16주차 정리

자연어 처리의 시작

1. 자연어 처리 활용 분야와 트렌드

자연어 처리 활용 분야

(1) Natural language processing

- 딥러닝 기술의 발전을 선도하는 핵심 분야
- 분야

1. Low-level parsing

   → 가장 low level의 task

   - **Tokenization:** 문장을 이해하기 위해 각 단어 단위로 쪼개나가는 과정 →토큰 만들기
   - **stemming**: 단어의 어미가 다양하게 변해도 동일한 단어의 어근을 추출

2. Word and phrase level

   - **NER(Named Entity Recognation)**: 고유 명사를 인식
   - **POS(Part-Of-Speech) tagging**: 단어들이 문장 내에서 어떤 품사나 성분인지 알아내기(주어, 목적어, 보어, 형용사, 무엇을 꾸미나)

3. Sentence level

   - **Sentiment Analysis(감성분류)**: 문장의 긍정, 부정 구분
   - **Machine Translation(기계 번역)**: 적절한 단어, 문법으로 번역 번역

4. Multi-sentence and paragraph level

   - **Entailment Prediction**(논리적 내포 및 모순관계 예측): 두 문장간의 논리적인 내포, 모순관계 예측
   - **question answering**(독해기반 질의응답): 질문 키워드가 포함된 문서 검색 → 독해를 통해 질문에 대한 정답을 정확하게 알아내서 제시해주는 과정
   - **dialog systems**(챗봇): 대화를 수행할 수 있는 기술
   - **summarization**(요약): 주어진 문서를 자동으로 요약하는 task

(2) Text mining (텍스트 마이닝)

- 빅데이터 분석과 관련된 분야
- 분야

1. 텍스트 및 문서 데이터→유용한 정보 추출
2. **Document clustering(문서 군집화)**:비슷한 의미를 가지는 키워드 grouping→ 분석
   ex) 토픽 모델링
3. **Highly related to computational social science** : 사회과학적 인사이트 산출

(3) Information retrieval (정보 검색)

- 검색 기술을 연구하는 분야
- 분야

1. **추천 시스템:** (수동으로 검색 안 해도 자동으로 해주는 것) ex) 광고, 음악

자연어 처리 분야의 트렌드

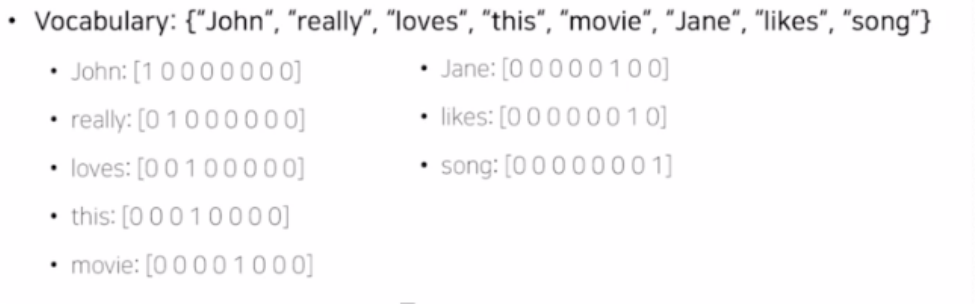
- 자연어 처리 분야는 컴퓨터 비전, 영상처리 분야와 더불어 인공지능이 가장 활발히 적용
- 주어진 텍스트 데이터를 숫자로 변환하는 워드 임베딩(Word Embedding) 과정을 거치게 됨
  - 워드 임베딩: 텍스트를 단어 단위로 분리하고 특정한 차원으로 이루어진 벡터로 표현하는 과정
- 순서 정보 반영 위한 모델
  - RNN 계열 모델 중 단점을 보완한 LSTM, LSTEM을 단순화하여 계산 속도를 빠르게 한 GRU 모델이 나와 사용되었음
- 새로운 발전- 룰 기반 X
  2017년에는 구글 'Attention is all YOU need' 의 '셀프 어텐션(Self-Attention)' 구조를 가진 '트랜스포머(Transformer) 모델'이 각광받기 시작
  - 트랜스포머 모델→현재는 영상처리/신약개발/시계열 예측 등에서도 다양하게 사용되고 있음
- 자가지도 학습
  - 자기지도 학습: 입력 문장이 주어져 있을 때 입력 중 일부단어를 가리고 맞추게 하는 문맥을 보고 맞추는 것.
  - BERT, GPT2/GPT3 와 같은 모델

- 대규모의 데이터, GPU 리소스를 필요

2. 기존의 자연어 처리 기법

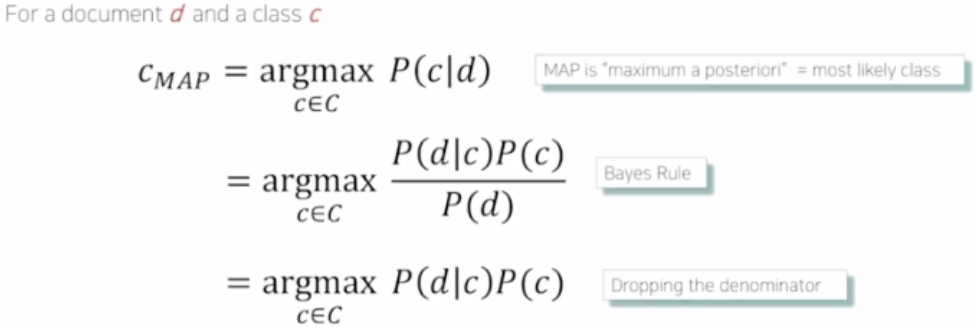
Bag-Of-Words (단어 가방 모형)

- 단어들의 순서는 고려 X, 단어들의 출현 빈도(frequency)에만 집중→ 텍스트 데이터의 수치화
- 사전상에서 가방을 준비하고 특정 문장에서 나타난 단어들을 가방에 넣어준 후(중복 x), 각 가방에 들어간 단어 수를 세서 벡터로 나타냄
- Step 1. unique한 단어들 사전(vocabulary) 형태로 저장
  - 저장된 단어들은 각각 중복 없는 카테고리 변수
- Step 2. categorical variable인 unique words를 one-hot vectors로 인코딩하여 벡터로 표현
  - 주어진 문장은 원-핫 벡터의 합, 즉 숫자로 표현할 수 있게 됨



Naive Bayes Classifier for Document Classification

- bag-of-words 벡터로 나타낸 문서를 정해진 카테고리/클래스로 분류하는 방법



예시

- 목표: 학습 데이터로 주어진 Traing 1~4 번 문장→ Test data(5번 문장)을 CV, NLP 두 클래스 중에 한 곳으로 분류

- Example
  - For a document *d*, which consists of sequence of words *w*, and a class *c*

	Doc(d)	Document (words, w)	Class (c)
Training	1	Image recognition uses convolutional neural networks	CV
	2	Transformer 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	?

