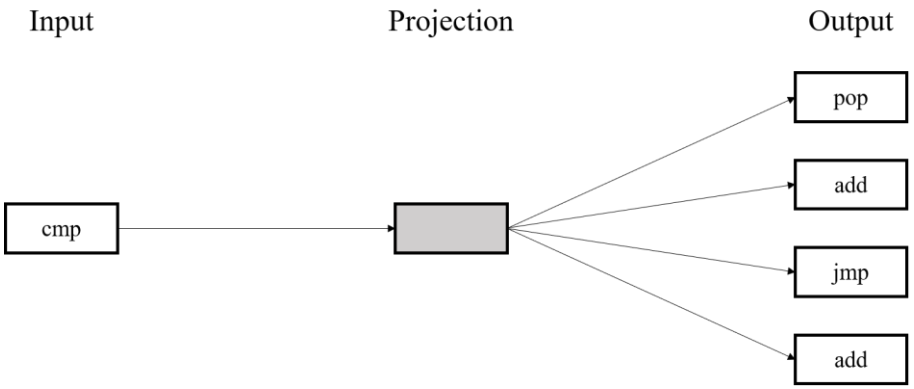
가) 관련 연구

RNN을 적절히 학습시키기 위해서는 니모닉을 적합한 벡터로 변환시키는 과정, 즉 단어 임베딩 과정을 적절히 수행해야 한다. 우리는 이를 위해 딥러닝 기반의 Word2Vec, Doc2Vec, FastText 알고리즘을 적용하여 임베딩 신경망을 구성한다.

Word2Vec은 가장 많이 사용되는 임베딩 방법론 중 하나로, 기존 원핫 인코딩(one-hot encoding) 방식과 다르게 더 낮은 차원의 벡터로 많은 단어를 임베딩할 수 있으며, 유사한 단어를 가까운 공간에 사상시키는 방식이다. 예를 들어 사과, 배, 서울 세 단어를 임베딩한다면 원핫 인코딩은 단어간의 관계를 고려하지 않고 일괄적으로 임베딩하지만, Word2Vec은 사과와 배는 가까운 공간에, 서울은 먼 공간에 사상시키는 방식이다. Word2Vec은 주변 단어를 이용해 중간 단어를 예측하는 방식인 CBOW(continuous bag of words)와 중간 단어로 주변 단어를 예측하는 방식인 Skip-Gram 방식이 있다. 예를 들어 니모닉이 pop, add, cmp, jmp, add이고, 윈도우 사이즈가 2이며, cmp 명령어에 대해 학습하는 단계라면 CBOW는 [그림 7]과 같이 주변 니모닉인 pop, add, jmp, add를 입력 받아 cmp를 예측하도록 학습하며, Skip-Gram은 [그림 8]과 같이 중심 니모닉인 cmp를 입력 받아 주변 니모닉인 pop, add, jmp, add를 예측하도록 학습한다. 일반적으로 Skip-Gram 방식이 CBOW 방식보다 효과적이라 알려져 있으며, 우리의 학습 결과 역시 Skip-gram이 CBOW에 비해 더 효과적으로 임베딩을 수행했다.



[그림 7] CBOW 학습 방안



[그림 8] Skip-Gram 학습 방안

Word2Vec 학습시 고려해야 할 주요 하이퍼 파라미터는 출력 벡터의 차원, 윈도우 크기, 최소 단어의 수, 에포크(epoch), 학습률이다. 출력 벡터의 차원은 임베딩 결과로 나오는 벡터의 차원을 의미하고, 윈도우 크기는 학습에 사용하는 주변 단어의 개수를 의미한다. 위의 그림은 윈도우 크기가 2인 경우이다. 최소 단어의 수는 등장 빈도가 적은 단어를 임베딩 시키지 않기 위해 필요하며, 이 숫자보다 단어가 적게 등장하는 경우 임베딩 시키지 않는다. 에포크와 학습률은 일반적인 딥러닝에서의 의미와 동일하다. 에포크는 전체 데이터의 학습을 반복하는 횟수를, 학습률은 오류 역전파시 가중치를 변화시키는 비중을 의미한다.

Doc2Vec은 Word2Vec의 변형으로 학습의 입력으로 단어뿐만 아니라 문서의 인덱스를 추가로 받는다. Doc2Vec은 각 문서별로 단어가 가까운 특징공간에 사상되는 것을 목적으로 한다. Word2Vec과 유사하게 주변 단어를 입력 받아 중심 단어를 예측하는 DBOW(distributed bag of words) 방식과 중심 단어를 입력 받아 주변 단어를 예측하는 Distributed-Memory 방식이 있다.

FastText는 Word2Vec의 변형으로 단어의 부분이 일치하는 단어는 유사한 단어임을 이용한다. 이를 위해 단어를 작은 단위로 나누는 방법론을 채택한다. 예를 들어 imul이라는 니모닉이 있고, 단어를 나누는 단위가 3이라면, imu, mul으로 나누어진 단어와 전체 단어 imul을 임베딩한다. 이후 mul이라는 니모닉을 임베딩할 때는 imul과 가까운 특징 공간으로 사상될 수 있다. FastText의 학습 방법은 Word2Vec과 동일하다.

나) 실험 결과

우리는 gensim 라이브러리를 이용하여 이상의 세 알고리즘을 구현했다. 하이퍼 파라미터는 기본 파라미터를 참고하여 윈도우 크기 10, 최소 단어 수 50, 에포크 10, 학습률 0.0002로 지정했다. RNN 구현시 모델의 너비를 적게 유지하면서 단어를 충분히 적합한 특징 공간으로 사상시키기 위

하여 우리는 특징 벡터의 차원을 8, 16, 32, 64, 128로 하여 실험했다. 학습 데이터는 5,000개를 활용했다. 현재 Word2Vec과 FastText에 대한 실험은 완료되었으며, Doc2Vec에 대한 실험은 4월 중으로 완료할 예정이다.

