T.T.T.S

Translated Text to Speech

Based on Text-to-Speech Deep Learning
Using Tensorflow and 'Naver papago' API

Team	4	**Major**	Software Engineering
Team member	Bang Sung-Ho	**ID**	201320891
	Lee Jeong-Eun		201420980
	Lee Chung-Hee		201420919

Contents

1. Overview

프로젝트 진행에 앞서 우선 Natural Language Processing을 Deep Learning을 통하여 구현한 오픈소스 프로젝트 중 하나를 선택하였다.

KSS 오픈소스 프로젝트는 한국어의 Text To Speech를 구현한 프로젝트이다. 기반 기술로는 Tensorflow API를 사용하여 Deep Learning을 구현하고 있으며 영어 알파벳과는 다른 한국어만의 특징을 반영하기 위한 특별한 자연어 가공 과정을 거치고 있다. 우리는 이 방법을 실습하여 익힘으로써 Natural Language Processing이 Deep Learning을 통하여 어떻게 이루어질 수 있는지 학습하는 것을 목표로 하였다. 또한 Deep Learning 기반 기술인 Tensorflow의 Flow를 학습하고 이를 바탕으로 기존의 Learning 방식을 개선하거나 그를 활용한 새로운 오픈소스 프로그램의 개발하였다.

우선 우리는 오픈소스 프로젝트의 매뉴얼을 바탕으로 Deep Learning을 통한 음성 인식 모델의 동작을 확인하였다. 그리고 그를 통하여 파악한 Deep Learning의 학습 시간과 난이도 등 여러 조건을 감안하여 프로젝트의 학습을 개선하는 것 대신 프로젝트를 새로운 프로젝트 개발을 목표로 하였다.

1.1. T.T.T.S Overview

영어는 만국공통의 언어이기 때문에 우리는 어디서든 쉽게 영어와 마주치게 된다. 영어를 번역해주는 프로그램은 시중에 이미 다수 존재하지만 우리는 번역을 해주는 동시에 번역 결과와 일치하는 음성의 파일을 생성해주는 프로젝트를 구상하였다.

T.T.T.S라는 이름은 ‘Translated Text to Speech의 약어이다. T.T.T.S는 영어로 입력된 텍스트를 한국어로 번역을 한 뒤에 음성파일로 생성을 해주는 서비스이다. 영어를 한국어로 번역하는 것은 Naver의 papago API를 사용한다.

1.2. Text-to-Speech

TTS는 Text-to-Speech의 약어로 사용자의 입력이 인식할 수 있는 문장의 형태로 주어졌을 때, 시스템에서 문장을 인식하고 그에 맞는 음성을 합성하여 제공하는 서비스를 의미한다. 이를 상용화하여 널리 사용되고 있는 서비스로는 최근 화두가 되고 있는 AI 비서에서부터 ARS의 자동응답과 시각장애인을 위한 지원 프로그램 등이 있다.

Text to Speech는 다음의 과정을 통하여 이루어진다. 우선 사람의 목소리 음파를 인식하여 일정한 음성 단위로 분리하고 각 음성 단위에 일치하는 음파를 저장하여 서비스 기반이 되는 음성 데이터를 습득한다. 이 후, 사용자로부터 특정 어휘가 주어졌을 때 어휘를 음성 단위로 분리한 후, 일치하는 음성 단위를 이전에 저장하였던 음성 데이터로부터 재조합하여 마지막으로 음성을 합성한다. 최근에는 Deep Learning을 기반으로 다수의 음성으로부터 어휘를 학습하고 학습된 모델을 기반으로 음성을 합성하는 프로젝트가 오픈소스는 물론 전문 연구기관과 산업체에 의하여 다수 진행되고 있다.

2. System Design

2.1. KSS Overview

'kss' 오픈소스 프로젝트보다 앞서 구현된 'dc_tss' 오픈소스 프로젝트는 2017년 10월 발표된 논문인 'Efficiently Trainable Text-to-Speech System Based on Deep Convolutional Networks with Guided Attention'에서 고안된 Deep Convolutional Network TTS를 기반으로 하고 있다. 'kss' 오픈소스 프로젝트 또한 이를 바탕으로 하고 있으며 기존의 영문 알파벳 학습 모델을 변형하여 한글 자모 인식과 그를 통한 음성 합성이 가능하다.

