
집교2 최종 보고서

Neural Japanese Transliteration



Team	4조	Major	Software Engineering
Team member	권태성	ID	201321019
	정다원		201620923
	신희수		201621048

목차

1 프로젝트 소개	3
A. 개요 및 요구사항	3
B. 사용 데이터 및 모델	4
2 RNN	5
A. 개념	5
B. BPTT	6
C. RNN의 문제점	7
3 Beam Search	8
A. 개념	8
B. Greedy	8
C. Top-down, Bottom-up Beam Search	10
D. Beam Size	11
4 결과	12
5. 참고문헌	14

1 프로젝트 소개

A. 개요 및 요구사항

현재의 디지털 환경에서 사람들은 Roman alphabet을 통해 일본어를 입력한다. 따라서 번역 엔진이 얼마나 사용자가 원하는 단어를 잘 예측하는가가 중요하다. 이에 이번 프로젝트에서는 뉴럴 네트워크가 알파벳을 일본어 스크립트로 얼마나 잘 변환할 수 있는가를 살펴본다. 정확도에 대한 평가는 이미 존재하는 여러 언어를 지원하는 SwiftKey™ keyboard의 퍼포먼스와 비교하면서 진행한다. 최신 SwiftKey 키보드 애플리케이션과 같은 문장에 대한 테스트는 다음 그림과 같다. 이 때 CER은 edit distance를 총 character의 개수로 나눈 error rate를 의미하는데 SwiftKey 키보드의 CER은 0.136이다.

swiftkey					
NUM	ROMAJI	EXPECTED	SwiftKey 6.4.8.57 (S)	# characters	edit distance
1	zuttosakinokotodakedone	ずっと先のことだけだね	ずっと先のことだけだね	11	0
2	onakahetteshinisoudayo	おなか減って死にそうだよ	お腹減って死にそうだよ	12	2
3	mizunosenseigaosshattayouni	水野先生がおっしゃったように	水の先生が仰ったように	14	5
4	chiryohougamitsukarukamoshirenai	治療法が見つかるかもしれない	治療法が見つかるかもしれない	14	0
5	okaasankawatteagerarenakutegomen'ne	お母さん代わってあげられなくてごめんね	お母さん?わってあげられなくてごめんね	19	1
6	omaegason'nakotododousundayo	お前がそんなことでどうすんだよ	お前がそんな事でどうすんだよ	15	2
7	aitsuhason'nayawajaneeyo	あいつはそんなヤワじゃねえよ	あいつはそんなやわじゃねえよ	14	2
8	aitsugaressouninattara	あいつが折れそうになったら	アイツが折れそうになったら	13	3
9	hirokimokokoshouyutsuiteru	ひろきもここようゆついでる	ひろきもここ?ゆついでる	13	7
10	watashinosekaihamarudekawatteshimatta	わたしの世界はまるで?ってしまった	私の世界はまるで?ってしまった	18	3
11	ushirokaratouan'youshiatsumete	後ろから営業用紙集めて	後ろから営業用紙集めて	11	0
12	kotoshiasondokanaitemouasobenaiyo	今年遊んどかないともう遊べないよ	今年遊んどかないともう遊べないよ	16	0
13	akuatopianinoritai	アクアトピアに?りたい	アクアとびあに?りたい	11	3
14	on'natteyokushaben'naatoomotte	女ってよくしゃべんなあとって	女ってよくしゃべんなあとって	15	0
15	kamenoessayantokakingyonesayaritoka	カメの餌やりとか金魚の餌やりとか	?の餌やりとか金魚のえさやりとか	16	4
16	son'nanodarekanitanomebaianai	そんな誰かに?めばいいじゃない	そんな誰かに?めばいいじゃない	16	0
17	kusurinkoukayarihabirinohouhouwo	?の?果やりハビリの方法を	?の?果やりハビリの方法を	13	0
18	kakuninshitaotomottomemasu	確認したいと思ってます	確認したいと思ってます	11	0
885	keisatsukanbunobougaigaattanokamo	警察幹部の妨害があったのかも	警察官分ぼうがいがあったのかも	14	6
886	sonoteidonousoshikatsukeneeyouja	その程度の?しかつけねえようじゃ	その程度の?しか付け難えようじゃ	16	2
887	watashinegashoujikidakarautosotsukenaino	わたしが正直だから?つけないの	私ねが正直だから?つけないの	16	6
888	taihonotokinittappurito	逮捕のときにたつぷりと	逮捕の時にたつぷりと	11	2
889	hatasehenkattakedona	果たせへんかったけどな	果たせへんかったけどな	11	0
890	sonosukinijoujirutsumoritya	そのすきに?じるつもりや	その際に常時るつもりや	12	4
891	koregashikakeraretakudan	これが仕掛けられたぞ?	これがしかけられたぞ?	11	2
892	anatashikawakarutogainaitoomoimasu	あなたしか分かる人がいないと思います	あなたしかわかる人がいないと思います	18	1
893	jikenkaiketsunotamenaratamawosageru	事件解決のためなら頭を下げる	事件解決のためなら頭をさげる	14	1
894	soregawatashinopuraido	それがわたしのプライド	それが私のプライド	11	3
895	konomaemoamaenihadamasareta	この前もお前にはだまされた	この前もお前には騙された	13	2
896	ochademononinagarayukkuriyariimashou	お茶でも?あながらゆっくりやりましょう。	お茶でも?あながらゆっくりやりましょう。	19	0
Total CER:	1640/12057=0.13				

