Intro to NLP

자연어 처리에는 다양한 분야가 있고, 해당 분야마다 주요 학회와 연구분야가 존재한다.

1. Natural Language Processing

학회 : ACL, EMNLP, NAACL

연구분야

가장 낮은 레벨(단어) → 높은 단계(많은 문장)로 이루어짐

- Low-level parsing : Tokenization, stemming
  - Tokenization : 문장에서 각 단어를 토큰이라고 부름. 주어진 문장을 단어단위로 쪼개나감. 이러한 토큰들이 sequence로 이루어짐.
  - Stemming : 어미(-ing, -ed, -지만,-고,-은) 이러한 어미가 달라도 같은 의미를 지님을 컴퓨터가 이해할 수 있어야 함. 의미를 보존하고 어근만을 추출
- Word and phrase level : NER(Named Entity Recognation), POS(Part-Of-Speech) tagging
  - NER : 단일 단어, 혹은 여러 단어를 하나의 고유명사로 인식하는 것 ex) Newyorktimes
  - POS : 문장 내 단어의 품사(주어, 목적어, 부사, 형용사 등)를 인식하는 것
- Sentence level : 감성 분류(Sentiment Analysis), 기계 번역(Machine Translation)
  - 감정 분석 : 해당 문장의 감정을 분석
  - 기계 번역 : 영어→ 한국어 로 번역할 때 그나라의 단어, 문법어순에 맞게 적절히 번역하는 것
- Multi-sentence and paragraph level : Entailment Prediction, 독해기반 질의응답(question answering), 챗봇(dialog systems), 요약(summarization)
  - Entailment Prediction : 논리적 내포 및 모순관계 예측
    - ex) 논리적 참, 거짓을 파악함. 두가지 문장을 예시로 보면, 어제 존이 결혼을 했다. 어제 최소한명이 결혼을 했다. 첫문장이 참인 경우 두번째 문장은 자동으

로 참. 이러한 관계를 예측하는 것.

- question answering : 검색시 해당 키워드가 들어간 사이트만 보여주는 것이 아니라, 검색 맥락에 알맞은 답변을 가장 위에 보여주는 것
  - ex) where did napoleon die : 키워드 napoleon, die등이 들어간 사이트를 보여주는 것이 아니라 해당 키워드들을 바탕으로 검색결과를 모으고 이를 바탕으로 정답을 정확히 알아내서(나폴레옹이 죽은 장소) 검색결과로 보여줌.
- dialog systems : 챗봇과 같은 대화 처리 기술
- summarization : 한줄 요약등의 요약 기술

2. Data Mining

빅데이터 분석과 관련. 데이터를 바탕으로 사회과학적 패턴, 경향성을 분석한다

학회 : KDD, The WebConf(前 WWW), WSDM, CIKM, ICWSM

연구분야

- Extract useful information and insights from text and document data
- 문서 군집화(Document clustering) ex) 토픽 모델링
- Highly related to computational social science : 통계적으로 사회과학적 인사이트 산출

3. Information retrieval

정보 검색 기술. 검색 시스템이 고도화 되어 이미 기술발전이 상대적으로 많이 진전된 상태.

학회 : SIGIR, WSDM, CIKM, Recsys

연구 분야

- Highly related to computational social science
- 추천 시스템 : 영상이나 광고등 개인화된 추천 시스템

NLP의 트렌드

Word Embedding

기존 머신러닝과 딥러닝 기술로 자연어 처리 문제를 해결하기 위해서는 주어진 텍스트 데이터를 숫자로 변환하는 '워드 임베딩(**Word Embedding**)' 과정을 거치게 됨

> ex) I love this movie. ⇒ 각 단어가 워드벡터로 저장됨. 워드벡터들이 특정 순서로 저장된 것이 한 문장이 됨. 자연어처리 모델은 이러한 sequence가 다르면 다른 문장임을 인식할 수 있어야함

Model

RNN ⇒ LSTM, GRU ⇒ Transformer

2017년에는 구글에서 발표한 'Attention is all YOU need' 라는 제목의 논문이 나오면서 '셀프 어텐션(Self-Attention)' 구조를 가진 '트랜스포머(**Transformer**) 모델'이 각광받기 시작. 대부분 NLP 모델은 transformer를 기반으로 함.

기계 번역을 위해 처음 제안된 모델임.

이전의 기계 번역 - 언어간 문장 구조, 어순을 규칙을 기반으로 번역 수행. 단어 번역 후 어순 변형. 너무 많은 예외상황과 많은 패턴에 대응하기 어렵

이후 - 영어-번역된한글문장 쌍을 학습하여 언어학적 규칙에 기반한 것이 아닌 sequence 기반의 RNN모델을 사용하여 번역 성능이 월등히 향상됨. 파파고, google translate는 이러한 기술을 활용한 서비스.

Transformer 전에는 customized models for diff NLP tasks. 그러나 현재는 self-attention 구조를 여러번 쌓아 모델 크기를 키우고 , 대규모 텍스트 데이터를 통해 모델을 학습하여 적용하는 방식으로 변하였다.

Self-supervised Learning

모델이 사전에 정의된 임의의 태스크를 수행하면서 데이터 전반에 대한 이해도를 높일 수 있도록 설계된 태스크를 의미한다. 모델은 pretext task를 수행하여 데이터에 대해 충분히 다룰 수 있게 되고, downstream task에 대해 fine tuning되어 효율적으로 성능을 확보할 수 있다.

ex) gpt, BERT - BERT에서는 일반적인 문장에서 하나의 단어를 지운다음, 해당 빈칸에 올 단어를 예측하는 것과, 그다음에 올 문장을 예측하는 방법으로 pre-trained 모델 학습을 진

행한다.

