
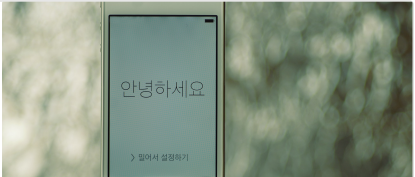


# 위클리 NLP week 13 ~ week 23

위클리 NLP - jiho-ml

구글 컴퓨터 언어학자가 쓰는 자연어 처리 (natural language processing)에 대한 튜토리얼! 매주 구독해서 받아 보세요.

 <https://jiho-ml.com/tag/weekly-nlp/>



## Week 13 - language model

language modeling ( LM ) - 자연스러운 문장을 모델링하는 것 : 인간의 뇌에는 이미 존재함

- 문장을 단어의 연속으로 보라
- 단어가 모여서 문장, 문장이 모여서 데이터, 모든 단어들이 있는 vocabulary
- 한 문장이 그럴듯함은 joint probability(동시 확률분포)로 계산

데이터(corpus)를 통해 LM을 학습함 ⇒ **문장이 주어졌을 때 그 문장이 얼마나 그럴 듯 한지를 확률로 계산하는 모델**

## Week 14 - N-gram language model

복잡한 input문장을 단순하게 보기 위해 uni-gram, bi-gram 사용

uni-gram (= 1gram) : 단어 1개씩 봄 ⇒ 계산은 단순하지만 단어의 순서를 생각 X

$$\begin{aligned} P(\text{"My name is Jimmy"}) &= p(\text{"My"}, \text{"name"}, \text{"is"}, \text{"Jimmy"}) \\ &= p(\text{"My"}) \times p(\text{"name"}) \times p(\text{"is"}) \times p(\text{"Jimmy"}) \end{aligned}$$

bi-gram (= 2gram) : 단어 2개씩 봄

조건부 확률로 봄 -  $P(W_2|W_1)$  =  $W_1$ 이 주어졌을 때  $W_2$ 가 올 확률

$$\begin{aligned} P(\text{"My name is Jimmy"}) &= p(\text{"My"} | \text{<start>}) \times p(\text{"name"} | \text{"My"}) \times p(\text{"is"} | \text{"name"}) \\ &\times p(\text{"Jimmy"} | \text{"is"}) \times p(\text{<end>} | \text{"Jimmy"}) \end{aligned}$$

start와 end는 문장의 시작과 끝

한번에 n개의 연속된 단어를 묶어 생각하는 것을 **n-gram LM**이라 함

- 학습된 n-gram 모델에 새로운 단어 (out-of-vocabulary : OOV)가 들어오면 smoothing이라는 기술로 확률이 0으로 계산되는 것을 막음

## Week 15 - ASR

자동 음성 인식 ( Automatic Speech Recognition : ASR ) → Nosi Channel Model을 기반으로 음성을 텍스트로 변환하는 기술

Nosi Channel Model : 발신된 메시지에 노이즈가 섞인 경우 노이즈를 제거해 decoding(해독)하여 수신하도록 함

ASR 모델 → 베이지안 이론을 통해 **소리 모델 (Acoustic Model)**, **언어 모델 (Language Model)**로 분리

ASR 모델의 input인 소리를 X, 해독할 텍스트를 W

$$\begin{aligned} P(W|X) &= P(X|W)P(W) / P(X) : \text{소리 } X \text{가 주어졌을 때, 텍스트 } W \text{는?} \\ W &= \operatorname{argmax} P(X|W)P(W) \quad ( P(X|W) - \text{소리 모델, } P(W) - \text{언어 모델} ) \end{aligned}$$

## Week 16 - AI의 음성 인식 방법

발음의 가장 기본 단위 : 음운( phoneme )

음운을 정리한 phoneme dictionary → CMUdict, IPA 두 종류

**AM (소리 모델)** : 주어진 음성을 연속된 음운으로 변환하는 모델

→ 과거에는 HMM, DTW 등의 모델이 쓰이다 현재는 딥러닝 모델이 쓰임

AM에서 예측된 음운들을 LM과 연계해 읽을 수 있는 단어로 바꾸는게 ASR의 마지막 단계

End-to-end - 여러개로 나누지 않고 한번에

**End-to-end ASR** - AM과 LM으로 나누지 않고 하나의 모델로 학습

- 음성과 텍스트로 모델을 학습
- Connectionist Temporal Classification (CTC) 알고리즘이 나옴
- 성능은 좋지만, 수많은 데이터가 필요

ASR의 성능 계산 지표 - **Word Error rate (WER)** : 말한 단어 중 몇 퍼센트의 단어를 틀리게 알아듣는지

Week 17 - 딥러닝이 language model에 필요한 이유

- LM ⇒ 어떤 문장의 그럴듯함을 확률로 계산하는 모델
- ⇒ corpus를 학습 데이터로 사용하여 통계 모델 구축
  - ⇒ N-gram LM : n개 연속되는 단어의 빈도를 계산하여 LM을 만든 모델
  - ⇒ LM으로 여러 문장이 주어졌을 때 가장 그럴듯한 문장을 뽑을 수 있음
  - ⇒ LM으로 문장의 앞부분이 주어졌을 때 다음 단어를 예측할 수 있음

