[Week16]_문가을

자연어 처리의 모든 것

1. 자연어 처리의 시작

자연어 처리 활용 분야와 트렌드

자연어 처리 Nautural Language Processing (NLP)

Low-level parsing

- Tokenization : 문장을 단어 단위로 쪼개기
- stemming : 단어의 어근(의미 단위)을 추출

Word and phrase level

- NER (Named Entity Recognition) : 단일 단어/ 여러 단위로 된 고유 명사 인식
- POS (Part-of-Speech) : 워드들이 문장 내에서 성분이 뭔지 알아내는 것 (목적어, 주어 등)

Sentence level

- Sentiment analysis
- machine translation

Multi-sentence and paragraph level

- Entailment prediction : 두 문장 간의 논리적인 내포나 모순 관계 예측
- question answering : 검색 창에 질문을 넣으면 답을 맨 위에 보여줌.
- dialog systems : chat-bot

- summarization

<Text mining>

- 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

<Information retrieval (정보 검색)>

- 정보 검색 분야, 추천 시스템

<Trends of NLP>

- text data → 단어 단위 → 차원을 가진 벡터로 (word embedding)
- Sequence data를 처리할 수 있어야 함.
- 대표적인 예로, RNN(Recurrent Neural Network)가 있고, RNN의 단점을 보완한 LSTM, GRU 모델이 나옴.
- 문법 내용을 적용한 rule 기반 모델도 존재함.
- '셀프 어텐션(Self-Attention)' 구조를 가진 '트랜스포머(Transformer) 모델'이 각광받기 시작함. 최근 발표된 대부분의 모델들은 트랜스포머 모델을 기반으로 하는 것이 많음.
- 최근에는 자가지도 학습(self-supervised Learning)이 가능한 BERT, GPT 와 같은 모델이 유행
- 자가지도 학습 : 문장에서 단어를 가리고 앞뒤 문맥을 통해 단어를 맞추도록 함.

`기존의 자연어 처리 기법`

딥러닝 기술이 적용되기 이전 기법

<Bag-of-Words>

step 1. text data 속에서 unique words를 인식하여 사전 구성

step 2. unique words를 one-hot vectors로 인코딩

- Vocabulary: {"John", "really", "loves", "this", "movie", "Jane", "likes", "song"}
  - John: [1 0 0 0 0 0 0 0]
  - really: [0 1 0 0 0 0 0 0]
  - loves: [0 0 1 0 0 0 0 0]
  - this: [0 0 0 1 0 0 0 0]
  - movie: [0 0 0 0 1 0 0 0]
  - Jane: [0 0 0 0 0 1 0 0]
  - likes: [0 0 0 0 0 0 1 0]
  - song: [0 0 0 0 0 0 0 1]

- for any pair of words, the uclidian distance is 2^1/2
- for any pair of words, cosine similarity(내적값/ 유사도) is 0

→ 단어의 의미와 상관 없이 모두 동일한 관계를 가진 것으로 표현됨.

step 3. one-hot vectors의 합으로 sentence/document 표현

- Sentence 1: "John really really loves this movie"
  - John + really + really + loves + this + movie: [1 2 1 1 1 0 0 0]
- Sentence 2: "Jane really likes this song"
  - Jane + really + likes + this + song: [0 1 0 1 0 1 1 1]

<NaiveBayes Classifier>

For a document d and a class c

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}}\ P(c|d)$$ 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

p(d)는 c 입장에서 상수기 때문임.

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

w1, w2,...,wn → 단어

예시 :

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	?

p(C_cv) = 2/4 = 1/2 // p(C_nlp) = 1/2

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

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

d_5 : Classification task uses transformer.

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

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

→ 확률 값이 가장 큰 클래스로 분류함.

기존 데이터에 존재하지 않는 단어가 등장하면 확률이 무조건 0으로 되기 때문에, 이를 보완하는 regulization 기법 또한 존재함.