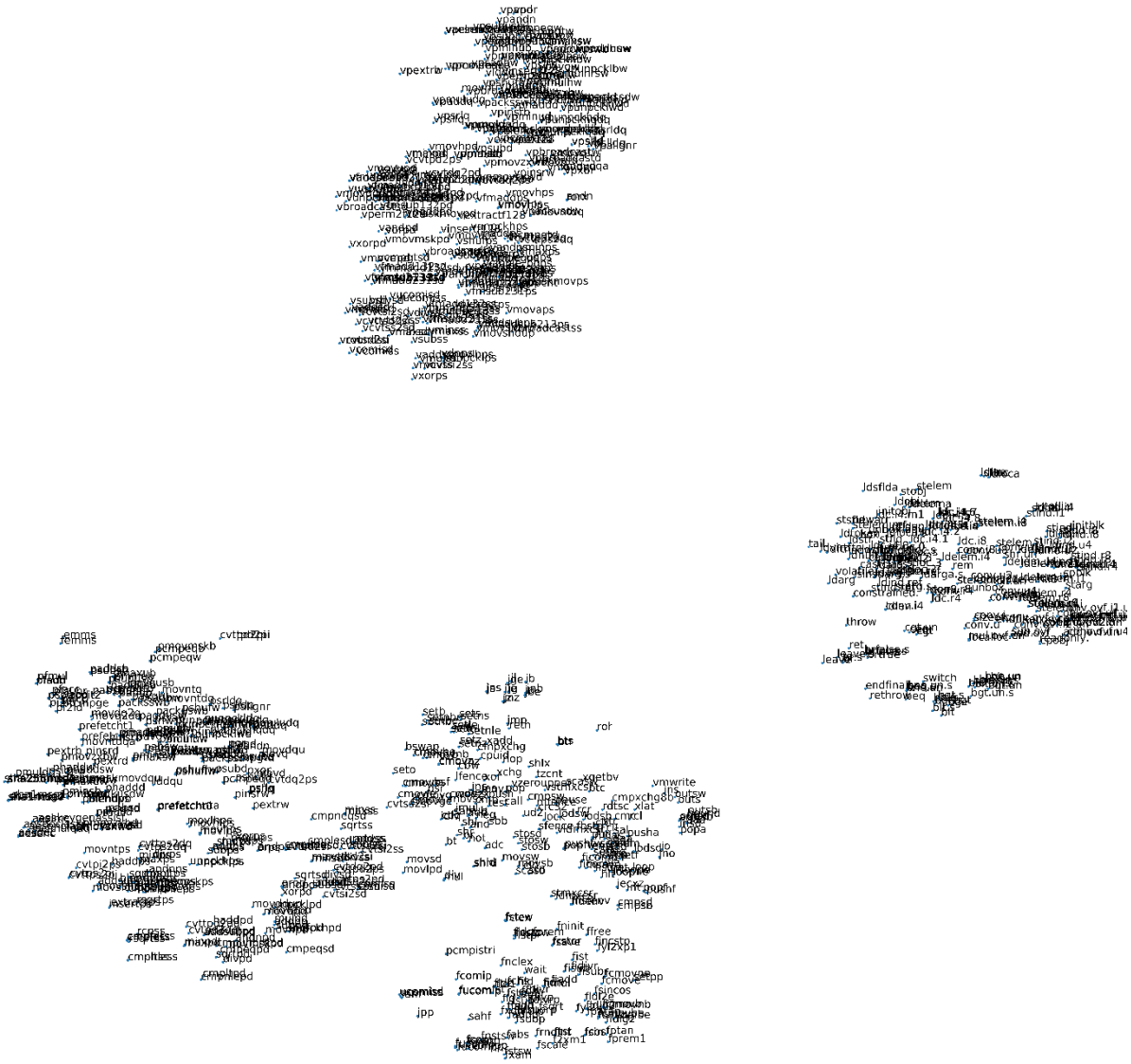
단어가 적합한 특징 공간으로 사상되었는지 확인하기 위해 우리는 다섯 개의 니모닉 mov, jmp, add, pop, push에 대해 가장 가까운 위치에 사상된 니모닉을 확인했으며, 사상시킨 결과를 t-SNE로 시각화했다. 그 결과 Word2Vec의 Skip-Gram 알고리즘을 적용했을 때 유사한 동작을 수행하는 니모닉끼리 가까운 특징 공간에 군집되는 것을 확인했으며, 벡터의 크기는 32 또는 64가 적합함을 확인했다. [표 4]는 각 특징 벡터의 크기별 가까운 니모닉을 나타낸 것이며,]와]는 t-SNE로 임베딩된 니모닉을 시각화한 결과이다. 나머지 실험 결과는 6장 부록에 정리하였다.

[표 4] Word2Vec(Skip-gram)의 각 니모닉별 가장 가까운 위치에 사상된 니모닉

Mnemomnic	vec 8	vec 16	vec 32	vec 64	vec 128
mov	lea setns xor test cmovnz cmovz push sets setz cmovs	lea xor cmovnz push cmovb shlx xchg cmovz add cmp	lea push xor xchg cmovnz cmovge cmovz inc sar movsx	lea push inc xchg setnz cmovge sar xor setnl cwde	lea push inc sar xor setnz cmovge imul xchg cmovg
jmp	jge jl jnz jnb jbe lea jb jg mov ja	jl jb jge jnz jg ja retn jle jz jnb	retn jl jge jg jb jle jnz jz jns jbe	retn jg lea jle jnz jl jge js ja jz	retn jl jle jns jnz jge js jb jbe jz
add	pextrb dec prefetcht1 phaddsw cmovz cmp	sub shl prefetcht0 mov movups dec	sub shl prefetcht0 prefetchnta mov movaps	sub shl imul sar mov lea	sub shl movaps mov imul lea

	stmxcsr psrldq movd movq	shlx adc inc cdq	sar pshufd cdq movups	neg subpd pshufd cmp	jg movups sar neg
pop	leave xor fninit vzeroupper fldl2e cmpsd retn sfence fcmovb fcmovnb	retn leave vzeroupper xor sfence pushf popf cmpltsd cmpnlesd cvtpd2pi	retn vzeroupper xor emms setz sfence leave setnl mov fninit	retn setz emms mov vzeroupper leave setns setnl setnz push	retn vzeroupper leave mov setz setnl setns jmp setnz push
push	jns lea setns js mov fnclex test stosd cmovs crc32	lea mov test fnclex jmp setns js fld1 fldz cmovns	mov lea cmovs cmovnz cmovge cmovns test cmovl fnclex movsx	lea mov stosd setnz setz cwde test fstp cmovnz cmovz	mov lea test stosd setz setnz fstp cmovnz fldz inc

