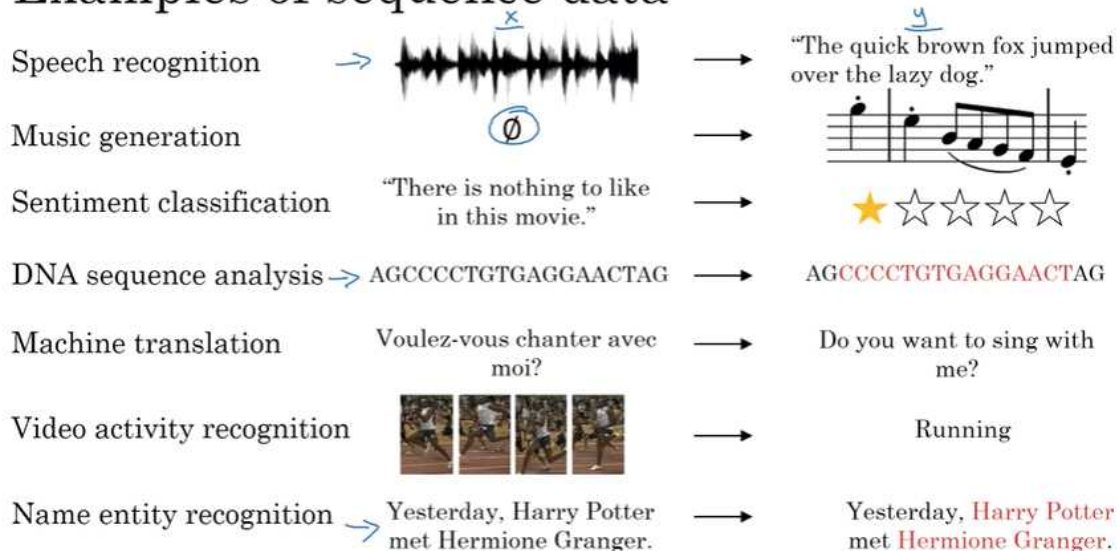


[Why sequence model]

Recurrent Neural Network(RNN) 모델은 음성 인식, 자연어 처리(NLP) 영역에 영향을 끼침

시퀀스 모델 사용 예시

Examples of sequence data



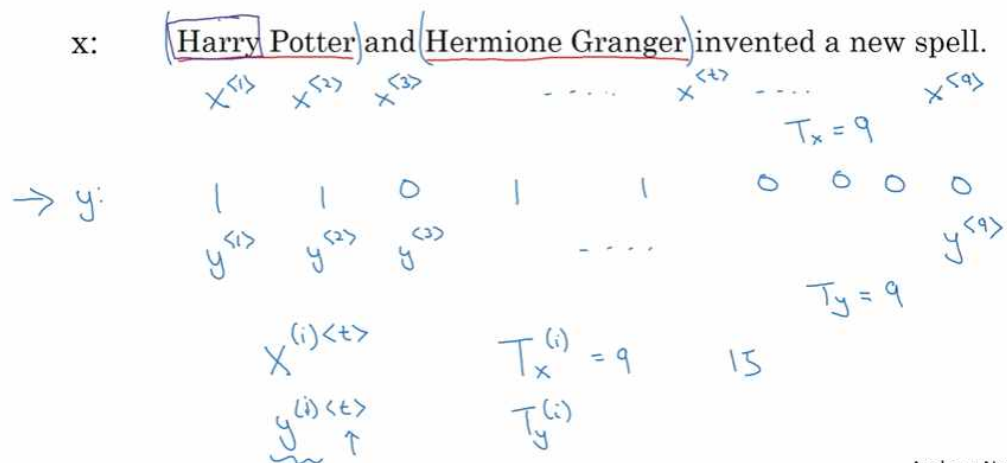
- 음성인식(Speech recognition): Input X인 오디오가 Text output Y에 매핑됨. 입력과 출력 모두 시퀀스 데이터(X: 시간에 따라 재생되는 음성, Y: 단어 시퀀스)
- 음악 생성(Music generation): Output Y: 시퀀스 데이터, X: 빈 집합이거나 단일 정수, 또는 생성하려는 음악의 장르나 원하는 음악의 처음 몇 개의 음
- 감정분류(Sentiment classification): X: 시퀀스 데이터, 주어진 문장에 대해서 Output: 별점
- DNA 분석: 염기서열 보고 단백질의 어느 부분과 상응하는지
- 기계번역, 비디오 동작 인식, 문장 내 이름 인식 ...

[Notation] - 시퀀스 데이터인 training set(x,y)를 묘사하는 방법

다음으로 RNN에서 사용되는 용어들을 정리

Motivating example

NLP



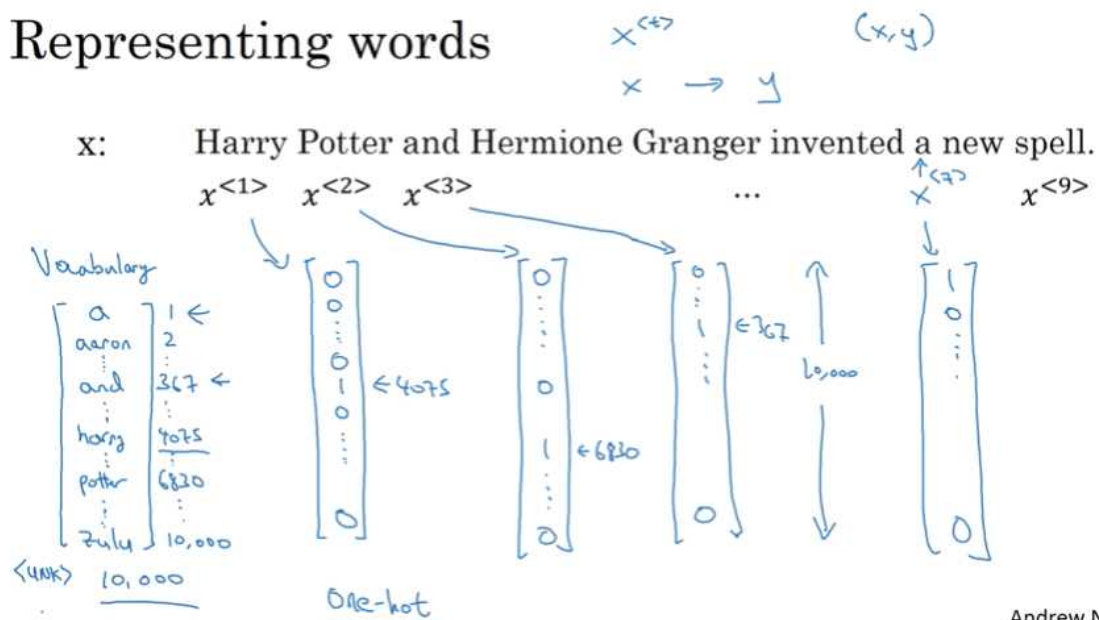
Andrew Ng

- 예를 들어, 'Harry Potter and Herimone Granger invented a new spell'라는 input이 있음
- 이 문장에서 자동으로 사람들의 이름을 인식(Name entity recognition)하는 시퀀스모델을 원할 때, 각 단어마다 output y 를 가지며, output 값은 이름인지 아닌지에 대한 1/0의 값을 가짐.
- 입력 시퀀스 x 는 9개의 단어로 구성된 feature set. 시퀀스의 순서를 나타내기 위해 $x^{<1>}, x^{<2>}, \dots, x^{<t>}, \dots, x^{<9>}$ 로 나타냄.(t 를 통해 순서의 중간을 나타냄(temporal sequence))
- output y 에 대해서도 $y^{<1>}, y^{<2>}, \dots, y^{<t>}, \dots, y^{<9>}$ 로 나타냄
- T_x : 입력 시퀀스의 길이, 여기서 $T_x=9$
- T_y : 출력 시퀀스의 길이(T_x 와 동일), T_x 와 동일한 값을 가질 수도 다른 값을 가질 수도 있음
- 여러 개의 training data가 있을 때, i 번째 training data는 $x^{(i) <t>}$ 로 나타냄 (t =sequence의 원소) - $x^{(i) <t>}$: i chain의 입력 시퀀스의 t 번째 원소
- i 번째 트레이닝 데이터의 길이: $T_x^{(i)}$, training set의 서로 다른 데이터의 길이는 다를 수 있음

NLP, 자연어 처리를 다루는 방법에 대해서 이야기해보자.

- NLP에서 먼저 결정해야할 것은 시퀀스 내 각각의 단어 표현 방법. 'Harry'라는 단어를 어떻게 표현해야할까?
- 시퀀스의 단어를 표현하기 위해서는 먼저 Vocabulary(or Dictionary)를 만들어야(선택해야) 함
- 예를 들어, Voca의 첫 번째 단어는 'a'이고, 두번째는 'Aaron', ... 'and', ... 'Harry', 'Potter'가 나오고 마지막에는 'Zulu'가 나올 수 있음 ('harry'는 4075번째, 'potter'는 6830번째)

Representing words



- 보통 상용 어플리케이션용의 사전은 대개 3~5만개의 단어를 포함. 여기서는 1만개를 사용한다고 가정 (단어를 고르는 방법은 자주 사용하는 단어 1만개를 고르거나, 다른 방법 사용 가능)
- 그 다음에 단어를 one hot encoding을 통해서 표현함
- ex) 'harry'를 나타내는 단어 $x^{<1>}$ 는 4075번째 위치만 1이고 나머지는 0인 벡터임(\because 사전의 4075번째 위치: 'harry'). $x^{<2>}$ 역시 6830번째만 1이고 나머지는 0인 벡터- 이 벡터는 1만차원의 벡터(사전이 만 개의 단어를 가질 때).

Andrew Ng

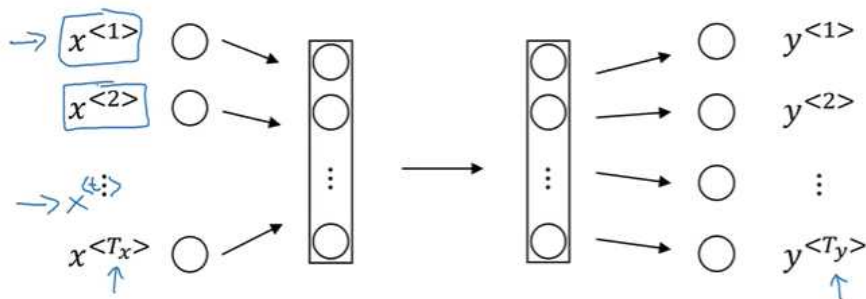
- 단어 표현 후, 다음 목표는 표현된 input x 를 output y 로 매핑하는 시퀀스 모델을 학습하는 것
- Supervised Learning으로 수행, x 와 y 를 포함하는 데이터(x, y) 사용
- 만약에 Voca에 없는 단어가 나오면?=> 새로운 Token이나 Unknown Word를 의미하는 가짜 단어를 만들면 됨. <UNK>와 같이 표시

[Recurrent Neural Network Model]

RNN 모델에 대해서 살펴보기 전에 기본적인 모델에 대해서 이야기해보자. - 이전 예제처럼 9개의 입력 단어가 있을 때, 9개의 단어를 입력으로 받는 모델

9개의 one-hot 벡터가 모델에 입력되고 몇 개의 hidden layer를 통해서 최종적으로 0 혹은 1의 값을 갖는 9개의 output이 출력.(1 = 사람 이름의 일부인지 판별)

Why not a standard network?



