Ratsgo.github.io 포스팅 글을 정리하였습니다.

그래프로 중요 기사 걸러내기

가정: 같은 시간대에 비슷한 제목의 기사가 많으면 중요한 이슈를 다루는 보도일 것

뉴스 소비자에게 중요한 기사를 걸러 보여주면서도 뉴스의 맥락을 알 수 있게 하면 좋겠다

➔ 스트럭처 저널리즘(Structurd Journalism) : 기사 및 기사 속 내재 정보를 계속 누적시켜 재맥락화(reconcextualize)하려는 노력을 총칭. 단순 팩트나 기존 기사들을 재조합하여 새로운 가치, 맥락을 만들어보려는 시도

중요 기사의 기준을 양적으로 접근

Graph-based News Representation

전제: 중요한 뉴스라면 모든 언론사가 취급할 것이다, 중요 기사는 제목이나 키워드가 비슷할 것이다

뉴스 제목을 노드(꼭짓점)과 엣지(간선)으로 표현.

뉴스 제목을 포스태깅하고 조사 등 불필요한 품사를 제거. 중복 단어 숫자를 웨이트(까중치)로 하는 무방향 엣지를 연결.

그래프 이론에서는 노드의 중요도를 뽑는 데 **중심성(Centrality) 개념을** 제시. 대표적인 것이
간선중심성(degree centrality) : 엣지에 해당하는 웨이트들의 합

다른 지표: **고유벡터 중심성(eigenvector centrality) :** 중요한 노드에 연결된 노드가 중요하다는 관점에서 창안된 개념.

그래프의 엣지로 이뤄진 인접행렬을 **고유값 분해(eigenvalue decomposition)**을 하여 얻을 수 있음

이렇게 구해진 i 번째 노드의 고유벡터 중심성 점수와 그 의미는 다음과 같음.

$$x_i = \frac{1}{\lambda}\sum_{i=1}^{n} A_{ij}x_j$$

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} A_{11}x_1 + A_{12}x_2 + \dots + A_{1n}x_n \\ A_{21}x_1 + A_{22}x_2 + \dots + A_{2n}x_n \\ \dots \\ A_{n1}x_1 + A_{n2}x_2 + \dots + A_{nn}x_n \end{pmatrix} = \begin{pmatrix} \lambda x_1 \\ \lambda x_2 \\ \dots \\ \lambda x_n \end{pmatrix}$$

중심성 점수가 높은 중요한 이웃과 연결된 i 번째 노드 또한 중심성 점수가 높아지게 됨.

중요한 노드에 연결된 노드가 중요함.

실험 설계 및 결과

뉴스 제목의 중복 단어를 가중치로 하는 그래프 구축

각 노드의 중심성은 간선중심성과 고유벡터 중심성 두 가지 지표를 구해 이를 곱하여 모두 고려

분석 대상: 조선일보, 한겨레 등 10 개 주요 조간 매체 10 개월치 기사

Deree: 간선중심성, eigen: 고유벡터중심성, result: 둘을 곱한 스코어

정렬해 상위 7 개 뽑음

Sequence-to-Sequence 모델로 뉴스 제목 추출하기

S2S 모델: Recurrent Neural Network 의 가장 발전된 형태의 아키텍처. LSTM, GRU 등 RNN cell 을 길고 깊게 쌓아서 복잡하고 방대한 시퀀스 데이터를 처리하는 데 특화된 모델.

-영어-한국어, 한국어-일본어 등 기계번역에 사용됨

-2014 년 Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation 논문에서 처음 소개 됨

-크게 encoder 와 decoder 두 파트

Ex. 영어->한국어 변환 기계번역

인코더는 소스랭귀지(source language)인 영어 텍스트 처리

디코더는 타겟랭귀지(target language)인 한국어 텍스트 맡음

디코더의 입력은 이 모델 정답에 해당하는 한국어 텍스트이며 출력 또한 한국어 텍스트임

〈go〉와 〈eos〉는 각각 정답 시작과 끝을 알리는 일종의 기호.

인코더 입력: Good morning?

디코더 입력: 〈go〉 좋은 아침입니다!

디코더 출력: 좋은 아침입니다! 〈eos〉

인코더는 소스랭귀지 정보를 압축함

디코더는 인코더가 압축해 보내준 정보를 받아서 타겟 랭귀지로 변환해 출력함

하지만 학습과정과 예측(테스트)과정이 조금 다름

학습과정에서 디코더는 인코더가 보내온 정보와 실제 정답('〈go〉좋은 아침입니다!')을 입력으로 받아 좋은 아침입니다〈eos〉를 출력함

예측 과정에서 디코더는 인코더가 보내온 정보와 '〈go〉'만 입력으로 받아 결과물을 차례대로 출력함.
띄어쓰기 기준으로 인풋을 만든다면 첫 결과물은 '좋은'. 이것을 다시 자신의 다음 단계 입력으로 넣어 '아침입니다'를 출력하게 됨.