[그림 9] Word2Vec(Skip-gram)의 벡터 크기가 32일 때 t-SNE로 시각화한 결과



[그림 10] Word2Vec(Skip-gram)의 벡터 크기가 64일 때 t-SNE로 시각화한 결과

2.2.3 딥러닝 기반 악성 행위 추출 도구

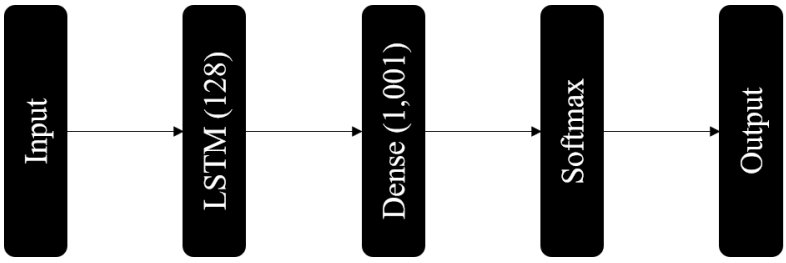
우리는 악성 행위 추출을 위해 두 가지 방법을 사용하여 실험을 수행했다. 첫 번째는 [그림 3], [그림 4]와 같이 하나의 주변 명령어로부터 하나의 명령어를 예측해 예측한 명령어와 다른 경우 해당 명령어가 이상 행위를 수행한다고 추정하는 방식이다. 두 번째는 함수 단위로 RNN 기반 오토인코더를 구현해 손실값이 특정 한계점보다 큰 경우 해당 함수가 이상 행위를 수행한다고 추정하는 방식이다. 실험에서 신경망은 Keras 라이브러리를 이용해 구현하였다.

가) 명령어별 이상탐지

주변 명령어로 하나의 명령어를 예측해 각 명령어의 이상 여부를 판정하는 방법은 다음과 같다.

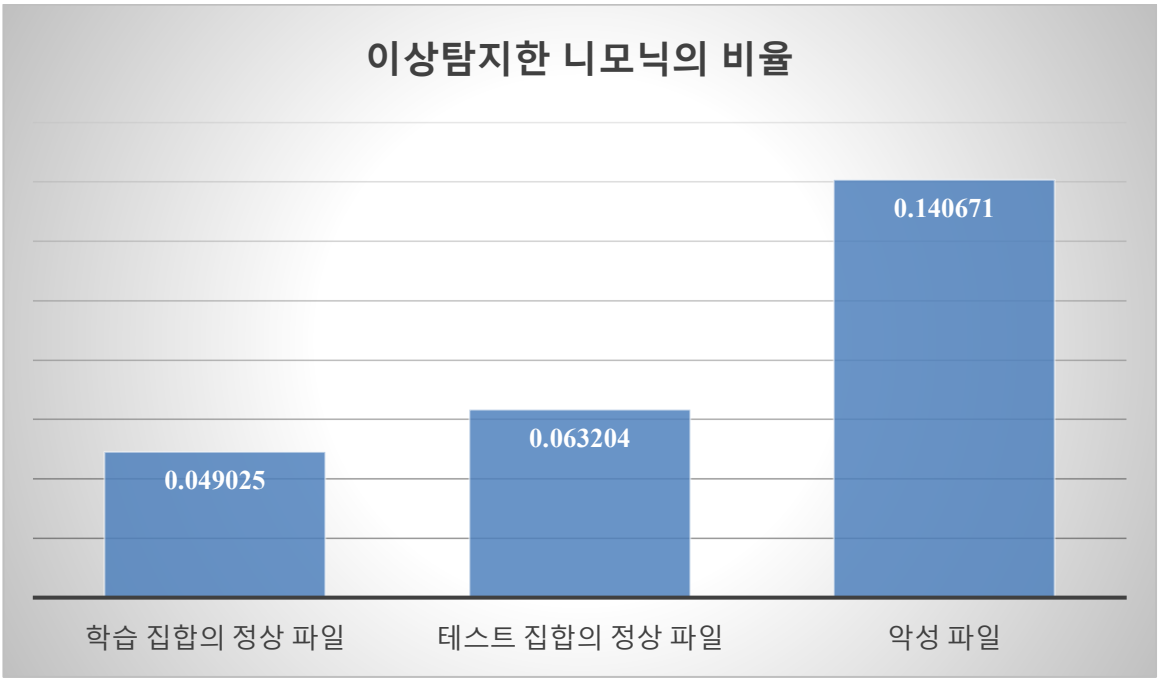
1. 특징 벡터를 임베딩 시킨다.
2. 특정 윈도우에 있는 명령어들을 입력 받아 다음 명령어, 혹은 중심 명령어를 예측한다.
3. 신경망이 예측한 명령어 중 상위 90% 이내에 포함되지 않으면 이상 명령어라 판정한다.

우리는 이전 명령어들로 다음 명령어를 예측하는 방식으로 실험을 수행했다. 하이퍼 파라미터는 다음과 같다. 단어 집합의 개수는 니모닉 1,000개와 알 수 없는 니모닉을 포함하여 1,001 개로 지정했으며, 윈도우 크기는 30으로 지정했다. 신경망은 [그림 11]과 같이 LSTM 한 층에 완전 연결층을 이은 신경망을 활용하였다.



[그림 11] 명령어별 이상탐지를 위한 신경망의 구조

1,000개의 파일에 대해 학습을 수행하고, 학습 집합의 정상 파일, 테스트 집합의 정상 파일, 악성 파일에 대해 이상탐지를 수행한 결과가 [그림 12]와 같다. 그림에서 악성 파일에 대한 이상탐지 비율은 정상 파일에 대한 이상탐지 비율의 두 배 이상이므로 학습된 신경망이 적절히 이상탐지를 수행할 것으로 추정할 수 있다. 그러나, 이 방법은 신경망의 학습 여부를 판정하기에 충분하지 않으므로 이후 3장에서 다룰 방식을 이용해 향후 추가 검증을 수행할 예정이다.



