

Chapter 1 자연어와 단어의 분산 표현

자연어 처리 \Rightarrow 컴퓨터가 인간의 언어를 이해할 수 있도록 하는 것!

자연어 처리

NLP의 핵심 \Rightarrow 단어의 의미를 어떻게 이해시킬 것인가?

↳ 시소러스 (유의어 사전) 등을 사용
통계 기반 기법
稠密 기반 기법 (word2vec)

시소러스

단어의 의미를 사람이 직접 입력 \rightarrow 일반적인 사전이 아니라 유의어 사전 형태의 시소러스 활용

Car = auto, automobile, machine 등등 \rightarrow 이걸 free 형태로 표현하기도 함 (단어 네트워크)

사람이 수작업으로 계속 분류해줘야 함 \Rightarrow 시대 변화에 대응 어렵다
↳ 사람에 대한 비용
단어의 미묘한 차이 구분 힘들

통계 기반 기법

말뭉치 (Corpus) 를 이용

↳ 대량의 텍스트 데이터, 연구나 활용을 하기 위해 수집된 것들

단어의 의미 \Rightarrow 벡터 표현으로 (분산 표현)

단어 자체의 의미보다 context가 중요

↳ '단어의 의미는 주변 단어에 의해 형성된다' 는 분포 가설에 기반

문맥 (맥락) 을 어디까지로 볼 것인가? \Rightarrow 윈도우의 크기

중요 키워드: 동시 발생 행렬, 벡터 간 유사도 (코사인 유사도)

통계 기반 기법 개선하기

동시 발생 행렬에서는 의미없는 단순 고빈도 단어를 거르기 힘들 \Rightarrow 점별 상호정보량 사용

PPMI 행렬을 구성하면 '의미있는' 단어 조합 추출 가능

(pointwise mutual information)
PMI

↳ 말뭉치의 어휘 수가 증가하면 행렬의 차원도 함께 증가

\Rightarrow 주로 사용하는 것은 양의 상호정보량 (PPMI)

특잇값 분석 (SVD) 등을 통해 차원 축소 \rightarrow 희소 (sparse) 벡터를 밀집 (dense) 벡터로 바꾸는 과정

chapter 2 word2vec

주요 기반 기법과 신경망

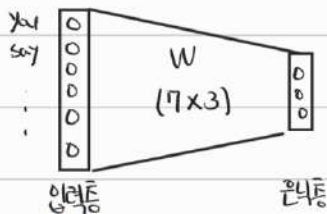
통계 기반 기법 \rightarrow 주변 단어의 빈도를 기초로 단어를 표현 \Rightarrow 대규모 말 문치 다룰 때 문제

\hookrightarrow SVD 계산이 너무 오래 걸린다

주요 기반 기법 \rightarrow 미니 배치를 활용한 신경망을 쓰기 때문에 대용량 처리 가능

신경망에서의 단어 처리 \rightarrow 고정 길이의 벡터 변환 (one-hot encoding 등)

\hookrightarrow 이것 이용해서 입력층의 차원을 결정 \Rightarrow 신경망을 이용해 단어 처리 가능



결국, 가중치 w 가 그 단어의 분산벡터

\hookrightarrow one-hot 표현이므로 결국 가중치 중에 행 한개를 선택하는 것과 같다

단순한 word2vec

CBOW - continuous bag of words \rightarrow '맥락'으로 부터 '중앙'을 예측

\hookrightarrow 주변 단어를 보고 그 단어를 예측

통계 기반 vs 주요 기반

- 통계 기반 기법 \rightarrow 말문자의 전체 통계 1회 학습 \rightarrow 새로운 단어가 들어오면 전부 갱신 필요
 - 주요 기반 기법 \rightarrow 말문자의 일부분을 여러 번 보면서 학습 (미니 배치) \rightarrow 지금 가중치를 바탕으로 새로 학습 가능
- 둘의 성능은 비슷