Problems:

- Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.

하지만 위와 같은 방법은 잘 동작하지 않음.

2가지 문제점

- 첫 번째: 입력과 출력 데이터의 길이가 트레이닝 데이터마다 다름. 모든 트레이닝 데이터가 같은 입력 길이 T_x 를 가지지 않고, 같은 T_y 를 가지지 않음. 입력에 임의의 값이나 0으로 채워서 동일한 길이로 맞출 수는 있지만 좋은 방법 X.
- 두 번째: 이와 같은 naive 신경망에서는 텍스트의 서로 다른 위치에 있는 동일한 feature(단어)가 공유되지 않음. 즉, 텍스트의 각 단어 위치에 관계없이(첫 번째나 t 번째나) 같은 단어가 나타나면 동일한 사람 이름이라고 추론해야함(일반적인 신경망에서는 단어의 위치에 따라 학습된 가중치가 달라, 다른 위치의 같은 단어가 다르게 해석될 수 O). 이는 CNN에서 학습한 이미지의 특성을 이미지의 다른 부분에 일반화시키는 것과 유사
- 또한, 이 경우에 각각의 단어($x^{<1>}$, $x^{<2>}$...)가 1만 차원의 one-hot vector이기 때문에 입력 layer가 매우 크고, 전체 입력 사이즈는 더 큼. 따라서 첫 번째 hidden layer의 weight matrix 파라미터의 수가 엄청나게 많아짐

Recurrent Neural Network(순환신경망)은 무엇일까? - 위의 두 문제점 해결

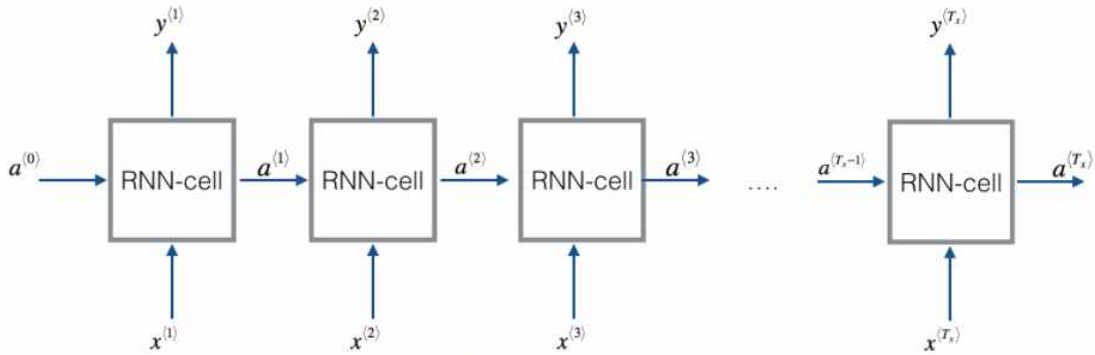
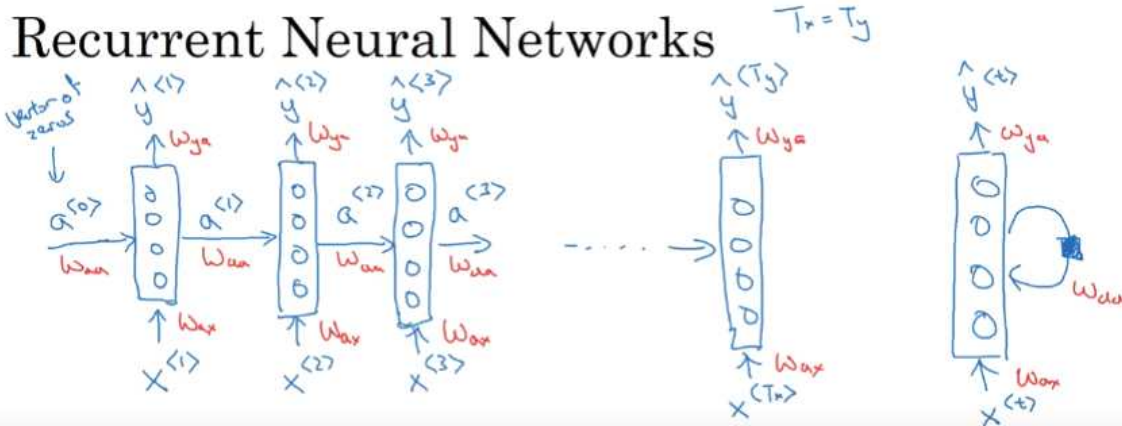


Figure 1: Basic RNN model

- RNN의 기본 구조



- 좌에서 우로 문장을 읽는다면 처음 읽는 단어 $x^{<1>}$ 를 신경망의 입력으로 사용
- 내부 신경망(hidden layer, output layer)를 통해 사람 이름의 일부인지 예측 $\hat{y}^{<1>}$
- 문장의 두 번째 단어 $x^{<2>}$ 를 읽고, $\hat{y}^{<2>}$ 를 예측할 때, $x^{<2>}$ 만 사용하는 것이 아니라, 첫 번째 단어로 연산한 정보의 일부를 사용(특히, 첫 번째 시점의 activation value가 전달됨)
- 동일하게 진행 ... -> 마지막 시점에서 $x^{<T_x>}$ 로 $\hat{y}^{<T_y>}$ 예측 (해당 예제에서는 $T_x = T_y$)
- 즉, 모든 각 step에서 다음 step으로 activation을 패스함
- RNN 신경망 학습을 시작하기 전에 $a^{<0>}$ 이 사용되는 것을 볼 수 있는데, 이것은 보통 무작위로 초기화하거나, 0으로 초기화함. 0으로 초기화하는 경우가 가장 일반적임.
- 위 이미지 오른쪽 주목) $x^{<T_x>} \rightarrow \hat{y}^{<T_y>}$. 루프 구조는 레이어가 스스로에게 연결되어 있고, shaded box는 한 step의 time delay를 나타내어 이전 시점의 hidden state가 현재 시점의 계산에 사용됨을 보여줌. 이는 모든 시간 단계에서 동일한 가중치 행렬을 사용하는 파라미터 공유를 의미함. (W_{aa}, W_{ax}, W_{ya} 가 사용됨. 모든 step의 입력에 가중치 W_{ax} 동일) (이 가중치가 어떻게 작용하는지는 다음 슬라이드에 설명)
- 위 기본 RNN 모델의 단점: 시퀀스에서 앞서 나온 정보만을 사용해서 예측을 한다는 것. $y^{<3>}$ 을 예측할 때, $x^{<4>}, x^{<5>}, x^{<6>}$ 의 정보를 사용 X. 이는 문제가 될 수 있음.

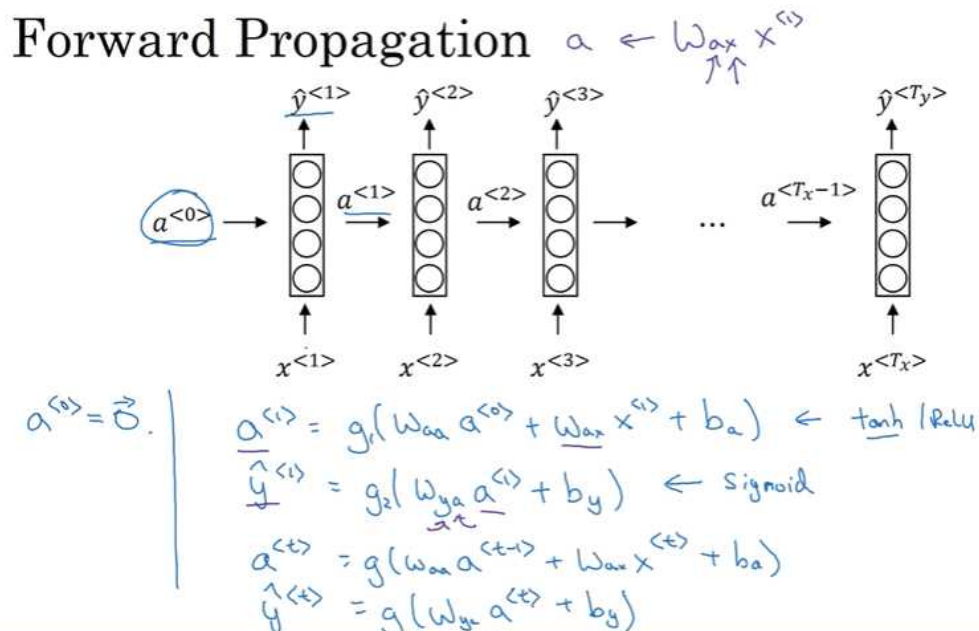
He said, "Teddy Roosevelt was a great President."

He said, "Teddy bears are on sale!"

위와 같은 문장에서, 첫 두 단어의 정보 뿐만 아니라 뒤의 나머지 단어들의 정보를 아는 것도 매우 유용함. 즉, 첫 두 세 마디만 고려한다면, Teddy라는 단어가 사람의 이름인지 확실하게 알 수 X
-> 양방향으로 반복되는 BRNN을 통해서 해결 가능

RNN의 Forward Propagation

아래는 신경망이 어떻게 계산되는지 보여줌



Andrew Ng

- 초기에 사용되는 $a^{(0)}$ 는 일반적으로 0 vector로 초기화
- $a^{(1)}$ 을 구하기 위해서 $W_{aa}a^{(0)} + W_{ax}x^{(1)} + b_a$ 에 activation function을 적용