プロジェクト 進捗に 있어 요구사항은 다음과 같다.

- NumPy >= 1.11.1
- TensorFlow == 1.2
- regex (Enables us to use convenient regular expression posix)
- janome (for morph analysis)
- romkan (for converting kana to romaji)

B. 사용 데이터 및 모델

기존 프로젝트는 텍스트를 음성으로 변환할 때 사용되는 Tacotron을 차용하였다. Tacotron은 CNN과 RNN의 조합을 기반으로 작동한다. 즉, 입력 문자의 시퀀스로부터 mel spectrogram frames의 시퀀스를 예측하는데 주의를 기울이는 반복적인 Sequence to sequence의 feature prediction 네트워크이며 이를 통해 예측된 mel spectrogram frames에 컨디셔닝 된 시간 영역 파형 샘플을 생성하는 WaveNet을 수정하였다. 즉 Tacotron은 텍스트를 사운드 주파수의 스펙트럼을 시각적으로 나타내는 스펙트로 그램으로 변환하고, 이의 요소를 해당 주파수로 변환하는 두 개의 심층 신경 네트워크를 중첩하는 원리로 작동한다.¹ 본 프로젝트는 Tacotron 모델을 차용하여 이를 알파벳을 일본어로 변환하는 데 사용하였다.

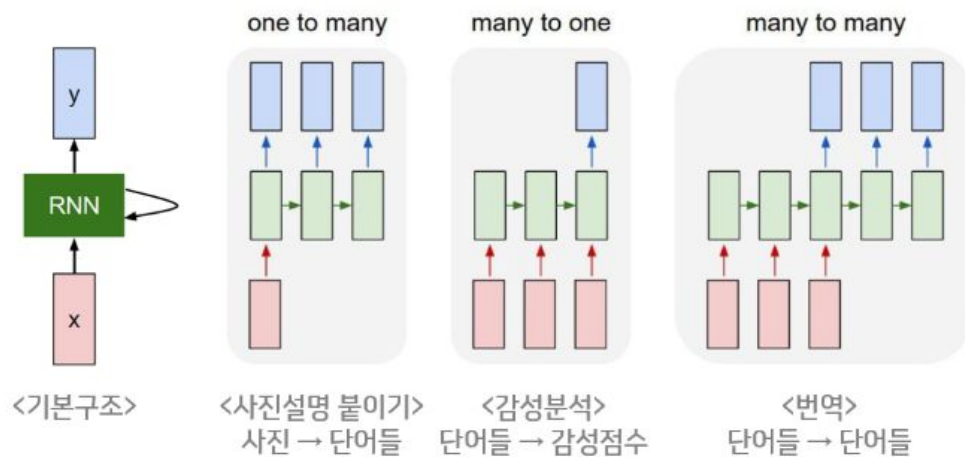
트레이닝을 위해 사용한 데이터는 Leipzig Japanese Corpus에서 jpn_news_2005-2008_1M-sentences를 다운받아 사용하였다. 이는 2005년에서 2008년까지의 일본어 뉴스에서 추출한 896 문장이다.