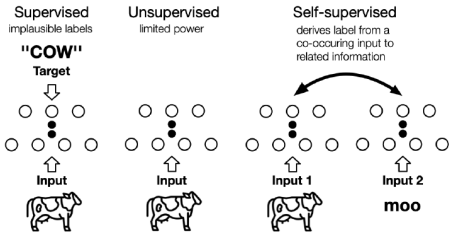
최근에는 많은 데이터를 학습시키기 위한 GPU 자원이 많이 필요함.

추가 참고 자료!

Self-Supervised Learning

처음 기계학습을 공부할 때를 생각해보면 세상엔 두가지 종류로 나눌 수 있다고 배운다. 지도학습 (supervised learning)비지도 학습 (unsupervised learning)여기서 지도학습은 레이블이 존재하는 특

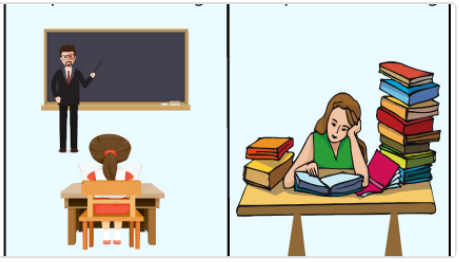
https://velog.io/@stapers/Self-Supervised-Learning



Self-supervised learning (자기지도 학습) 이란?

요즘 Deep learning에서 주목받고 있는 학습법인 Self-supervised learning을 소개할까 합니다. 이번 글에서는 Self-supervised learning에 대한 간단한 소개 이후 왜 이 학습법이 주목받는지, Self-

https://lifeisenjoyable.tistory.com/15



+자기지도학습vs 비지도 학습

비지도학습은 데이터 내부의 패턴을 잡아내어 clustering, community dection, anomaly dection을 수행해내는데 반해, 자기지도학습은 단순히 데이터를 복원하는 과정을 통해 학습이 이루어진다.

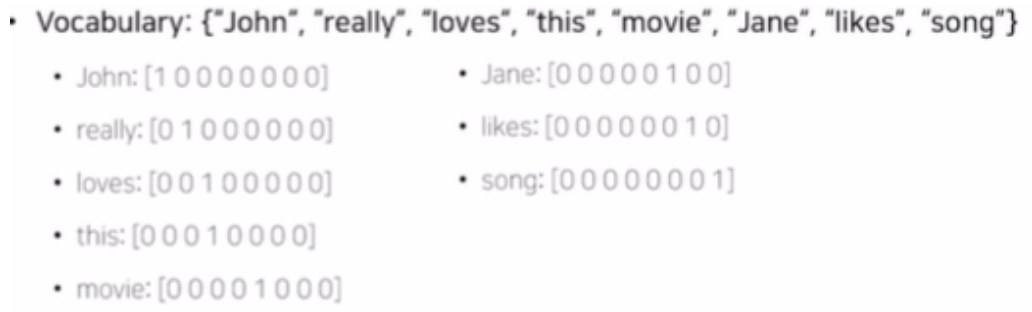
Naive Bayes Classifier

딥러닝 기술이 적용되기 이전에 자연어 처리 분야에서 많이 활용되던 기법인 'Bag-Of-Words' 표현형과 이를 활용한 대표적인 문서 분류 기법인 '나이브 베이즈 분류기(Naive Bayes Classifier)'에 대해서 알아보자.

Bag-Of-Words (단어 가방 모형)

단어들의 순서는 전혀 고려하지 않고, 단어들의 **출현 빈도(frequency)**에만 집중하는 텍스트 데이터의 수치화 표현 방법

- 단어를 벡터로 표현하기 위해서 주어진 문장에 쓰인 단어들을 사전(Vocabulary) 형태로 저장(중복 X)
- 저장된 단어들은 각각 유니크한 카테고리 변수(Categorical variable)이므로, 원-핫 인코딩(**One-hot Encoding**)를 이용해 벡터로 표현.



  - 모든 단어들의 쌍은 유클리드 거리가 루트2 이고 코사인유사도는 0. 즉, 단어의 의미에 관계없이 모든 단어가 동일한 관계를 가지게 됨.
- 이를 통해 주어진 문장을 원-핫 벡터의 합, 즉 숫자로 표현할 수 있게 됨

이제 이러한 단어들의 합으로 나타낸 문서를 클래스 중에 하나로 분류하는 방법 ⇒ Naive bayes classifier

Naive Bayes Classifier for Document Classification

- C: 클래스의 개수
- P(c|d) : 특정 문서 d가 클래스 c에 속할 조건부 확률

*P(d)는 상수 값이므로 무시하여 최종적으로 아래의 수식이 도출됨

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}}\ P(c|d)$$ MAP is "maximum a posteriori" = most likely class

$$= \underset{c \in C}{\operatorname{argmax}}\ \frac{P(d|c)P(c)}{P(d)}$$ Bayes Rule

$$= \underset{c \in C}{\operatorname{argmax}}\ P(d|c)P(c)$$ Dropping the denominator

특정 클래스 c가 고정되었을 때, 문서 d가 나타날 확률은 아래와 같다

$$P(d|c)\mathrm{P}(c) = P(w_1, w_2, \ldots, w_n|c)\mathrm{P}(c) \rightarrow \mathrm{P}(c)\prod_{w_i \in W} P(\mathrm{w}_i|c)$$

- w는 일련의 단어들, c는 클래스 , d는 문서를 의미
- 각 단어가 나타날 확률이 독립이라고 가정한다면, 오른쪽 처럼 확률의 곱으로 나타낼 수 있게 됨

예시를 통해 이해해보자.