디코더가 예측 과정에서 자신의 직전 출력(좋은)을 다음입력으로 다시 넣는 이유: 예측 단계에서는 학습 때와 달리 정답이 없기 때문. 그래야 정답이 주어지지 않은 상태(예컨태 구글 번역기에 우리가 번역하고 싶은 문장을 입력할 때)에서도 예측이 가능해짐.

문제 정의

인코더에 뉴스 본문을 넣고 디코더에 제목을 넣는 것

정답이 있는 데이터만 S2S 학습이 가능함. 하지만 자연언어처리 분야에서 정답이 있는 데이터를 얻기는 매우 어려움, 그런데 뉴스 기사는 비교적 수월하게 구할 수 있는 정답 있는(제목 있는) 데이터.

데이터 전처리

2016 년 1 월초 사흘치 기사 1000 건 모음.

기사를 제목과 본문으로 나눔.

KoNLPy 를 사용하다가 형태소 분석 과정에서 잘못된 태깅으로 말뭉치 정보가 왜곡되거나 손실될 염려를 배제하기 위해 다른 방법 사용,

단어를 띄어쓰기 기준으로 나누고 3 글자까지만 잘라서 노말라이즈 함.

Ex. 감정가, 감정가의, 감정가격에,,, 등을 모두 같은 단어로 봄

단어수 줄여서 효율적으로 하기 위해서.

실제로 단어 단위 S2S 모델은 학습 말뭉치의 단어 수가 분류해야 할 클래스 개수가 되기 때문에 단어수가 지나치게 크면 학습의 질은 물론 속도도 급격하게 감소하는 문제

S2S 에 넣은 인풋: 텍스트를 숫자로 바꾸어 넣음

경기도 0, 준예산 1, 사태 2… 이렇게 하기 위해서는 단어와 숫자가 매칭된 사전(dictionary)가 있어야 함

예측 과정에서는 숫자를 단어로 바꿔야 함. 그러므로 숫자와 단어가 매칭된 사전도 별도로 만들었음.

데이터를 모델에 넣기 전에 고려할 것: 데이터 차원수

RNN: 시퀀스에 길이에 유연한 네트워크 구조

But S2S 처럼 복잡하고 방대한 네트워크는 시퀀스 길이를 맞춰주는 게 좋음

그래서 디코더에 넣을 시쿠너스 길이를 뉴스 제목 최대 길이(18 단어)로 맞추고, 인코더에 넣을 시퀀스 길이를 기사 앞부분 100 개로 맞춤

만약 길이보다 데이터 시퀀스 길이가 짧다면 〈PAD〉를 넣어 길이를 맞춰줌. Word_to_ix 와 ix_to_word 에 각각 〈PAD〉라는 요소를 추가해주고 길이가 부족한 데이터에 끼워 넣었음

모델 구현

Cell 은 두 가지를 실험. LSTM/GRU

은닉층 여러 개 쌓을 수 있음

중요한 부분을 중점적으로 검토하게 해 분석의 정확도를 높이는 attention 기법을 적용/미적용한 것의 차이도 실험

데이터 시퀀스의 순방향과 역방향 정보를 모두 고려하는 bidirectional 방법론 가운데 후자만 차용

즉, 인코더 입력 시퀀스를 모델에 입력으로 넣을 때 역방향으로 넣었다는 이야기(뉴스 본문 첫 단어가 인코더의 맨 마지막 입력이 됨)

코드는 ipynb 파일에 있음

Toolloading_data 함수는 csv(utf-8)을 읽어 title 과 contents 로 나눠줌

Tool.make_dict_all_cut 은 학습 말뭉치들을 읽어들여 위 예시처럼 word_to_ix 와 ix_to_word 를 생성함

Multi 는 은닉층 1 개 할건지, 여러 개 할건지

Forward_only 는 학습 때는 false, 추론(테스트) 때는 true 로 지정

Vocab_size 는 말뭉치 사전 단어수, num_layers 는 은닉층 개수, batch_size 는 1 회당 학습하는 데이터 수(배치 크기)

Encoder_size 는 인코더에 넣을 입력 시퀀스의 최대 길이, decoder_size 는 디코더에 넣을 시퀀스의 최대 길이

Tool.check_doclength 함수는 입력 컨텐츠의 최대 길이 반환함

Tool.make_inputs 는 단어를 숫자로 바꾸는 과정

클래스 seq2seq 에서 정의한 네트워크 구조

변수 선언 부분: encoder_size 만크므이 placeholder 를 생성해 encoder_inputs 를 만듦

Decoder_inputs, targets, target_weights 를 만듦

Target_weights 는 target(제목_에 대응하는 요소가 〈PAD〉일 경우 0, 〈PAD〉가 아닐 경우 1 로 채워 넣은 리스트

네트워크 선언 부분: cell 은 tf.nn.rnn_cell.GRUCell 이라고 선언. tf.nn.rnn_cell.LSTMCell 이라고 선언하면 GRU 를 간단하게 LSTM cell 로 변환 가능. 층을 여러 개 쌓고 싶으면 tf.nn.rnn_cell.MultiRNNCell 이라는 함수를 쓰면 됨.