¹ <https://www.analyticsindiamag.com/tacotron-2-google-ai-text-to-speech-system>

2 RNN

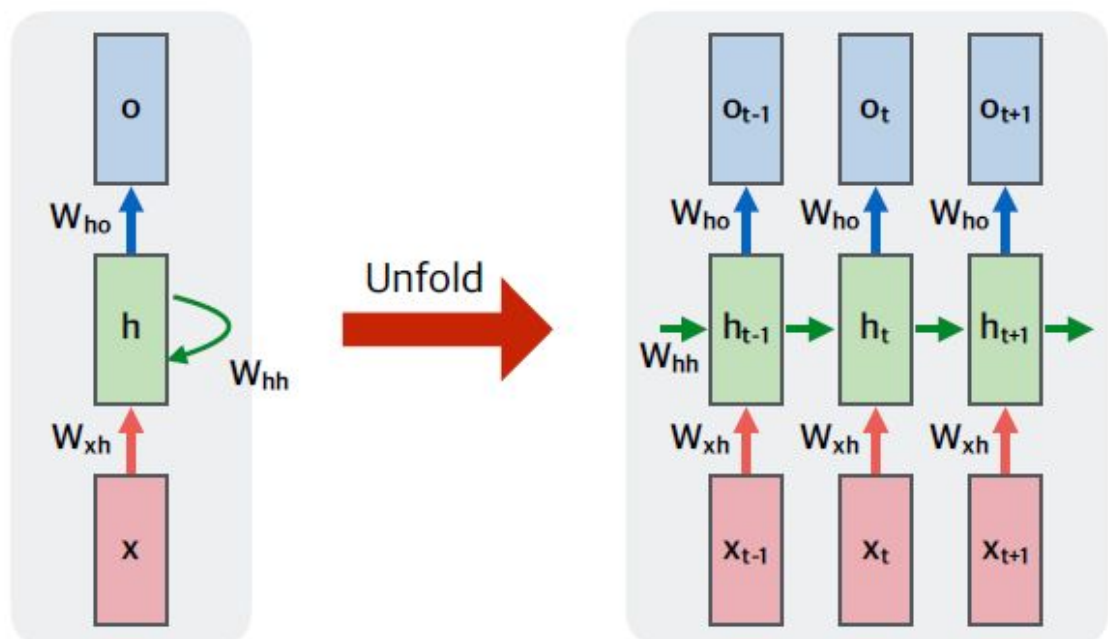
A. 개념

RNN(Recurrent Neural Network)은 순차적으로 정보를 처리하는 인공지능의 한 종류이다.



<그림 2-1>

RNN은 히든 노드가 방향을 가진 엣지로 연결돼 순환구조(directed cycle)를 이룬다(그림 2-1). RNN은 음성, 문자 등 순차적으로 등장하는 데이터 처리에 적합한 모델로 알려져 있으며, CNN과 더불어 최근 들어 각광 받고 있는 알고리즘이다.



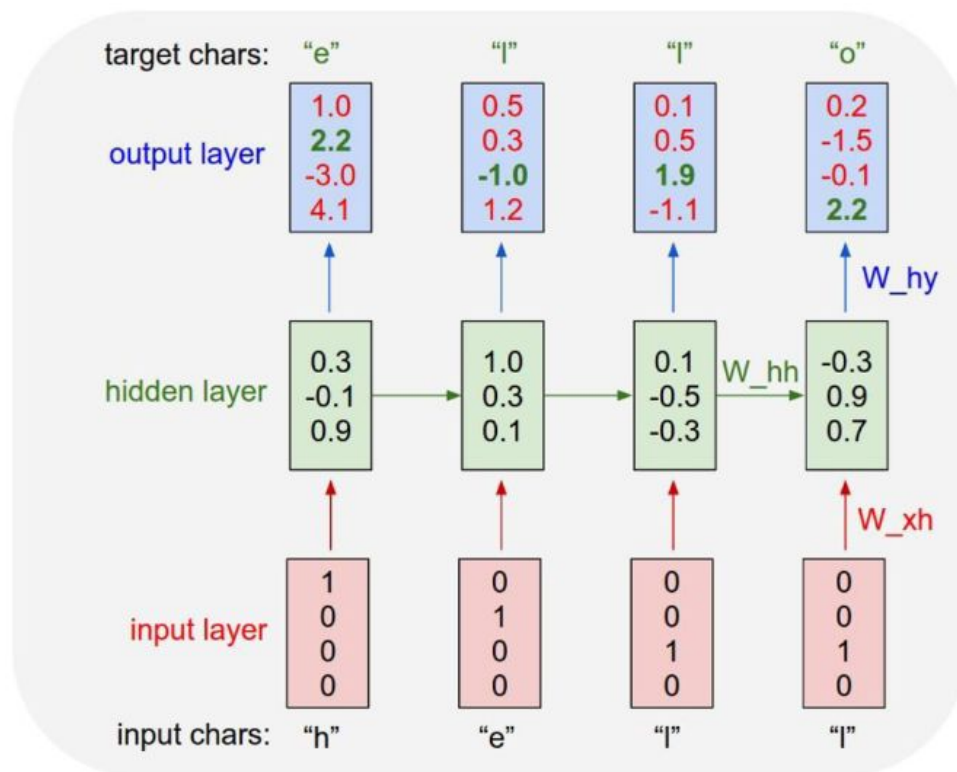
<그림 2-2>

위의 그림에서 x_t 는 시간 스텝 t 에서의 입력값을 의미한다. h_t 는 시간 스텝 t 에서의 hidden state 값을 의미한다. 현재 상태 h_t 는 이전 스텝의 상태 h_{t-1} 과 현재 스텝의 입력값 x_t 에 의해 계산된다.