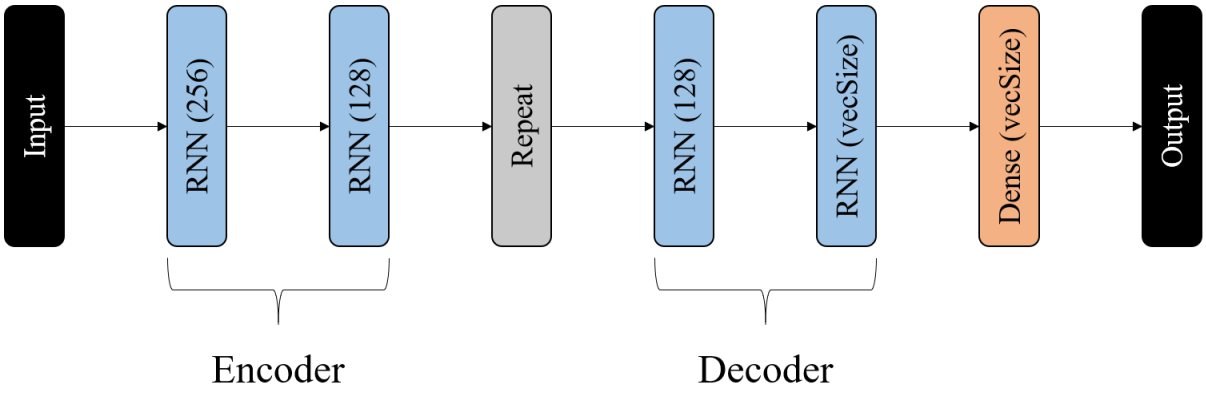
[그림 12] 테스트 집합별 이상탐지한 니모닉의 비율

나) 오토인코더 기반 이상탐지

오토인코더를 이용해 각 함수별 이상탐지를 수행하는 방법은 다음과 같다. 아래의 방법에서 우리는 한계점(threshold)을 0.1로 지정했다.

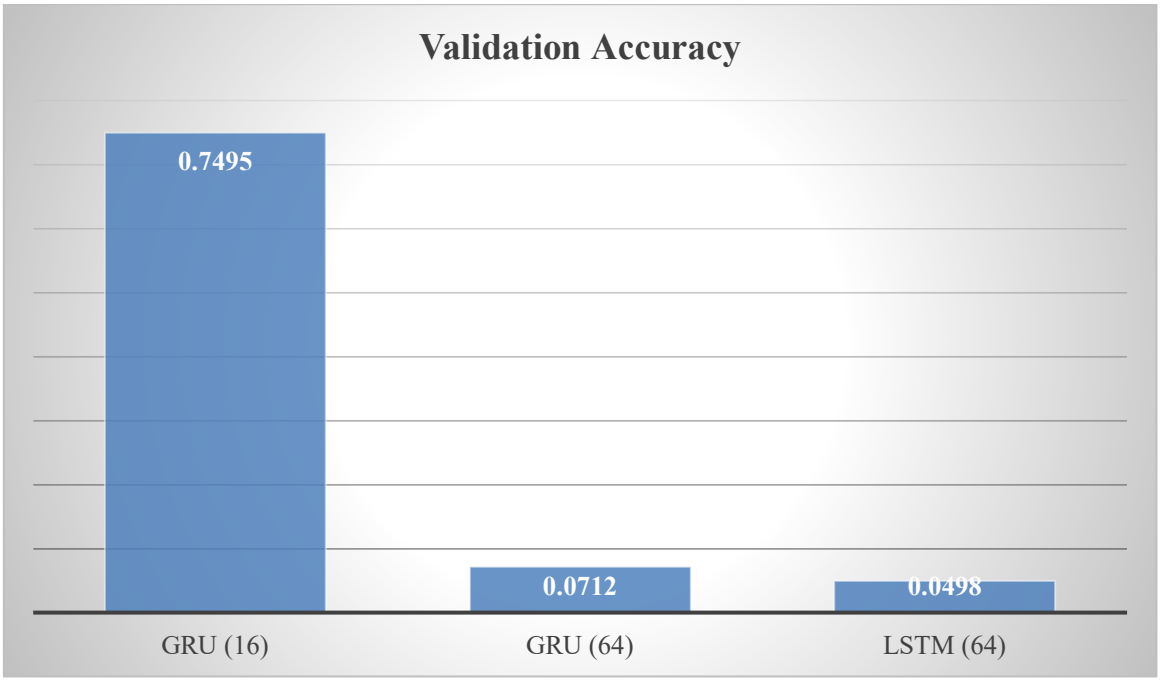
1. 특징 벡터를 임베딩 시킨다.
2. 각 함수별 명령어를 입력 받아 해당 함수의 명령어를 예측한다.
3. 신경망이 예측한 명령어와 원본 명령어의 MAE(mean absolute error)를 계산하여, MAE가 특정 한계점보다 크면 이상행위를 수행하는 함수라 판정한다.

실험한 오토인코더의 구조는 [그림 13]과 같다. 그림에서 RNN으로는 GRU와 LSTM을 사용했으며 임베딩 벡터의 크기(vecSize)는 16, 64로 지정해 실험했다. 학습 데이터는 정상 파일 10,000개를 사용했다.



[그림 13] 오토인코더 기반 이상탐지를 위한 신경망의 구조

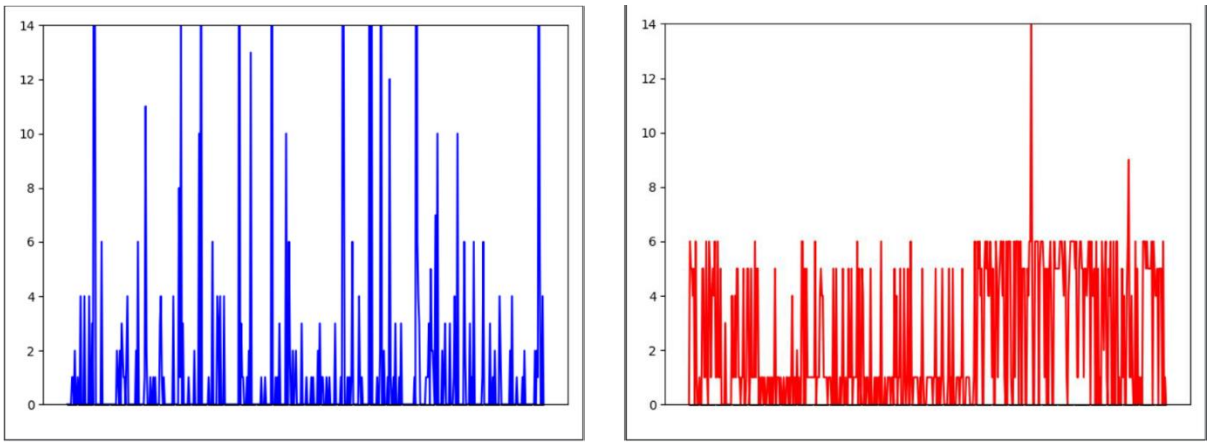
신경망 구조별 학습 결과가 [그림 17]와 같다. 그림에서 x축은 사용한 RNN 구조와 특징 벡터의 차원을 나타낸다. 학습 결과 GRU (16)은 테스트 집합의 손실값이 큰 반면, GRU (64)와 LSTM (64)는 낮으므로 실험 결과는 2.2.2 목의 결과와 같이 특징 벡터의 차원은 최소 32차원 이상이어야 함을 시사한다.