학습과 예측 과정:

If not forward_only 구문이 실행되는 부분이 학습 과정

Tf.nn.seq2seq.embedding_attention_seq2seq 함수는 output 을 반환

Feed_previous 에 flase 를 씀

Attention 을 쓰지 않으려면 tf.nn.seq2seq.embedding_seq2seq 함수를 사용해보기. 이 output 에 W 를 내적하고 b 를 더해 logit 을 만들고, 이를 바탕으로 cross-entropy loss 를 구함

추론 과정

Else: 구문이 실행되는 부분

Tf.nn.seq2seq.embedding_attnetion_seq2seq2 함수의 feed_previous 에 True 를 집어넣음

디코더가 직전에서 출력한 결과를 다음 과정의 인풋으로 받아들여 추록하라는 지시

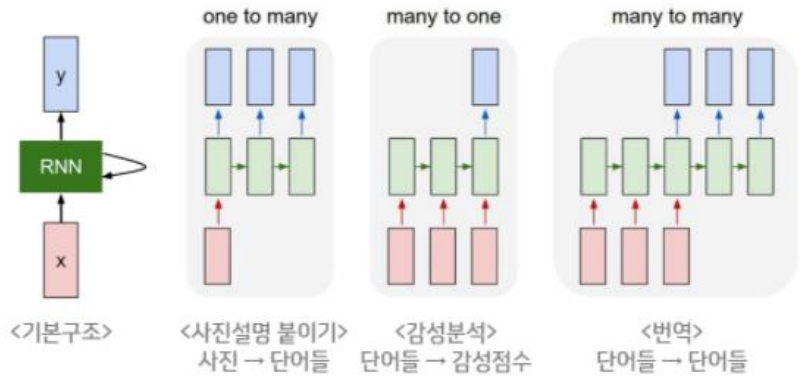
결과

학습에 쓰이지 않은 데이터 넣어보면 조금 아쉽

이유: 번역은 Good morning->좋은 아침입니다, 안녕하세요 등 몇 가지 안 되는 닫힌 정답

뉴스 제목은 언론사마다 천차만별인 열린 정답이라는 것

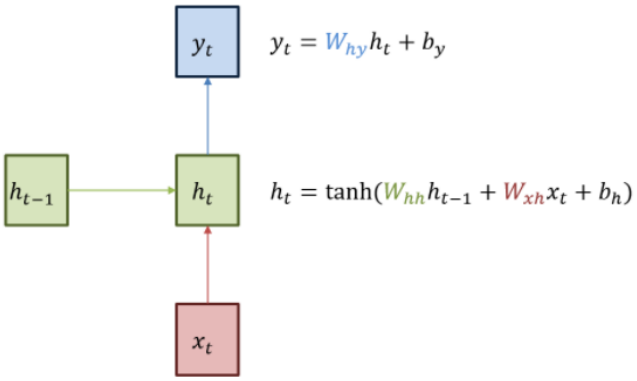
RNN 과 LSTM 을 이해해보자!



RNN: 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망의 한 종류

음성, 문자 등 순차적으로 등장하는 데이터 처리에 적합한 모델

시퀀스 길이에 관계 없이 인풋과 아웃풋을 받아들일 수 있는 네트워크 구조이기 때문에 필요에 따라 다양하고 유연하게 구조를 만들 수 있다는 점



녹색 박스는 hidden state, 빨간 박스는 input x, 파란 박스는 output y

현재 상태의 hidden state ht 는 직전 시점의 hidden state ht-1 을 받아 갱신됨

현재 상태의 아웃풋 yt 는 ht 를 전달받아 갱신되는 구조

Hidden state 의 활성함수(activation function)은 비선형함수인 hyperbolic tanget(tanh)

이유: [밑바닥부터 시작하는 딥러닝] h(x)=cx 를 활성함수로 표현한 3 층 네트워크. Y(x)=h(h(h(x)))

Y(x) c*c*c*x 처럼 세 번의 곱셈을 수행하지만 실은 y(x)=ax 와 똑 같은 식. a=c^3 이라고만 하면 끝, 즉 히든레이어가 없는 네트워크로 표현 가능. 그래서 층을 쌓는 혜택을 얻고 싶다면 활성함수로는 반드시 비선형함수를 사용해야 함.