o_t 는 시간 스텝 t 에서의 출력값을 의미한다. RNN은 상태 h_t 를 네트워크의 메모리라고 생각할 수 있는데, 이는 RNN의 구조에서 하나의 시간 스텝은 이전 시간 스텝의 영향을 받으며, 이에 대한 정보를 담고 있기 때문이다. 또한, 위의 그림에서 유추할 수 있듯이, 출력값 o_t 는 현재 상태 h_t 에만 의존하고 있다. 가장 큰 RNN의 특징 중 하나가 t 스텝의 출력 값이 $t+1$ 스텝의 입력값이 된다는 점 이다. 또한 W_{hh} , W_{xh} , W_{ho} 와 같은 파라미터들이 일반적인 뉴럴 네트워크와 달리 모든 스텝에서 공유되고 있다는 특징을 갖는다. 이는 학습해야 할 파라미터를 대폭 낮춰준다. 만약 파라미터가 공유되지 않는다면 시퀀스 길이가 긴 경우에 학습해야 할 파라미터의 개수가 대폭적으로 증가하는 문제가 생기기 때문에, 모든 스텝에서 파라미터가 공유된다는 것은 RNN의 가장 큰 특징 중 하나이다.

B. BPTT

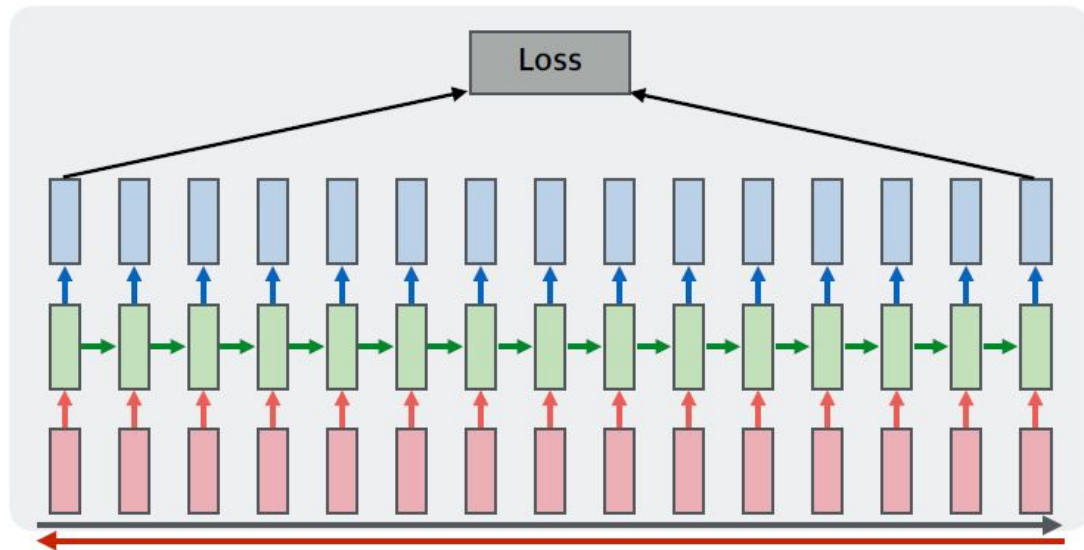
RNN은 backpropagation을 확장한 BPTT(Backpropagation Through Time)을 사용해서 학습한다. RNN은 문장과 같은 시퀀스를 입력으로 받기 때문에 학습 시 backpropagation을 시간에 대해서도 수행하는 것이다.



<그림 2-3>

BPTT를 설명하기에 앞서, forward propagation(순전파)와 backpropagation(역전파)에 대해 간단하게 언급할 필요가 있다. 예를 들어, <그림 2-3>은, RNN 모델에 'hell'을 넣으면 'o'를 반환하게 해 결과적으로 'hello'를 출력하게 하는 경우이다. 위에서 가진 학습데이터의 글자는 'h','e','l','o' 4개 뿐이고, 이를 one-hot-vector로 바꾸면 <그림 2-3>의 input layer와 같이 바뀐다. input layer을 기반으로 hidden layer를 생성하고

이를 바탕으로 output layer을 생성한다. 마찬가지로 두번째, 세번째, 네번째 단계들도 모두 갱신할 수 있고 이 과정을 forward propagation이라고 부른다. 모델에 정답을 알려주어야 모델이 parameter을 적절하게 갱신해 나갈 수 있다. <그림 2-3>의 output layer에 진한 녹색으로 표시된 숫자들은 정답에 해당하는 인덱스를 가리킨다. 이 정보를 바탕으로 parameter 값들을 갱신해 나가는 것을 역전파(backpropagation)라고 한다.