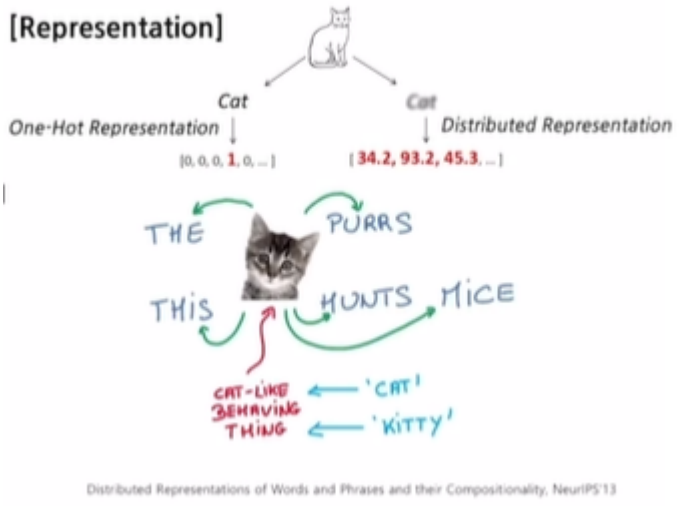
Word Embedding - (1) Word2Vec

워드 임베딩 : 단어를 벡터(좌표)로 표현하는 기법

→ 비슷한 의미를 가진 단어들이 비슷한 좌표를 가지도록 mapping되게 하여 의미 상의 유사도가 반영되도록 함.

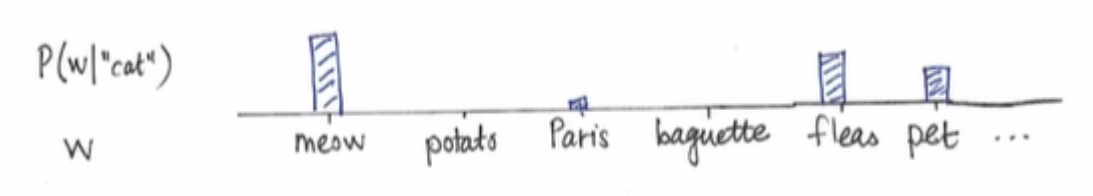
<Word2Vec>

- 같은 문장에서 나타난 단어는 의미적으로 관련성이 높다고 가정함.



Distributed Representations of Words and Phrases and their Compositionality, NeurIPS'13

- 특정 단어와 함께 등장할 확률을 나타낸 확률 분포를 나타낼 수 있음.



알고리즘 과정 자세히

1. 문장을 단어 단위로 쪼개서 단어 사전을 만들고
2. 각 단어를 one-hot 벡터로 만든 후
3. 슬라이딩 윈도우 라는 기법을 적용하여, 한 단어를 중심으로 앞뒤로 나타난 단어 각각과 입출력 단어 쌍을 이룸 (윈도우 size가 3이면 앞뒤로 하나씩)
4. 벡터가 가중치와 내적한 후 최종적으로 softmax 함수를 거쳐 확률분포를 결과로 내고, 원핫벡터로 변환해 판단.

word2Vec 계산 과정

- 문장의 단어의 갯수만큼에 Input, Output 벡터 사이즈를 입력/출력해줌. 이 때 연산에 사용되는 히든 레이어(hidden layer,은닉 층)의 차원(dim)은 사용자가 파라미터로 지정할 수 있음.
- 실제로 Tensorflow나 Pytorch와 같은 프레임워크에서는 임베딩 레이어와의 연산은 0이 아닌 1인 부분, 예를 들어 [0,0,1]의 벡터인 경우는 3번째 원소와 곱해지는 부분의 컬럼(column)만 뽑아서 계산함.
- 마지막 결과값으로 나온 벡터는 softmax 연산을 통해 가장 큰 값이 1, 나머지는 0으로 출력됨.
- 위의 연산이 반복되면서, 같이 등장하는 단어들 간의 벡터표현이 유사해짐.
- 단어들의 벡터를 더하거나 빼서 다른 단어를 유추할 수도 있음.

Word2Vec 응용

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

Word Embedding - (2) Glove

Word2Vec과의 차이점

→ 단어쌍들이 한 윈도우 내에서 몇 번 등장했는지 사전에 계산(P_ij), u_i(입력 word)와 v_j(출력 vector)의 내적과 가까워지도록 함.

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

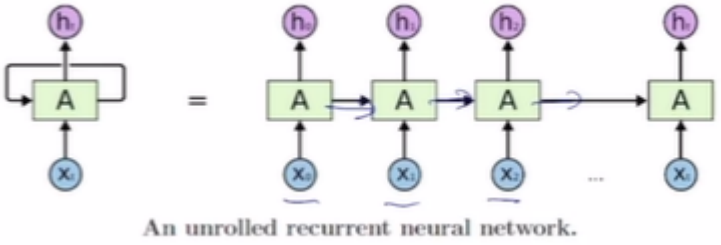
glove website에서 기학습된 워드 임베딩 이용 가능함.