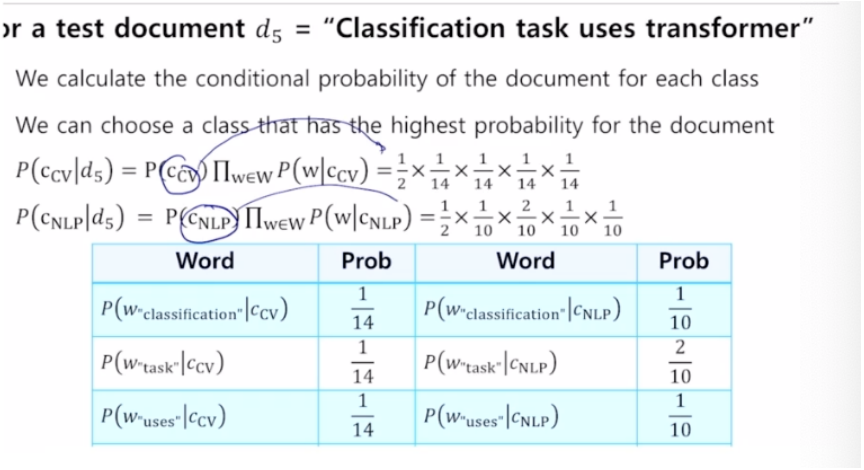
- $P(c_{CV}) = \frac{2}{4} = \frac{1}{2}$
- $P(c_{NLP}) = \frac{2}{4} = \frac{1}{2}$

- 5번 문장에 있는 각 단어들이 1~4번 문장에 몇 번 등장했는지를 조건부 확률로 계산

Example

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



or a test document d_5 = "Classification task uses transformer"

We calculate the conditional probability of the document for each class

We can choose a class that has the highest probability for the document

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

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

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

- 파라미터 추정 방식은 최대우도법(MLE)을 기반으로 유도

3. Word Embedding - (1)Word2Vec

Word Embedding

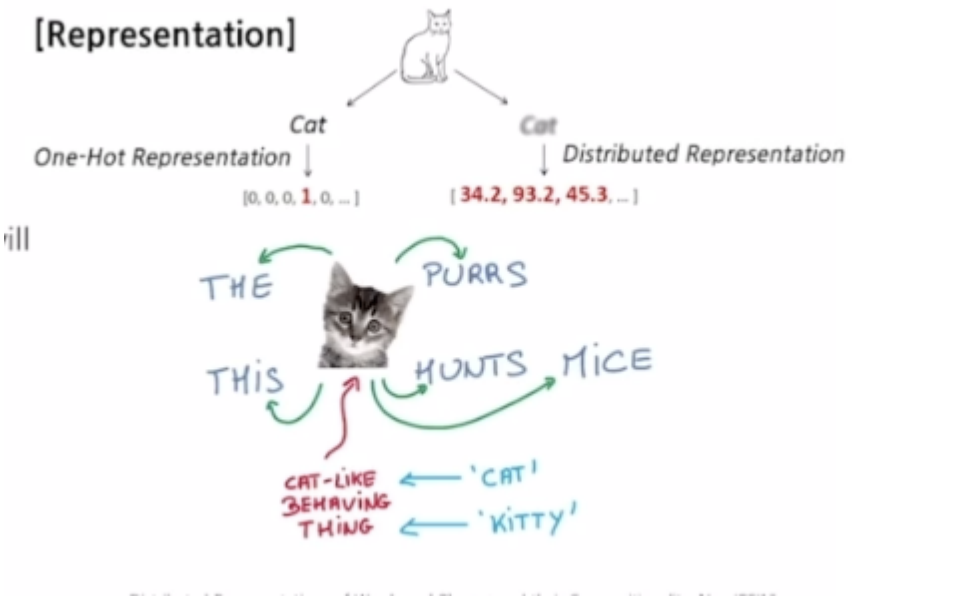
워드 임베딩: 각 단어를 좌표공간 상의 한 점의 좌표를 나타내 벡터로 표현하는 기법

ex) (1,3,4) → cat

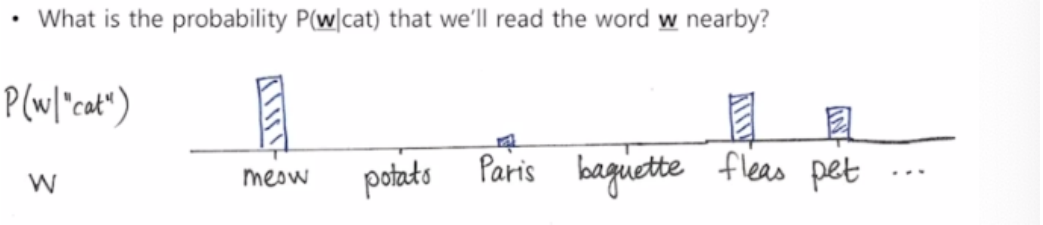
- 비슷한 의미를 →좌표공간 상의 비슷한 위치 mapping(유사도를 반영)
  - cat-kitty는 비슷한 위치, hamburger는 먼 위치
  - love-like는 비슷한 위치

Word2Vec Idea

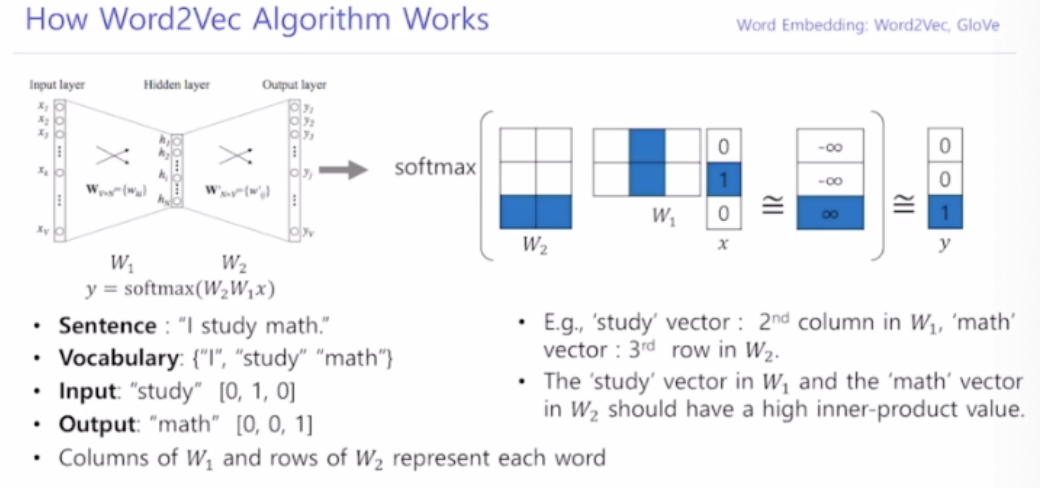
- 같은 문장 내에서 인접한 단어들간의 의미가 비슷할 것이라고 가정



- 주변에 등장하는 단어들을 통해 중심 단어의 의미가 표현될 수 있다는 것을 착안
  - 확률 분포를 예측하여 학습 진행



구체적 학습 과정

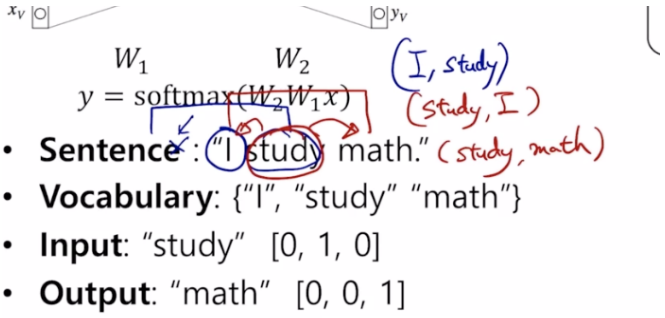


1) 워드를 Tokenization→유니크한 단어만 모아서 사전(Vocabulary)을 구축

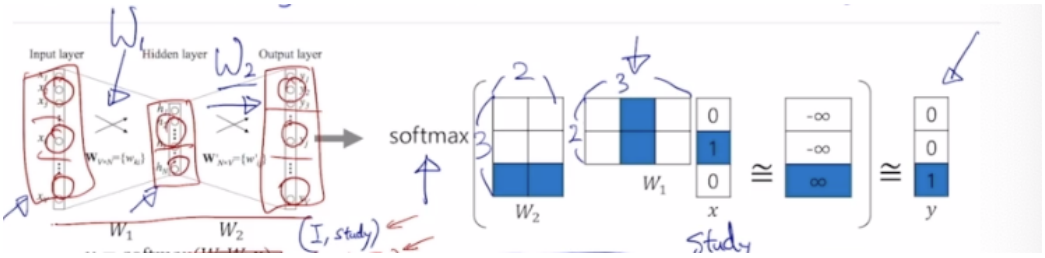
- 사전의 각 단어는 사전 사이즈만큼의 dimension을 가지는 one-hot vector