<그림 2-4>

<그림 2-4>에서 보는 것 처럼 모든 시퀀스에 대해 forward propagation을 진행하고 loss를 계산 한 뒤에 backpropagation시 모든 시퀀스에 대해 그라디언트를 계산한다. 이러한 경우 시퀀스가 길면 계산 속도가 매우 느려지는 문제가 발생한다. 따라서 Forward와 backward를 전체 시퀀스에 대해 수행하지 않고 부분(chunk) 시퀀스에 대해서만 수행하는 Truncated BPTT방법이 있다.


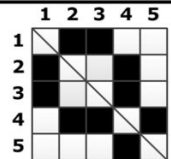
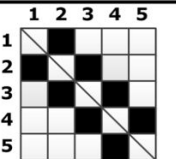
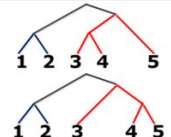
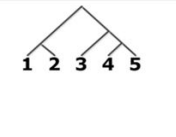
C. RNN의 문제점

RNN은 연관된 단어들 사이에 여러 스텝이 지난 경우 이를 까먹는 문제가 발생한다. 또한 현재 단어가 예전에 등장한 단어와 연관있는 경우에도 문제가 발생하는데, 각 스텝마다 그라디언트를 곱하면서 vanishing이 일어나기 때문이다. 요약하자면, RNN의 경우 긴 시퀀스를 잘 처리하지 못한다는 문제점이 존재하고 이를 Vanishing gradient라고 한다. 해결책으로는 활성화 함수를 tanh에서 ReLU로 바꾸거나, LSTM, GRU와 같은 RNN의 변형 모델을 사용하는 방법이 존재한다.

3 Beam Search

A. 개념

RNN은 응집성이 높은 입력값을 트리 형태로 결합해 가면서 입력값의 구조를 추상화하는 기법이다.² 따라서 RNN에서 트리 구성은 중요한데, RNN에서 트리를 구성하는 방법은 그림과 같다.

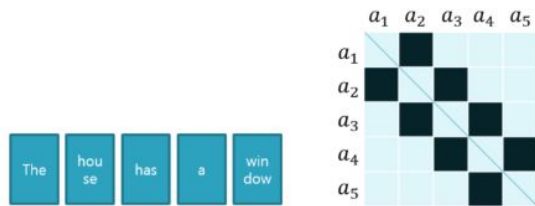
	Image	Text
Input Instance		The house has 1 2 3 a window 4 5
Adjacency Matrix		
Set of Correct Tree Structures		

우선 입력값의 이웃끼리 결합해 트리를 만든다. 그림을 예로 들면 1번 영역의 이웃은 2번과 3번입니다. 이를 인접행렬(Adjacency Matrix)로 나타내면 (1,2), (2,1), (1,3), (3,1) 위치의 요소값이 1이 된다. 이렇게 이웃이 될 수 있는 가능한 모든 경우의 수를 고려해 행렬로 나타낸 것이 좌측 두번째 그림이 됩니다. 이미지에 비해 텍스트는 한 단어의 이웃이 좌, 우 두 개 뿐이라 인접행렬이 보다 간단하다. 이웃이 될 수 있는 모든 경우의 수에서 그 가짓수를 하나씩 제거해 정답 트리 구조(Correct Tree Structure)로 나아가는 것이 트리 탐색의 핵심이다. 트리 탐색의 방법으로는 Greedy Search와 이를 개선한 Beam Search 기법이 있다.

B. Greedy

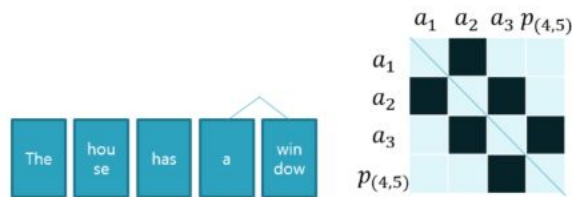
Greedy Search의 방법은 다음과 같다. 주어진 문장은 The house has a window 이며, 이를 단어 별로 나누어 {The, house, has, a, window}를 {a1, a2, a3, a4, a5}로 간주한다. 이를 기반으로 인접행렬을 구하면 다음 그림과 같이 된다. 그림에서 C는 이웃들의 쌍으로 이루어진 집합이다.