성리

- 주요 기반 기법은 추측하는 것이 목적인데 그 부산물로 단어의 분산표현을 얻을 수 있다.
- word2vec은 주요 기반 기법이며 단순한 2층 신경망이다
- word2vec은 CBOW와 skip-gram으로 나뉜다.
- CBOW는 맥락으로부터 단어를 추측하며 skip-gram은 단어로부터 맥락을 추측한다
- word2vec은 가중치를 다시 학습할 수 있으므로 단어의 분산표현 갱신이나 새로운 단어 추가를 효율적으로 할 수 있다

Chapter 3. Word2Vec 속도 개선

CBow 모델: 말뭉치 ↑ ⇒ 계산량 ↑ 속도 저하

⇒ Embedding 과 네거티브 샘플링으로 해결

CBow 모델: 말뭉치 증가 ⇒ 뉴런 수 증가

원 벡터 { 입력층의 one-hot 표현과 가중치 행렬 곱 ⇒ Embedding
은닉층과 가중치 행렬 곱과 Softmax 계층 ⇒ 네거티브 샘플링

W, n
Wout

Embedding 계층

말뭉치 (어휘 수) 가 100만 ⇒ one-hot도 100만 ⇒ 뉴런도 100만

↳ 결국 계산하는 것은 가중치 행렬의 특성 행 → 이것을 대체하는 계층이 Embedding 계층

Embedding 계층에 단어 Embedding (분산표현)을 저장 → 행렬곱 대신에 이거 사용

네거티브 샘플링

은닉층 이후에서 계산 오래 걸리는 곳 ⇒ { 은닉층 × 가중치 (중력)
소프트 맥스

네거티브 샘플링: 다중분류 ⇒ 이진분류 (yes/no로 바꾸기)
Softmax

이진 분류: Sigmoid 함수 + cross entropy → $L = -(t \log y + (1-t) \log (1-y))$

확률로 변환

Sigmoid의 출력
실제 label: 0 또는 1

타깃이 say 이고 맥락이 you, hello 일 때

$$\begin{cases} t=1 \text{ 이면 } -\log y \\ t=0 \text{ 이면 } -\log (1-y) \end{cases}$$

say가 정답인가? → 이렇게 만드는 거 이진분류라

답이 아닌 경우도 학습 필요 → 부정적 예 (오답)을 몇 개 선택해서 학습 (negative sampling)

네거티브 샘플링: 긍정적 예를 타깃으로 손실 계산 + 부정적 예 샘플링 후 손실 계산

↳ 각각의 손실을 더한 것이 최종 손실

네거티브 샘플링 기법 → 말뭉치의 통계 데이터를 기초로 샘플링 → 말뭉치에서 자주 등장 ⇒ 많이 "확률분포" 이용
드물게 등장 ⇒ 적게

word2vec 을 사용한 애플리케이션

word2vec 으로 얻은 분산표현 ⇒ 비슷한 단어 찾기 → 유사하게 전이학습도 가능
↙
이미 학습한 분산 표현을 이용하여 다른 작업 수행

정리

- ① Embedding 계층은 단어의 분산 표현을 담고 있고 순전파 시 해당 벡터 출력
- ② Word2Vec은 어휘 수 증가에 비례하여 계산량 증가 → 근사치 사용
- ③ 네거티브 샘플링: 부정적 예시 몇 개 사용 ⇒ 다진 분류를 이진 분류로 취급
- ④ word2vec 에서 얻은 분산표현: 단어의 의미 내포, 비슷한 맥락의 단어 ⇒ 단어 벡터 공간에서 가까이 위치

chapter 4 순환신경망

피드포워드 신경망 (FFP) ⇒ 순환신경망의 등장
↳ 시계열 데이터 처리에 약점 (recurrent neural network, RNN)

확률과 언어모델

CBOW 모델 → 맥락으로부터 타깃을 예측
한쪽(이전) 맥락만 고려
 $w_{t-2} \quad w_{t-1} \quad w_t \quad w_{t+1} \quad w_{t+2}$
↳ 확률로 나타내면 $p = (w_t | w_{t-1}, w_{t+1})$
⇒ '언어모델'의 등장