RNN 의 기본 구조

글자 주어졌을 때 바로 다음 글자 예측하는 character-level-model

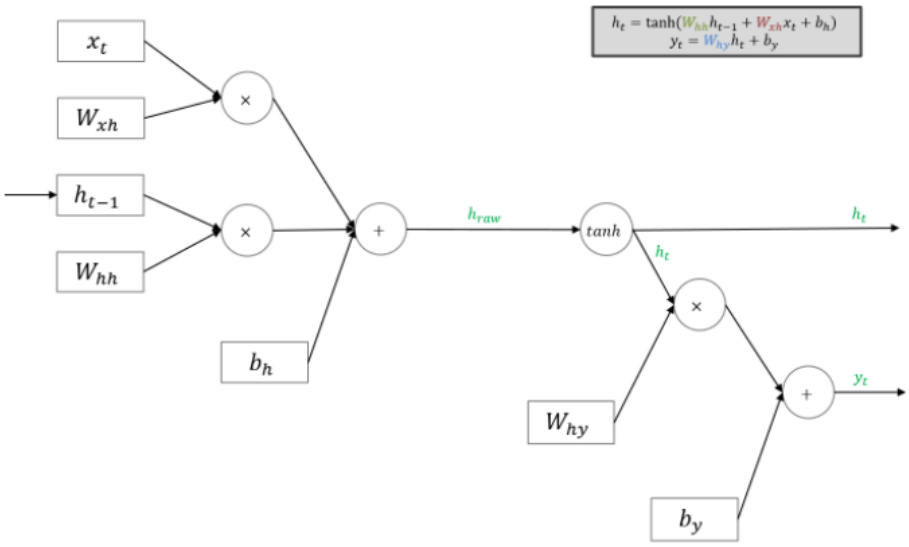
Hello 넣으면 o 를 반환하여 hello 출력하게

모델에 정답을 알려줘야 모델이 parameter 를 적절히 갱신해 나감

Wxh, Whh, Why (1 번째 스텝끼리 같고, 2 번째 스텝끼리 같고…)

모든 시점의 state 에서 이 파아미터는 동일하게 적용됨(shared weights)

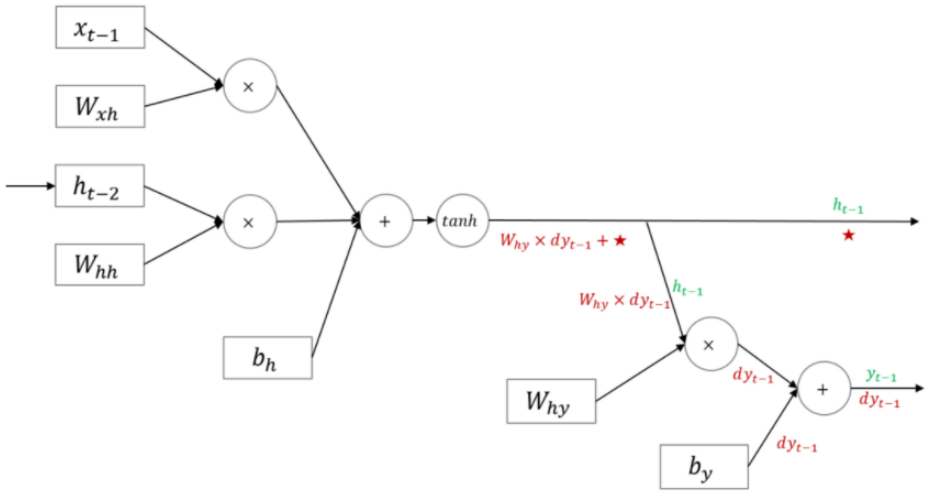
RNN 의 순전파(forward propagation)



RNN 의 역전파(backpropagation)