- N-gram LM의 한계
1. 못 본 단어의 조합의 확률은 0 → 일반화 능력이 떨어진다고 함
    - ⇒ 단어를 독립적인 특징으로 보기 때문에 예측에 한계가 있음
    - ⇒ 따라서 **Neural LM (딥러닝 언어 모델)**은 **word embedding**을 통해 해결
  2. n이 커질 수록 멀리 있는 과거의 단어를 고려하는 것이 힘들 → long term dependency 문제
    - ⇒ **RNN**을 사용해 문제 해결

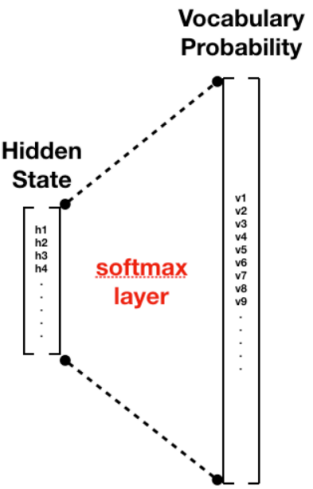
Week 18 - RNN LM

**RNN LM** - 딥러닝 언어 모델 (neural LM)

**This is a very long sentence explaining about a long sentence.**

rnn은 다 본다!                      trigram   bigram   Target

- RNN의 hidden state에 단어들의 정보가 압축되어 들어있음
- ⇒ input vector로 word embedding이 한 개 들어갈 때, 나오는 output vecotr를 hidden state라고 함
- 학습 데이터에 존재하는 단어들로 vocabulary 리스트를 만들
- ∴ hidden state로 RNN모델을 학습해서 단어 예측 → classification



- hidden state 벡터와 softmax layer를 연결해 vocabulary에 있는 단어의 숫자만큼 벡터로 만들 → vocabulary probability
- softmax : 확률 점수로 만드는 함수
- 학습 데이터를 통해 모델을 개선 - loss function으로
- softmax를 쓰면 loss function으로 cross-entropy function
- ex) "I love you", "This is a sentence" 가 있을 때:

```
(input x, output y) # x가 주어졌을 때, y를 예측하기
([], "I")
(["I"], "love")
(["I", "love"], "you")

([], "This")
(["This"], "is")
(["This", "is"], "a")
(["This", "is", "a"], "sentence")
```

⇒ 라벨링을 하지 않고 자동으로 생성할 수 있어 LM은 비지도 학습으로 구분됨

## Week 19 - neural LM 평가 방법

LM의 평가 지표 : **perplexity** (held-out test data) -  $2^{\text{entropy}}$

entropy - 불확실성이 커지면 entropy가 높아지고, 확실하면 낮아짐

∴ perplexity가 낮은 모델이 좋다

Text generation : LM으로 새로운 글 생성

→ LM이 확률 분포가 생성되는 probabilistic model임

→ softmax를 통해 확률 점수를 만들고, 그를 통해 다음 단어 예측하는 것을 반복 (샘플링)

ex) [Math is a]가 주어졌을 때, 다음 단어로 "common"이 뽑히고, [Math is a common]이 주어졌을 때, 다음 단어로 "democrat"이 뽑힘 .. 글이 끝날 때 까지 반복

## Week 20 - 구글 번역기

기계 번역 (Machine Translation: MT)

MT ⇒ NLP에서 가장 확실한 응용 분야

Parallel Corpora : 번역을 하기 위해 필요한 데이터 ⇒ source 언어와 target 언어 (한국어, 영어 쌍)

Phrase-based Machine Translation : 문장을 구절로 나눠서 접근

1. 구절 (phrase) 또는 단어 간 대응되는 사전(dictionary)을 만들

ex) play = 놀다 80%, 연극 20% 로 확률 점수로 저장

2. 사전을 이용해 문장을 구절로 나눔

ex) "나는 너와 라면을 먹었다" → 나는, 너와, 라면을, 먹었다

3. 나눈 부분을 번역한 후 순서를 알맞게 맞춤 (언어의 어순을 맞춰줌)

ex) I, with you, ramen, eat → I ate ramen with you

⇒ Word alignment : HMM같은 확률 모델로 만들어 학습 후 순서를 찾아냄

⇒ dependency parsing (의존 구조 분석) : 문장 구조를 자동 분석하는 모델

## Week 21 - seq2seq 모델

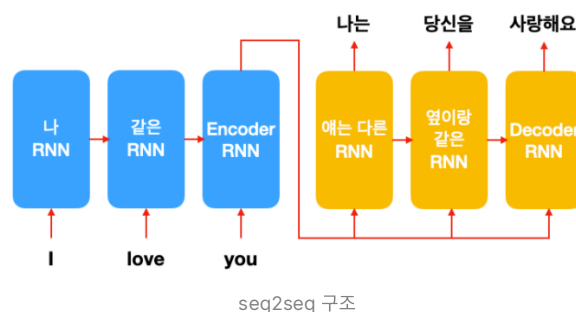
신경망 기계 번역 (Neural Machine Translation: NMT) 등장