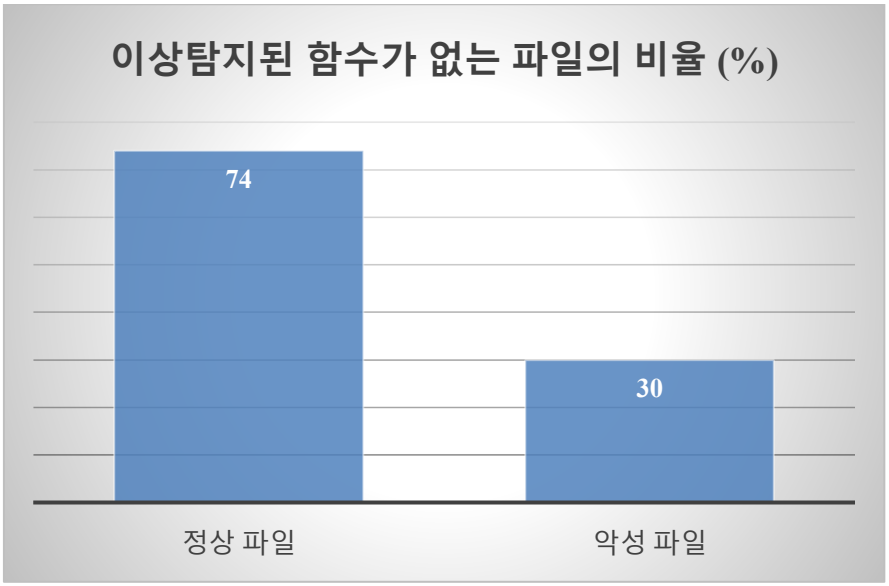
[그림 14] 신경망의 구조별 테스트 집합에 대한 손실값

신경망이 학습을 제대로 수행했는지 확인하기 위해 정상 파일 500개와 악성 파일 500개에 대해 이상 탐지된 함수의 개수를 확인하였다. [그림 15]는 각 파일별 이상 탐지된 함수의 개수를 도

식화한 것이다. 그림에서 정상 파일은 이상탐지가 되지 않거나, 10개 이상의 많은 함수에 대해 이상탐지가 되는 반면, 악성 파일은 대부분 6개의 함수에 대해 이상탐지가 되는 것을 볼 수 있다. 특정 정상 파일의 함수들에 대해이 이상탐지가 되는 이유는 학습에 충분히 많은 데이터를 사용하지 않았기 때문으로 추정된다. 한편, 모든 함수에 대해 이상탐지가 되지 않은 파일의 비율은 [그림 16]과 같다. 그림에서 정상 파일은 전체 500개의 파일 중 약 74%가 이상탐지된 함수가 없는 반면, 악성 파일은 약 30%가 이상탐지된 함수가 없었다. 학습 데이터 집합의 크기를 늘린다면 정상 파일의 비율은 향상되고, 악성 파일의 비율은 감소할 것으로 기대된다.



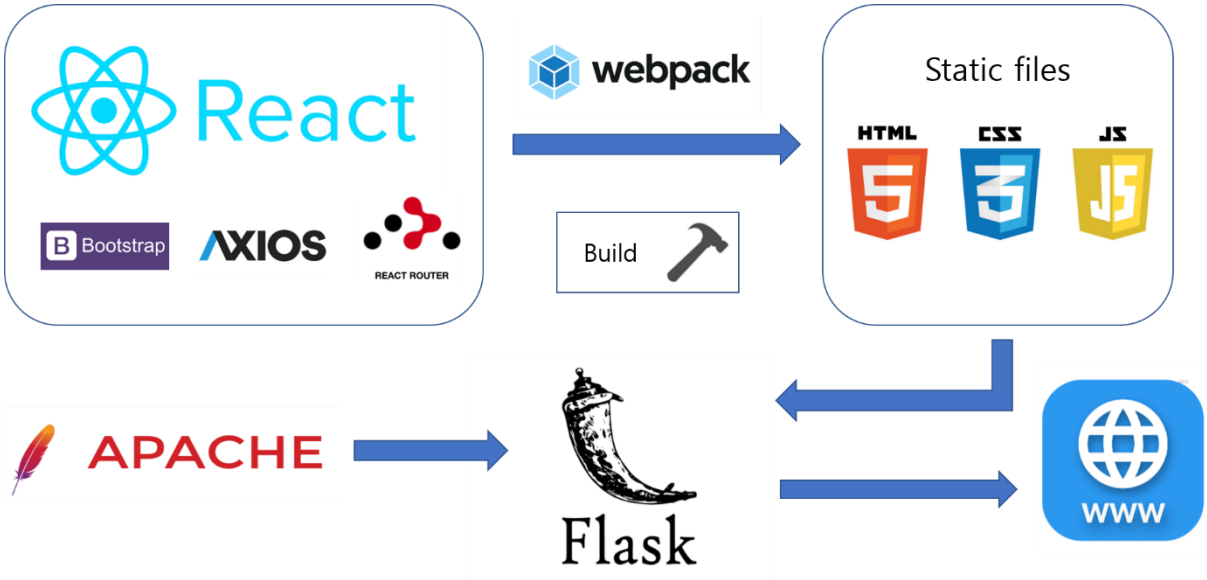
[그림 15] 정상/악성 파일의 이상탐지한 함수의 개수 (x축: 파일, y축: 이상탐지한 함수의 개수)
(왼쪽: 정상 파일, 오른쪽: 악성 파일)