$$a^{(1)} = g_1(W_{aa}a^{(0)} + W_{ax}x^{(1)} + b_a)$$

- activation function으로는 tanh나 ReLU 사용
- 첫 번째 단어의 예측은

$$\hat{y}^{(1)} = g_2(W_{ya}a^{(1)} + b_y)$$

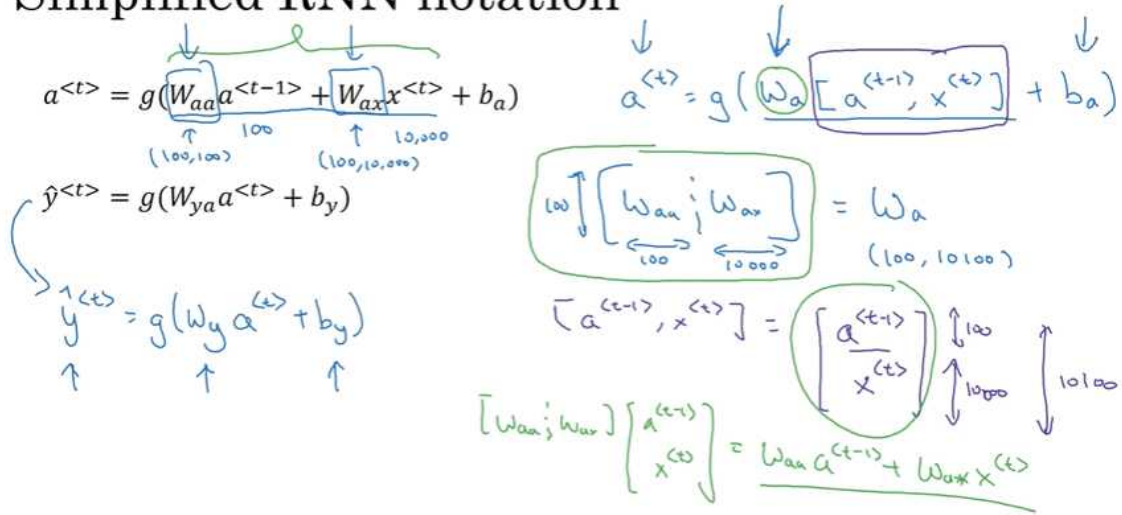
- 일반화하면

$$a^{(t)} = g(W_{aa}a^{(t-1)} + W_{ax}x^{(t)} + b_a)$$

$$\hat{y}^{(t)} = g(W_{ya}a^{(t)} + b_y)$$

위의 식 단순화

Simplified RNN notation



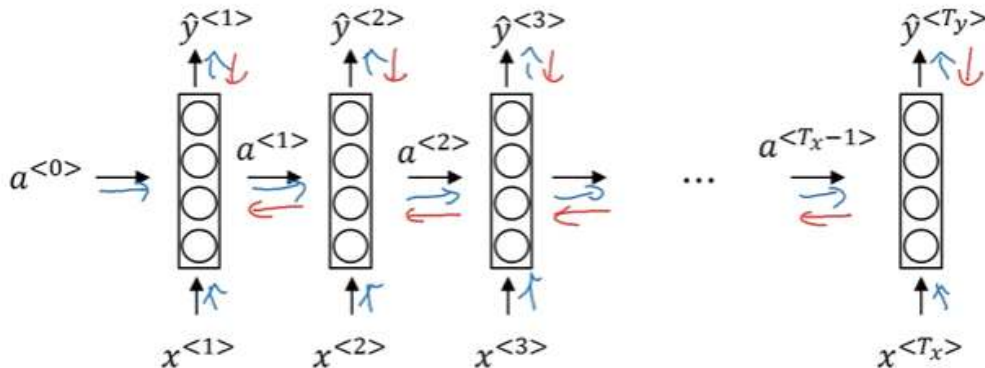
RNN의 hidden state 업데이트 수식: $a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$

- W_a 는 $[W_{aa} : W_{ax}]$ 를 의미하는데, 만약 a 가 100차원이고, x 가 10,000차원의 벡터라면 W_{aa} 는 (100, 100)의 차원을 갖고, W_{ax} 는 (100, 10000)의 차원을 가짐. 두 가중치를 연결하면 W_a 는 (100, 10100) 차원의 가중치가 됨
- * W_{aa} : 이전 hidden state와 현재 hidden state를 연결하는 가중치 행렬
- * W_{ax} : 현재 입력 $x^{<t>}$ 와 hidden state를 연결하는 가중치 행렬.
- W_a 와 연산되는 $[a^{<t-1>}, x^{<t>}]$ 는 $\begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$ 를 의미, 10100차원의 벡터가 됨(두 차원의 합)
- * $(a^{<t-1>})$: 이전 시점의 hidden state, $x^{<t>}$: 현재 입력

Output 계산: $\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$

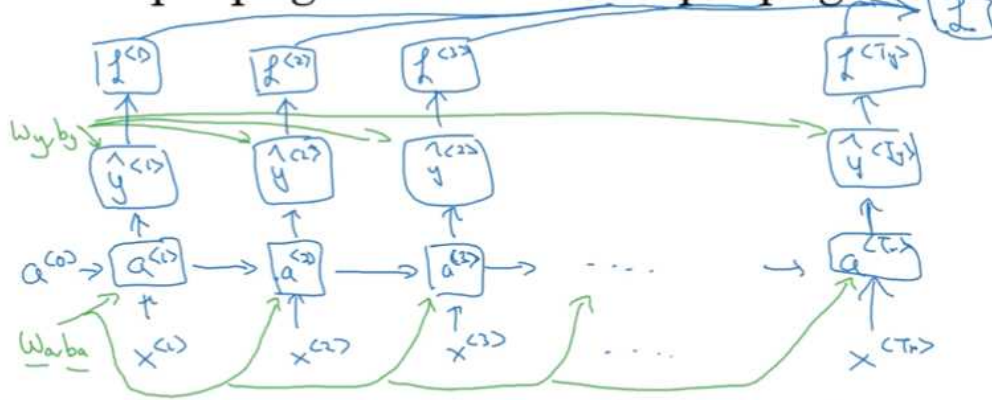
[Backpropagation through time]

다음으로 RNN의 Backpropagation(역전파)가 어떻게 동작하는지 대략적으로 살펴보자.



이미 알겠지만, BP는 FP의 반대방향으로 진행됨

Forward propagation and backpropagation



$$\mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)}) = -y^{(t)} \log \hat{y}^{(t)} - (1 - \hat{y}^{(t)}) \log (1 - \hat{y}^{(t)})$$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^T \mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)}) \leftarrow$$

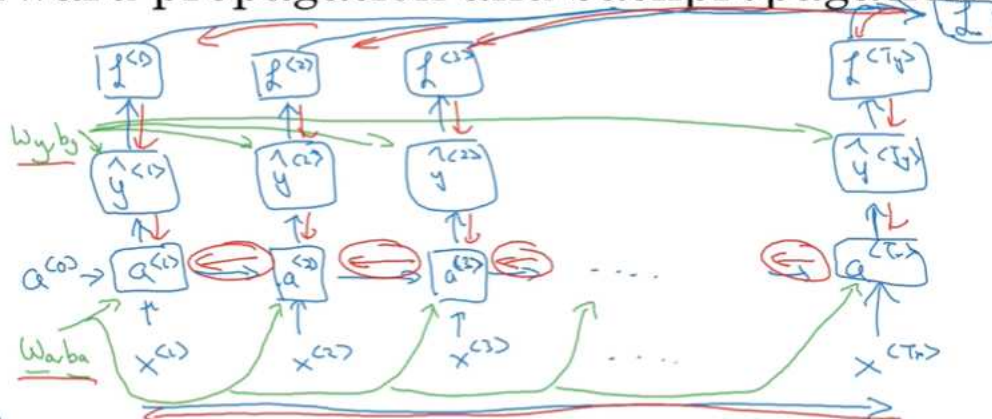
- 이전 강의 내용에서 살펴봤듯이, W_a, b_a 를 통해서 $a^{(t)}$ 를 구하고, W_y, b_y 통해서 예측 값 $\hat{y}^{(t)}$ 를 구함
- BP 계산을 위해 Loss function을 통해 실제 결과와 예측값의 Loss 계산 -> 총 Loss의 값 계산
- Cross Entropy Loss(Standard Logistic Regression Loss)를 사용할 수 있는데, 각 element의 Loss와 총 Loss는 다음과 같음

$$\mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)}) = -y^{(t)} \log \hat{y}^{(t)} - (1 - \hat{y}^{(t)}) \log (1 - \hat{y}^{(t)})$$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^T \mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)}) \leftarrow$$

- 최종 Loss는 시간별 개별 Loss의 합이다.
- 그 후 BP를 진행. BP는 오른쪽에서 왼쪽으로 향하는 계산임 - 파라미터 업데이트

Forward propagation and backpropagation

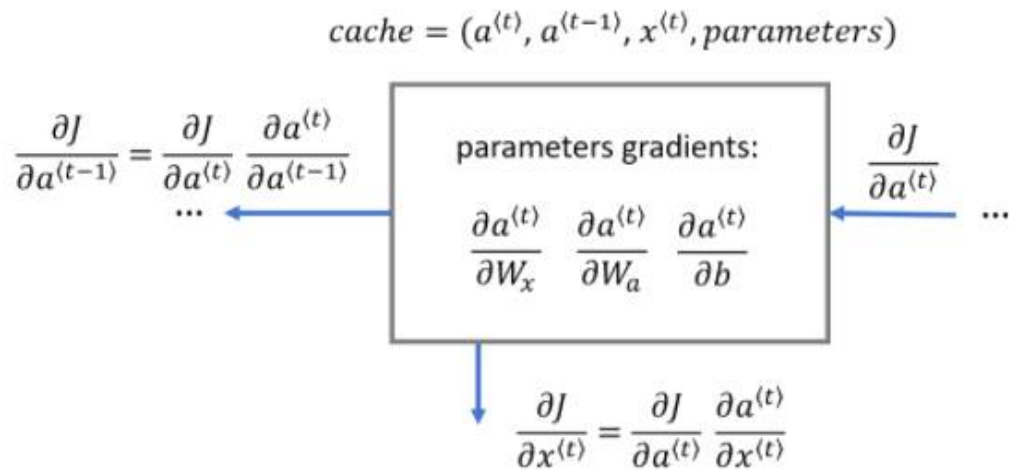


$$\mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)}) = -y^{(t)} \log \hat{y}^{(t)} - (1 - \hat{y}^{(t)}) \log (1 - \hat{y}^{(t)})$$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^T \mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)}) \leftarrow$$

Backpropagation through time

FP가 시간 순서로 전달하는 것과 반대로 BP는 시간을 거꾸로 가는 것과 같아서 Backpropagation through time(BPTT)라고 부름








34:58

[Difference types of RNNs]

지금까지 RNN 아키텍처에서는 T_x 가 T_y 와 동일한 경우를 살펴보았음

T_x, T_y 는 항상 같지 않을 수 있음. 다양한 RNN 아키텍처를 살펴보도록 하자.

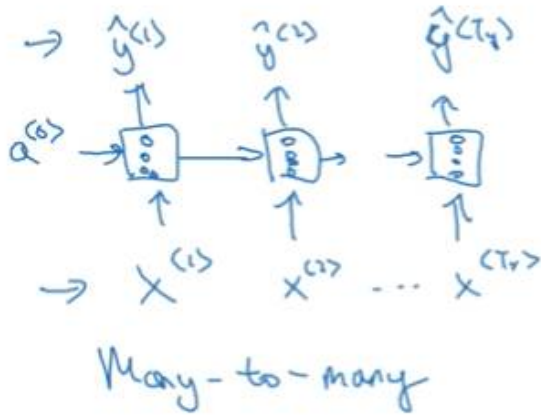
Examples of sequence data

Speech recognition		→	T_x "The quick brown fox jumped over the lazy dog."
Music generation		→	T_y 
Sentiment classification	"There is nothing to like in this movie."	→	y 
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger . <small>Andrew Mc</small>

이전에 본 시퀀스 데이터의 예시들

x, y 가 모두 시퀀스 데이터거나, x 가 빈 집합이거나, 하나의 입력일 수 있으며, y 가 시퀀스가 아닐 수도 있음.

$$T_x = T_y$$



Many-to-many 아키텍처

이전 예제들처럼, $T_x = T_y$ 이고

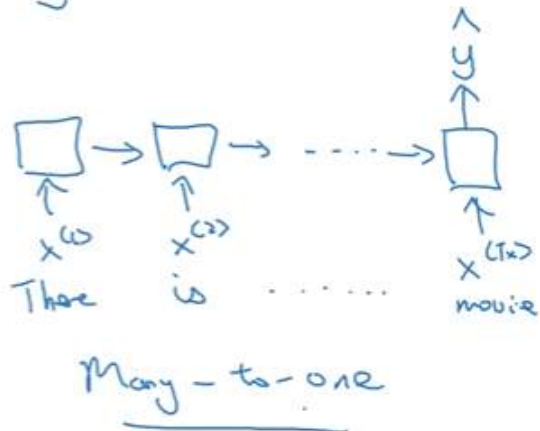
input sequence도 많은 words, output sequence도 많은 words