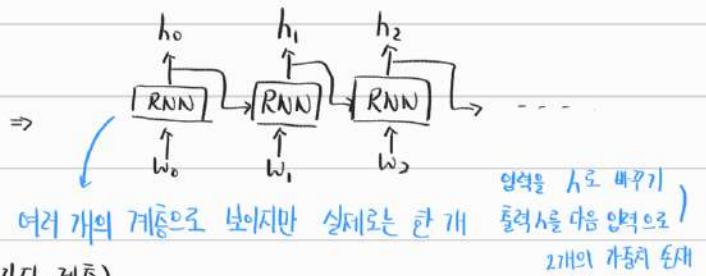
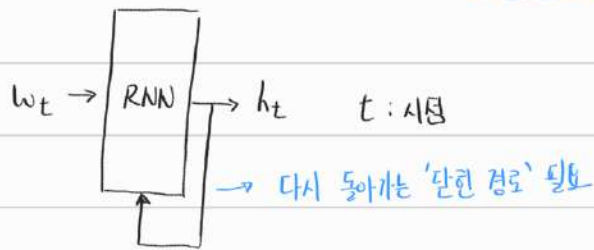
언어모델 - 단어나열에 확률을 부여 특정 시퀀스에 대해 그 시퀀스가 일어날 가능성(확률) 평가
↳ 기계 번역, 음성 인식에 사용됨

CBOW 모델 → 언어모델 정리

↳ CBOW의 경우 맥락의 크기가 '고정' ⇒ 맥락 크기를 계속 조정해야함
↳ 크기를 계속 키워도 단어간의 순서가 무시되는 문제 발생 } ⇒ RNN으로 해결

RNN이란

'순환' 하는 신경망 → 순환을 위해선 단한 경로가 필요



RNN → 'h'라는 상태를 가지는 계층 (메모리를 가진 계층)

역전파 시에도 일반적인 FFP의 역전파와 동일 (BPTT: Backpropagation through time)

↳ 긴 시계열 데이터를 학습할 때 문제 → 역전파가 불안정 계산 메모리 소요

Truncated BPTT - 큰 시계열 데이터를 처리할 때 신경망 연결을 적당한 길이로 끊기

↳ 시간적 길이로 너무 길어진 걸 잘라내어 작은 신경망 여러 개로 만들

그리고 이 작은 신경망에서 오차 역전파 시행

순전파의 연결은 유지하고 역전파의 연결만 끊어줘야 함 → 데이터를 순서대로 입력해야 함

언어모델의 예측성능 평가 → 퍼플렉시티 (perplexity, 혼란도) 사용

↳ 확률의 역수: you 뒤에 say가 올 확률이 0.8 $\Rightarrow \frac{1}{0.8} = 1.25$
0.5 $\Rightarrow \frac{1}{0.5} = 2$

입력데이터가 여러 개인 경우 $\text{perplexity} = e^L$
 $L = \frac{1}{N} \sum_{n,k} t_k \log y_{nk}$

값을수록 더 좋다
= 선택지를 더 줄였다

정리

- ① RNN은 순환하는 경로가 있고 이를 통해 내부에 '은닉상태' 기억 가능
- ② RNN의 순환경로를 펼쳐면 다수의 RNN 계층이 연결된 신경망으로 해석할 수 있다
- ③ 긴 시계열 데이터를 학습할 때는 데이터를 블록단위로 쪼개고 BPTT에 의한 학습
- ④ 언어 모델은 단어 시퀀스를 확률로 해석

chapter 5 게이트가 추가된 RNN


기본 RNN의 성능 안좋은 \rightarrow 장기 의존관계를 학습 못함

\Rightarrow '게이트'가 추가된 LSTM이나 GRU 주로 사용 (장기 의존성 학습 가능)