² <https://ratsgo.github.io/deep%20learning/2017/06/26/beamsearch/> Socher et al.(2011)



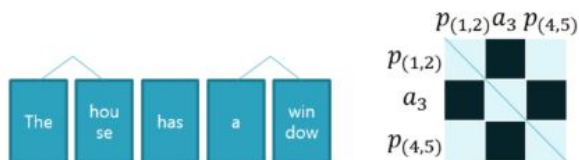
$$C = \{[a_1, a_2], [a_2, a_3], [a_3, a_4], [a_4, a_5], [a_2, a_1], [a_3, a_2], [a_4, a_3], [a_5, a_4]\}$$

이 때 RNN 모델의 score가 [a4, a5]가 가장 높다고 가정하면 이 둘을 이어서 트리를 만드는 것이 첫번째 과정이다. 이에 맞추어 인접행렬과 C를 갱신하는데 그 결과는 다음 그림과 같다.



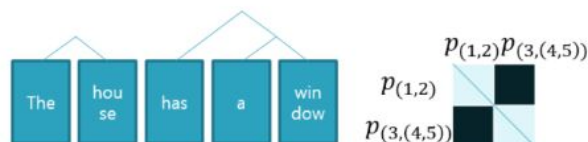
$$C = \{[a_1, a_2], [a_2, a_3], [a_3, p_{(4,5)}], [a_2, a_1], [a_3, a_2], [p_{(4,5)}, a_3]\}$$

다음 단계에서 RNN이 출력한 score가 [a1, a2]가 가장 높다고 가정한다. 이 때 트리, 인접행렬, C의 구성은 다음 그림과 같이 갱신된다.



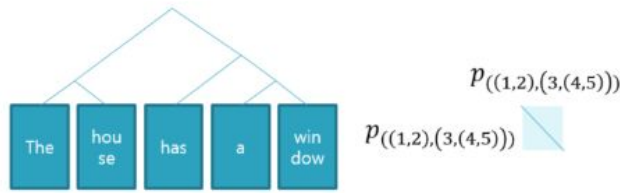
$$C = \{[p_{(1,2)}, a_3], [a_3, p_{(4,5)}], [a_3, p_{(1,2)}], [p_{(4,5)}, a_3]\}$$

그 다음 단계에서 RNN이 출력한 score가 has라는 단어와 'a window'라는 구가 가장 높다고 가정하면 이 때 트리, 인접행렬, C의 구성은 다음과 같이 갱신된다.



$$C = \{[p_{(1,2)}, p_{3,(4,5)}], [p_{3,(4,5)}, p_{(1,2)}]\}$$

최종적으로 연결해야 하는 노드는 두 가지이고, 최종 정답 트리 구조는 다음 그림과 같다.

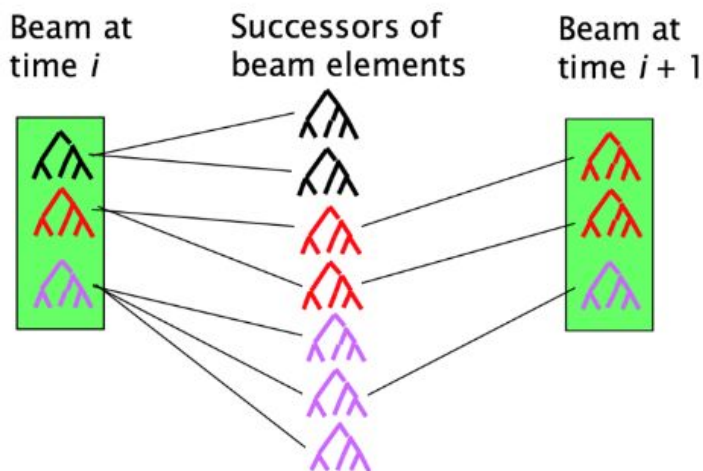


$$C = \{p_{((1,2),(3,(4,5)))}\}$$

C. Top-down, Bottom-up Beam Search

앞서 언급했듯이 문장에서 한 단어의 이웃은 좌, 우 2개 뿐이므로 less-greedy search 알고리즘인 bottom-up beam search가 사용될 수 있다.³ 이 때 Beam Search란 최고 우선 탐색(Best-First Search)를 기반으로 하되, 기억해야 하는 노드의 수를 제한해 효율성을 높인 방식이다.