2) 문장에서 중심단어를 위주로 학습 데이터를 구축

- sliding window 기법 사용
- ex) “I study math”→ 중심단어가 study → (I study), (study I) , (study math) 와 같은 단어쌍을 학습 데이터로 구축

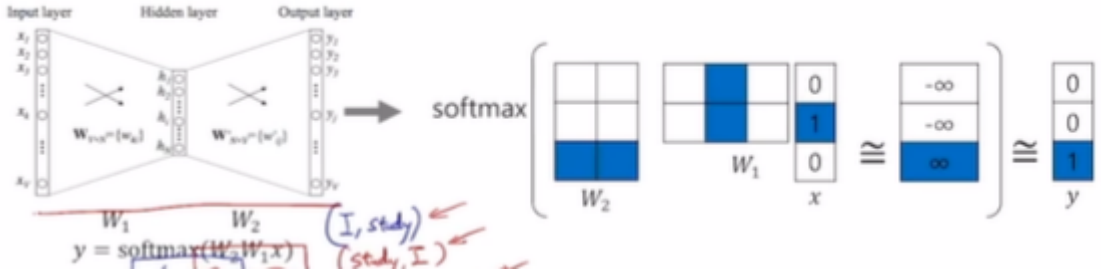


3) 입출력 단어쌍들에 대해 예측 task 수행하는 2 layer neural network

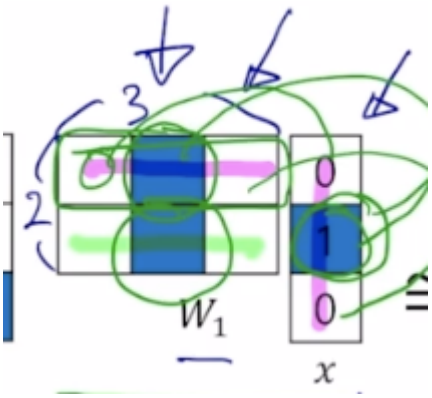


- 입출력 노드는 모두 3차원 one-hot vector(사전에 단어 수가 3개이므로)

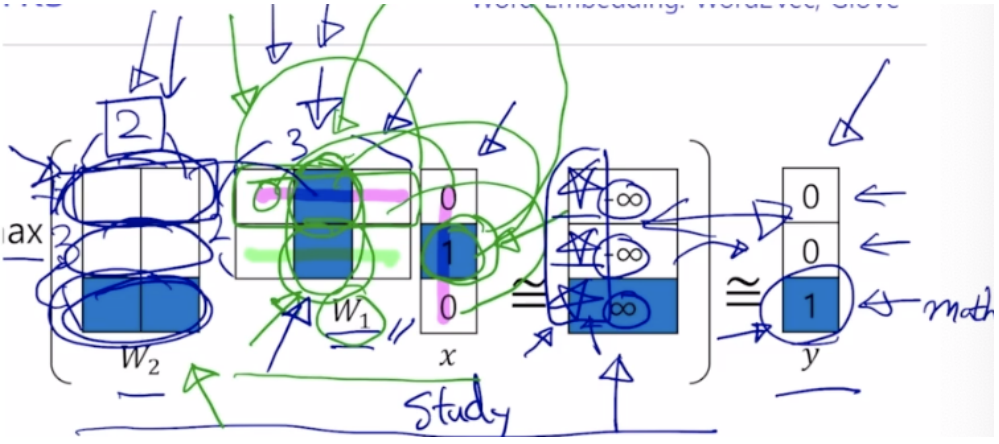
Word2Vec의 계산



- 문장의 단어의 갯수만큼에 Input, Output 벡터 사이즈를 입력/출력
  - 연산에 사용되는 은닉 층의 차원(dim)은 사용자가 파라미터로 지정
- 임베딩 레이어와의 연산은 0이 아닌 1인 부분만 추출해서 계산



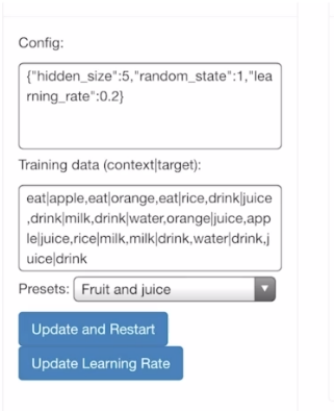
- one-hot vector와 첫번째 선형 변환 (W1) matrix가 곱해지는 과정 -> 임베딩 레이어라고 함
- [0,1,0] 벡터인 경우는 2번째 원소와 곱해지는 부분의 column만 추출하여 계산해줌

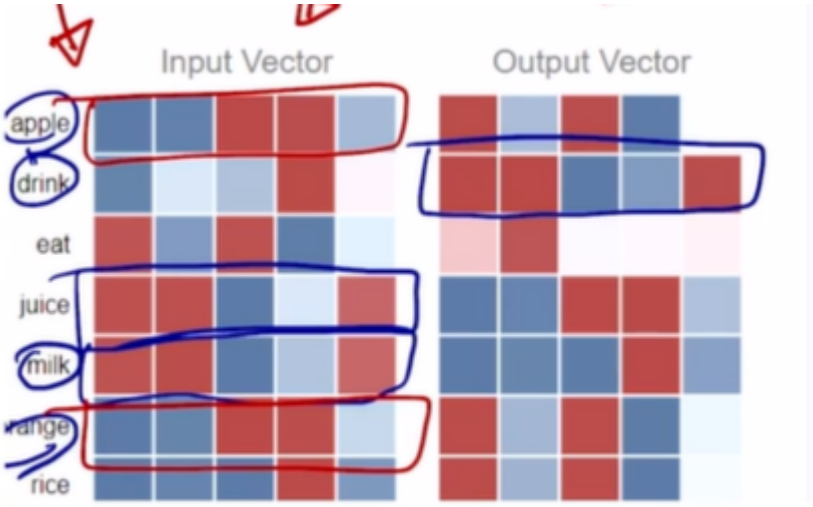


- 마지막 결과값으로 나온 벡터는 softmax 연산을 통해 가장 큰 값이 1, 나머지는 0으로 출력

- W1, W2에 속한 파라미터들을 조정하며 학습 진행
- 내적 연산을 반복되면서, 같이 등장하는 단어들 간의 벡터표현이 유사도가 커지게 됨.