---

2. **자연어 처리와 딥러닝**

Recurrent Neural Network (RNN)

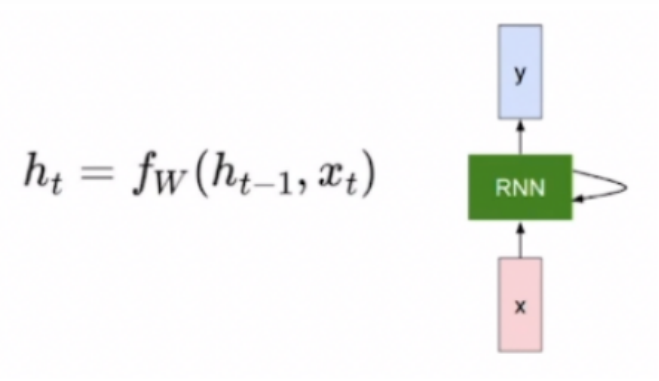
기본 구조



An unrolled recurrent neural network.

- RNN은 현재 타임스텝에 대해 이전 스텝까지의 정보를 기반으로 예측값을 산출하는 구조의 딥러닝 모델.
- 문장의 각 단어는 각 타임스텝의 입력으로써 순차적으로 들어감.
- 매 타임스텝마다 동일한 파라미터를 가진 모듈을 사용하므로, '재귀적인 호출'의 특성을 보여주어 'Recurrent Neural Network'라는 이름을 가지게 됨.

Input and outputs of RNNs (rolled version)



- hidden state vector가 다음 타임 스텝의 입력으로 쓰임과 동시에, 출력값 계산함.

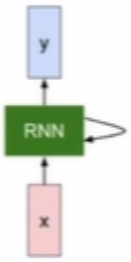
  - t : 현재 타임스텝(time step) , w : 웨이트(weight)
  - $h_t - 1$: old hidden-state vector
  - x_t : input vector at some time step
  - h_t : new hidden-state vector
  - f_w : RNN function with parameters W
  - y_t : output vector at time step t

- y_t는 매 타임스텝마다 계산할 수도 있고, 마지막 output만 계산하기도 함. (예측이 단어마다 필요한지, 문장마다 필요한지 등에 따라 다름)

> 💡 The same function and the same set of parameters are uesd at every time step!

hidden state를 계산하는 방법

- The state consists of a single "hidden" vector **h**

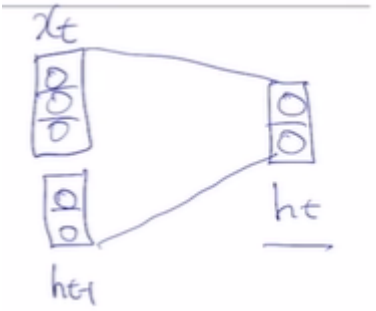


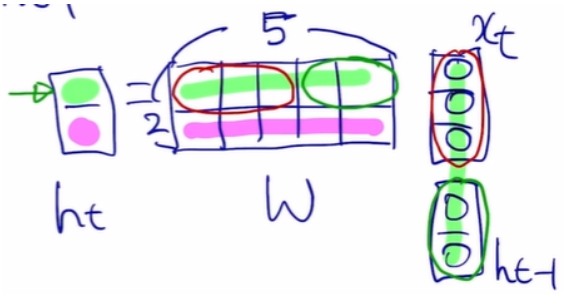
$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

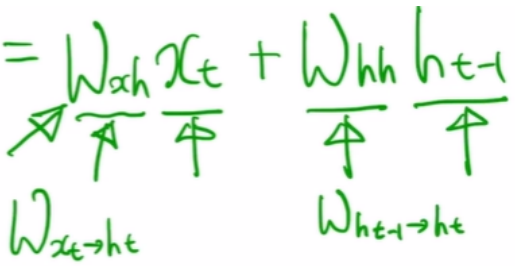
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