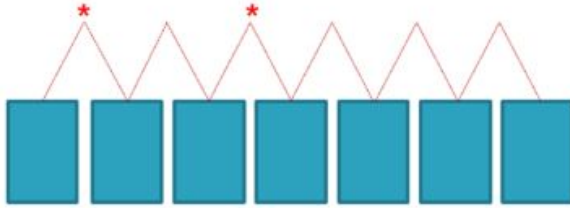
Beam Search는 Top-down Beam Search와 Bottom-up Beam Search로 나눌 수 있는데, 먼저 Top-down Beam Search는 상위 노드에서 하위 노드로 분기해 나가는 Beam Search 방식이다.



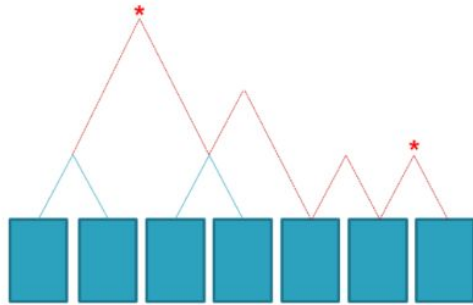
위 그림에서, 사용자가 기억해야 하는 노드 수(Beam size)를 3으로 정했다고 가정한다. 그러면 i 번째 step에서 다음 step에 선택될 수 있는 가능한 모든 경우의 수를 계산해본 뒤 Beam Search의 기본이 되는 알고리즘(예컨대 RNN)이 내놓는 최상위 3개 결과만 취해서 $i+1$ 번째 step의 결과물로 반환한다.

이와 같은 Top-down Beam Search 보다는 자연어 처리 분야에서는 Bottom-Up Beam Search가 더욱 자주 쓰인다. 이는 하위 노드에서 상위 노드로 트리를 결합해 가는 방식을 말한다. 임의의 7개 단어가 있고 이로부터 파싱 트리를 구축해야 한다고 가정해보자. 이 때 사용자가 정한 Beam Size는 2라고 한다. 초기 상태는 아래 그림과 같다. 각 단어들은 좌, 우의 두 이웃뿐이기 때문에 다음 step에서 선택될 수 있는 가능한 모든 경우의 수는 빨간색 실선과 같다. 이 때 RNN이 예측한 가장 응집성이 높은 두 가지가 첫 번째-세번째 단어, 세번째-네번째 단어를 결합하는 것이라고 가정하자며 이는 *로 표시한다.

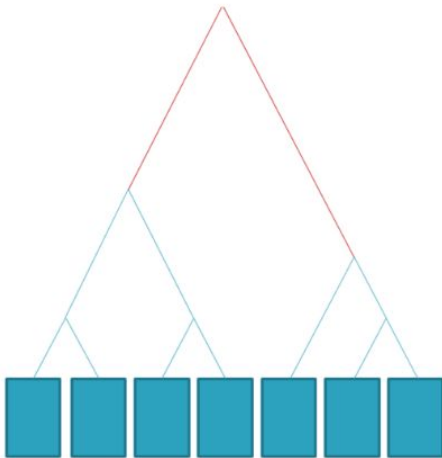
³ Socher et al.(2011)



이제 두번째 과정에서 이 두 가지를 잇고, 마찬가지로 다음 step에서 선택될 수 있는 가능한 모든 경우의 수를 계산한 뒤 최적의 두 개의 결과를 선택한다. 그 결과는 다음 그림과 같다.



이와 같은 작업을 반복하다 보면 한 단어가 두 개의 트리에 모두 속해야 하는 경우가 발생할 수 있는데, 이 때는 score가 더 높은 곳에 할당되게 된다. 최종적으로 얻는 정답 트리 구조는 다음 그림과 같다.



D. Beam Size

Beam Size는 보편적으로 3~10의 값을 가장 많이 사용한다. Beam Size가 클수록 알고리즘이 고려할 가능성이 많아지기 때문에 보편적으로는 더 나은 문장을 예측할 확률이 높아진다. 그러나 이에 수반되는 Cost가 크기 때문에 Beam Size가 크면 더 나은 결과를 얻을 수 있지만 구현 과정이 느려진다. 반면에 Beam Size가 작으면 정확도는 높지 않지만 빠르게 실행된다.⁴

4 결과

ln_model_epoch_01_gs_6146_beam_width_1

NUM	EXPECTED	model_epoch_01_gs_6146_beam_width_1	# characters	edit distance
1	ずっと先のことだけどね	ずっと開機のことだけどね	11	2
2	おなか減って死にそうだよ	同かってってにそうだよ	12	5
⋮				
896	お茶でも飲みながらゆっくりやりましょう	今でものみじがらくくくりやりましょう	19	6
Total CER:	4681/12057=0.39			