Word2Vec의 특성

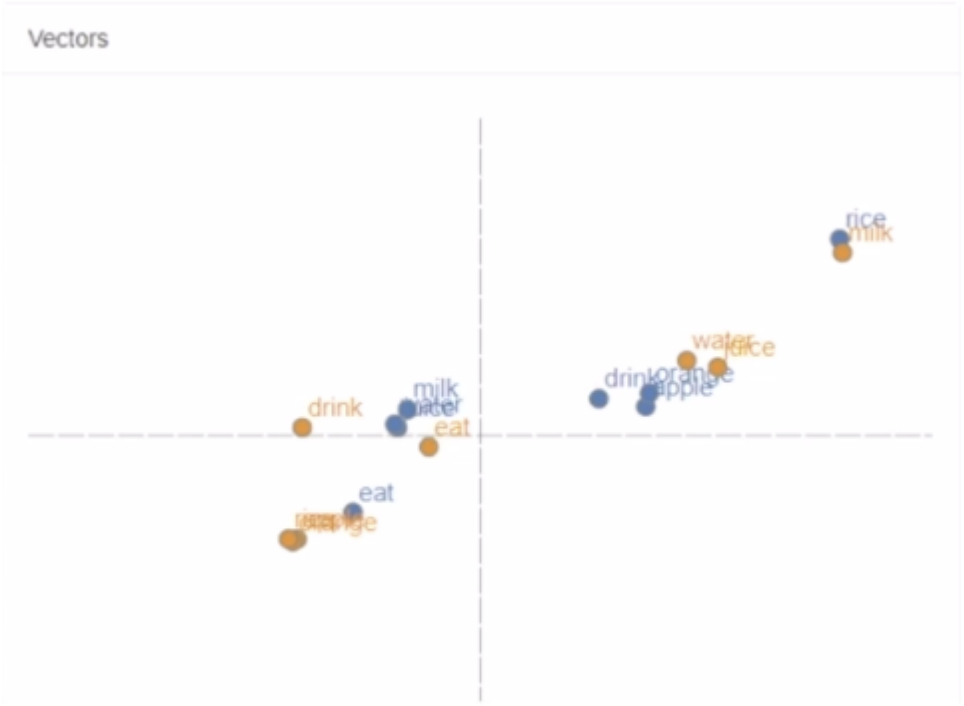


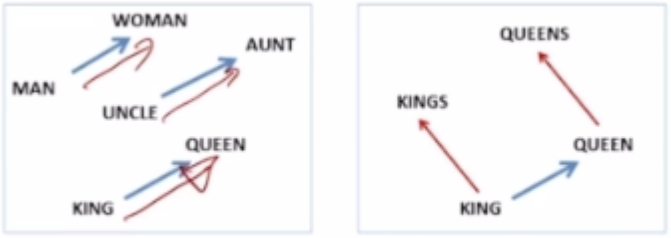


- drink와 milk, water의 input vector 유사한 형태를 가짐
- juice의 input vector와 drink의 output vector 거의 유사한 형태를 가짐

  → 둘의 내적값을 최대로

- 아래는 입력, 출력 단어 2차원으로 차원 축소한 후 시각화한 것





(Mikolov et al., NAACL HLT, 2013)

- queen - king 그리고 woman - man , 마지막으로 aunt - uncle 의 벡터가 비슷→ 여성과 남성의 관계성을 잘 학습했다는 것.
- 연세대 축제 이름인 "아카라카"에서 연세대를 빼고 고려대를 더해주면, 고려대의 축제인 "입실렌티"가 나옴

Application of Word2Vec

- Word intrusion detection: 의미가 가장 다른 단어 찾아내기
  - 단어들간의 유클리드 거리를 계산→평균을 구한 후 가장 큰 단어를 고름

- 기계번역, 감정분석, 이미지 캡션 등에서 임베딩 벡터 사용

  - Machine translation : 단어 유사도를 학습하여 번역 성능을 더 높여줍니다.
  - Sentiment analysis : 감정분석, 긍부정분류를 돕습니다.
  - Image Captioning : 이미지의 특성을 추출해 문장으로 표현하는 테스크를 돕습니다.

4. Word Embedding - (2)GloVe

GloVe : Global Vectors for Word Representation

GloVe와 Word2Vec의 차이점

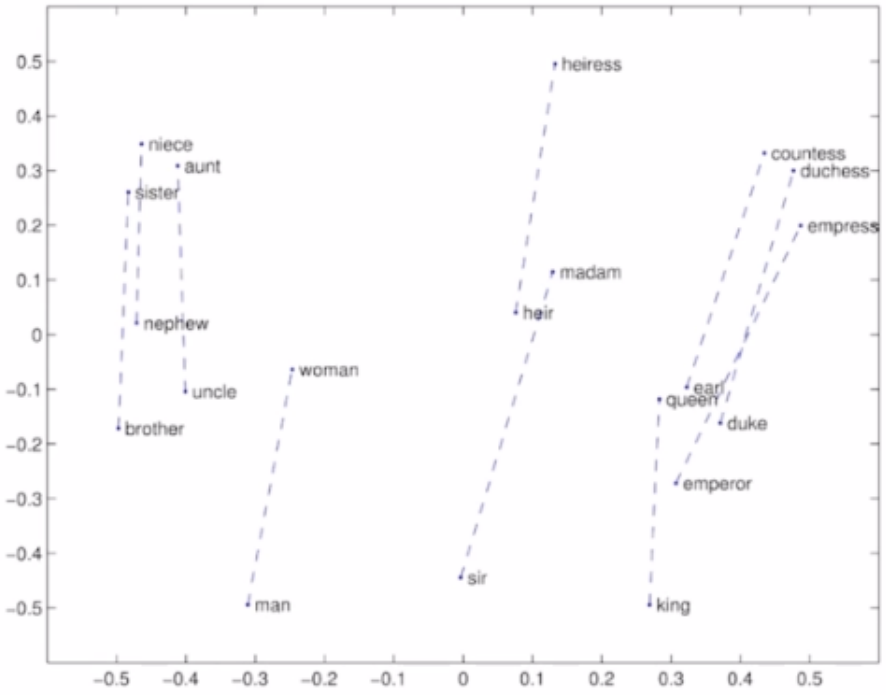
- 각 입출력 단어쌍들에 대해 학습 데이터에서 두 단어가 한 window에서 몇번 동시에 등장했는지 사전에 계산→ log 값→ Ground Truth로
- 단어 간의 내적값이 가까워질 수 있도록 하는 loss 함수 사용