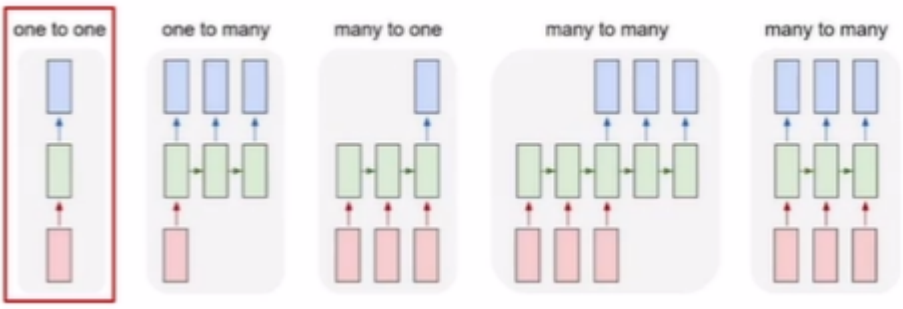




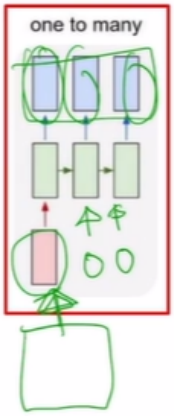
h_t-1과 h_t의 노드 수는 같아야 함.



Types of RNNs

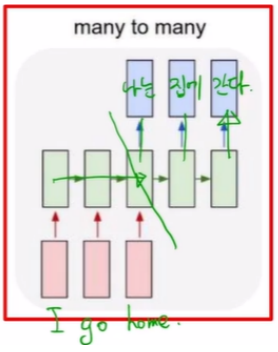


- one to one : 입력과 출력이 모두 sequence data가 아님, 예시) [키, 몸무게, 나이]와 같은 정보를 입력값으로 할 때, 이를 통해 저혈압/고혈압인지 분류하는 형태의 테스크
- one to many : 입력은 하나의 타임 스텝, 출력은 여러 타임 스텝으로 이루어짐. 예시) 이미지 캡션 → 하나의 이미지를 입력값으로 주면 설명글을 생성하는 테스크

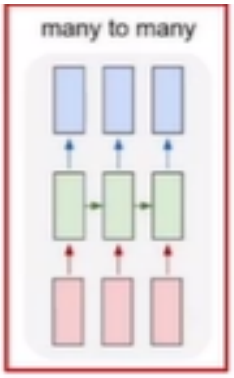


추가적으로 넣어줄 입력이 없을 경우, 같은 사이즈의 벡터/텐서/행렬이 들어가되 0으로 채워진 것으로 입력을 줌.

- many to one : 입력 sequence를 최종 값을 마지막 타임 스텝에서 출력함. 예시) 감성 분석과 같이 문장을 넣으면 긍/부정 중 하나의 레이블로 분류하는 테스크
- many to many : 입출력이 모두 sequence data



예시) 기계 번역과 같이 입력값을 끝까지 다 읽은 후, 번역된 문장을 출력해주는 테스크



입력을 받을 때마다 예측 수행.

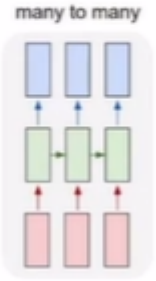
예시) 비디오 분류와 같이 영상의 프레임 레벨에서 예측하는 태스크

혹은 각 단어의 품사에 대해 태깅하는 POS와 같은 태스크

Character-level Language Model (언어 모델)

문자열이나 단어 순서를 받고, 다음 단어를 예측하는 모델

- Example of training sequence "hello"
  - Vocabulary: [h, e, l, o]
  - Example training sequence: "hello"