한번 학습된 데이터를 바탕으로 수행한 결과를 평가할 때, beam 길이를 1로 주었을 때 오차율은 약 0.388이 나오는 것을 볼 수 있었다. 이와 비교하여 같은 데이터를 평가함에 있어, beam 길이를 5로 늘려줬을 때 결과를 확인해봤는데,

ln_model_epoch_01_gs_6146_beam_width_5

NUM	EXPECTED	model_epoch_01_gs_6146_beam_width_5	# characters	edit distance
1	ずっと先のことだけどね	ずっと先のことだけどね	11	0
2	おなか減って死にそうだよ	同かってってにそうだよ	12	5
⋮				
895	この前もお前にはだまされた	この前も今前にはだまされた	13	1
896	お茶でも飲みながらゆっくりやりましょう	男でものみながライクリリやりましょう	19	9
Total CER:	4311/12057=0.36			

다음과 같이, 오차율이 약 0.357로, beam 길이가 1일 때 보다 오차율이 더 줄어들었다. 이를 통해 beam의 길이에 따라 오차율이 더 좋고, 나빠질 수 있음을 확인 할 수 있었다.

ln_model_epoch_13_gs_79898_beam_width_1

NUM	EXPECTED	model_epoch_13_gs_79898_beam_width_1	# characters	edit distance
1	ずっと先のことだけどね	ずっと先のことだけどね	11	0
2	おなか減って死にそうだよ	お腹へって死にそうだよ	12	3
⋮				
896	お茶でも飲みながらゆっくりやりましょう	お茶でも飲みながらゆっくりやりましょう	19	0
Total CER:	1595/12057=0.13			

ln_model_epoch_13_gs_79898_beam_width_2

NUM	EXPECTED	model_epoch_13_gs_79898_beam_width_2	# characters	edit distance
1	ずっと先のことだけどね	ずっと先のことだけどね	11	0
2	おなか減って死にそうだよ	お腹へって死にそうだよ	12	3
⋮				
896	お茶でも飲みながらゆっくりやりましょう	お茶でも飲みながらゆっくりやりましょう	19	0
Total CER:	1518/12057=0.13			

In_model_epoch_13_gs_79898_beam_width_3

NUM	EXPECTED	model_epoch_13_gs_79898_beam_width_3	# characters	edit distance
1	ずっと先のことだけどね	ずっと先のことだけどね	11	0
2	おなか減って死にそうだよ	お腹へって死にそうだよ	12	3
⋮				
896	お茶でも飲みながらゆっくりやりましょう	お茶でも飲みながらゆっくりやりましょう	19	0
Total CER:	1525/12057=0.13			

In_model_epoch_13_gs_79898_beam_width_5

NUM	EXPECTED	model_epoch_02_gs_79898_beam_width_5	# characters	edit distance
1	ずっと先のことだけどね	ずっと先のことだけどね	11	0
2	おなか減って死にそうだよ	お腹へって死にそうだよ	12	3
3	水曜日生がおへ！あーちとろけ	水の生がおへ！あーちとろけ	14	4
⋮				
896	お茶でも飲みながらゆっくりやりましょう	お茶でも飲みながらゆっくりやりましょう	19	0
Total CER:	1517/12057=0.13			

In_model_epoch_13_gs_79898_beam_width_8

NUM	EXPECTED	model_epoch_02_gs_79898_beam_width_5	# characters	edit distance
1	ずっと先のことだけどね	ずっと先のことだけどね	11	0
2	おなか減って死にそうだよ	お腹へって死にそうだよ	12	3
3	水曜日生がおへ！あーちとろけ	水の生がおへ！あーちとろけ	14	4
⋮				
896	お茶でも飲みながらゆっくりやりましょう	お茶でも飲みながらゆっくりやりましょう	19	0
Total CER:	1517/12057=0.13			

그 다음은 13번 학습된 데이터를 여러가지 빔 길이를 주면서 비교를 해보았다. 학습이 여러번 된 데이터들은 평균적으로 한번 학습된 데이터보다 훨씬 더 나은 오차율을 보이고 있다. 이를 통해 학습이 여러 번 될 수록 오차율이 줄어듦을 알 수 있었다. 또 beam 길이별 결과를 보았을 때 위의 그림들에서 보이는 오차율은 약 0.13으로 비슷하지만 자세히 살펴보면 beam 길이별로 edit distance에 차이가 있음을 알 수 있다. 우리는 beam 길이가 2일 때 가장 좋은 결과를 낼 것이라고 생각했지만 빔 길이가 5일때와 8일 때가 조금 더 좋은 결과를 보이는 것을 확인할 수 있었다.

5. 참고문헌

<https://www.analyticsindiamag.com/tacotron-2-google-ai-text-to-speech-system/tacotron>

<https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/>