RNN의 주요 문제점 \rightarrow 기울기 소실, 기울기 폭발

\hookrightarrow 기울기 소실과 폭발이 일어나는 이유

① 활성화 함수 tanh 사용 \rightarrow 기울기가 계속 사그라들

 \rightarrow 미분 그래프

\nearrow ReLU면
이런정도 해결

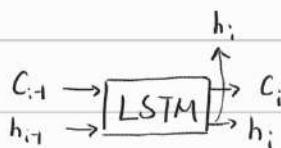
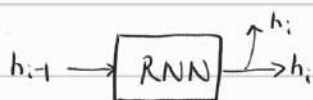
② 반복적인 행렬곱 \rightarrow 행렬의 특이도에 따라 소실 혹은 폭발

기울기 폭발 대책 \rightarrow 기울기 클리핑 (gradient clipping) 사용

$\text{if } \|\hat{g}\| \geq \text{threshold}$

$$\hat{g} = \frac{\text{threshold}}{\|\hat{g}\|} \hat{g} \rightarrow \text{특성값보다 크면 기울기를 고정}$$

RNN 계층과 LSTM 계층



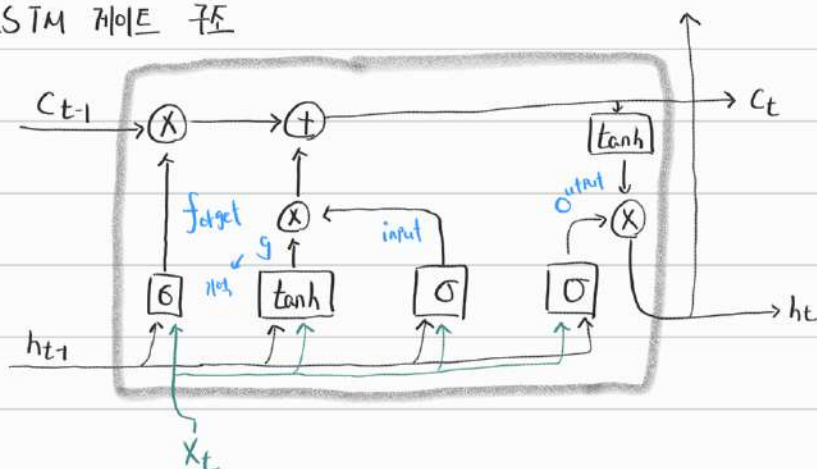
LSTM에는 기억셀 C가 존재!

\nearrow LSTM내에서만 데이터 주고 받음

기억셀 C에 이전까지 필요한 정보 모두 저장 (그렇게 되도록 학습 수행) \rightarrow C의 존재로 인해 가중치 손실률
낮음

Output Gate, forget Gate 등이 존재 \rightarrow Gated RNN

LSTM 게이트 구조



과적합 개선 \rightarrow 드롭아웃 이용 \rightarrow 시간축 방향 (y_t, y_{t+1}) 말고 계층 깊이에 넣기

\hookrightarrow 변형 드롭아웃을 이용하면 시간축 방향 가능

Embedding 계층과 Affine 계층의 가중치를 공유하면 성능 향상

정리

① RNN은 기울기 소실, 폭발이 문제

② LSTM에는 input, forget, output 3개의 게이트와 각각의 가중치 존재 (시그모이드)

chapter 6 RNN을 사용한 문장 생성

언어모델을 사용한 문장 생성

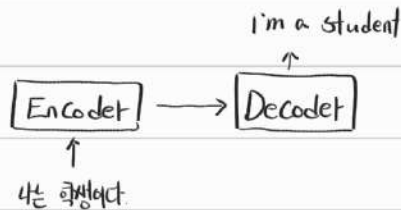
학습된 언어모델에 단어를 입력 \Rightarrow 다음에 나올 단어들이 확률적으로 출현