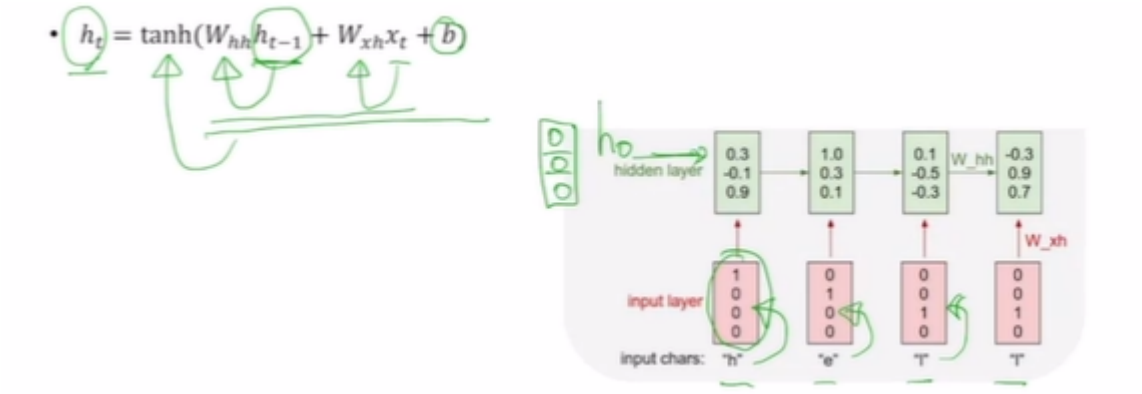


1) Character-level 의 사전 구축

2) 각각의 character는 총 사전의 개수 만큼의 dimension을 가지는 one-hot vector로 표현됨.

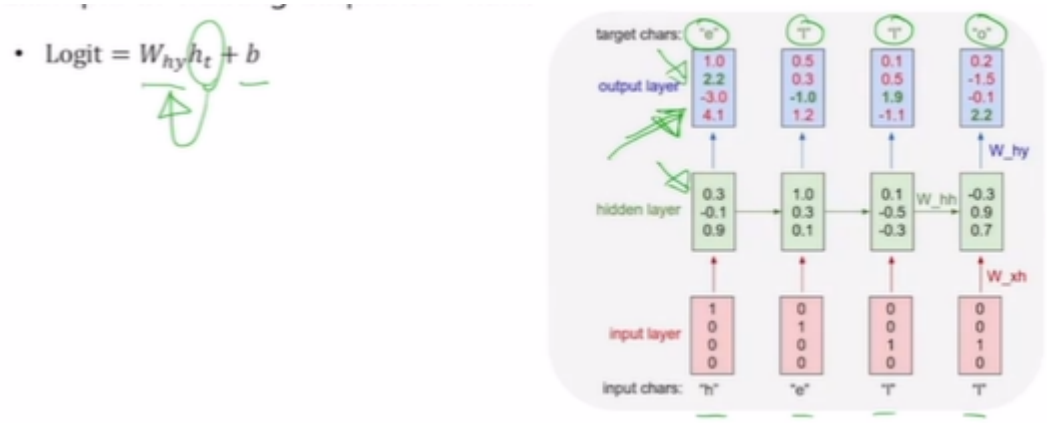
(h를 받으면 e를 예측하고, e를 받으면 l을 예측하는 그런 구조)

3) hidden state 계산

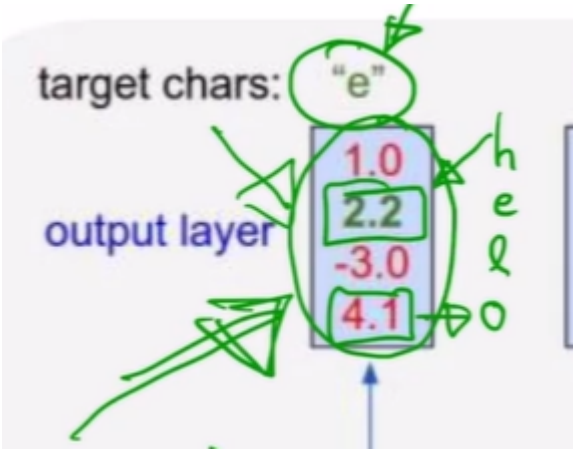


h_0는 이전 스텝이 없기 때문에 0으로 채운 것을 입력으로 줌.

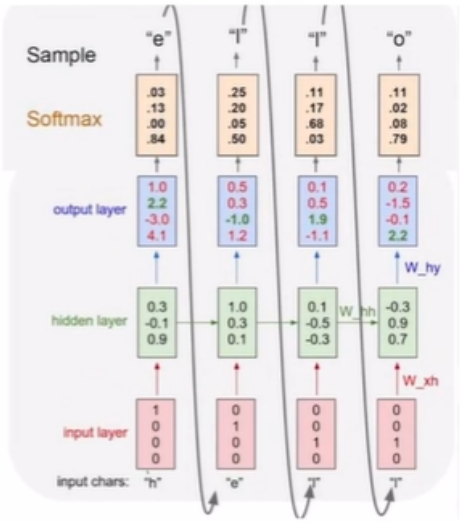
4) output vector 계산



logit : softmax layer를 통과 시켜 one-hot 벡터 형태의 출력값이 나오게 함.



'e'로 예측해야 하는데, 'o'가 예측으로 나옴.