[그림 16] 이상탐지된 함수가 없는 파일의 비율

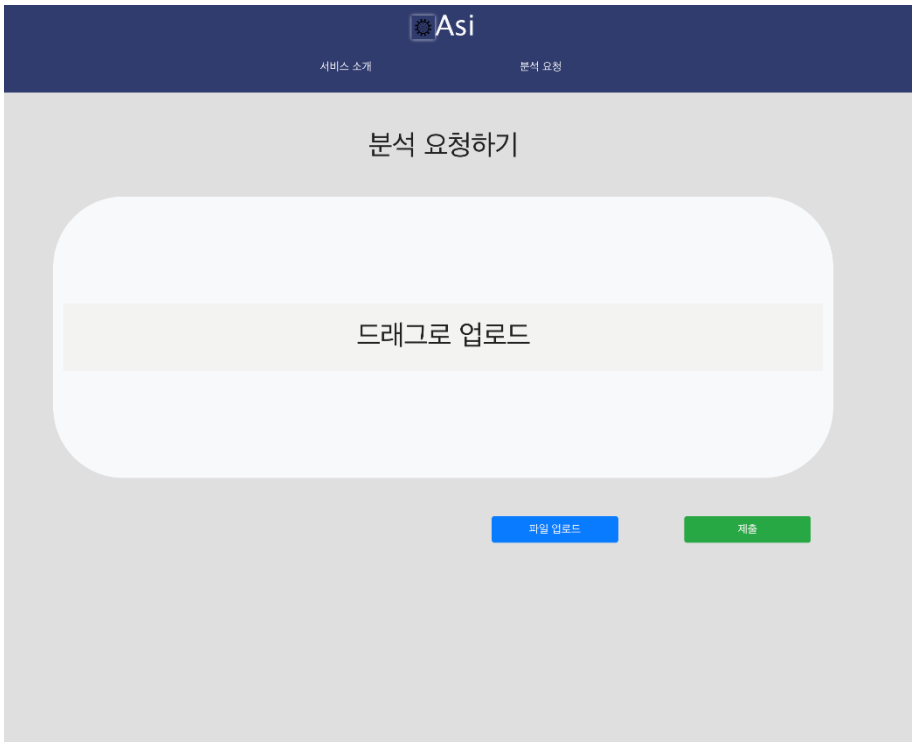
2.2.4 웹 서비스

사용자 친화적인 인터페이스 구축을 위해 웹 서비스를 구축하였다. 웹 구축을 개발 환경은 [그림 17]과 같다. 프론트엔드는 React 라이브러리를 기반으로 만들었으며 Axois 라이브러리를 이용해 Flast와 React 간의 파일 송수신을 구현했다. 서버는 Apache http로 구현하였다.

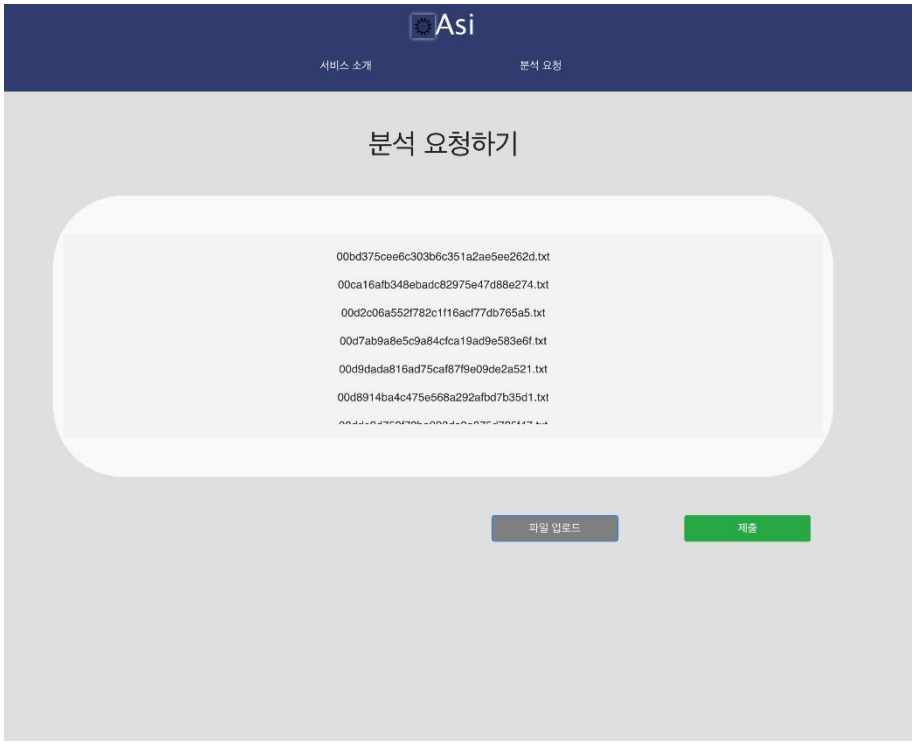


[그림 17] 웹 서비스 구축을 위한 라이브러리 구조

현재 웹은 파일 업로드 및 프로젝트 소개 화면을 구현하였다. [그림 18]은 초기 화면으로 파일을 직관적으로 업로드 할 수 있도록 드래그 앤 드롭(drag and drop) 방식으로 구현하였다. 업로드가 완료되면 업로드한 파일의 목록이 [그림 19]와 같이 나타난다.



[그림 18] 초기 화면



[그림 19] 업로드가 완료된 화면

또한, 사용자는 본 프로젝트의 개요를 웹에서 확인할 수 있도록 구현했다.

[그림 20] 프로젝트 소개 화면

3 수정된 연구내용 및 추진 방향

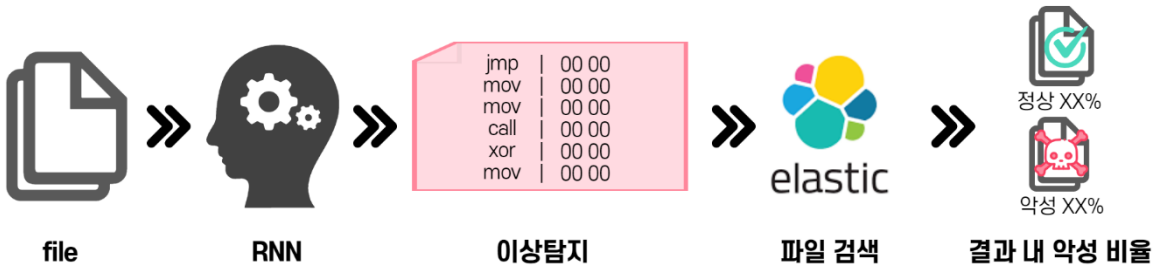
3.1 수정사항

상용 프로그램 중 본 프로젝트와 유사한 기능을 수행하는 프로그램이 없어 학습된 신경망의 성능을 측정하기 어렵다. 우리는 이를 해결하기 위해 기존 Elasticsearch를 이용해 판정한 이상 행위 코드가 적절한 코드인지 판별할 수 있는 기술을 추가로 개발한다. Elasticsearch 기반의 악성 여부 판정 방법은 다음과 같다.