Many-to-one 아키텍처

예) Sentiment classification

x는 텍스트(영화리뷰)처럼 many words고,
y가 0/1의 값이나 1~5점의 평점인 하나의 숫자라면

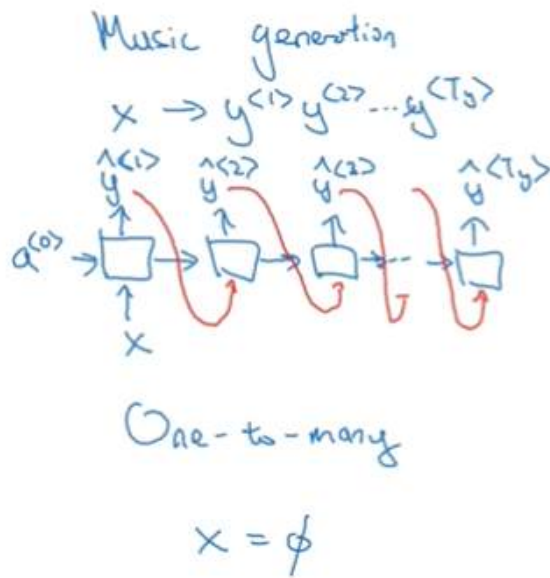
Sentiment classification
 $x = \text{text}$
 $y = 0/1 \quad 1 \dots 5$



one-to-one 아키텍처

단순한 신경망과 동일

One-to-one



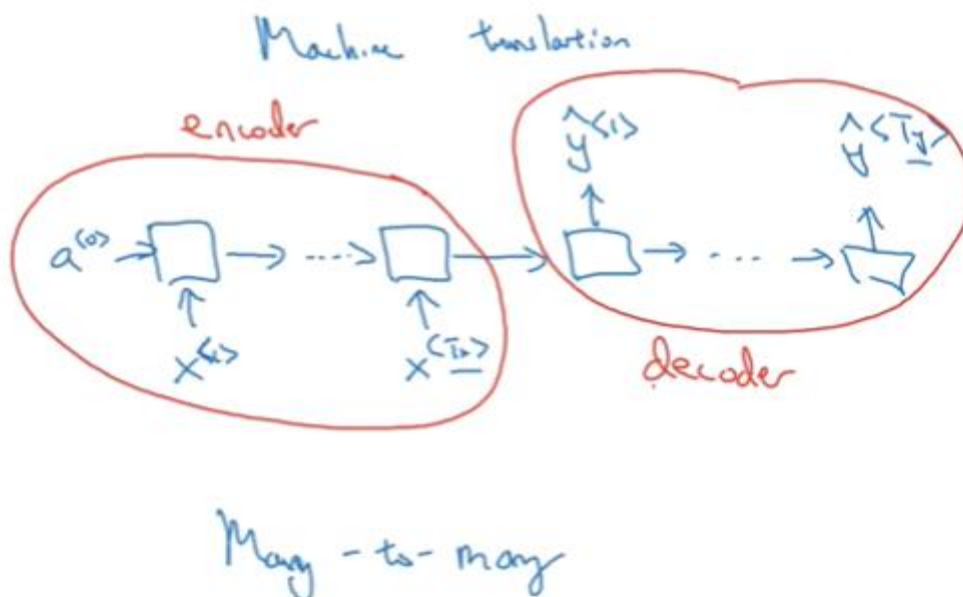
One-to-many 아키텍처

예) Music generation

입력이 원하는 장르가 무엇인지 알려주는 정수 일 수도 있고, 원하는 음악의 첫 음표일 수도 있으며, 어느 것도 입력하고 싶지 않다면 null 입력일 수도 있고, 항상 0일 수도 있음
다음과 같이 하나의 input x 와 시퀀스(set of notes) output y 를 가질 수 있는 모델이 됨

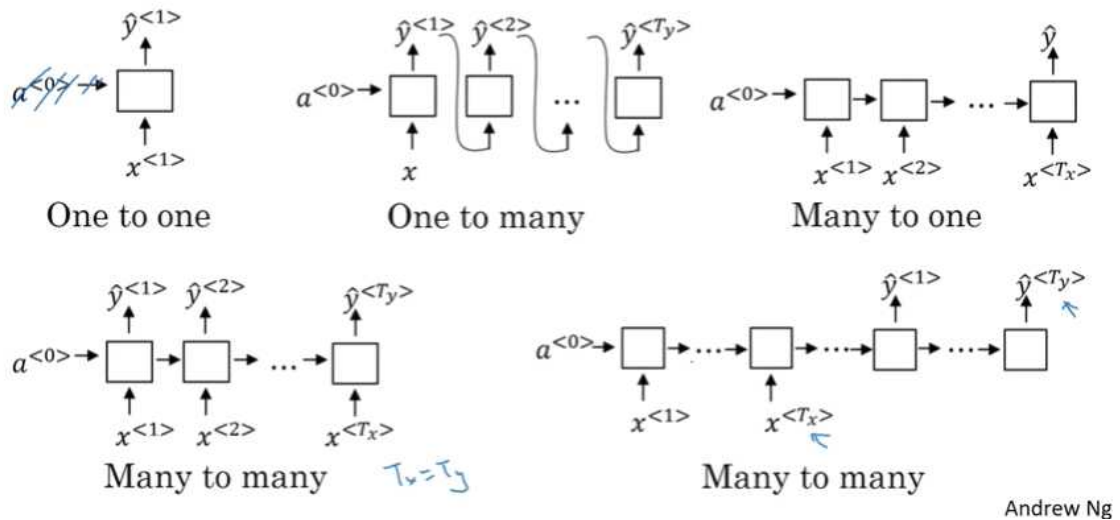
아키텍처는 이렇게 Many-to-many, Many-to-one, One-to-one, One-to-many으로 분류 가능.
Many-to-many에 흥미로운 예 하나 더 있음 - 입력과 출력 길이가 다름

기계번역(machine translation): Many-to-many 아키텍처, 입력과 출력의 길이가 다를 수 있다.
($T_x \neq T_y$)



위와 같은 모델에서 입력(한국어)을 받는 부분은 encoder, 그리고 출력(영어)을 내는 부분을 decoder라고 부름

Summary of RNN types



[Language model and sequence generation]

Language Modelling: Natural Language Processing에서 가장 기본적이고 중요한 작업 중 하나
RNN을 사용해서 Language model을 만드는 방법을 배워보자

language modelling(언어 모델)이란? - 우리가 음성 인식(Speech recognition) 시스템을 만들고 있다고 가정해보자.

What is language modelling?

Speech recognition

The apple and pair salad.

→ The apple and pear salad.

The apple and 페어 salad라는 문장을 들었을 때, 두 가지 문장 중 두 번째 문장이 더 가능성이 있다고 생각할 것이다. 두 문장이 정확히 똑같이 들리더라도 말이다!

speech recognition system은 두 문장의 확률이 얼마인지 알려주는 language model을 사용해 두 번째 문장을 선택할 것이다.

- 예를 들어, 모델이 첫 번째 문장에 대한 확률은 3.2×10^{-13} 으로 지정하고, 두 번째 문장의 확률은 5.7×10^{-10} 으로 지정. 따라서 음성 인식 시스템은 비교적 두 번째 문장을 선택
- 이러한 언어 모델은 RNN을 사용해서 만들 수 있는데, 이런 모델을 만드려면 large corpus of english text를 포함하는 training set이 필요 (corpus란 NLP 용어로 큰 영어 문장들을 의미)

$$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$$

$$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$$

$$P(\text{Sentence}) = ? \quad P(y^{<1>}, y^{<2>}, \dots, y^{<T_y>})$$

언어 모델의 기본 작업은 내 문장을 입력하는 것인데, 이 문장은 $y^{<1>}, y^{<2>}, \dots$ 로 표현
언어 모델에서 문장을 input x 가 아닌 output y 로 표시하는 것이 유용함 → 그 확률을 추정함

어떻게 language model을 만들 것인가?

Language modelling with an RNN

Training set: large corpus of english text.

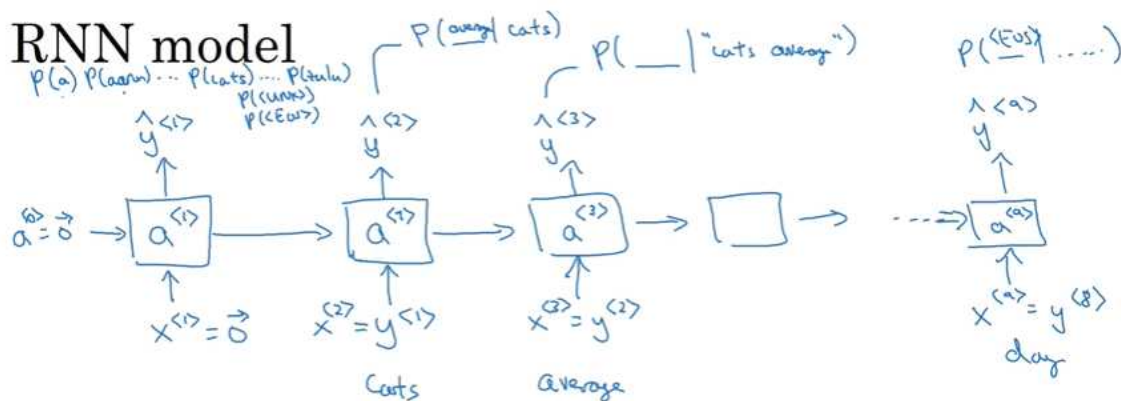
Tokenize
Cats average 15 hours of sleep a day. <EOS>
 $y^{(1)}$ $y^{(2)}$ $y^{(3)}$... $y^{(8)}$ $y^{(9)}$

The Egyptian Mau is a breed of cat. <EOS>

10,000
<UNK>
위와 같은 'Cats average 15 hours of sleep a day.'라는 문장이 있다고 해보자.

- 첫 번째로 해야 할 일: 이 문장을 토큰화하는 것. (토큰화를 통해서 단어들로 분리하고, 단어들을 Voca의 index를 통해 one-hot vector로 매핑함.)
- 추가 작업: training set의 모든 문장에 문장이 끝난다는 의미인 <EOS>라는 토큰을 추가함. (마침표를 토큰으로 사용할 수 있는데, 여기서 마침표는 무시함)
- 만약 아래 문장처럼 Mau라는 단어가 Voca에 없다면 <UNK>라는 토큰으로 대체 가능.

RNN 아키텍처는 다음과 같다.