2.1.1. DCTTS Model

Neural Speech Synthesis Model의 연구가 처음 시작된 이래로 대부분이 학습에 RNN(Recurrent Neural Networks)을 사용했으나, DCTTS는 RNN을 사용하지 않고 Convolution으로 구성된다. DCTTS는 기존 학습 방식에 비하여 학습이 상당히 빠르며, 학습 완료후 음성 합성 또한 합리적인 시간 내에 이루어진다.
학습에 앞서 학습 데이터의 Quality가 중요하다. 이는 학습하는 과정에서 음성을 직접 재생하며 음성의 Fidelity에 따라 음성 단위의 인식률이 결정되며 인식한 음성 단위와 transcript의 비교하며 학습하는 것이 이유이다.

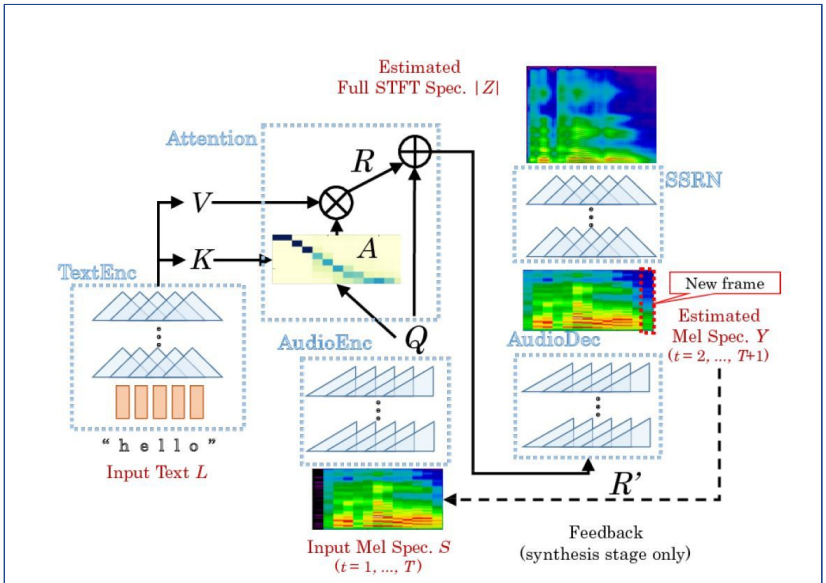


Fig. 1. Network architecture.

DCTTS는 텍스트를 멜 스펙트로그램(mel spectrogram)으로 변환하는 네트워크(Text2Mel)와 스펙트로그램 변환 네트워크(SSRN, Spectrogram Super-resolution Network)로 구성된다. 'KSS' 프로젝트는 음성 합성에 앞서 이 2개의 네트워크를 기준으로 모델 학습을 진행한다. 프로젝트에서 제시된 hyper-parameter는 400만 회의 학습을 권장하고 있으며 실제로 테스트한 결과 30만 회의 학습으로는 실제 음성과 유사한 음성을 합성해낼 수 없었다.