1. Elasticsearch에 악성 파일과 정상 파일의 어셈블리 코드 데이터를 저장한다.
2. 제안된 신경망이 판정한 구문을 Elasticsearch로 검색하여 해당 구문을 포함하는 파일을 검색한다.

3. 검색된 파일 중 악성 파일의 비율이 한계값보다 크면 신경망이 적합한 판정을 수행했다고 판별한다.

즉, [그림 21]과 같은 흐름으로 신경망의 학습 여부를 판정한다.



[그림 21] 신경망의 학습 여부 판정 흐름도

한국인터넷진흥원의 정보보호 R&D 챌린지의 정상 파일과 악성 파일 각각 7,000개를 업로드한 뒤, stosd stosd stosd and mov 니모닉으로 파일을 검색한 결과가 [그림 22]와 같다. 검색 결과 16개의 파일이 검색된 것을 확인할 수 있다.

```
GET _search
{
  "query": {
    "match_phrase": {"ngram":"stosd stosd stosd and mov"}
  } ,
  "stored_fields": []
}
```

```
"took" : 96,
"timed_out" : false,
"_shards" : {
  "total" : 5,
  "successful" : 5,
  "skipped" : 0,
  "failed" : 0
},
"hits" : {
  "total" : {
    "value" : 16,
    "relation" : "eq"
  },
  "max_score" : 3.2914119,
  "hits" : [
    {
      "_index" : "version1",
      "_type" : "mal",
      "_id" : "e302fb9a6f9b53847b553d455bf21ca4",
      "_score" : 3.2914119
    },
    {
      "_index" : "version1",
      "_type" : "mal",
      "_id" : "40a86f3c160cbd2552ca71f99bbaf31d",
      "_score" : 2.612845
    },
    {
      "_index" : "version1",
      "_type" : "mal",
      "_id" : "144340c241b3e3eb7cfe9e47448a95b6",
      "_score" : 2.5041578
    },
    {
      "_index" : "version1",
      "_type" : "mal",
      "_id" : "eda9c3b268e810aefea888ab468c479f",
      "_score" : 2.073002
```

[그림 22] Elasticsearch 검색 예제

4 향후 추진계획

4.1 향후 계획의 세부 내용

4.1.1 딥러닝 기반 악성 행위 추출 도구

현재 신경망 구현을 위한 기술이 확보되었고, 학습 여부를 판정하기 위한 기술 또한 판정되었다. 향후에는 다양한 구조와 파라미터에 대한 신경망을 학습시켜 최적의 신경망 구조를 도출할 예정이다.

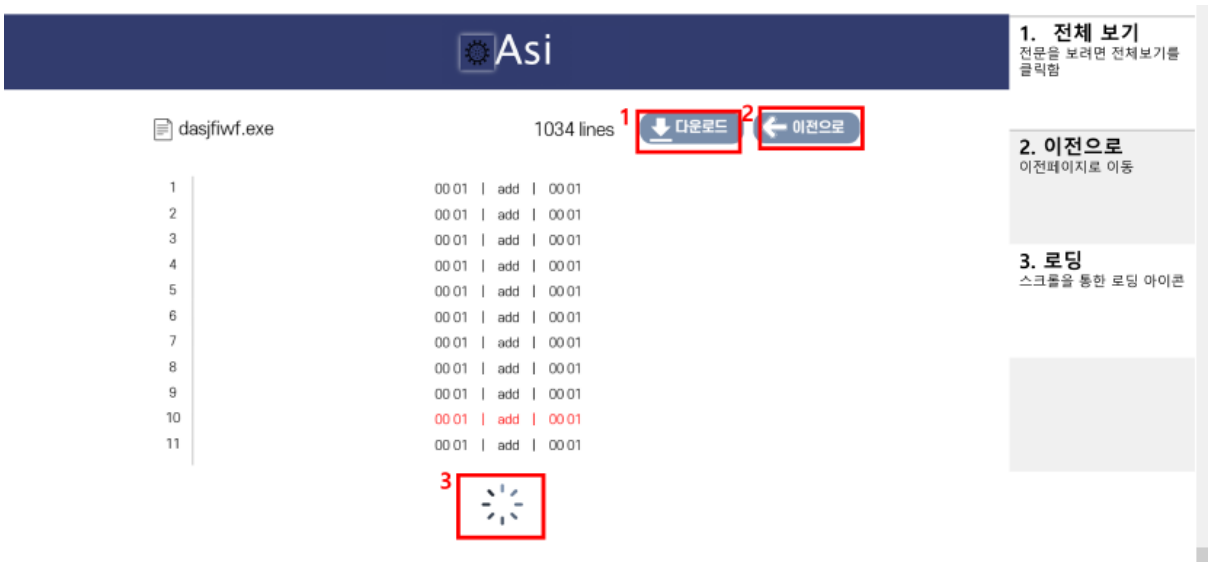
4.1.2 웹 서비스

프론트엔드는 와 같이 분석한 결과를 확인하고, 다운로드 하는 기능을 추가할 예정이다.

[그림 23] 분석 결과 목록 및 다운로드 페이지

웹에서 결과를 쉽게 확인할 수 있도록 악성 행위를 수행하는 부분을 강조하여 나타내고, 전문을 확인할 수 있는 기능을 추가할 예정이다.

[그림 24] 분석 결과 확인 페이지



[그림 25] 분석 결과 전문 확인 페이지

서버는 학습한 신경망으로부터 악성 행위를 수행하는 부분을 추출하고, 이를 이용자에게 전달하는 기능을 추가할 예정이다.

5 고충 및 건의사항

신경망을 빠르게 학습시키기 위해서는 고사양 PC를 필요로 합니다. 현재 학교에서 제공하는 GPU 서버는 사용량이 많아 사용하기 어려운 실정이라 Colab과 같은 공개 개발 환경에서 실험을 수행하고 있습니다. 공개 개발 환경은 시간 제한이 있고 사양이 낮아 다양한 실험을 진행하기 어렵습니다. GPU 서버를 증설해주시거나 사용하지 않는 자원을 적극적으로 회수해주신다면 실험에 큰 도움이 될 것으로 기대합니다.