학습 데이터로 주어진 Traing 1~4 번 문장을 통해 우리는 Test data(5번 문장)을 CV, NLP 두 클래스 중에 한 곳으로 분류하려 한다. 어떻게 해야 할까?

Data	Doc(d)	Document (words, w)	Class (c)
Training	1	Image recognition used convolutional neural networks	CV
	2	Transformers can be used for image classification task	CV
	3	Language modeling uses transformer	NLP
	4	Document classification task is language task	NLP
Test	5	Classification task uses transformer	?

5번 문장에 있는 각 단어들이 1~4번 문장에 몇 번 등장했는지를 조건부 확률로 계산하면 쉽게 알 수 있다. 하나의 클래스로 고정되었을 때, 각각의 단어(classification, task, uses, transformer)가 나타날 조건부 확률을 추정하면, 아래와 같다.

For each word w_i, we can calculate conditional probability for class c

- $P(w_k|c_i) = \frac{n_k}{n}$, where n_k is occurrences of w_k in documents of topic c_i

Word	Prob	Word	Prob
$P(w_{\text{'classification'}}\|c_{CV})$	$\frac{1}{14}$	$P(w_{\text{'classification'}}\|c_{NLP})$	$\frac{1}{10}$
$P(w_{\text{'task'}}\|c_{CV})$	$\frac{1}{14}$	$P(w_{\text{'task'}}\|c_{NLP})$	$\frac{2}{10}$
$P(w_{\text{'uses'}}\|c_{CV})$	$\frac{1}{14}$	$P(w_{\text{'uses'}}\|c_{NLP})$	$\frac{1}{10}$
$P(w_{\text{'transformer'}}\|c_{CV})$	$\frac{1}{14}$	$P(w_{\text{'transformer'}}\|c_{NLP})$	$\frac{1}{10}$

첫번째 조건부 확률로 연산방식을 설명해보겠다.

- C_cv : CV라고 분류된 문장에서 전체 단어의 개수 ⇒ 14
- w_classification : classfication이라는 단어가 CV라고 분류된 문장에서 등장한 수 ⇒ 1

따라서 1/14 이다.

이러한 연산을 바탕으로, 위에서 배운 확률 곱 공식(단어들은 서로 독립이라는 가정이 있으므로)을 통해 CV와 NLP에 각각 속할 확률을 구하면 아래와 같다.

- $P(c_{CV}|d_5) = P(c_{CV}) \prod_{w \in W} P(w|c_{CV}) = \frac{1}{2} \times \frac{1}{14} \times \frac{1}{14} \times \frac{1}{14} \times \frac{1}{14}$
- $P(c_{NLP}|d_5) = P(c_{NLP}) \prod_{w \in W} P(w|c_{NLP}) = \frac{1}{2} \times \frac{1}{10} \times \frac{2}{10} \times \frac{1}{10} \times \frac{1}{10}$

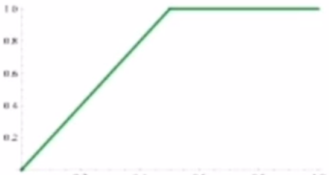
다만 이 방식의 맹점은 다른 단어들이 분류하고자 하는 문장에 많이 등장했을지라도, Training data 에서 1번이라도 등장하지 않았다면 모든 단어들의 확률 곱으로 인해 0이 된다는 문제가 있다.

이를 완화하기 위해 regularization 기법들이 활용되기도 한다.

Word Embedding _ Glove

Glove : Global Vectors for Word Representation

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2 \quad f \sim$$



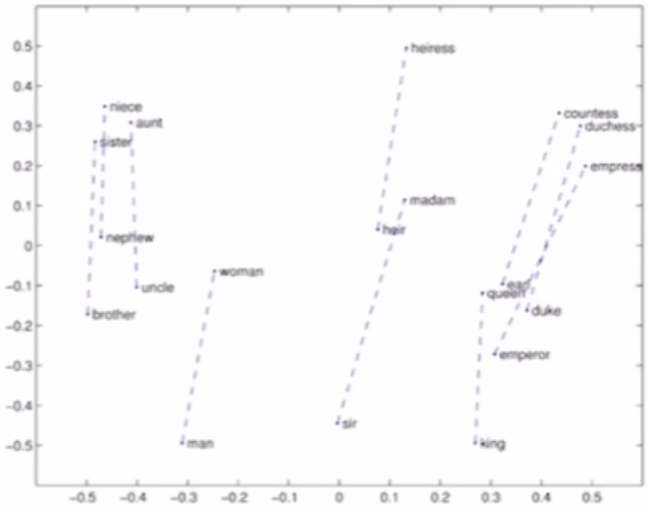
사전에 미리 각 단어들의 동시 등장 빈도 수를 계산(logP_ij)하며, 단어 간의 내적값과 사전에 계산된 값의 차이를 줄여가는 형태로 학습합니다. 특정 두 단어 세트가 자주 등장할 경우, 자동으로 여러번에 거쳐 학습됨으로써 내적값이 등장 횟수에 비례하여 커진다. Glove에서는 동시 등장 횟수를 미리 계산하고(P_ij), 이 값을 ground truth 값으로 사용한다.