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

- Word2Vec보다 빠르게 동작하며 더 적은 데이터에서도 잘 동작함

GloVe의 특징

- 의미 차이가 일정한 방향과 크기의 벡터로 나타남



- 형용사들에 대해 원형, 비교급/최상급 간에도 일정한 크기와 방향을 가진 벡터들이 나타남-> 문법적 의미와 관계도 효과적으로 학습함

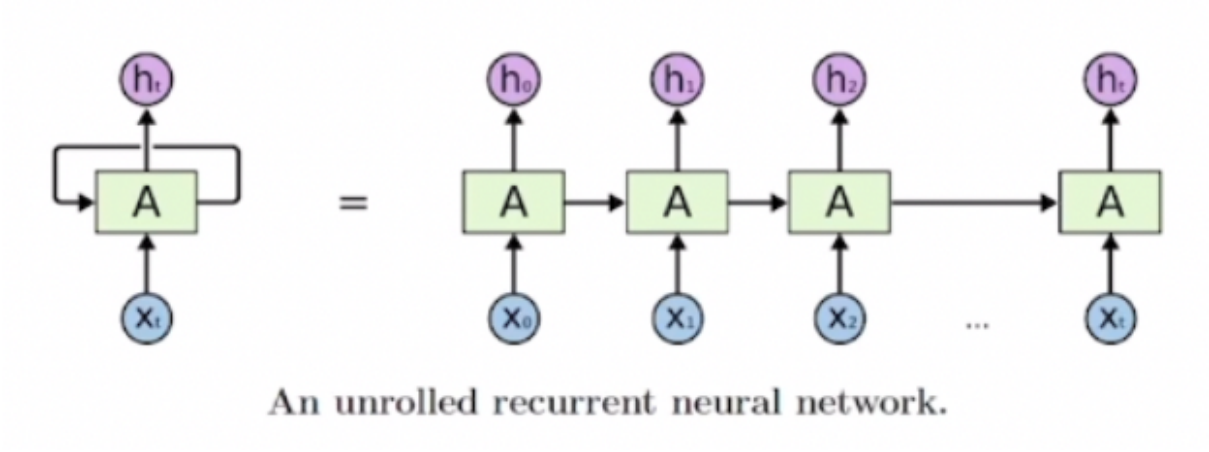
사전 학습된 Glove 모델

- 사전에 이미 대규모 데이터로 학습된 모델이 오픈소스로 공개
- 위키피디아 데이터를 기반으로 하여 6B token만큼 학습 되었으며, 중복 제거 후 사전을 구축할 때도 단어의 개수가 무려 40만개(400k)에 달함
- uncased: 대문자 소문자를 구분 x ↔ cased: 대소문자를 구분

자연어 처리와 딥러닝

1. Recurrent Neural Network (RNN)

RNN

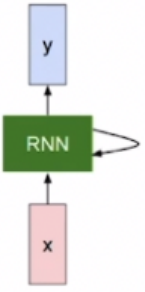


An unrolled recurrent neural network.

- 시퀀스 데이터가 입력, 출력 벡터
- 이전 스텝까지의 정보(x*t*, *h*(t-1))→ 현재 타임스텝 예측값(h_t)을 산출하는 구조
- 매 타임스텝마다 동일한 파라미터를 가진 모듈(A)을 반복적으로 사용= 재귀적인 호출-> Recurrent Neural Network(RNN) 계산 방법

RNN 계산 방법

$$h_t = f_W(h_{t-1}, x_t)$$

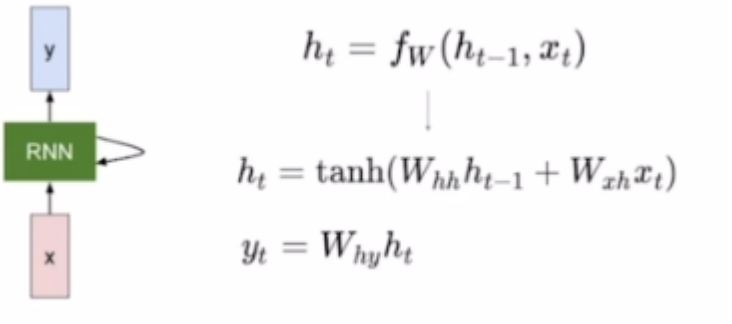


- t : 현재 타임스텝(time step) , w : 웨이트(weight)
- $h_t - 1$: old hidden-state vector
- x_t : input vector at some time step
- h_t : new hidden-state vector
- f_w : RNN function with parameters W
- y_t : output vector at time step t

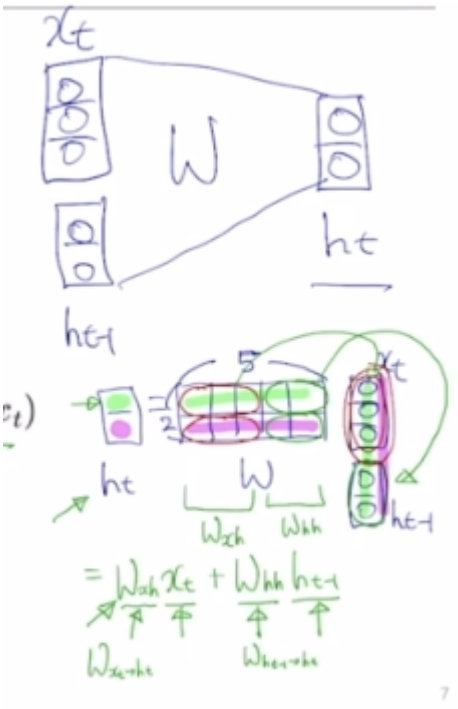
1. 위의 변수들에 대하여, $h_t = f_w(h_t - 1, x_t)$ 의 함수를 통해 매 타임스텝마다 hidden state를 다시 구해준다.
2. 이 때, W와 입력값($x_t, h_t - 1$)으로 tan_h 를 곱해서 h_t 를 구해준다.
3. 구해진 h_t, x_t 를 입력으로 y_t 값을 산출하게 된다.

- y는 매 타입스텝마다 구해야 할 수도 있고 마지막에만 구하는 경우도 있음

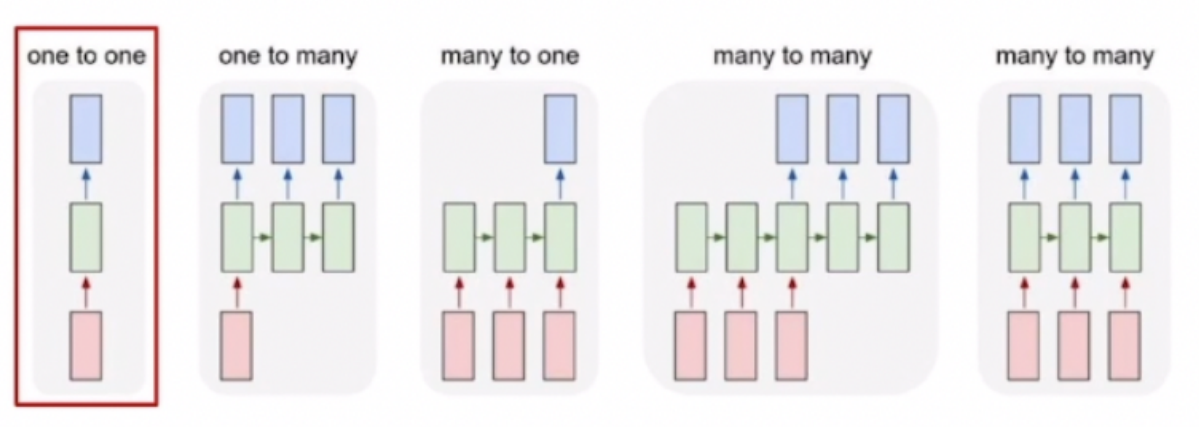
- 파라미터 W는 모든 타임스텝에서 동일한 값을 공유



- *Wxh*: X(*t*)−>*ht* 로 변환
- *Whh*: W(*t-1*)−>*ht* 로 변환
- *Why*: *Wht*−>*yt* 로 변환



RNN 종류



1. one-to-one
- 입출력 데이터의 타임스텝이 1개
- ex) [키, 몸무게, 나이] → 저혈압/고혈압인지 분류
2. one-to-many
- 입력은 1개의 타임스텝, 출력은 여러 개의 타임스텝
- ex) 이미지 캡션, 입력: 이미지 -> 출력: 여러 설명
3. many-to-one
- 입력 시퀀스를 입력→최종값을 마지막 스텝에서 출력
- ex) 감성 분석: 긍정/부정 중 하나의 레이블로 분류
4. many-to-many
- 입력값을 끝까지 다 읽은 후 마지막 타임스텝에서 번역된 문장(예측값)을 출력해주는 테스크
- ex) 기계 번역
5. many-to-many
- 딜레이가 존재하지 않고 입력할 때마다 예측값을 출력
- ex) 단어별 품사 태깅→ POS 테스크, 비디오 프레임별 분류 테스크