6 부록

[표 5] Word2Vec(CBOW)의 각 니모닉별 가장 가까운 위치에 사상된 니모닉

Mnemomnic	vec 8	vec 16	vec 32	vec 64	vec 128
mov	jmp lea jz jnz jle jbe call push jge jl	jmp jle jnz xor jbe jz jge jl jg ldarg.1	jnz jmp jle jz jg xor jge jbe jl cmpunordps	jmp jle jnz jz jg xor jge lea jbe jl	jmp jz jnz jle jge lea jl xor jg push
jmp	mov jnz jl jz jbe jge jg jle lea call	mov jnz ret jz lea jl ldarg.1 box jle jge	jnz mov jz jg jle lea jge setnz setz call	mov lea jnz jz call setz jge jle jl jg	mov lea jnz jz call jge jl push jle jg
add	jbe ja sub jnb jg jle jge jz jnz jb	ldloc.2 imul sub adc stloc.3 ldloc.3 ldloc.0 ja stloc.1 sar	ldc.i4.1 imul ldc.i4.3 ldc.i4.2 jg ja stelem.i4 ldelem.i4 sub bge.s	imul ja jb jnb jg jno adc sub jbe jl	jb ja jg jnb imul sub jl jbe adc shl
pop	leave retn cgt.un pushf stosd setns	leave retn callvirt castclass brfalse.s isinst	retn leave ldsfld callvirt ldelem.ref xor	leave retn vfmsub231ss divpd vfmsub213sd vpminsd	retn leave mov xor leave.s jmp

	vzeroupper fsincos ldnull fclex	ldelem.ref blt.s stloc.2 brtrue.s	pushf stloc.s vzeroupper mov	vpackusdw movdq2q pextrb pmovzxwd	vcmpgtsd cmpsd vdivss call
push	lea call mov jmp test throw ldtoken vfmsub231sd jl jge	lea call jz jnz mov jle jmp stloc.0 ldarg jge	call lea jz jnz js jmp box jle dup ldc.i4.0	lea call jz jnz jmp jle mov js setnz test	lea call jz jnz jmp mov jle test js jl

[표 6] FastText(Skip-Gram)의 각 니모닉별 가장 가까운 위치에 사상된 니모닉

Mnemomnic	vec 8	vec 16	vec 32	vec 64	vec 128
mov	setz cmovnz cmovz setnz setns xor setnl sets setl test	lea push add vmovsd shlx cmp xor inc shl jno	lea xor push shlx add shl sar inc setnl cmovz	lea push xor add xchg inc setnz vmovsd cmp sar	lea push add xor inc cmp cmovz movzx movsd setz
jmp	jl jbe jb jnz jge jnb ja jle jg fsincos	jl jge jb jnb ja jbe jg jle retn jnz	retn jl jle jge jb jg jnb jbe jnz js	retn jle jg jl jge jns jnz lea js jb	retn jle jl jnz jge jg jns js jz jb
add	dec cmpneqsd cmovbe	sub prefetcht0 cmpltpd	sub prefetcht0 prefetcht1	sub shl mov	sub mov imul

	setnbe stmxcsr ucomisd btr cvtss2si maxpd cmp	prefetcht1 mov movups adc cmpnlepd movddup prefetchnta	shl prefetch sar unpckhpd imul mov prefetchnta	prefetcht1 prefetcht0 lea cmp imul prefetch pmuldq	lea cmp shl movaps xorps addpd mulpd
pop	leave popf pushf jecxz sti stc vroundsd vzeroupper cld loopne	retn retf leave vzeroupper cmpltsd popfw cmplesd pushf xor les	retn popfw cmpsw xor roundsd cmpsd popf vzeroupper vroundsd vcmpgtsd	retn leave push vroundsd popf cld setz roundsd vpbroadcastw popfw	retn push leave mov call lea cmplesd vzeroupper jmp setz
push	jns js lea fnclex call stosd test cmovs setns jz	lea mov setns fnclex haddpd fsincos cmovns cmovnz cmovle cmovz	lea mov test fnclex setns cmovs setnz fld1 xor setz	lea mov test setnz setz cmovnz stosd xor cmovns ficom	lea mov setnz setz test stosd cmovnz jz xor setns

[표 7] FastText(CBOW)의 각 니모닉별 가장 가까운 위치에 사상된 니모닉

Mnemomnic	vec 8	vec 16	vec 32	vec 64	vec 128
mov	jmp jg jl cmovnz jnz jge jb setz jbe ldloc.3	jmp jle jbe jnz jge push jz jnb jl lea	jle jnz jmp jge jz jl jg jbe jnb jb	jle jnz jz jmp jge jg xor jbe jnb jl	jmp jnz jle jz jge jl jg jnb push jbe

jmp	mov jg jl ldloc.3 cmovnz jnz jge jb ldloc.2 jbe	mov jl jb jnz jge jbe fcmovnbe lea fcmovnb jg	mov jge lea jl jnz jg jle jb pushfw ja	mov lea jge jnz movss setp push jg jle jnb	mov lea movss jge jnz jl push movaps test jz
add	ldind.ref ldelem ldelema stmxcsr movzx ldelem.r8 ldelem.r4 endfinally stloc starg.s	movddup movshdup movsldup inc imul movntq movd movnti movsx movntdq	jno imul adc psignw movnti cvttps2dq fcmovb cvtps2dq paddusw pavgw	jno shlx jg imul adc movnti fiadd cmovb fimul movntdq	imul jg adc fimul fiadd addsd jno jl sub ja
pop	leave newobj retn ldnull ldvirtftn rethrow ldsfld ret leave.s iret	retn leave popfw popcnt brtrue popf fsave lds callvirt brtrue.s	retn leave popfw popcnt popf popa castclass leave.s isinst pushf	retn leave popfw leave.s popf movhlps popcnt br.s movlhps ldsfld	retn leave popfw leave.s popf popa popcnt xor br.s ldsflda
push	lea call jl jmp mov ldelem.ref jle jge ldstr jnz	call lea ldtoken mov ldstr pushfw box calli jnz pusha	lea call pushfw calli jmp jnz jz pusha pmulhw mov	lea call pushfw pusha jnz jz jmp mov jle jge	lea call pushfw jmp jnz jz mov jl pusha test