- u_i : 입력 단어의 embedding vector
- v_j : 출력 단어의 embedding vector
- P_ij : i와 j 가 한 윈도우에서 동시에 나타날 확률

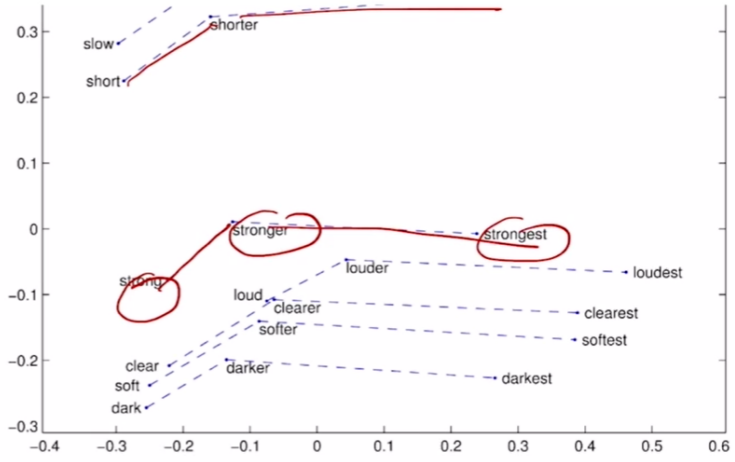
Word2Vec 과의 차이점

- Word2Vec는 모든 연산을 반복하지만, Glove는 사전에 계산된 Ground Truth를 사용해 반복계산을 줄인다.
- 따라서 Word2Vec보다 더 빠르게 동작하며, 더 적은 데이터에서도 잘 동작한다.

예시



- Glove 모델로 학습한 후, PCA 방식으로 2차원 형태로 나타낸 그림. 성별의 차이가 대체로 일정한방향과 크기를 가진 벡터로 나타남을 알 수 있음.



- 단어도 마찬가지로 잘 분류함을 알 수 있음.

사전 학습된 Glove 모델

- 사전에 이미 대규모 데이터로 학습된 모델이 오픈소스로 공개되어 있습니다. 해당 모델은 위키피디아 데이터를 기반으로 하여 6B token만큼 학습 되었으며, 중복 제거 시에도 단어의 개수가 무려 40만개(400k)에 달합니다.
- 학습된 모델을 나타낼 때 뒤에 붙는 "uncased"는 대문자 소문자를 구분하지 않는다는 의미이며, 반대로 "cased"는 대소문자를 구분한다는 의미입니다. 예를 들어 Cat과 cat이 uncased에서는 같은 토큰으로 취급되지만, cased에서는 다른 토큰으로 취급됩니다.

- Glove 깃헙 주소 : https://github.com/stanfordnlp/GloVe (기존에 학습된 워드 임베딩도 다운로드 받아 사용할 수 있다 참고!)

Character-level Language Model

다음에 올 단어/문자를 예측하는 언어모델에 대해서 학습하고, 그 중에서도 RNN을 활용한 언어모델의 원리를 살펴보자.

Character-level Language Model

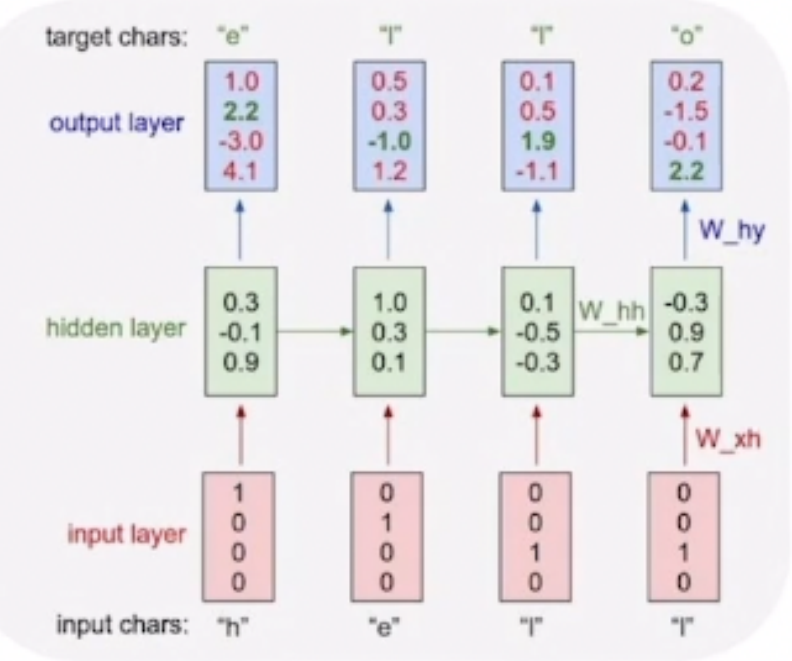
언어 모델이란 이전에 등장한 문자열을 기반으로 다음 단어를 예측하는 것. 그 중에서도 character-level Language Model은 **문자 단위**로 다음에 올 문자를 예측하는 언어 모델.

TRAIN

- 예를 들어, 그림과 같이 맨 처음에 "h"가 주어지면 "e"를 예측하고, "e"가 주어지면 "l"을 예측하고, "l"이 주어지면 다음 "l"을 예측하도록 hidden state가 학습돼야 함.
  1. 사전 구축

     Vocabulary : [h, e, l, o]

     각 단어는 총 단어의 개수만큼의 차원을 가지는 one-hot vector로 나타낸다.
  2. 첫번째 time step인 h에서는 다음에 h t-1의 값과 현재 입력값의 선형 결합후 비선형 활성함수를 통해 h_t를 만들게 됨.
  3. h1은 이전의 h값이 없으므로 우리가 h0를 모두 0으로 줌.
  4. 이 output vector를 multi-class classification을 수행하려면 softmax layer를 통과시켜 확률값으로 변환한다.