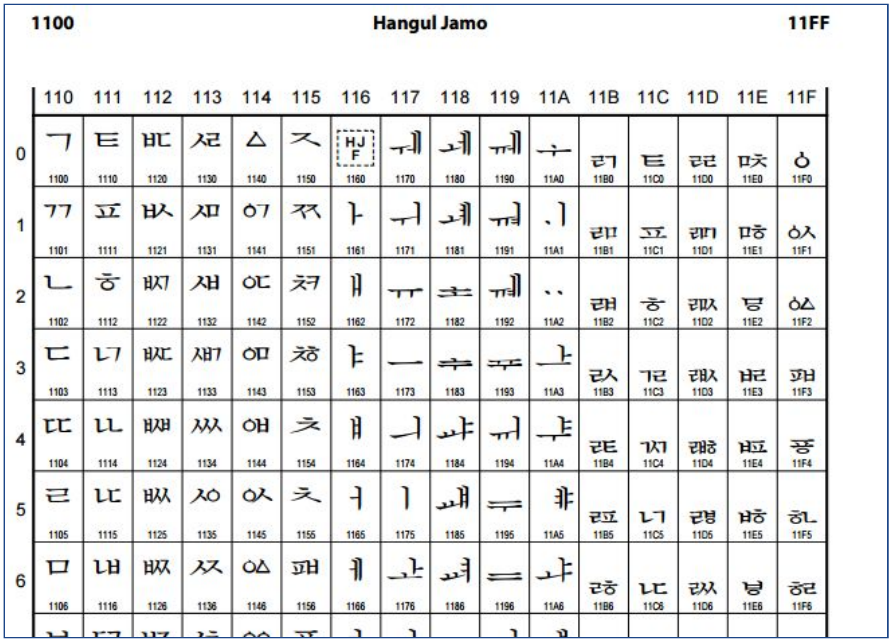
2.2. Korean Alphabetizer

2.2.1. Unicode with Korean Alphabet

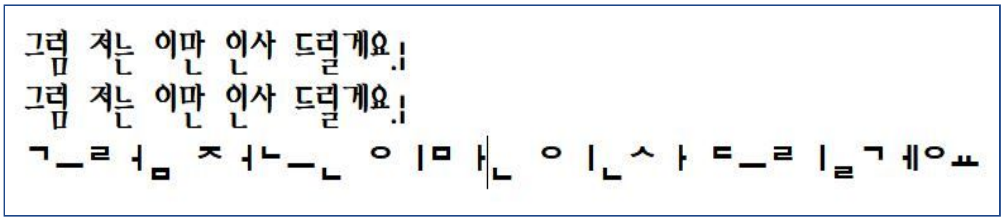
한글은 영어의 입력 방식과는 다르게 초성 중성 종성을 조합한 완성형 글자이면서 Semantic에 따라 음성 단위의 읽히는 소리가 달라지는 언어이다. 따라서 영어 알파벳을 학습하는 TTS와는 다른 전략이 요구된다.
이 프로젝트에 있어서 취하는 전략은 5가지 이다.

- ❏ 음운을 사용한 방식 (0x01100-0x011FF)
- ❏ 초성 중성 종성으로 나누어 판단하는 방식 (0x01100-0x011FF)
- ❏ 종성의 소리를 초성에 결합하여 판단하는 방식
- ❏ 쌍 자음을 무시하는 방식(0x01100-0x011FF)
- ❏ 받침이 없다고 판단하는 방식

입력값을 합성하기 위하여 유니 코드상에서 어떻게 표현되었는지 확인해야 한다. 한글의 유니코드는 다음과 같다.



한글 자모는 1100 부터 11FF 까지로 구성되어 있으며 이는 같은 ㄱ 에 관해서도 초성이냐 종성 이냐 에 따라 구분되는 유니코드 이다 5가지 전략을 취하기 위해서 입력해주는 텍스트는 다음 과 같은 형식을 취해야만 한다.



위의 그림처럼 자음을 초성과 종성을 구분해줘야 하는데, 키보드의 자판이 제공하는 자모는 유니코드 0x01100-0x011FF 의 범위에

존재하지 않는다. 기존 프로젝트에서는 음성 데이터와 일치하는 transcript를 작성하기 위해서 초성, 중성, 종성에 일치하는 유니코드의 값을 수작업으로 찾아서 옮겨적으며 작성하여야 했다. 이러한 불편함을 해소하고 보다 빠른 data-set의 생성을 위하여 입력받은 문장으로부터 초성, 중성, 종성을 분리하고 각각이 독립된 자모가 아닌 초성, 중성, 종성으로 가지는 유니코드의 값을 0x01100-0x011FF 영역으로 맵핑 해주는 코드를 작성하였다. 초성, 중성, 종성은 다음과 같은 계산을 통하여 완성된 글자를 만든다.

```
한글 완성형 character의 유니코드의 값
   = ((초성 * 21) + 중성) * 28 + 종성 + 0xAC00
```

위와 같이 한 글자가 만들어진다. 그리고 이를 역으로 연산하여 문장의 각 초성, 중성, 종성을 분석할 수 있다. 완성된 한글 문자 유니코드 X에 대하여 초성, 중성, 종성이 가지는 유니코드 값을 계산하는 방법은 다음 과 같다.

```
초성 = ( ( X - 0xAC00 ) / 28 ) / 21
중성 = ( ( X - 0xAC00 ) / 28 ) % 21
종성 = ( X - 0xAC00 ) % 28
```