RNN 은 히든 노드가 순환 구조를 띄는 신경망

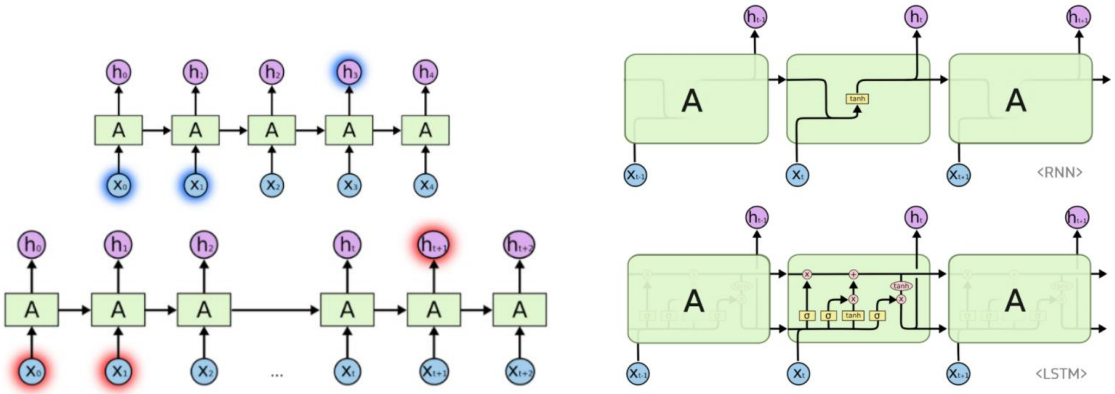
즉, ht 를 만들 때 ht-1 이 반영됨. 바꿔 말하면 아래 그림의 dh t-1 은 t-1 시점의 Loss 에서 흘러들어온 그래디언트인 Why x dy t-1 분 아니라 별표에 해당하는 그래디언트 또한 더해져 동시에 반영된다는 뜻



LSTM 의 기본 구조

RNN 은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 역전파시 그래디언트가 점차 줄어 학습 능력이 크게 저하되는 것으로 알려져 있음

이를 vanishing gradient problem 이라고 함



이를 극복하기 위해 LSTM 이 고안됨. LSTM 은 RNN 의 hidden state 에 cell-state 를 추가한 구조

Cell-state 는 일종의 컨베이어 벨트 역할을 하여 state 가 꽤 오래 경과하더라도 그래디언트가 비교적 전파가 잘 되게 됨.

Forget gate Ft 는 과거 정보를 잊기 위한 게이트

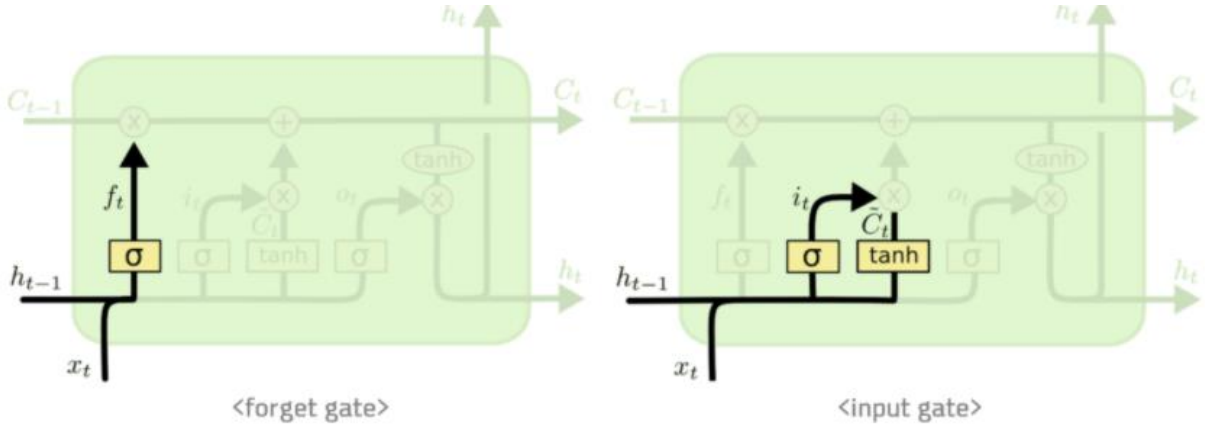
h t-1 과 xt 를 받아 시그모이드를 취해준 값이 바로 forget gate 가 내보내는 값이 됨

시그모이드는 0-1 범위. 0 이면 이전 상태의 정보는 잊고, 1 이라면 이전 상태의 정보를 온전히 기억

Input gate it ⦿ gt 는 현재 정보를 기억하기 위한 게이트

h t-1 과 xt 를 받아 시그모이드를 취하고, 또 같은 입력으로 tanh 를 취한 다음 Hadamard product 연산을 한 값이 바로 input gate 가 내보내는 값