↳ 제일 높은 확률만 선택 \Rightarrow 결정적
확률을 이용한 샘플링 \Rightarrow 확률적

\Rightarrow 학습이 더 잘된 언어모델을 쓸수록 좋은 결과를 나타냄

Seq2Seq 모델

시계열 데이터를 다른 시계열데이터로 변환하는 모델 (Encoder-Decoder 모델)



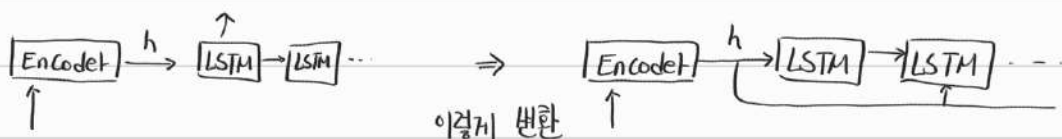
encoder에서 decoder로 전달하는 정보는 마지막 계층의 hidden state

\Rightarrow encoder를 이용해서 input을 고정길이 벡터로 바꾸는 것과 같다

Seq2Seq의 개선: Reverse와 Pecky

Reverse: 입력 데이터 반전

pecky: decoder에 한번만 선택시킨 'h'를 모든 RNN에 선택



Chapter 7 어텐션 (Attention)

어텐션 매커니즘 → Seq2Seq의 개선

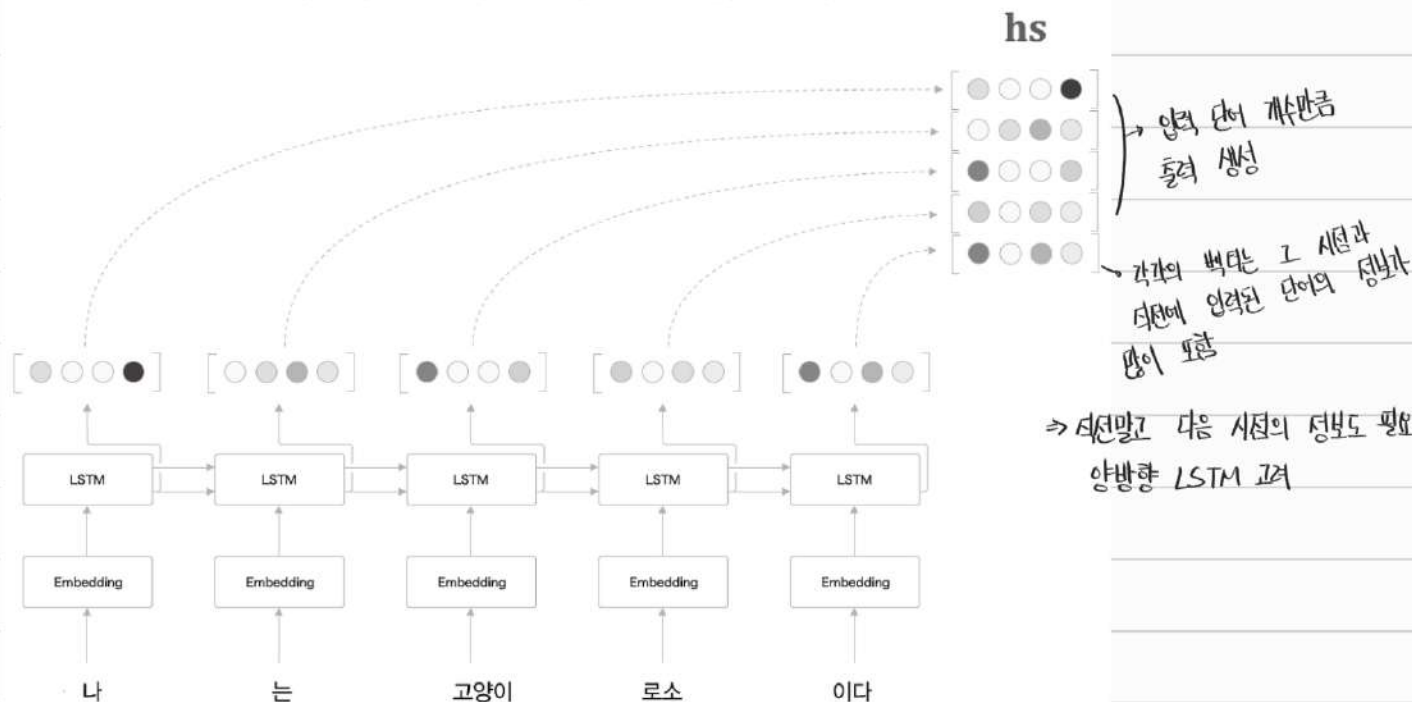
↳ 필요한 정보에만 주목

Seq2Seq Encoder ⇒ 고정길이 벡터 h 반환

⇒ 문장 길이에 상관없이 고정길이 반환 ⇒ 필요한 정보만