- 이때 각 타임스텝별로 output layer를 통해 차원이 4(유니크한 문자의 개수) 벡터를 출력해주는데 이를 logit이라고 부르며, softmax layer를 통과시키면 원-핫 벡터 형태의 출력값이 나오게 된다.
- 위의 그림을 보면, 아직 h를 입력으로 넣었을 때, 다음문자로 o를 예측하는 결과가 나온다. 이를 e값의 확률이 높아지도록 계속 학습한다.

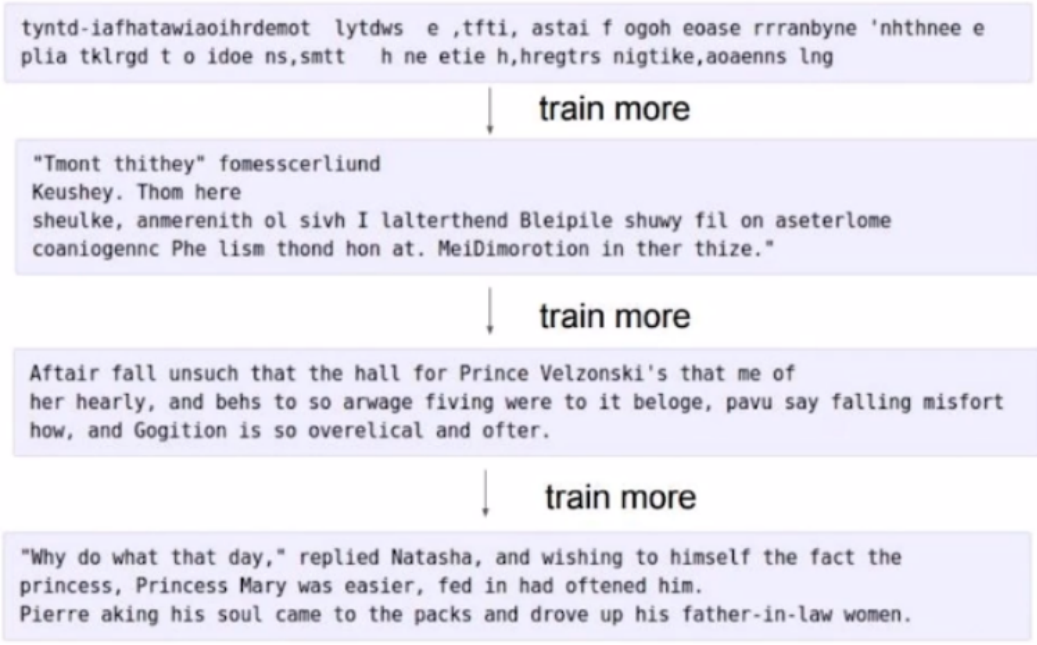
TEST
학습이 완료되고 추론을 할 때를 보자

예측값을 얻어내면, 이 값을 다음 time step의 입력값으로 적용하게 된다.

다양한 언어모델의 예시

문단에 대한 예측 모델을 보자.

공백이나 콤마, 등 특수문자도 하나의 vocabulary 로 기록되어, 긴 문장을 가진 문단이여도 하나의 벡터로 나타낼 수 있게 된다..



1. 셰익스피어의 글을 활용해 더 많은 학습이 진행될수록 완전한 형태의 문장을 출력하는 것을 확인할 수 있습니다.
2. 전 타임스텝까지의 주식값을 활용하여 다음날 주식값을 예측하는 형태로도 수행할 수 있습니다.
3. 그 외에도 인물 별 대사, Latex로 쓰여진 논문, C 프로그래밍 언어와 같은 경우에도 다양한 언어적 특성을 학습하여 텍스트를 생성할 수도 있습니다.

Backpropagation through time and Long-Term-Dependency

지난 시간까지 배웠던 RNN은 자연어 처리에서 널리 사용되던 모델이었습니다. 하지만 치명적인 단점이 있었고, 이를 보완한 모델들이 나오기 시작했습니다. LSTM, GRU 그리고 Transformer 까지의 모델들이 나오게 된 이유에 대해 알아봅시다.