→ Cats average 15 hours of sleep a day. <EOS>

- 첫 번째 $x^{(1)}$ 와 $a^{(0)}$ 은 0 vector로 설정하고, $a^{(1)}$ 과 $y^{(1)}$ 를 계산
- 여기서 $y^{(1)}$ 는 softmax output이며, 첫 번째 단어의 확률을 의미함. 이 예시에서는 소프트맥스에 상응하는 출력으로 첫 번째 단어 cats가 될 것임.
- 다음 RNN step이 진행되고 $a^{(1)}$ 를 가지고 다음 단계를 진행함. 여기서 $x^{(2)}$ 는 $y^{(1)}$ (= cats)가 되며, 모델에 입력함. 그리고 이 단계에서 softmax output에 의해서 다시 예측되고 그 결과는 첫 번째 단어가 cats일 때의 다른 Voca 단어들의 확률이 될 것임
- 세 번째 단계에서 $x^{(3)} = y^{(2)}$ 이고, 이전 단어 'cats average'를 가지고 세 번째 단어의 확률 예측
- 계속 진행하면 마지막 $y^{(9)}$ output을 내보내고, 이 값은 EOS가 될 것임.
- 이렇게 예측한 후에 Loss를 구하는데, 전체 Loss는 각 output의 Loss의 합이 됨.

$$\mathcal{L}(\hat{y}^{<t>, y^{<t>}) = - \sum_i y_i^{<t> \log \hat{y}_i^{<t>}$$

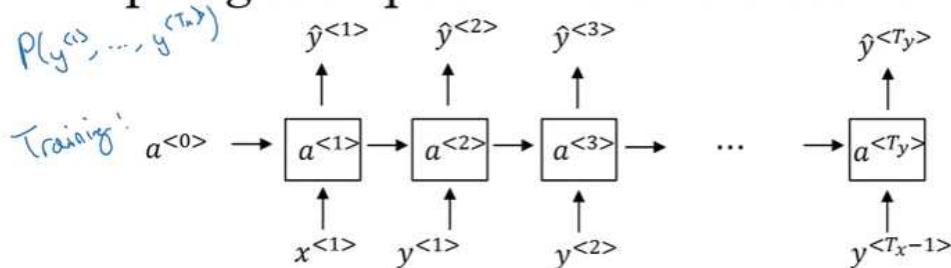
$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>, y^{<t>})$$

- 예를 들어 첫 세 개의 단어(Cats average 15)로 이루어진 시퀀스가 있다고 하면, 첫 번째 단계에서 softmax 함수를 통해 $y^{<1>}$ 의 확률 생성, 두 번째는 이전 단어 cats를 바탕으로 $y^{<2>}$ 의 확률 예측, 세 번째는 'average'를 바탕으로 $y^{<3>}$ 의 확률 예측.
 - 즉, $P(y^{<1>, y^{<2>, y^{<3>}) = P(y^{<1>})P(y^{<2>}|y^{<1>})P(y^{<3>}|y^{<1>, y^{<2>})$
 - 모델이 전체 시퀀스를 예측하는 과정은 각 시점의 조건부 확률을 곱한 형태로 나타냄
- $$P(y^{<1>, y^{<2>, \dots, y^{<T_y>}) = P(y^{<1>})P(y^{<2>}|y^{<1>})P(y^{<3>}|y^{<1>, y^{<2>}) \dots P(y^{<T_y>}|y^{<1>, \dots, y^{<T_y-1>})$$
- 즉, 각 시점에서 모델은 이전 단어와 hidden state 정보를 사용해 다음 단어의 확률을 예측

[Sampling novel sequences]

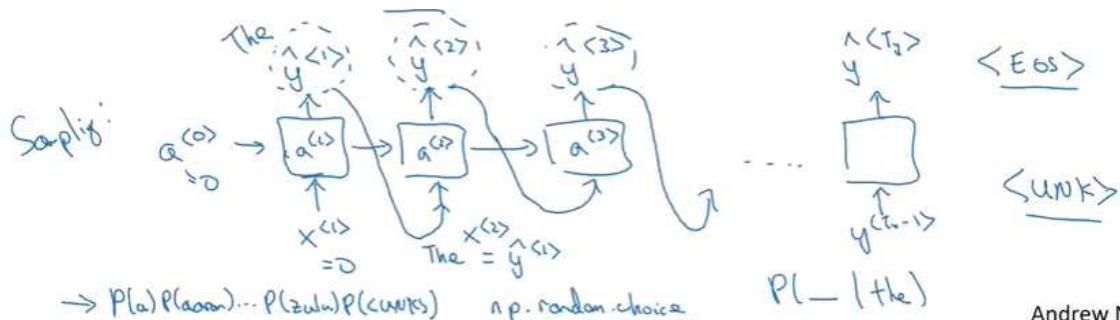
샘플링(sampling): 훈련된 RNN 모델로부터 시퀀스를 생성하는 과정 - 모델이 학습한 확률 분포를 바탕으로 새로운 시퀀스를 만들기 위해 사용됨

Sampling a sequence from a trained RNN



네트워크는 위와 같은 모델을 가지고 학습이 되었다.

그리고 sampling을 진행하는데, 샘플링의 과정은 다음과 같다.

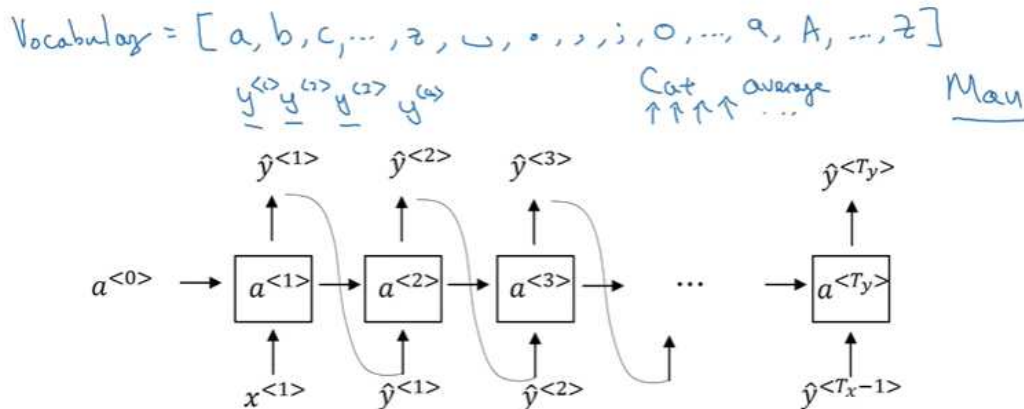


- $a^{<0>}$, $x^{<1>}$ 는 0 vector이고, 첫 번째로 softmax 확률분포인 $\hat{y}^{<1>}$ 를 출력함. 이 확률 분포를 기반으로 np.random.choice를 사용해 첫 번째 단어를 샘플링 함
- 두 번째 단계에서는 샘플링한 값을 $x^{<2>}$ 로 사용하고 softmax를 통해 다음 단어의 확률 출력
- EOS 토큰이 나오거나 설정한 시퀀스의 최대 길이에 도달할 때까지 위의 과정 반복
- 이것이 RNN 모델에서 무작위로 선택된 문장을 생성하는 방법임

지금까지 우리는 단어 수준의 RNN을 만들었지만, 문자(character) 수준의 RNN도 만들 수 있음

Character-level language model

Vocabulary = [a, aaron, ..., zulu, <UNK>] ←



- 즉, Training set는 개별 문자로 토큰화됨 (a, b, c, ... ,z, ,(space). , ;, A, ... ,Z) 등 텍스트의 각 문자를 하나의 토큰(token)으로 취급
- 문자 수준의 언어 모델을 사용하면 알 수 없는 단어에 대해 걱정할 필요가 없음 - unknown words나 specialized words에 적용 가능
- 하지만, 문자 수준은 입력이 훨씬 더 긴 sequences가 되기 때문에 문장의 초기 부분이 문장의 뒷부분에도 영향을 많이 줄 때의 성능이 단어 수준의 언어 모델보다 낮음

다음은 실제 언어 모델의 사례이다.

Sequence generation

News

President Enrique Peña Nieto, announced
sench's sulk former coming football langston
paring.

"I was not at all surprised," said hich langston.

"Concussion epidemic", to be examined. ←

The gray football the told some and this has on
the uefa icon, should money as.

Shakespeare

The mortal moon hath her eclipse in love.

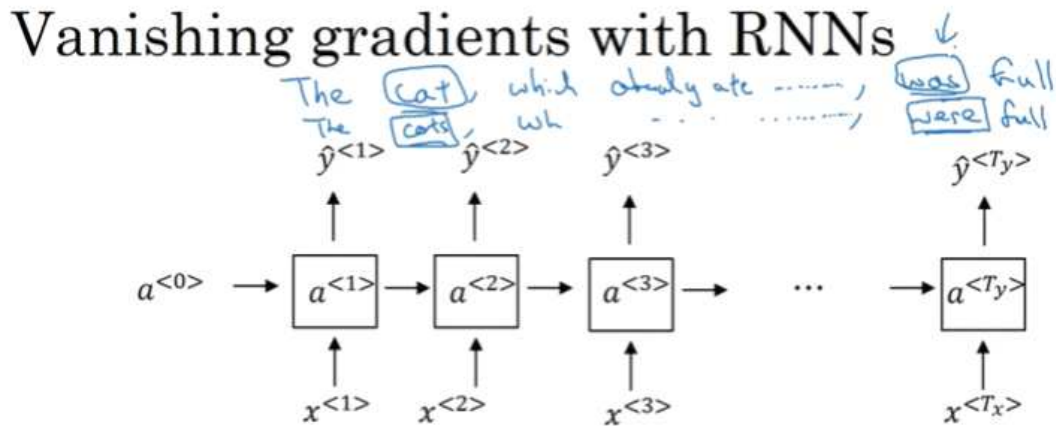
And subject of this thou art another this fold.

When besser be my love to me see sabl's.

For whose are ruse of mine eyes heaves.

뉴스 기사로 학습된 시퀀스 모델은 왼쪽과 같은 텍스트를 생성하고, 셰익스피어의 책에 의해 훈련된 모델은 셰익스피어가 쓴 것과 같은 텍스트를 생성

[Vanishing gradients with RNNs]