Seq2Seq Encoder ⇒ 문장 길이에 따라 출력 길이도 달라지도록

그림 8-2 Encoder의 시각별(단어별) LSTM 계층의 은닉 상태를 모두 이용(h_s 로 표기)

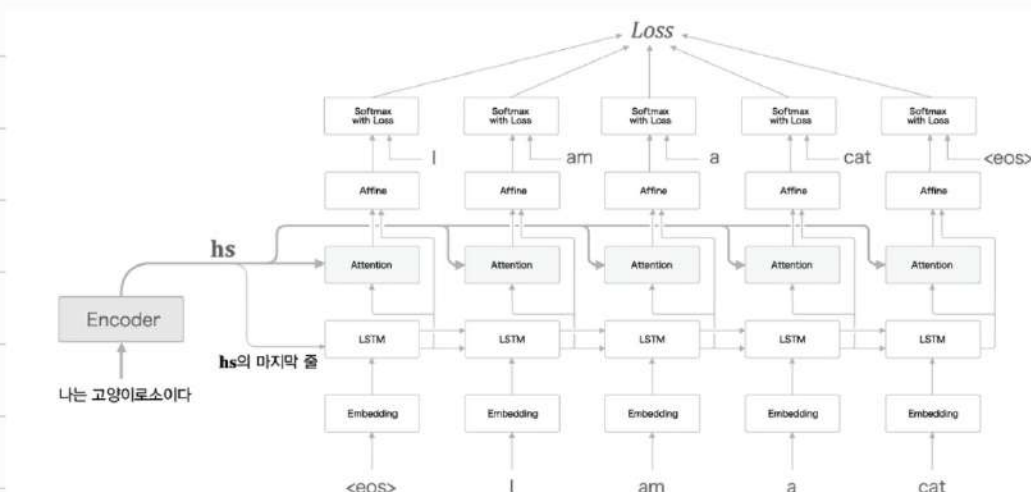


Decoder의 개선

Encoder의 출력 h_s 를 모두 활용하면서 어떤 단어끼리 대응관계가 있는지 학습

↳ 알라인먼트 (alignment)

필요한 정보에만 주목하여 그 정보로부터 시계열 변환 수행 ⇒ 어텐션



어텐션 계층에서 하는 일

⇒ 각 시점에 Decoder에 입력된

단어와 대응관계를 h_s 에서 선택

'I'라는 가중치를 이용 (가중치 합 1)

→ 맥락 벡터 'c' 획득

가중치 α 는 h_s 와 h_i 벡터 내적으로 계산

정리

- ① seq2seq 등의 시계열 데이터를 변환하는 과정은 두 시계열 사이에 대응관계가 많이 존재
- ② 어텐션은 시계열 사이의 대응관계를 데이터로부터 학습
- ③ 어텐션에서는 벡터 사이의 유사도를 이용해 가중합을 구하고 이 가중합을 출력으로 사용
- ④ 어텐션이 출력하는 가중치(확률)를 시각화 하면 입출력의 대응관계를 볼 수 있다