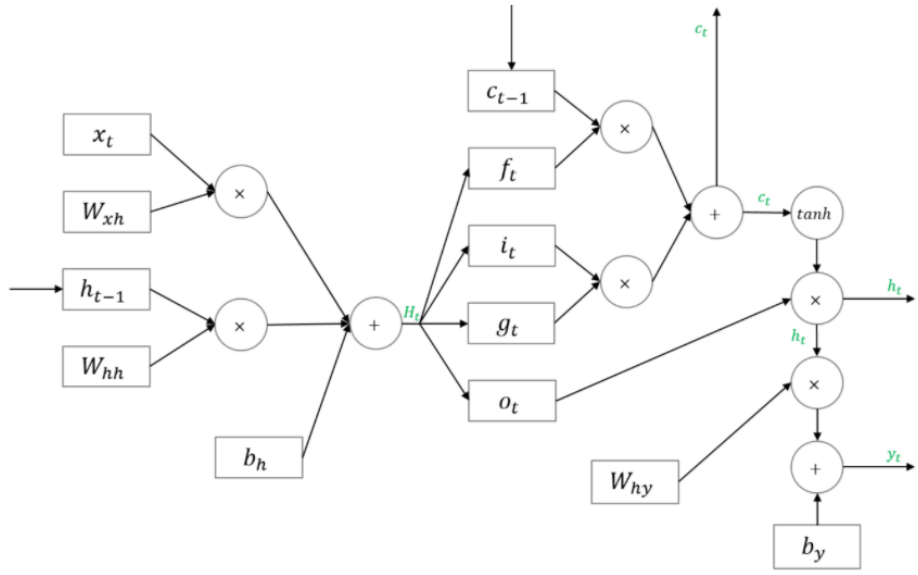
it 의 범위는 0-1, gt 의 범위는 -1 ~ 1 이기 때문에 각각 강도와 방향을 나타냄