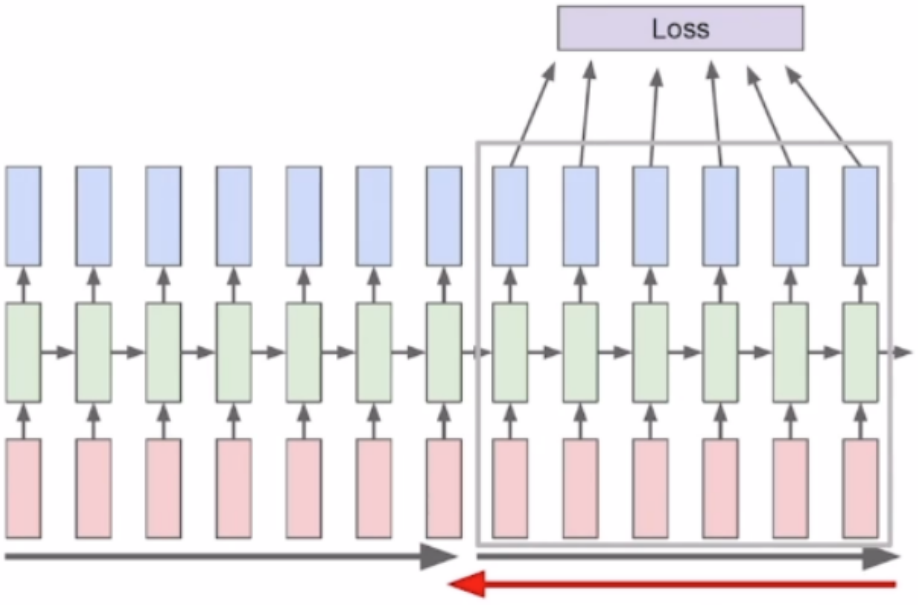
RNN 모델이 학습하는 방법 : Truncation , BPTT

Truncation

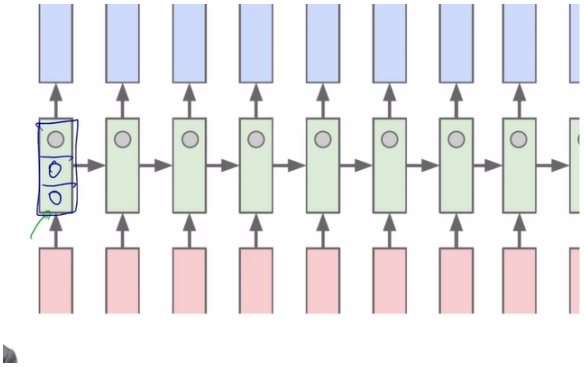
각 time step마다 주어진 문자가 있을 것이고, 각 time step의 hidden state vector를 통해 output을 각자 가지고, 이 예측값과 ground truth 값과의 비교를 통한 loss function을 통해 학습이 진행된다.

Truncation이란, 제한된 리소스(메모리) 내에서 모든 시퀀스를 학습할 수 없기때문에 아래 사진과 같이 잘라서 학습에 사용하는 것

⇒ 문장이 매우 길면 한정된 GPU 리소스로 한번에 계산할 수 없음 → Truncation 을 통해 제한된 길이의 sequence로 학습.

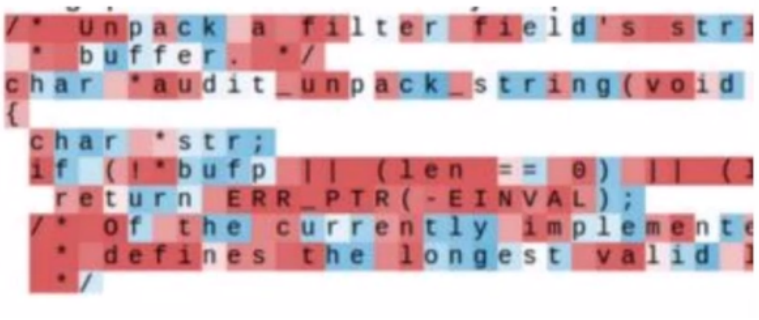


- 딥러닝 모델은 forward propagation을 통해 계산된 W를, backward propagation을 지나면서 W를 미분한 값인 gradient를 통해 학습하게 됩니다.



ht의 역할

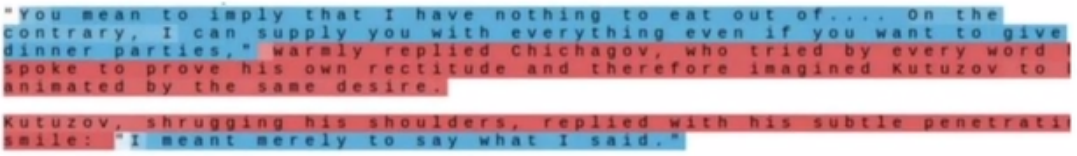
이전의 데이터에 대한 정보는 ht(hidden state vector) 에 담겨져야 함. 만약 hidden state vector가 3차원이라면, 어느 위치의 값이 유의미한지 분석하여 역추적할 수 있다.



특정한 hidden state vector의 차원을 고정하고, 크기가 어떻게 변화하는지를 나타낸 것. 크기가 음수로 크면 파란색, 양수로 크면 빨간색으로 나타냄.

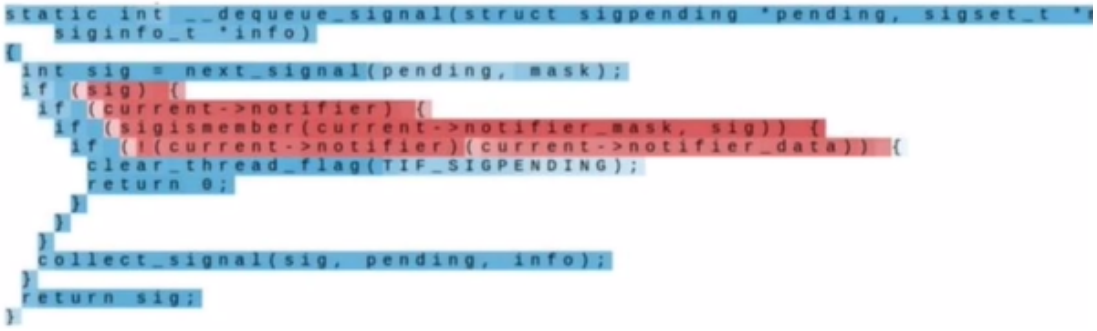
- BPTT란, Backpropagation through time의 줄임말로 RNN에서 타임스텝마다 계산된 weight를 backward propagation을 통해 학습하는 방식을 의미합니다.