하지만 결정한 값은 소리 글자의 코드 값이 아니기 때문에 이들이 각각 초성, 중성, 종성 중 어떤 위치의 문자인가를 나타내기 위해 다시 다음과 같은 계산이 필요하다.

```
초성문자코드 = 초성 +  0x1100 // ㄱ
중성문자코드 = 중성 +  0x1161 // ㅏ
종성문자코드 = 종성 +  0x11A8 -1
```

위의 코드에서 종성의 유니코드 값을 결정하는 '0x11A8 -1'는 종성이 없는 경우를 반영한 결과이다. 이를 (0x01100-0x011FF) 영역내의 글자로 맵핑한다.

```
sentense =[]
if __name__ == '__main__':
    test_keyword = "이러지도 못하는데"
    split_keyword_list = list(test_keyword)
    print(split_keyword_list)

    result = list()
    for keyword in split_keyword_list:
        # 한글 여부 check 후 분리
        if re.match('.*[ㄱ-ㅎㅏ-ㅣ가-힣]+.*', keyword) is not None:
            char_code = ord(keyword) - BASE_CODE
            char1 = int(char_code / CHOSUNG)
            result.append(CHOSUNG_LIST[char1])
            sentense.append(CHOSUNG_LIST[char1])
            char2 = int((char_code - (CHOSUNG * char1)) / JUNGSUNG)
            result.append(JUNGSUNG_LIST[char2])
            sentense.append(JUNGSUNG_LIST[char2])
            char3 = int((char_code - (CHOSUNG * char1) - (JUNGSUNG * char2)))
            if (char3!=0):
                result.append(JONGSUNG_LIST[char3])
                sentense.append(JONGSUNG_LIST[char3])

        else:
            result.append(keyword)
            sentense.append(keyword)
    # result
    print("".join(sentense))
```

```
['이', '러', '지', '도', ' ', '못', '하', '는', '데']
이러지도 못하는데
```

마지막으로 완성된 코드는 위의 그림과 같으며 실행 결과는 그림 하단의 출력과 같다. 출력이 나타나는 터미널이 가지는 특성에 따라 출력이 합성형으로 나오는 경우가 있으나 학습 data-set을 위한 텍스트 파일로 출력하는 경우 한글 자모의 초,중,종성 분리가 이루어진 형태 (0x01100-0x011FF)의 출력이 나타남을 확인 하였다.

2.3. Translated Text to Speech

```
if __name__ == '__main__':
    ko_txt = input('> Text to Speech : ')
    encText = urllib.parse.quote(ko_txt)
    data = "source=en&target=ko&text=" + encText
    request = urllib.request.Request(url)
    request.add_header("X-Naver-Client-Id", client_id)
    request.add_header("X-Naver-Client-Secret", client_secret)
    response = urllib.request.urlopen(request, data=data.encode("utf-8"))
    rescode = response.getcode()
    if (rescode == 200):
        response_body = response.read()
        res_text = response_body.decode('utf-8')
        res_obj = json.loads(res_text)
        translated = res_obj['message']['result']['translatedText']
        print(translated)
        ko_txt = translated
        ko_txt.replace("\n", "")
        ko_txt = ko_txt + "¦" + ko_txt + "¦" + ko_txt
        synthesize(ko_txt)
    else:
        print("error")
        print("Error Code:" + rescode)

    print("Done")
```

번역하고싶은 영문을 입력받고 네이버 파파고 API를 사용하여 한국어로 번역을 해주는 코드이다. “source=en&target=ko&text”부분에서 영어에서 한국어로 번역을 요청하는 뜻이다. 필요에 따라 번역할 언어, 번역될 언어의 변경이 가능하다. 또한, 네이버 파파고 API를 사용하기 위해서는 네이버 개발자 센터에 방문해서 인증키를 발급받아야 한다. 발급받은 인증키를 전역변수인 ‘client-id’, ’client_secret’로 정의하여 주면 된다.

u

2.3.1. Text Translation

영문 입력을 받아 한글로 번역한 결과를 papago API를 통하여 얻는다. 그리고 번역 결과를 바탕으로 한 음성을 합성한다.