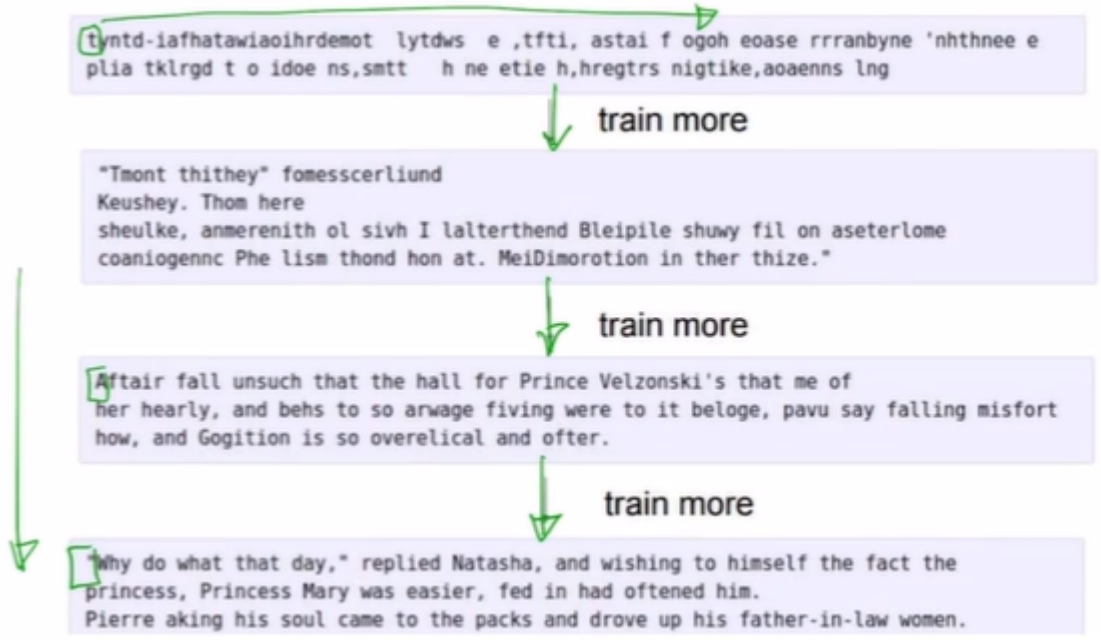


첫번째 charcter만 입력으로 주고, 예측값을 다음

타입 스텝의 입력으로 사용.

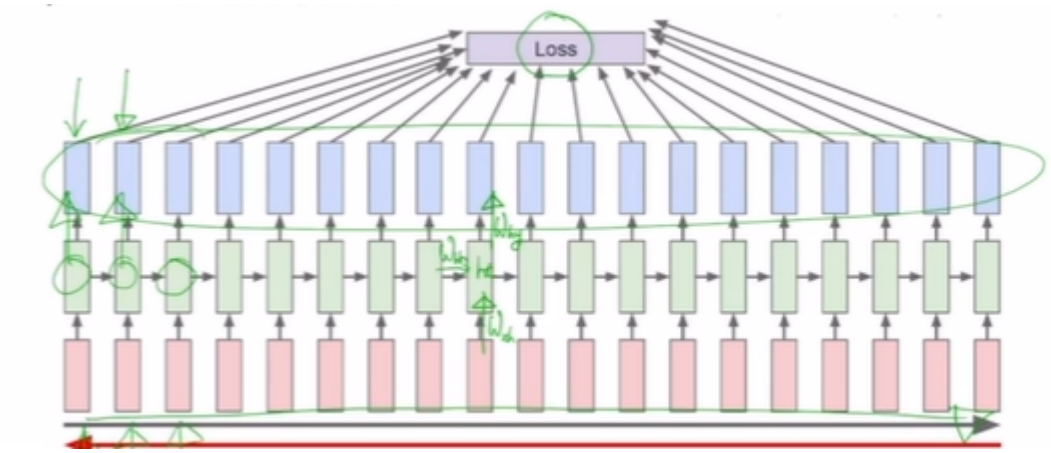
과거 주식 데이터를 통해 바로 다음 주가를 예측하는 모델이더라도, 더 이후의 주가를 예측하는 모델과 같음. (긴 sequence 예측)

다른 예시 : 문단이나 주가도 가능



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



- Truncation : 제한된 리소스(메모리) 내에서 모든 시퀀스를 학습할 수 없기때문에, 잘라서 학습에 사용하는 것

- 딥러닝 모델은 forward propagation을 통해 계산된 W를, backward propagation을 지나면서 W를 미분한 값인 gradient를 통해 학습하게 됨.
- BPTT란, Backpropagation through time의 줄임말로 RNN에서 타임스텝마다 계산된 weight를 backward propagation을 통해 학습하는 방식을 의미함.