1. Quote Detection Cell



- 흥미로운 패턴. 따옴표를 기준으로 빨간색, 파란색이 나뉨. 이는 hidden state vector가 어떤 역할을 하고 있는지 알려줌.⇒ 따옴표의 열고 닫음의 상태를 기억하는 것이 ht의 역할임을 알수 있다.

2. If statement cell



- if 라는 단어가 나왔을 때의 조건문을 기억하는 셀.
- 아래 그림은 특정 hidden state를 시각화한 그림으로, BPTT를 반복하게되면 다음과 같이 빨강(긍정)과 파랑(부정)으로 해당 time step에서의 중요한 부분을 잘 학습하는 것을 볼 수 있습니다.

Vanishing/Exploding Gradient Problem in RNN

하지만, 기존의 Vanilla RNN으로는 위와 같이 학습될 수 없습니다. 그 이유는 gradient가 전파되면서 소실되거나 증폭되면서 멀리까지 학습정보를 잘 전달하지 못하는 Long-Term-Dependency가 발생하기 때문입니다.

왜냐, ht연산에서는 동일한 matrix를 매 time step마다 곱하게 되는데, Whh가 반복적으로 곱해지므로(등비수열과 같은) 기울기가 기하급수적으로 커지거나 값이 작아짐.

따라서 유의미한 값을 먼 time step까지 전달할 수 없게 됨.

그렇다면 위의 hidden state 시각화는 어떻게 잘 이뤄진걸까요? 바로 RNN의 Long-Term-Dependency 문제를 보완한 LSTM 모델로 학습한 결과이기 때문입니다. LSTM에 대한 자세한 내용은 다음 강의에서 배우도록 하겠습니다.

Toy Example

- $h_t = \tanh(w_{xh}x_t + w_{hh}h_{t-1} + b), t = 1,2,3$
- For $w_{hh} = 3, w_{xh} = 2, b = 1$

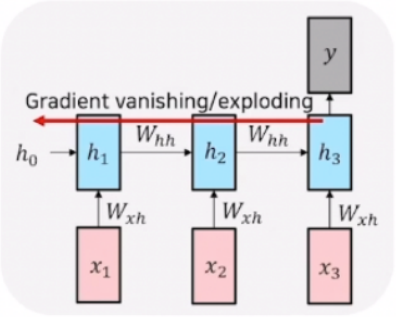
$h_3 = \tanh(2x_3 + 3h_2 + 1)$

$h_2 = \tanh(2x_2 + 3h_1 + 1))$

$h_1 = \tanh(2x_1 + 3h_0 + 1))$

...

$h_3 = \tanh(2x_3 + 3\tanh(2x_2 + 3h_1 + 1) + 1)$

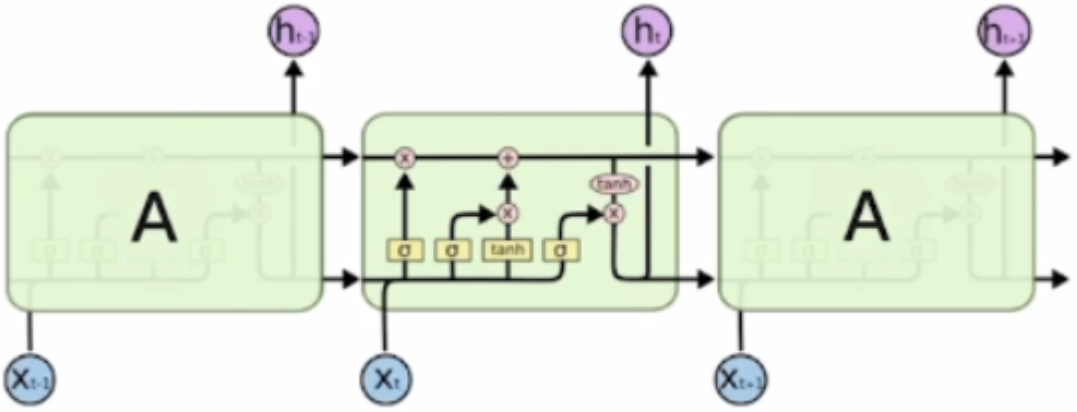


- 위 그림을 통해 time step이 3인 RNN의 BPTT과정을 살펴보겠습니다.
- 3번째 time step의 hidden state 인 *h*3 를 *h*1 으로 표현하면 맨 아랫줄의 식과 같이 표현되는 것을 확인할 수 있습니다. BPTT를 통해 h1에 대한 gradient를 계산해주면, t*anh* 로 감싸진 괄호안의 값들 중에 3 값이 속미분되어 나오게 됩니다. (자세한 과정은 chain rule에 대해서 학습이 필요합니다.) 13:00~
- 위 예시는 time step이 3이므로 3이 2번 곱해지지만, 만약 길이가 더 길어진다면 미분값은 기하급수적으로 커질 것이고, 만약 속미분되어 나오는 W의 값이 1보다 작다면 미분값은 기하급수적으로 작아질 것입니다.
- 이 계산과정을 통해 Gradient Vanishing/Exploding 문제가 발생하고 이 문제가 Long-Term-Dependency를 일으킴.0.

LSTM

길이가 길어질수록 학습이 잘 이뤄지지 않는다는 문제점이 있는데, 이를 해결하기 위해 LSTM(Long Short-Term Memory)라는 모델이 등장.