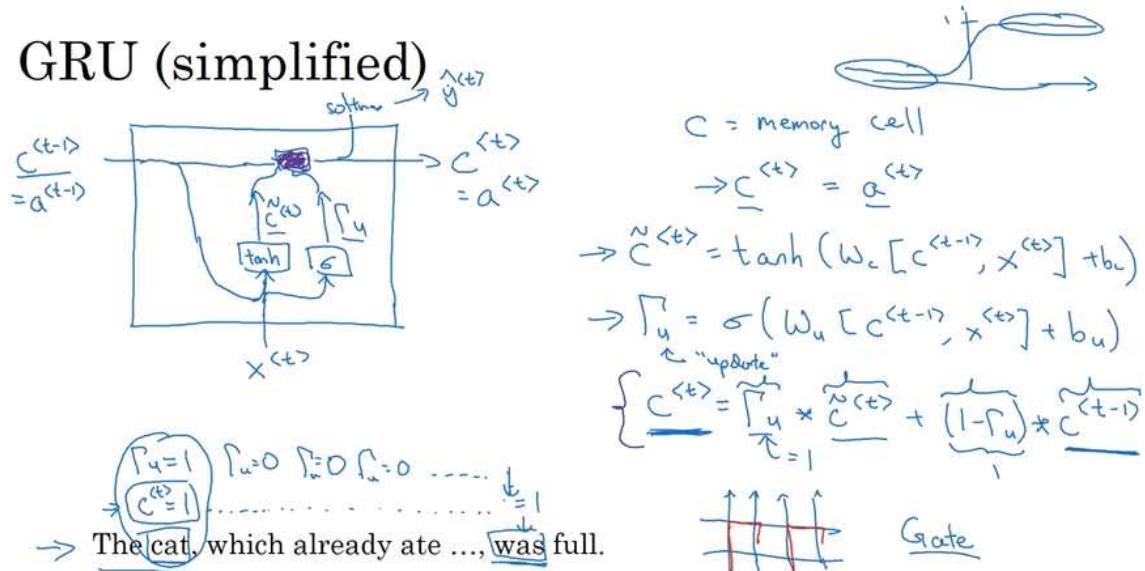


- Basic RNN 모델의 문제점 중의 하나: Vanishing gradient 문제가 발생할 수 있다는 것
- 아래와 같은 문장을 살펴보자.
'The cat, which already ate, was full'
'The cats, which already ate, were full'
- 예시는 문장의 단어가 long-term dependencies를 가지는 예시. 즉, 문장 초반부의 단어가 문장 후반부에 영향을 끼칠 수 있다는 것, RNN은 이와 같은 장기적인 의존성을 확인하는데 효과적이지 X (긴 시퀀스를 처리하는데 한계가 존재함)
- 즉, cat나 cats처럼 단수/복수 명사가 문장 초반에 존재했다는 것을 기억하고 있어야 제대로 된 예측을 할 수 있는데, RNN의 경우에는 가까이 있는 것에 영향을 더 많이 받음. $y^{<3>}$ 은 $y^{<3>}$ 에 가까이 있는 값(입력)에 영향을 받는다. => Basic RNN 알고리즘의 약점
- Deep layer NN에 대해서 언급할 때, gradient가 기하급수적으로 감소해서는 안되고, 또한 기하급수적으로 증가해서도 안된다고 했음. 여기서는 Vanishing gradient에 대해서 포커스를 맞추고 있지만, Exploding gradient가 최근 더 큰 문제로 대두됨. gradient가 너무 커져버리면, 신경망의 매개 변수가 완전히 엉망이 되기 때문-> gradient clipping을 통해서 어느 정도 해결 가능

[Gated Recurrent Unit(GRU)]

지금까지 Basic RNN이 어떻게 동작하는지 살펴보았고, 이번에는 조금 더 긴 시퀀스를 잘 캡처 (long-term dependencies 처리)하고, Vanishing Gradient 문제를 해소할 수 있는 GRU에 대해서 살펴보도록 하자. GRU는 LSTM과 유사하지만, 조금 더 간략한 구조를 가지고 있다.

GRU (simplified)



[Cho et al., 2014. On the properties of neural machine translation: Encoder-decoder approaches]

[Chung et al., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling]

Andrew Ng

c 는 memory cell을 의미하며 이전 정보들이 저장되는 공간이다. 이전 시점의 정보가 $c^{<t-1>} = a^{<t-1>}$ 를 통해 전달되고, 현재 input $x^{<t>}$ 와 결합하여 다음 time step으로 전달된다. 즉, 이전 예시처럼 문장 앞쪽의 단수/복수 정보가 뒤쪽까지 전달되어서 'was'를 예측하는데 영향을 끼칠 수 있게 된다.

여기서 크게 두 가지 연산 존재: 1) 이전 정보와 현재 입력을 통해서 $\tilde{c}^{<t>}$ 연산

$$\tilde{c}^{<t>} = \tanh(W_c [c^{<t-1>}, x^{<t>}] + b_c)$$

이 연산은 현재 time step에서 다음 time step으로 전달할 정보들의 후보군을 업데이트하는 것

2) Update gate Γ_u 를 통해 어떤 정보를 업데이트할지 결정하게 되는데(이전 정보의 중요도 조절), 아래와 같이 계산

$$\Gamma_u = \sigma(W_u [c^{<t-1>}, x^{<t>}] + b_u)$$

그리고 최종적인 메모리셀 $c^{<t>}$ 는 다음과 같이 계산

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

즉, 현재 time step에서 업데이트할 후보군과 이전 기억 정보들의 가중치 합으로 계산

여기서 Gate는 어떤 정보를 더 포함시킬지 결정하는 역할 (*는 element-wise 곱을 의미)

GRU는 reset gate와 update gate의 두 게이트만 사용하며, LSTM의 forget gate와 input gate의 기능을 결합함-> LSTM보다 파라미터의 수가 적고, 더 단순한 구조.

Full GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\underbrace{\Gamma_r}_{\text{reset gate}} * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

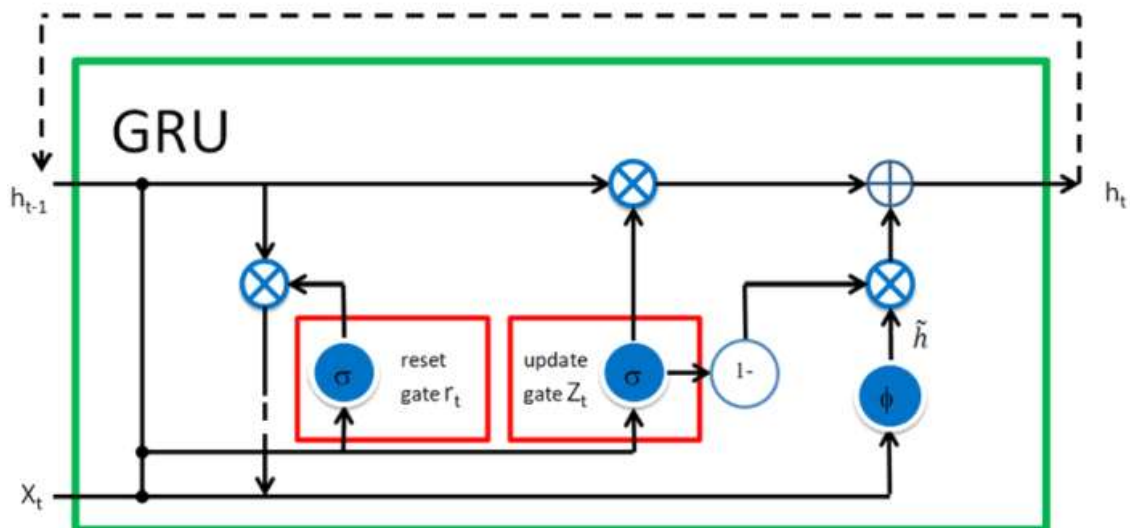
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

LSTM

The cat, which ate already, was full.

Full GRU에서는 Update gate 외에도 Reset Gate가 존재하며, 현재 시점에서 얼마나 이전 시점의 정보를 리셋할지 결정하는 게이트임. 값을 [0, 1] 범위로 조정하는 sigmoid 함수를 출력으로 사용해 이전 정보에 곱해주게 된다. (0에 가까우면 이전 정보가 무시되고, 1에 가까우면 이전 정보가 현재 입력에 큰 영향 끼침)

Full GRU는 다음과 같은 구조를 갖는다. 여기서 h는 c=a를 의미한다.(cell state와 hidden state가 통합)



[Long Short Term Memory(LSTM)]

LSTM은 GRU보다 일반적으로 많이 사용되는 장시간 단기 메모리 Unit (GRU보다 조금 더 복잡)

LSTM in pictures

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

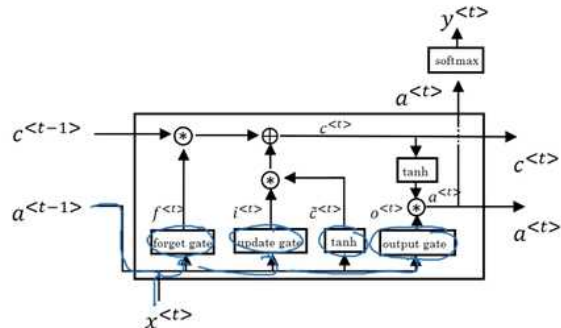
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

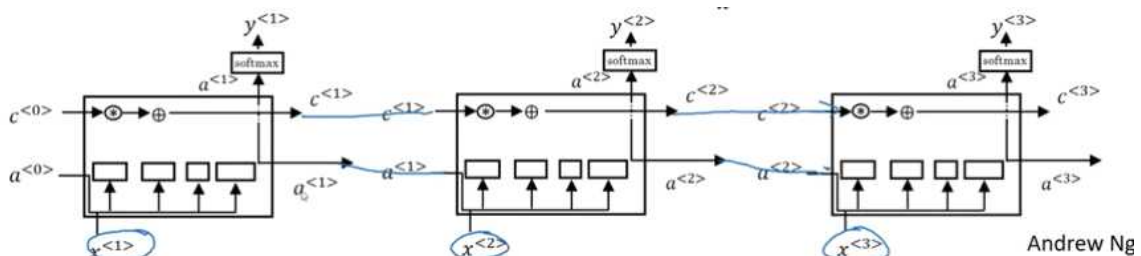


우선 이전 time step에서 전달받는 input이 $c^{<t-1>}, a^{<t-1>}$ 로 추가된다.

그리고 Forget gate, Update gate, Output gate를 통해서 각각의 연산을 수행하고, tanh(비선형 변환)를 통해서 $\sim c^{<t>}$ 를 연산. 현재 time step에서 다음 time step으로 업데이트할 정보들의 후보군을 의미한다.

핵심 아이디어는 현재 time step의 정보를 바탕으로 이전 time step의 정보를 얼마나 잊을지 정해주고(Forget gate), 그 결과에 현재 time step의 정보의 일부(Forget Gate와 Update Gate의 값 곱합)를 더해서 다음 time step으로 정보를 전달한다.

LSTM은 다음과 같은 과정을 통해서 계산된다.



GRU와 LSTM을 비교했을때, 어느 것이 우월하다고 할 수 없고, 강점을 보이는 분야가 조금 다르다. LSTM이 조금 더 먼저 발견되었고, GRU는 비교적 복잡한 LSTM 모델을 단순화한 것이다.

GRU의 장점은 LSTM에 비해서 훨씬 단순하고, 따라서 훨씬 더 큰 네트워크를 만드는 것이 쉽다. (두 개의 gate만 존재하기 때문에 연산량이 적다.)

LSTM의 다소 모델이 크지만, 3개의 gate가 존재해서, 보다 성능이 좋고 효과적일 수 있다.

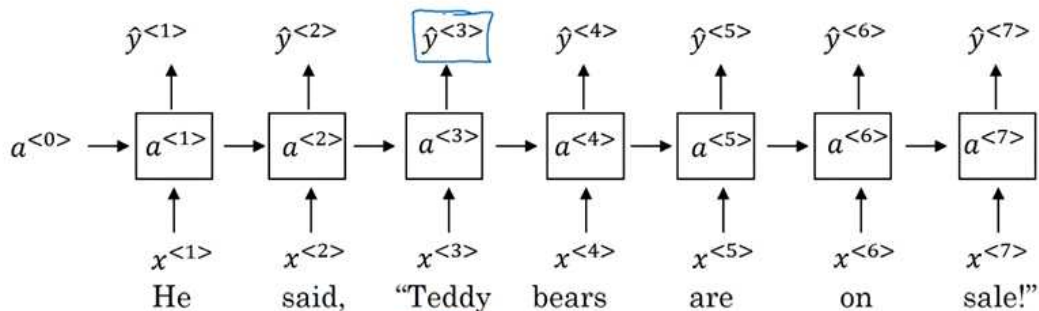
[Bidirectional RNN(BRNN)]

지금까지 RNN의 가장 기본적인 building block들을 살펴보았다. 추가로 훨씬 더 강력한 모델을 만들 수 있는 두 가지 아이디어가 있는데, 1) Bidirectional RNN(BRNN): 주어진 시퀀스에서 이전 시점과 이후 시점의 모든 정보를 사용해 예측 수행. 2) deep RNNs

Getting information from the future

He said, "Teddy bears are on sale!"