Vanishing / Exploding Gradient Problem in RNN

vanilla RNN에서는 back propagation 동안, 동일한 행렬을 매 타임스텝마다 곱하게 됨.

→ Vanishing / Exploding Gradient → Long-Term-Dependency

Toy Example

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

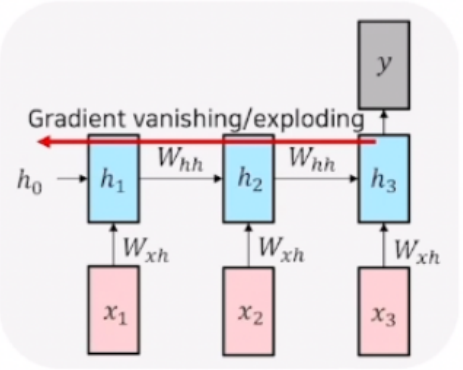
$h_3 = \tanh(2x_3 + 3h_2 + 1)$

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

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

...

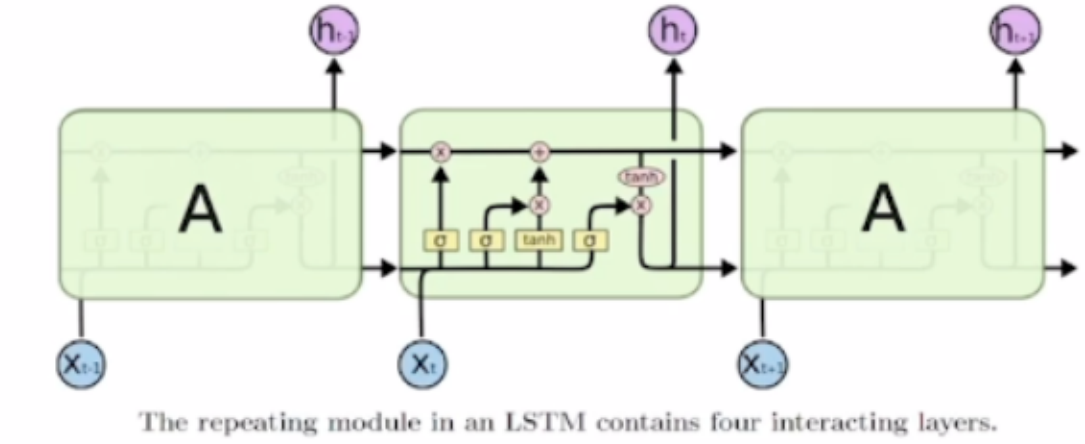
$h_3 = \tanh(2x_3 + 3\tanh(2x_2 + 3h_1 + 1) + 1)$



→ h_3을 h_1에 대해 미분하면, 3이 타임스텝만큼 거듭제곱.

Long Short-Term Memory (LSTM)

original RNN 의 Long term dependency를 해결하고자 만들어진 모델임.



The repeating module in an LSTM contains four interacting layers.

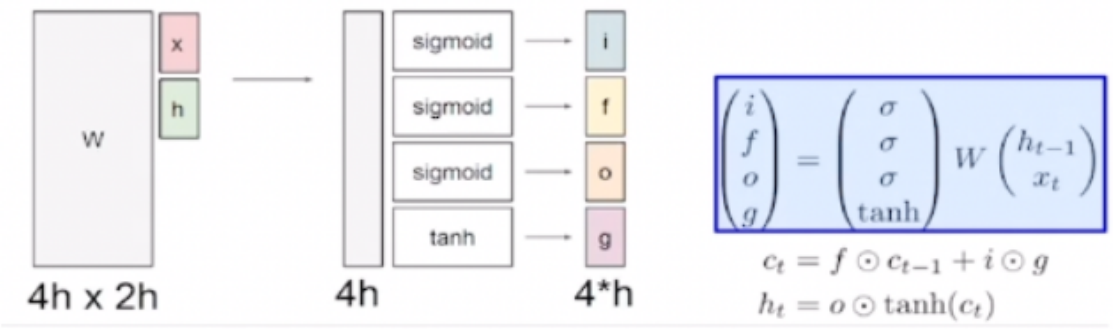
이전 스텝의 아웃풋으로 현재 스텝의 인풋으로 들어오는 두 가지

- c_t : cell state vector (핵심적 정보)