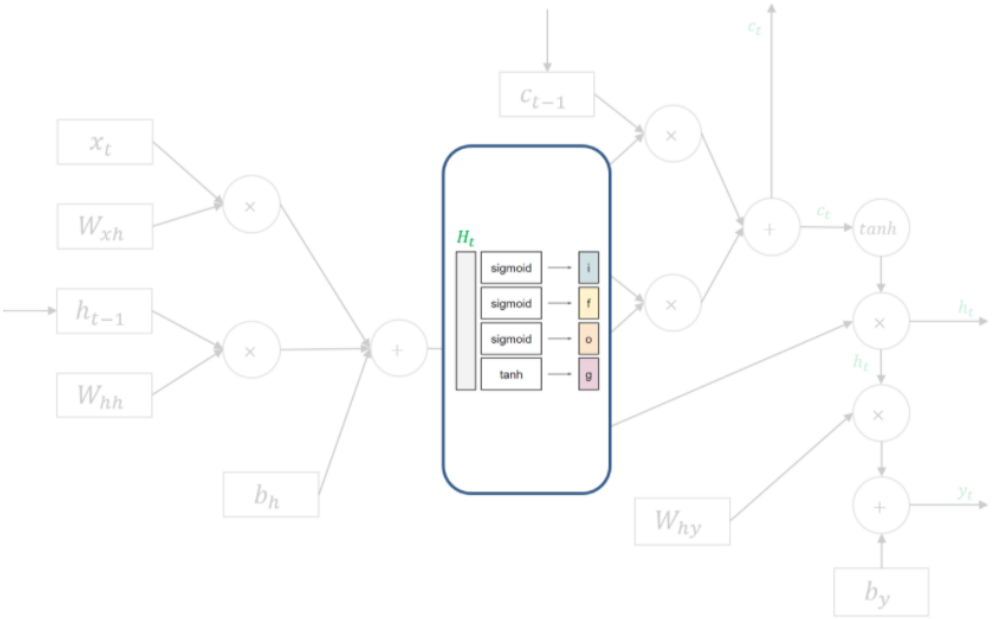


<forget gate> <input gate>

LSTM 의 순전파

Ht(행렬)을 행 기준으로 4 등분하여 I, f, o, j 각각 해당하는 활성함수를 적용하는 방식으로 I, f, o, g 를 계산함





LSTM 의 역전파

d ft, d it, d gt, d ot 를 구하기 까지 backward pass 는 RNN 과 유사

d Ht 를 구하는 과정이 핵심

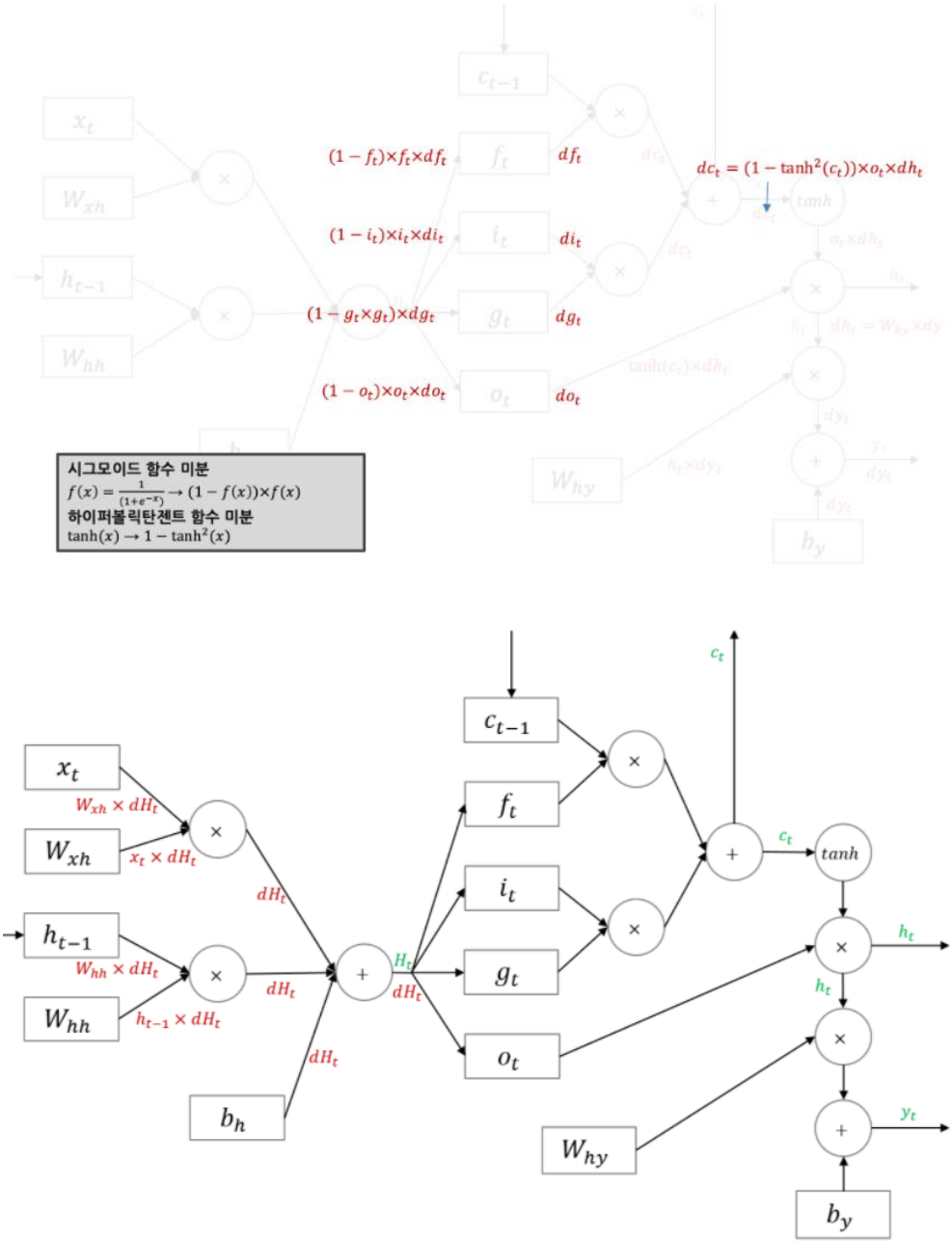
Ht 는 it, ft, ot, gt 로 구성된 행렬. 각각에 해당하는 그래디언트를 merge 하면 dHt 를 만들 수 있음

I, f, o 의 활성함수는 시그모이드이고, g 만 하이퍼볼릭탄젠트임

각각의 활성함수에 대한 로컬 그래디언트를 구해 흘러들어온 그래디언트를 곱해주면 됨

순전파 과정에서 4 등분했던 것처럼, backward pass 에서는 di, df, do, dg 를 다시 합쳐 dHt 를 만듦

이렇게 구한 dHt 는 다시 RNN 과 같은 방식으로 역전파가 되는 구조



LSTM 은 cell state 와 hidden state 가 재귀적으로 구해지는 네트워크

따라서 cell state 의 그래디언트와 hidden sate 의 그래디언트는 직전 시점의 그래디언트 값에 영향을 받음. 이는 RNN 과 마찬가지. 이를 역전파 시 반영해야 함.

Word2Vec 으로 문장 분류하기

휴대폰 리뷰 분류. 휴대폰 리뷰 문장을 배터리, 카메라, 디자인, 사운드 따위의 기능(범주)로 분류하는 문제

Word2Vec 으로 단어 임베딩하기

2013 년 구글 개발. 단어를 벡터로 바꿔주는 방법. 이를 임베딩이라고 함

단어를 벡터화할 때 단어의 문맥적 의미를 보존함. MAN 과 WOMAN 의 거리는 KING 과 QUEEN 의 거리와 유사.