실행 결과는 다음의 그림과 같다.

```
$ python synthesize.py
> Text to Speech : In this course, we learn how machine can learn natural language.
이 과정에서는 기계가 어떻게 자연의 언어를 배울 수 있는지 배운다.
Graph loaded
2018-12-21 11:46:27.141379: I C:\tf_jenkins\workspace\rel-win\M\windows-gpu\PY\36\t
se: AVX AVX2
2018-12-21 11:46:27.902038: I C:\tf_jenkins\workspace\rel-win\M\windows-gpu\PY\36\t
name: GeForce GTX 1060 major: 6 minor: 1 memoryClockRate(GHz): 1.6705
pciBusID: 0000:01:00.0
totalMemory: 6.00GiB freeMemory: 4.97GiB
2018-12-21 11:46:27.918166: I C:\tf_jenkins\workspace\rel-win\M\windows-gpu\PY\36\t
1060, pci bus id: 0000:01:00.0, compute capability: 6.1)
Text2Mel Restored!
SSRN Restored!
100%|
Working on file 1
Done
```

3. Conclusion

3.1. Roles of Each Members

- **Bang SungHo**
  KSS 오픈소스 프로젝트 파악 및 모델 학습, 응용 총괄
- **Lee JeongEun**
  Naver papago API의 도입, 번역 텍스트의 음성 합성 개발
- **Lee ChungHee**
  한글 자모 분리 프로그램 개발

3.2. Current State of System

KSS 오픈소스 프로젝트의 이미 학습된 모델을 기반으로 프로젝트를 진행하였다. 미리 학습되어있는 모델은 한국어 텍스트 파일을 음성 합성을 통해서 한국어 텍스트에 맞는 음성 파일을 생성해준다.

우리는 여기에 영-한 번역 기능을 추가하였다. Naver의 papago 번역 API를 사용하여 영어 문장이 입력되면 한국어로 번역된 값을 받아온 후 그에 맞는 음성을 합성하여 파일을 생성한다.

또한 한국어 음성 학습 data-set을 생성할 때, 번거로울 수 있는 natural language processing 의 한글 자모 분리과정을 프로그램으로 처리함으로써 확실하고 정확한 transcript의 생성을 할 수 있도록 하였다.

3.3. Additional Work

현재는 입력을 command line을 통하여 받을 수 밖에 없다. 이것을 사용자에게 편리한 인터페이스에 이식하는 것으로 프로젝트를 개선할 수 있을 것이다. 예를 들어 영문 텍스트 파일이나 웹페이지의 url 등을 입력받으면 자동으로 영어 텍스트를 인식하고 학습시켜 영어 텍스트 파일을 입력받아 결과물을 출력해낸다.

3.4. What We've Learned

프로젝트에 앞서 제시된 과제는 기존 학습 모델의 hyper parameter와 가중치 연산 방법을 개선하여 학습 향상을 이뤄내는 것 혹은 프로젝트를 활용한 새로운 프로젝트의 개발이었다. 전자를 달성하는 것이 Deep Learning을 학습하는 측면에서 더 이로울 수 있으나 앞서 서술된 바와 같이 TTS는 납득할 수 있는 결과물을 얻어내기 위해서는 그 학습시간이 상당히 증가하며 우리에게 주어진 시간 내에는 그런 학습과정을 변형하며 여러번 실시하기에는 부족하였다.

Tensorflow API의 동작을 위한 환경설정에도 많은 시간이 소모되었다. 또한 단순히 cpu 연산만으로 음성 학습이라는 무거운 작업을 빠른 시간 내에 수행하기에 무리가 있었다.만약 다중 gpu 환경이 주어졌다면 이러한 제약을 해결할 수도 있었을 것으로 보인다.

이 프로젝트를 통하여 Natural Language를 컴퓨터에서 인식하기 위한 전처리 과정에서 필요한 기술이 어떤 것인지 학습할 수 있었다. 예를 들면 ‘kss’ 프로젝트에서는 음성 data와 일치하는 transcript를 학습 data-set으로 제공해야 한다. Transcript는 한글이라는 언어의 특성과 컴퓨터가 인식하기 위한 형태인 unicode로 한글을 나타내야 한다는 제약 들을 이유로 영문의 학습과는 다른 특별한 처리를 요구하게 된다.

Deep Learning 기반의 오픈 소스를 활용한 프로젝트를 설계하고 개발하는 방법을 익혔다.

4. References

1. Hideyuki Tachibana, Katsuya Uenoyama, Shunsuke Aihara. “Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention.” *ICASSP* (2018)