- h_t : hidden state vector (필터링된 정보를 담은 벡터)

$$\{C_t, h_t\} = LSTM(x_t, C_{t-1}, h_{t-1})$$

LSTM의 여러 게이트



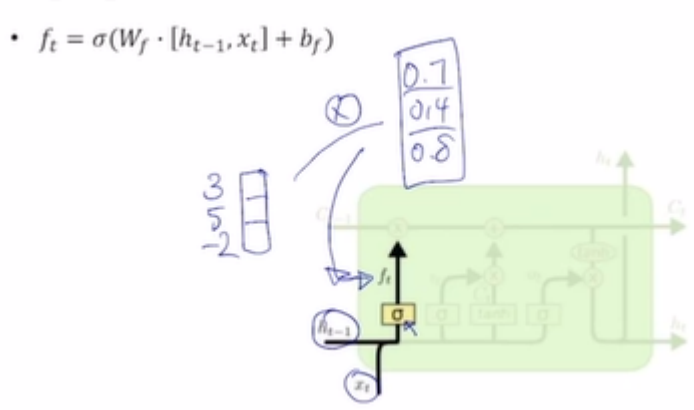
시그모이드를 통해 나온 벡터는 0과 1 사이의 값이 나옴.

→ 곱해지는 값을 얼마나 보존할지 결정

tanh를 토애 나온 벡터는 -1과 1 사이의 값이 나옴

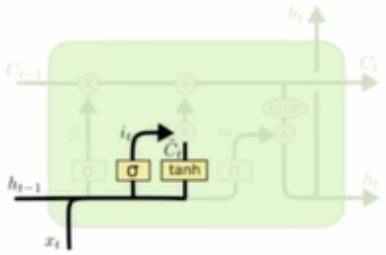
→ 현재 타임스텝에서 계산되는 유의미한 정보

- Forget gate



- Gate gate

- Generate information to be added and cut it by input gate
  - $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
  - $\widetilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
- Generate new cell state by adding current information to previous cell state
  - $C_t = f_t \cdot C_{t-1} + i_t \cdot \widetilde{C}_t$

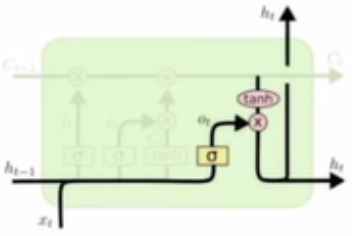


/colah.github.io/posts/2015-08-Understanding-LSTMs/

- Output gate

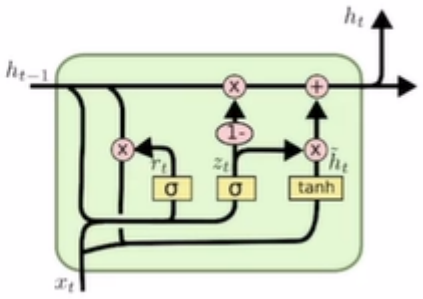
- Generate hidden state by passing cell state to tanh and output gate
- Pass this hidden state to next time step, and output or next layer if needed
  - $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$
  - $h_t = o_t \cdot \tanh(C_t)$

$-1 \sim 1$



Gated Recurrent Unit (GRU)

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



c_t가 따로 존재하지 않음.

h_t가 lstm의 c_t와 유사한 역할함. (중요한 정보 담음)

z_t : input gate

forget gate 대신 (1-input gate)를 사용하여 h_t*ht* 를 구할때 가중평균의 형태로 계산하게 됨.

→ 계산량과 메모리 줄임.

Backpropagation in LSTM, GRU

주된 정보를 담는 c_t가 업데이트 되는 과정이 동일한 가중치 행렬을 계속 곱해주는 것이 아니라, 타임스텝마다 다른 값으로 이루어진 forget gate를 곱해주거나 덧셈으로 원하는 정보를 만드는 것을 통해 경사 소실/ 폭발 문제가 사라짐.

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Backward flow of gradients in RNN can explode or vanish
- Common to use LSTM or GRU: their additive interactions improve gradient flow