벡터로 바뀐 단어들은 유클리디안 거리나 코사인 유사도 등의 방식으로 거리를 잴 수 있고, 거리가 가까울수록 비슷한 단어라고 해석할 수 있음

5 개 사이트에서 29 만 7906 개의 리뷰 수집

Cohesion tokenizer 로 포스태깅(형태소분석) 작업. KoNLPy 처럼 품사정보까지 반환하지는 않지만 토크나이징을 분석 대상 코퍼스의 출현 빈도를 학습한 결과를 토대로 토큰을 나눠주기 때문에 붐석 품질이 비교적 조음

Word2Vec 에 품사정보까지 넣을 필요도 없기도 함

파이썬 genism 패키지를 활용해 Word2Vec 방법론을 적용

Word2Vec embedding

from gensim.models import Word2Vec

embedding_model = Word2Vec(tokenized_contents, size=100, window = 2, min_count=50, workers=4, iter=100, sg=1)

most_similar 함수로 두 벡터 사이의 코사인 유사도 구함

 # check embedding result

print(embedding_model.most_similar(positive=["디자인"], topn=100))

단어 벡터로 가중치 행렬 만들기

문맥적 정보가 보존된 상태의 단어 벡터 사이의 거리(유사도)를 구하고 이를 가중치 삼아 각 문장별로 스코어를 구함

이렇게 구한 스코어를 바탕으로 각 리뷰 문장을 특정 기능에 할당/분류

거리행렬(distance matrix)로 바꾸기

유클리디안 방법을 사용하여 거리행렬 구함

Ex. 배터리, 발열은 거리가 1, 배터리와 은은 10. 은보다는 발열이 배터리와 유사한 단어. 이것이 우리가 만들 가중치 행렬이 지향하는 핵심 원리.

즉, 특정 쿼리 단어(배터리)와 거리가 가까운 단어는 높은 가중치, 그렇지 않은 단어는 낮은 가중치

$$W_{ij} = exp\left(-\frac{d(x_i, x_j)^2}{2\sigma^2}\right)$$

Exp 앞의 계수를 제외하면 이는 정규분포 식과 같음

들쭉날쭉한 벡터간 거리들을 아주 예쁜 모양의 정규 분포로, 특정 범위까지 스케일링까지 해서 반환한다는 얘기

쿼리 단어인 배터리와 의미가 유사한 발열이라는 단어는 높은 가중치를, 배터리와 별 상관이 없는 조사 '은'은 낮은 가중치를 가짐

가중치 행렬로 문장별 스코어 구하기

단어 가중치를 모두 더한 값이 바로 이 문장이 배터리 기능을 얼마나 중시하는지를 알려주는 지표가 됨

즉, 배터리 스코어는 배터리라는 쿼리 단어와 나머지 단어 사이의 가중치들의 **선형 결합(linear combination)**

그런데 단어 하나하나마다 가중치행렬에서 Lookup 형식으로 가중치들을 가져오는 것은 너무 비효율적. 더 빠르고 편리하게 하기 위해 단어문서행렬(Term-Document Matrix) 등장.

TDM 의 행은 단어, 열은 문서가 됨

만약 행에 해당하는 단어가 쓰이지 않은 리뷰의 요소값은 0, 쓰였다면 1 이 됨

TDM 과 가중치행렬을 내적(inner product)해주면 문장별로 스코어를 한 방에 구할 수 있음

있도록 아래 예시를 준비했는데요. 우선 말뭉치에 등장하는 전체 단어 수가 7개이고, 우리가 조사하고 싶은 기능은 f_1, f_2, f_3 이렇게 세 가지가 있다고 가정해보겠습니다. 두번째와 여섯번째 단어로 구성된 리뷰의 각 기능별 스코어는 다음과 같습니다.

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{ccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{array} \\ \begin{bmatrix} 1.1 & 0.2 & 0.1 & 0.7 & 0.8 & 0.6 & 0.4 \\ 0.8 & 0.5 & 0.8 & 1.0 & 0.8 & 0.2 & 0.1 \\ 0.2 & 0.3 & 0.5 & 0.6 & 0.2 & 0.1 & 0.7 \end{bmatrix} \end{array} \times \begin{array}{c} \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{array} = \begin{bmatrix} 0.8 \\ 0.7 \\ 0.4 \end{bmatrix}$$

전체 내적의 결과는 (쿼리 단어의 수 x 문서의 수) 행렬 형태가 됨

Doc1 의 배터리 스코어가 이 스코어 행렬의 1 행 1 열에 해당하는 요소가 됨

배터리 뿐만 아니라 사운드, 카메라까지 지정했다면 같은 방식으로 Doc1 의 '사운드' 스코어는 2 행 1 열, Doc1 의 '카메라' 스코어는 3 행 1 열이 됨. 마찬가지로 Doc2 의 '배터리', '사운드', '카메라' 스코어는 1 행 2 열, 2 행 2 열, 3 행 2 열이 됨