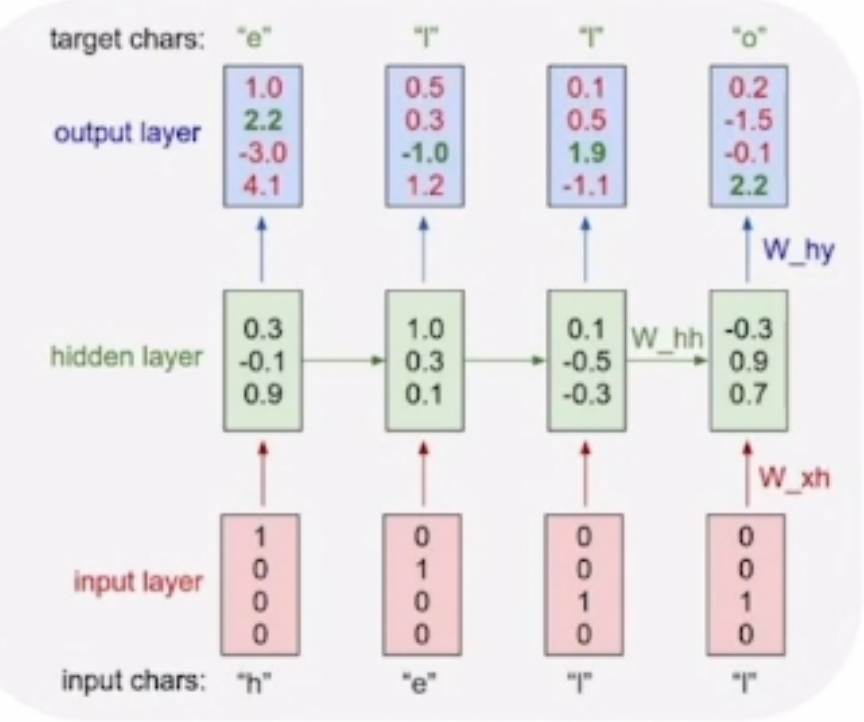
2. Character-level Language Model

character-level Language Model

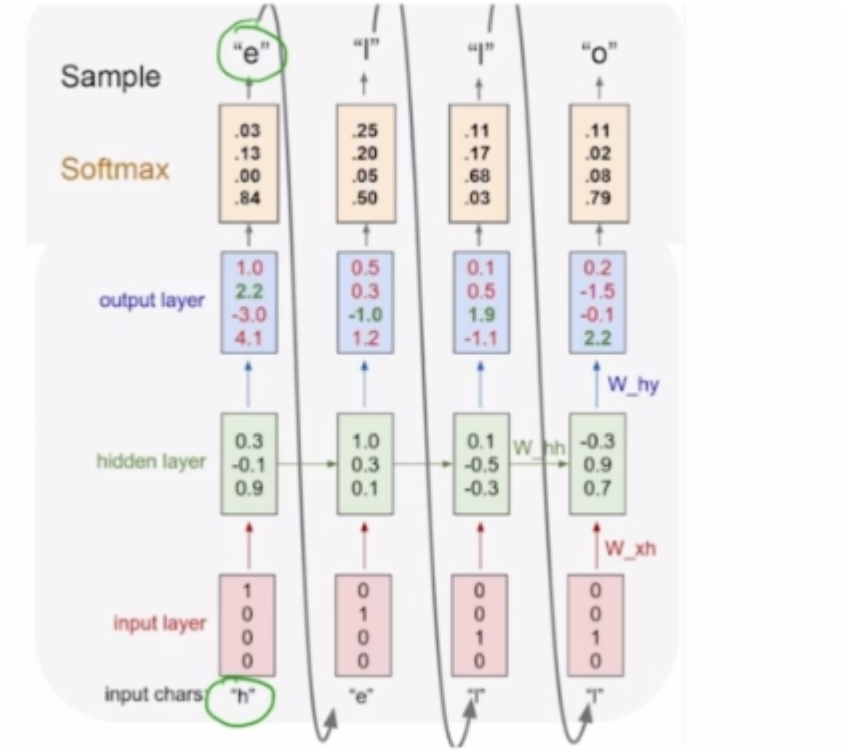
- 언어 모델: 이전에 등장한 문자열을 기반으로 다음 단어를 예측하는 테스크
  → 캐릭터 레벨 언어 모델(character-level Language Model): 문자 단위로 다음에

올 문자를 예측하는 언어 모델

Example of training sequence "hello"



- vocabulary: [h,e,l,o]> one hot vector 형태의 input
- "h"가 주어지면 "e"를 예측→"e"가 주어지면 "l"을 예측→ "l"이 주어지면 다음 "l"을 예측하도록 hidden state가 학습
- hidden layer: $ht=tanh(Whh\ ht-1+Wxh\ xt+b)$
- output layer: $Logit=Why\ ht+b$
  - many to many task에 해당
  - 각 타임스텝별로 output layer를 통해 출력하는 차원 4 벡터 - softmax layer를 통과→ one hot vector 형태의 출력값이 나옴
- ground truth vector와 가까워지도록 학습