He said, "Teddy Roosevelt was a great President!"

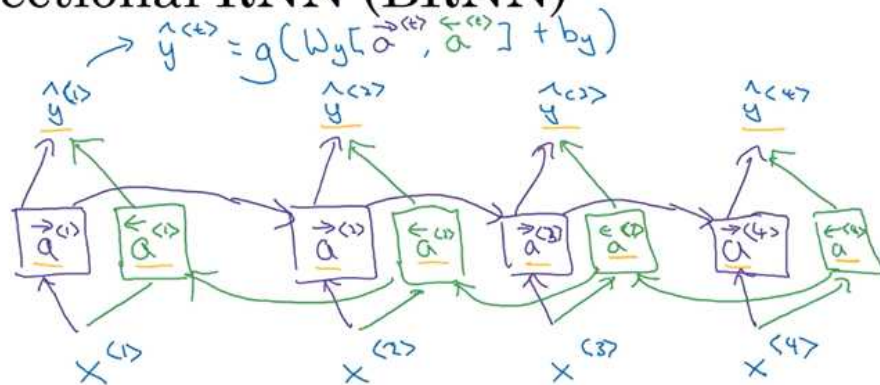


단방향 RNN 모델은 이전 시점의 정보만을 사용해 현재 시점의 예측을 수행 - 세번째 단어인 Teddy가 이름인지 아닌지 확인하기 위해 첫번째, 두번째 단어만으로는 충분하지 않고, 더 많은 정보(미래)가 필요하다.

이런 경우에 BRNN이 위 문제를 해결할 수 있다.

예를 들어 4개의 입력(4개의 단어로 이루어진 문장)이 있다고 가정해보자.

Bidirectional RNN (BRNN)



Acyclic graph

BRNN은 위와 같은 구성을 가진다.

- Forward RNN(보라색): 입력 시퀀스를 앞에서부터 뒤로 순서대로 읽으며 hidden state를 계산
 - Backward RNN(초록색): 입력 시퀀스를 뒤에서부터 앞으로 거꾸로 읽으며 hidden state를 계산
- 두 RNN의 hidden state를 결합하여 최종 예측을 수행-> 과거와 미래의 정보가 모두 고려된 hidden state를 얻을 수 있다.

Bidirectional 구조를 구성할 때 그 내부의 RNN 셀이 어떤 형태의 RNN이든 사용할 수 있다

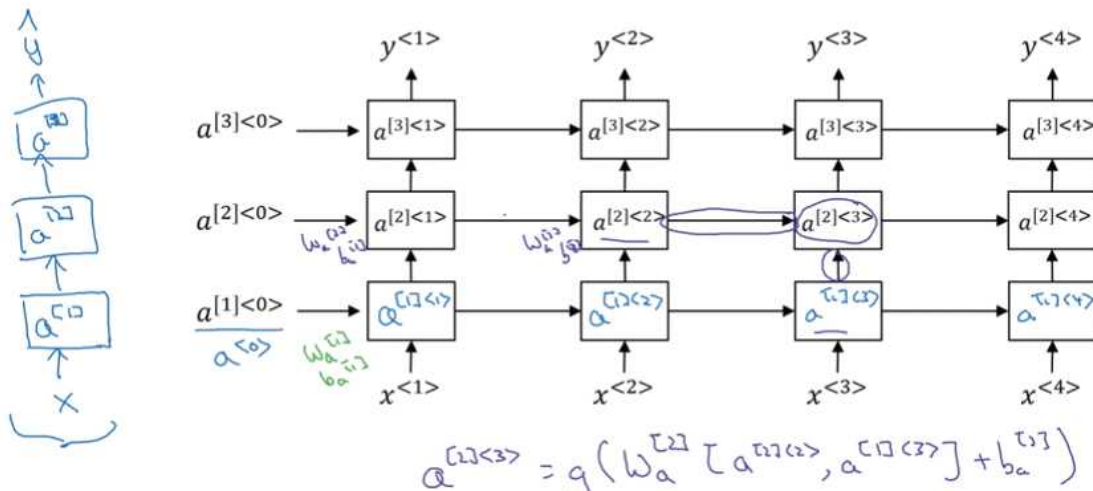
이처럼 BRNN을 통해서 전체 시퀀스의 정보들을 참조해서 정확하게 예측할 수 있지만, 단점은 예측을 하기 전에 전체 데이터 시퀀스가 필요하다는 것이다.

예를 들어, 음성 인식 시스템을 만들어서 연설을 입력받을 때, 연설이 끝날 때까지 기다렸다가 그 연설 데이터를 처리해서 예측할 수 있을 것이다. (실시간으로 순차적인 예측을 수행하기 어려움)

[Deep RNNs]

지금까지 본 단일 층 RNN이 시퀀스의 패턴을 잘 학습할 수 있지만, 더 복잡하고 추상적인 특징을 학습하기 위해 여러 층으로 구성된 RNN을 사용할 수 있다.

Deep RNN example



여기서는 3개의 layer로 쌓았지만, 더 많이 쌓을 수 있다.

1) 수평의 정보 흐름 (시간 축):

각 층의 RNN은 시퀀스의 시간에 따른 변화를 처리하며, 각 시점 t 의 hidden state를 이전 시점의 hidden state를 기반으로 업데이트한다.

2) 수직의 정보 흐름 (층 간 연결):

각 층의 RNN은 그 이전 층의 hidden state를 입력으로 받아 다음 층의 hidden state를 계산한다. 이를 통해 데이터의 추상화 레벨이 층을 거치며 점점 깊어진다.

따라서, 위와 같은 문제를 해결하기 위해서 Word Embedding을 통해서 각각의 단어들에 대해 features와 values를 학습하는 것이 필요하다.

Word Embedding: 각 단어를 밀집된 저차원 벡터로 표현하는 방법으로, 단어 간의 의미적 관계와 유사성을 수치적으로 나타낼 수 있도록 학습된다.

Featurized representation: word embedding

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
size
cost
alike
verb

I want a glass of orange juice.
 I want a glass of apple juice.

Andrew Ng

- 단어 임베딩은 위와 같이 단어와 특징을 매핑하는 행렬(matrix)를 갖는다. row: feature(Gender/Royal/Age/... 등) col: Vocabulary에 존재하는 단어들
- Man은 Gender에 해당하는 값이 -1에 가깝고, Woman은 1에 가까운 것을 볼 수 있다. 서로 반대되는 개념이기 때문에 두 합이 0에 가깝게 되는 것이다. Apple이나 Orange의 경우에는 성별과 거의 연관이 없기 때문에 Gender에 해당되는 값이 0에 가까운 것을 확인할 수 있다.
- 이처럼 워드 임베딩을 통해서 각 특징에 대한 값들을 갖는 임베딩 행렬(Embedding Matrix)를 얻을 수 있다.
- Man의 임베딩 벡터는 e_{5391} 로 표현할 수 있고, e는 embedding을 의미한다.

I want a glass of orange juice.

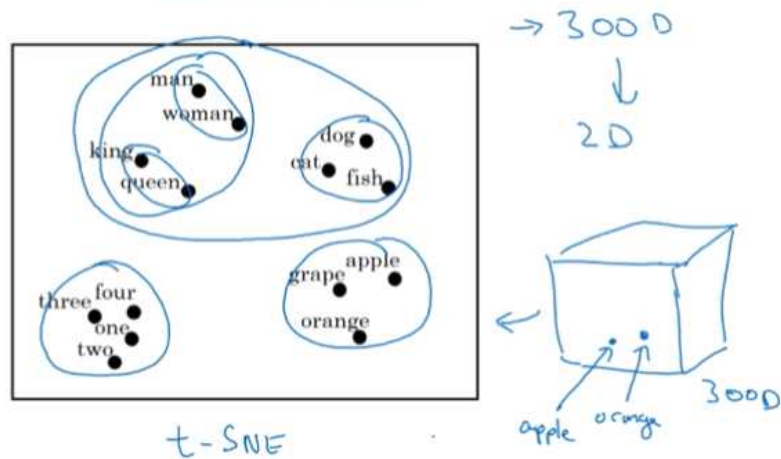
I want a glass of apple _____.

워드 임베딩을 통해서 juice 앞의 apple과 orange과 유사하다는 것을 추론할 수 있기 때문에(유사한 벡터로 표현되어있음) apple 다음에 juice가 온다는 것을 더 쉽게 예측할 수 있다.

다만, 워드 임베딩에서 각 row가 어떤 특징을 의미하는지 해석하기는 어렵지만, one-hot encoding보다 단어 간의 유사점과 차이점을 더 쉽게 알아낼 수 있는 장점을 갖고 있다.

보통 feature의 개수로는 300 dimension을 많이 사용한다.

Visualizing word embeddings



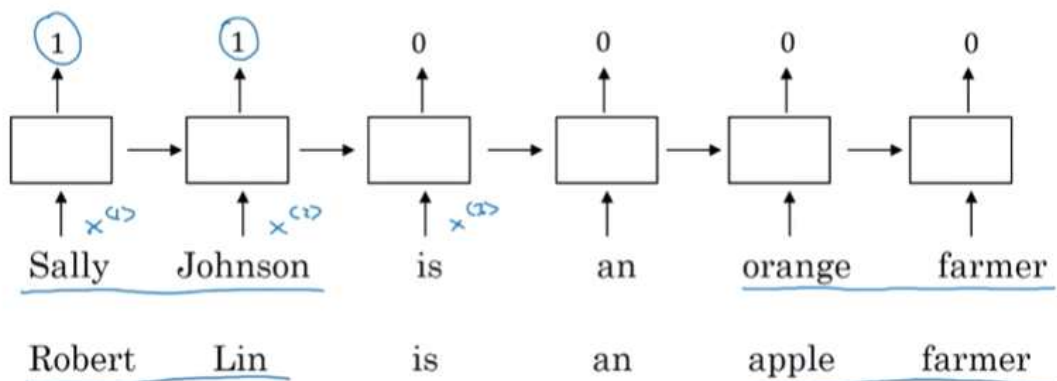
일반적으로 Word Embedding은 300차원 벡터로 표현되는데, 이는 단어 간의 다양한 의미적 관계를 나타내기에는 적합하지만 인간이 이를 직접 이해하고 시각화하기에는 어렵다.

300차원의 Word Embedding 행렬을 조금 더 쉽게 이해하기 위해서 차원을 축소하여 저차원으로 변환하여 시각화해야 한다. - 각 단어의 벡터가 어떤 의미적 관계를 가지는지 파악 가능
이 시각화 작업을 위해 사용되는 알고리즘은 t-SNE 알고리즘이다.

t-SNE 알고리즘: 임베딩 행렬을 더 낮은 차원으로(여기서는 300차원을 2차원(2D)으로) 축소함. 유사한 단어는 가까이, 서로 다른 단어는 멀리 떨어지도록 배치한다. -> 단어 간의 유사성이나 의미적 관계를 시각적으로 쉽게 파악할 수 있도록 해준다.