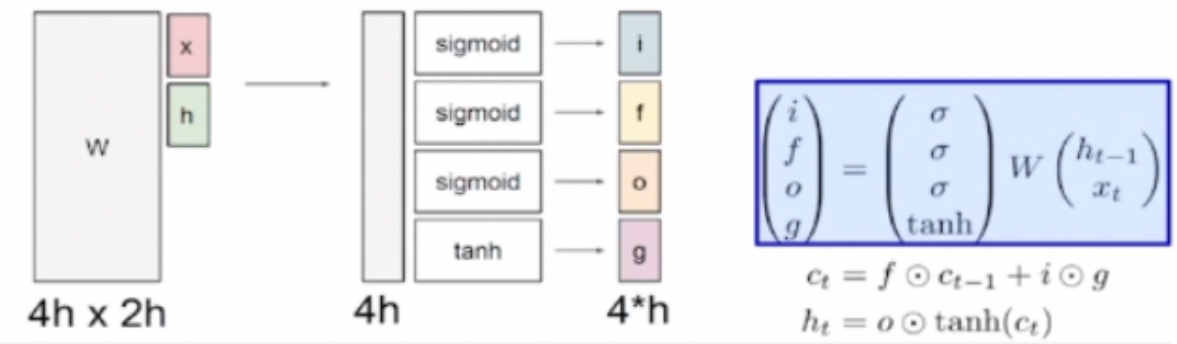
LSTM : Long Short-Term Memory



The repeating module in an LSTM contains four interacting layers.

LSTM의 중심 아이디어는 단기 기억으로 저장하여 이걸 때에 따라 꺼내 사용함으로 더 오래 기억할 수 있도록 개선하는 것입니다. 다시 말해 Cell state에는 핵심 정보들을 모두 담아두고, 필요할 때마다 Hidden state를 가공해 time step에 필요한 정보만 노출하는 형태로 정보가 전파됩니다.

LSTM의 gate



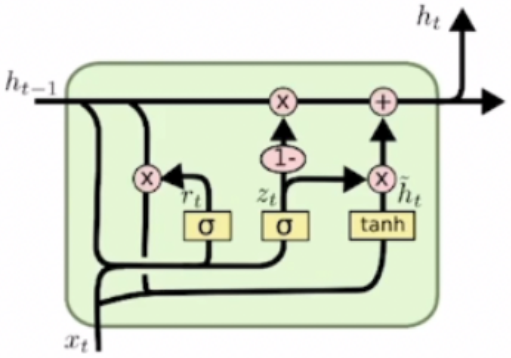
- 위 그림을 살펴보면 input으로 x_t, h_t-1 이 들어오게 되고 이를 W에 곱해준 후 각각 sigmoid or tanh로 연산해줍니다. 게이트별로 설명은 아래와 같습니다. Ifog 로 불리는 게이트들은 그림에서 순서대로 나타납니다.
- I : Input gate로 불리며, cell 에 쓸 지말지를 결정하는 게이트입니다. 즉, 들어오는 input에 대해서 마지막에 sigmoid를 거쳐 0-1 사이 값으로 표현해줍니다. 이 값은 cell state와 hidden state 두 갈래로 흐르게 됩니다.
  - 표현식 : sigmoid(W(xt, ht-1)
- f : Forget gate 로 불리며, 정보를 어느정도로 지울지를 0~1사이의 값으로 나타냅니다.
  - 표현식 : sigmoid(W(xt, ht-1)
- o : Output gate로 불리며, Cell 정보를 어느정도 hidden state에서 사용해야할 지를 0~1사이 값으로 나타냅니다.
  - 표현식 : sigmoid(W(xt, ht-1)
- g : Gate gate로 불리며, 어느정도로 Cell state에 반영해야할 지를 -1 ~ 1 사이의 값으로 나타냅니다.
  - 표현식 : tanh(W(xt, ht-1)

LSTM이 RNN과 다른점

- LSTM의 특징은 각 time step마다 필요한 정보를 단기 기억으로 hidden state에 저장하여 관리되도록 학습하는 것입니다.
- 오차역전파(backpropagation) 진행시 가중치(W)를 계속해서 곱해주는 연산이 아니라, forget gate를 거친 값에 대해 필요로하는 정보를 덧셈을 통해 연산하여 그레디언트 소실/증폭 문제를 방지합니다.

GRU : Gated Recurrent Unit

- $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$
- $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$
- $\tilde{h_t} = \tanh(W \cdot [r_t \cdot h_{t-1}, x_t])$
- $h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h_t}$
- c.f) $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C_t}$
  in LSTM



- LSTM과 전체적인 동작원리는 유사하지만, Cell state, Hidden state를 일원화하여 경량화한 모델입니다.
- GRU에서 사용되는 h_t-1*ht*−1 은 LSTM에서의 c_t*ct* 와 비슷한 역할을 합니다.
- forget gate 대신 (1-input gate)를 사용하여 h_t*ht* 를 구할때 가중평균의 형태로 계산하게 됩니다.
- 계산량과 메모리 요구량을 LSTM에 비해 줄여준 모델이면서 동시에 성능면에서도 LSTM과 비슷하거나 더 좋은 성능을 내는 모델입니다.

RNN , LSTM , GRU 요약

- RNN은 들어오는 입력값에 대해서, 많은 유연성을 가지고 학습되는 딥러닝 모델입니다.
- RNN에서는 그레디언트 소실/증폭 문제가 있어 실제로 많이 사용되지는 않지만, RNN 계열의 LSTM, GRU 모델은 현재도 많이 사용되고 있습니다.
- LSTM과 GRU 모델은 RNN과 달리 가중치를 곱셈이 아닌 덧셈을 통한 그레디언트 복사로 그레디언트 소실/증폭 문제를 해결했습니다.