번역 :

⇒ source언어 x가 주어졌을 때, target언어 y의 확률 : 조건부 언어 모델 (**conditional language modeling**)

⇒ ASR에서 쓰인 nosiy channel model과 같이 번역도 **encoder-decoder model**

conditional language modeling, encoder-decoder model과 RNN을 이용한 neural LM이 합쳐져 **seq2seq 모델** 탄생



encoder RNN 1개, decoder RNN 1개

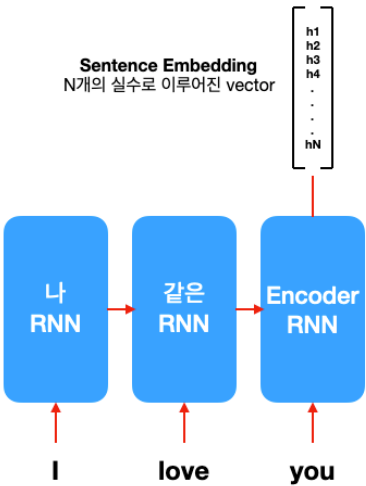
decoder의 RNN은 encoder RNN의 최종 hidden state를 계속 참고함

## Week 22 - seq2seq 모델

### seq2seq

encoder RNN은 번역해야하는 문장(input source)를 한 단어씩 읽음 → 한 단어가 들어갈 때마다 hidden state가 바뀜

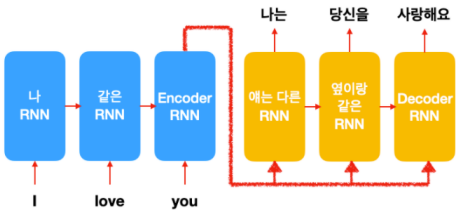
encoder RNN의 최종 hidden state : **문장 임베딩 (sentence embedding)** - 이 vector에 번역해야 할 문장에 대한 모든 정보가 들어있음



decoder RNN은 순차적으로 한 단어씩 출력

⇒ RNN LM과 비슷하지만 다른점 2개

- 1. sentence embedding도 같이 input으로 넣음



- 2. 랜덤으로 샘플링 하지 않고 확률이 높은 단어를 선택

RNN LM과 마찬가지로 softmax가 생성한 확률 점수와 target 정답을 가지고 cross-entropy loss를 계산 → 정답에 가까운 것에 높은 확률

첫번째 단어를 잘못 고르게 될 수도 있기 때문에 Top-K beam search 방법을 사용 → 확률이 높은 단어 K개를 골라 선택

번역 퀄리티 계산 : **BLEU score** → 얼마나 정답과 공통 단어가 많은냐에 따른 점수

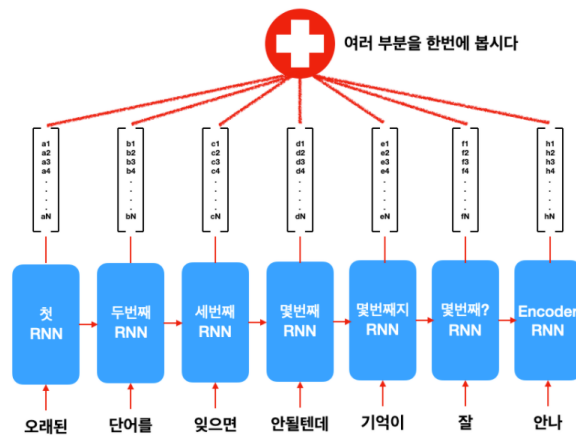
## Week 23 - 관심법

RNN과 LSTM, RNN 두개를 쓴 seq2seq 모두 기억력에 문제 → 긴 문장을 번역하려고 하면 힘들

seq2seq의 경우 - sentence embedding에 한 문장의 정보를 다 넣기 때문에 앞의 단어들은 까먹게 됨

∴ **Attention Mechanism (관심법)**이 등장

⇒ 문장을 한 vector에 넣지 않고, 매 단어를 넣는 RNN hidden state를 모두 이용하여 여러 vector를 얻음 → 여러 vector를 한번에 묶음



⇒ 앞 부분 번역할 때 앞 단어에 더 가중치, 뒷 부분 번역 할 때 뒷 단어에 더 가중치를 갖게 해서 자연스럽게 번역

⇒ 어느 부분의 단어들에 더 가중치를 둘지는 [pharse alignment](#)로 단어 간 관계를 찾을

transformer, Bert 모델 기반이 Attention Mechanism