활용: 전 타임스텝까지의 주식값→ 다음날 주식값을 예측

다양한 언어모델의 예시

- 문단 학습도 가능

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

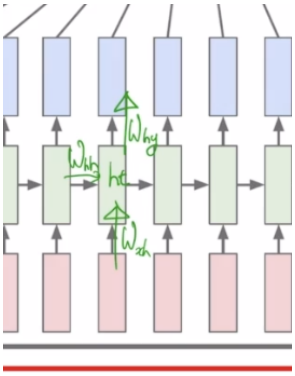
→ 학습이 진행될수록 고품질의 문자열이 생성됨

- 인물별 대사
- Latex로 논문 작성
- C 언어

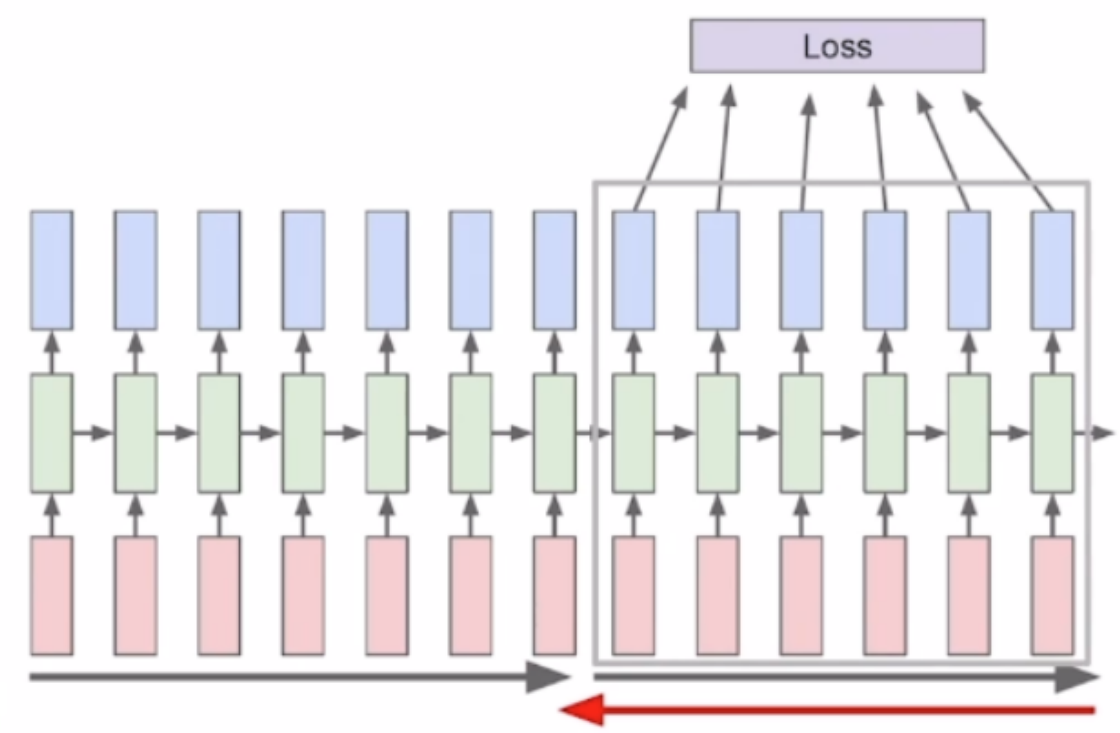
3. Backpropagation through time and Long-Term-Dependency

RNN 모델이 학습하는 방법

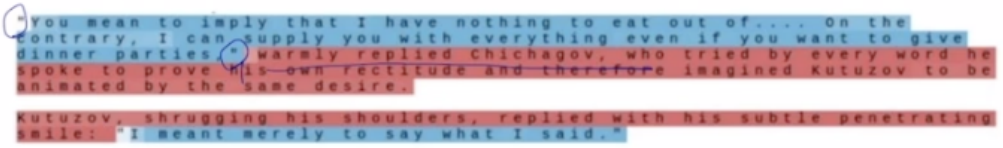
- W_xh, W_hh, h_t, W_hy 학습 진행



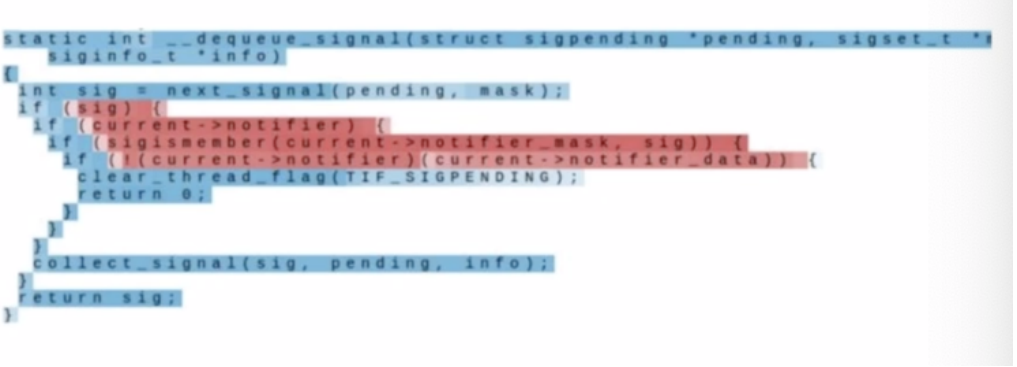
- **Truncation**: 잘라서 제한된 길이의 시퀀스만으로 학습을 진행하는 것



- **BPTT(Backpropagation through time)**: RNN에서 타임스텝마다 계산된 weight를 backward propagation을 통해 학습하는 방식
  - 특정 dimension의 hidden state를 빨강은 긍정, 파랑은 부정으로 시각화한 그림

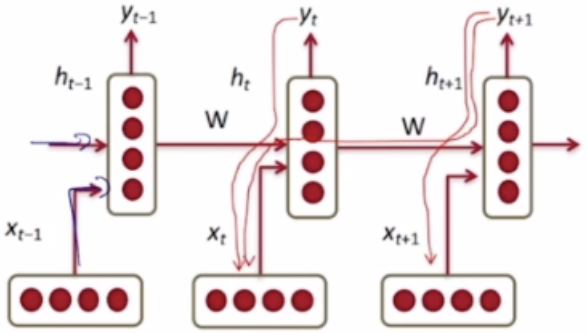


  - 조건문에 해당하는 부분이 빨강으로 나타냄



- Vanilla RNN문제: 같은 숫자 계속 곱하는 문제가 생길 수 있음

nishing or exploding



Vanishing/Exploding Gradient Problem in RNN

- RNN보다 진보: LSTM, GRU
- Vanilla의 문제: gradient가 전파되면서 기하급수적으로 소실, 증폭→ 멀리까지 학습정보를 잘 전달하지 못하는 Long-Term-Dependency

Toy Example

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

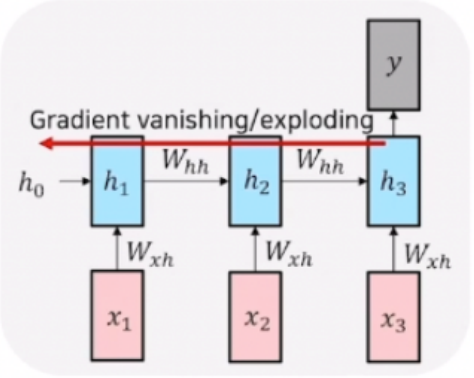
$h_3 = \tanh(2x_3 + 3h_2 + 1)$

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

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

...

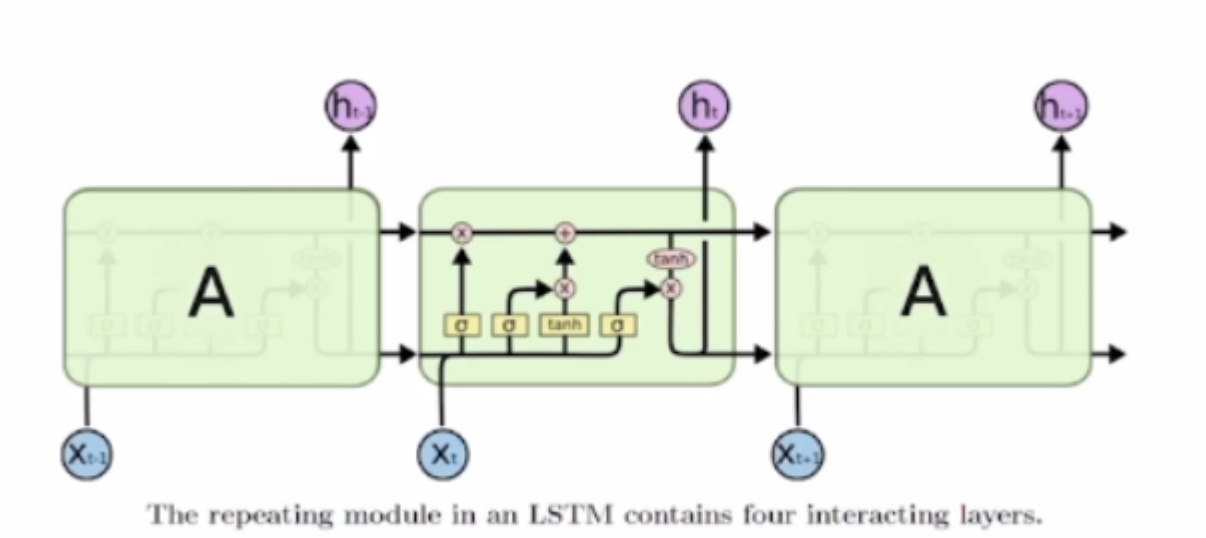
$h_3 = \tanh(2x_3 + 3\tanh(2x_2 + 3h_1 + 1) + 1)$



- back propagation 과정에서 h3에 대한 h1 편미분 값을 구할 때, 접선의 기울기들을 곱해 3이 거듭 곱해짐-> 모델 자체가 전체적으로 학습이 잘 되지 않는 현상 발생(폭발, 소멸 문재 발생 가능)

4. Long Short-Term Memory (LSTM)

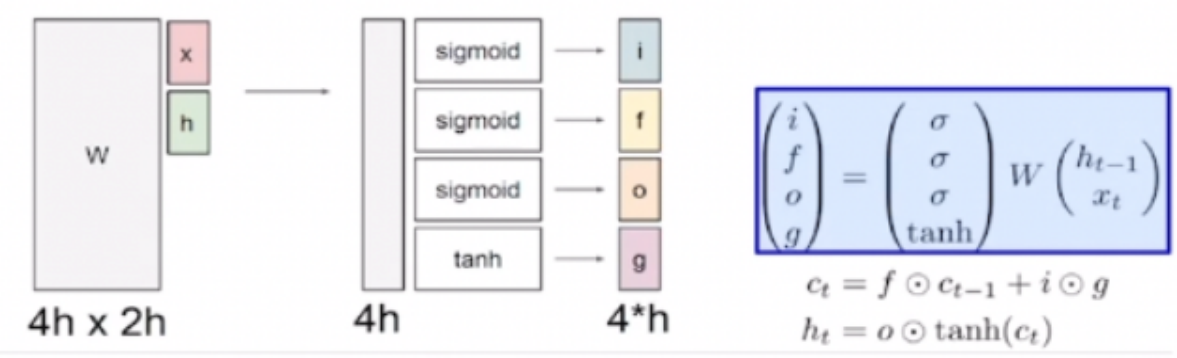
LSTM : Long Short-Term Memory



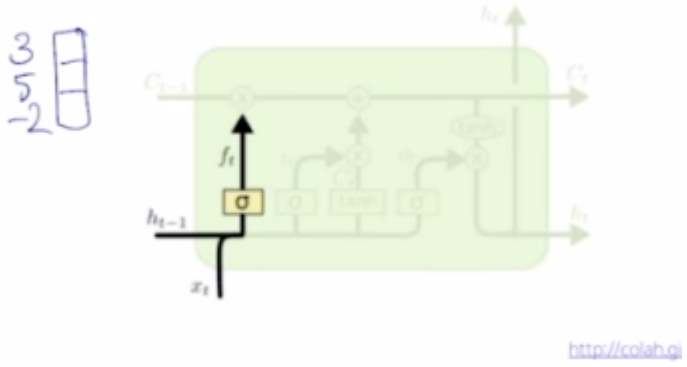
The repeating module in an LSTM contains four interacting layers.

- vanilla RNN을 완벽하게 대체 가능하고 더 좋은 성능을 보임
  → gradient vanishing/exploding과 Long-Term-Dependency를 해결
- 중심 아이디어: 단기 기억으로 저장하여 때에 따라 꺼내 사용함으로 더 오래 기억할 수 있도록 개선
- Cell state에는 핵심 정보들을 모두 담아두고, 필요할 때마다 Hidden state를 가공해 time step에 필요한 정보만 노출하는 형태로 정보 전파
  - $Ct,ht=LSTM(xt,Ct-1,ht-1)$
    - $Ct-1$: cell state vector
    - $ht-1$: hidden state vector

LSTM의 연산 과정



- input으로 x_t, h_t-1이 들어오게 되고 이를 W에 곱해준 후 각각 sigmoid or tanh로 연산해줌
- 4개의 gate 존재→양을 조절하는 역할
- i : Input gate이며, cell 에 쓸지말지를 결정하는 게이트
  - 들어오는 input에 대해서 마지막에 sigmoid를 거쳐 0-1 사이 값으로 표현해줌
  - 표현식 : sigmoid(W(xt, ht-1))
- f : Forget gate이며, cell 정보를 어느정도로 지울지를 sigmoid를 거쳐 0~1사이의 값으로 나타냄
  - 표현식 : sigmoid(W(xt, ht-1))



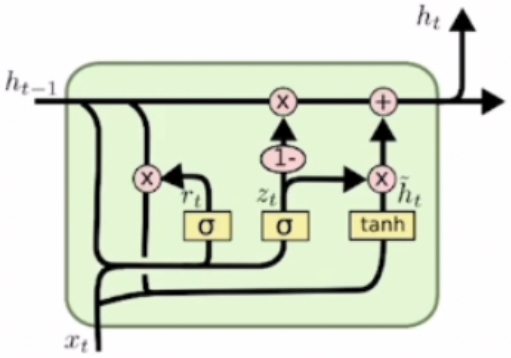
- o : Output gate이며, Cell 정보를 어느정도 hidden state에서 사용해야할 지를 sigmoid를 거쳐 0~1사이 값으로 나타냄
  - 표현식 : sigmoid(W(xt, ht-1))
- g : Gate gate이며, 어느정도로 Cell state에 반영해야할 지를 tanh를 거쳐 -1~1 사이의 값으로 나타냄
  - 표현식 : tanh(W(xt, ht-1))

LSTM이 RNN과 다른점

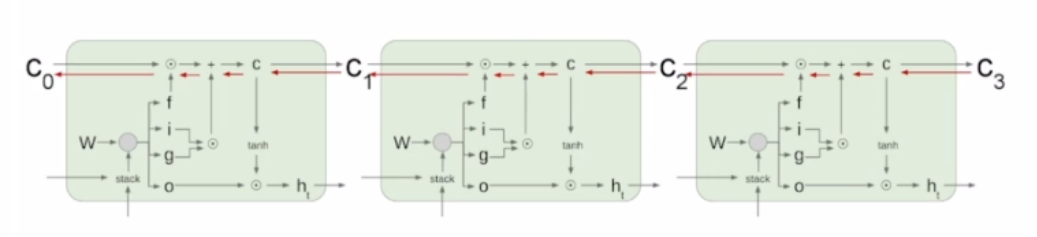
- LSTM은 각 time step마다 필요한 정보를 단기 기억으로 hidden state에 저장하여 관리되도록 학습함
- backpropagation 진행시 forget gate를 거친 값에 대해 필요로 하는 정보를 덧셈을 통해 연산하여 그레디언트 소실/증폭 문제를 방지

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



- GRU: LSTM과 전체적인 동작원리는 유사하지만, Cell state, Hidden state를 일원화하여 경량화한 모델
- GRU에서 사용되는 *ht*−1은 LSTM에서의 *ct*와 비슷한 역할을 함
- forget gate 대신 1-input gate를 사용
- LSTM에 비해 계산량과 메모리 요구량을 줄이면서 성능도 더 좋음



RNN ,LSTM ,GRU 요약

- RNN은 다양한 길이의 시퀀스에 특화된 유연성을 가진 딥러닝 모델
- RNN에서는 gradient vanishing/exploding 문제가 있어 실제로 많이 사용되지는 않지만, LSTM, GRU 모델은 현재도 많이 사용되고 있음