[Using word embeddings]

Named entity recognition example




문장에서 이름이나 특정 개체를 인식하는 예제를 살펴보자.

'Sally Johnson is an orange farmer'라는 문장에서 Sally Johnson이 이름이라는 것을 확실하기 위한 방법 중의 하나는 orange farmer가 사람임을 알아내는 것이다. one-hot encoding이 아닌 word embedding을 사용해서 학습한 후에, 새로운 example 'Robert Lin is an apple farmer'에 대해서 apple이 orange와 유사하다는 것을 알기 때문에 Robert Lin이 사람 이름이라는 것을 더 쉽게 예측할 수 있다.

만약 apple farmer가 아닌 많이 사용되지 않는 과일인 'durian cultivator'라면 어떻게 될까?
training set의 수가 적고, training set에 durian과 cultivator가 포함되지 않을 수 있다. 하지만, durian이 과일이라는 것을 학습하고, cultivator가 사람을 나타낸다는 것을 학습한다면 training set에서 학습한 orange farmer에서 학습한 것을 durian cultivator에도 일반화하게 될 것이다.
단어 임베딩이 이렇게 일반화할 수 있는 이유 중의 하나는 단어 임베딩을 학습하는 알고리즘이 매우 큰 단어 뭉치들을 통해서 학습하기 때문이다.(10억~1000억개의 단어)

만약 충분한 크기의 training set이 없더라도, 이미 다른 대규모 데이터셋으로 학습된 Word Embedding을 가져와 사용할 수 있다. 이를 Transfer Learning이라고 한다.
Transfer Learning을 통해 이미 학습된 임베딩을 적용하면, 새로운 도메인이나 적은 데이터에서도 좋은 성능을 보여줄 수 있다.(꽤 좋은 성능을 보여준다)

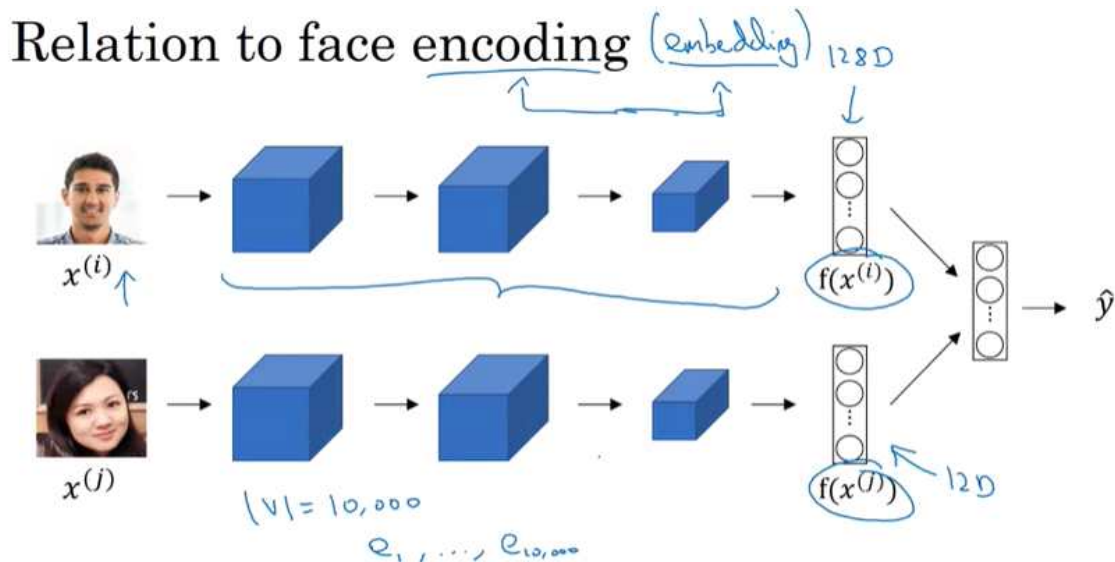
Transfer learning and word embeddings

- 
1. Learn word embeddings from large text corpus. (1-100B words)
(Or download pre-trained embedding online.)
 2. Transfer embedding to new task with smaller training set.
(say, 100k words) → 10,000 → 300
 3. Optional: Continue to finetune the word embeddings with new data.

일반적인 NLP 작업에서 단어임베딩은 위와 같은 과정으로 진행된다. 하지만, Language model이나 Machine Translation에는 유용하지 않다. 앞서 배운 Transfer Learning처럼, A의 data는 많고 B의 data는 적을 때 더욱 효과적이다.

1. 대규모 코퍼스에서 Word Embedding 학습: 방대한 텍스트를 통해 단어 간의 의미적 관계와 맥락을 학습.
2. 임베딩을 새로운 과제에 전이: 사전 학습된 임베딩을 가져와 작은 데이터셋이 있는 새로운 과제에 적용 - 기존에 학습한 단어의 관계와 맥락을 새로운 데이터셋에서도 효과적으로 활용할 수 있음
3. 필요에 따라 임베딩 미세 조정(Fine-Tuning): 새로운 과제의 데이터에 맞게 -> 더 나은 성능

단어 임베딩은 Face encoding과 유사하다고 볼 수 있다.



Face Encoding과 Word Embedding은 모두 원래의 고차원 데이터를 저차원 벡터로 표현하는 방법으로, 그 안의 의미적 유사성과 관계를 벡터 공간에서 학습한다. Face Encoding은 얼굴 이미지의 특징을 인코딩하고, Word Embedding은 단어의 의미적 특징을 인코딩하는 과정이지만, 유사성을 인코딩하고 유사한 것들은 가까운 벡터로 배치한다는 점에서 그 원리가 유사하다.

- 위 사진에서 이미지를 128D vector로 변환해서 이미지들을 비교한다.
- 한 가지 차이점은 사진의 경우에는 처음 보는 이미지더라도 벡터로 encoding이 되지만, 단어 임베딩의 경우에는 사전에 정의된 단어들만 인코딩할 수 있다. 어휘 사전에 없는 단어는 UNK로 표현된다. 즉, 정해진 단어만 학습한다는 의미이다.

[Properties of word embeddings]

Analogies

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

Handwritten notes below the table:

- e_{5391} (under Man), e_{9853} (under Woman)
- e_{man} (under Man), e_{woman} (under Woman)
- $e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
- $e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
- $e_{man} - e_{woman} \approx e_{king} - e_{?}$ (with e_{queen} written below the question mark)
- $Man \rightarrow Woman \approx King \rightarrow ? \rightarrow Queen$

[Mikolov et. al., 2013, Linguistic regularities in continuous space word representations]

Andrew Ng

단어임베딩의 중요한 부분은 이것이 유추에 큰 도움을 준다는 것이다. 이에 대한 예제를 통해서 단어 임베딩의 특징을 살펴보자.

'남자(man)와 여자(woman)는 왕(king)과 ____과 같다'라는 유추 문제가 있을 때, 어떻게 예측할 수 있을까? 위 표와 같이 Man은 4차원 벡터로 표현되며, $e_{5391} = e_{man}$ 으로 나타낼 수 있다. 그리고 woman의 임베딩은 e_{woman} 으로 표현하며, king/queen도 동일하게 표현된다. (실제로는 50~1000 차원을 사용한다)

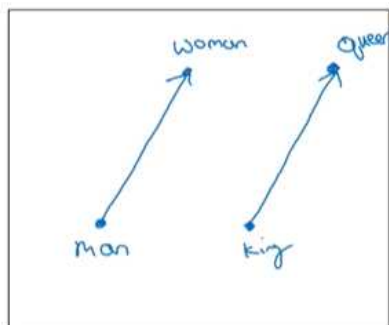
그리고 $e_{man} - e_{woman}$ 과 $e_{king} - e_{queen}$ 을 시행해보면 다음과 같은 흥미로운 결과를 얻을 수 있다.

$$e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

"man"과 "woman"의 벡터 차이와 "king"과 "queen"의 벡터 차이가 유사하다는 것을 발견함으로써, 두 관계가 유사하다는 것을 추론할 수 있다.

직관적으로 이해하기 위해서 아래 슬라이드를 보자.

Analogy using word vectors



300D

Find word w : $\arg \max_w$

$$e_{man} - e_{woman} \approx e_{king} - e_w$$

$$\sim \text{sim}(e_w, e_{king} - e_{man} + e_{woman})$$

30-75%

Andrew Ng

Word Embedding에서는 각 단어가 고차원 벡터 공간(예: 300차원)에서 점으로 표현된다.

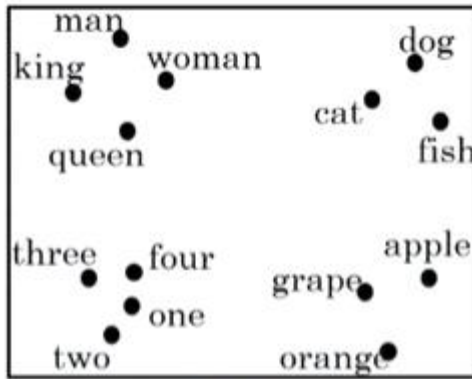
그리고, man과 woman의 차이와 king과 queen의 차이 벡터는 매우 유사할 것이다. 위에서 나타난 벡터(화살표)는 성별의 차이를 나타내는 벡터를 의미한다. 주의해야할 점은 300차원 안에서 그려진 벡터이다.(2차원이 아님)

여기서 우리가 해야할 것은 'man -> woman as king -> ____' 에서 우리는 빈칸의 단어 w 를 찾는 것이고, 아래 방정식으로 찾을 수 있다.

$$\text{Find word } w : \arg \max_w \text{sim}(e_w, e_{king} - e_{man} + e_{woman})$$

여기서 sim은 similarity function을 의미하며, 두 단어 사이의 유사성을 계산한다. 위 식에서 유사성을 최대화하는 단어를 찾게 되고, 따라서 queen이라는 단어라고 예측할 수 있을 것이다. (적절한 similarity function이 필요하다)

실제 논문에서는 30~75%의 정확성을 보여주고 있는데, 유추 문제에서 완전히 정확한 단어를 예측해야 정답으로 인정되기 때문에 정확성이 다소 낮아 보일 수 있지만, 벡터 공간 내에서 상당한 유사성을 표현하고 있다.



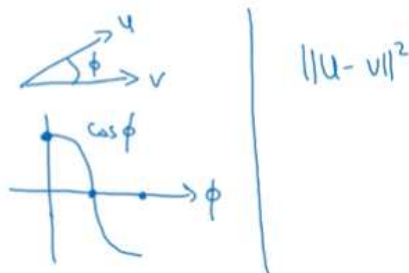
t-SNE

그리고 t-SNE 알고리즘에 대해서 언급하자면, 이 알고리즘은 300D를 2D로 축소하여 매핑하는데, 매우 복잡하고 비선형적인 매핑이다. 유사한 단어들을 서로 가깝게 배치하여 단어 간의 의미적 유사성을 보여준다. 따라서 시각화는 단지 300차원에서 2D로 축소된 것이므로 실제 연산 및 관계 비교는 고차원(300D)에서 이루어져야한다.

Cosine similarity

$$\rightarrow \text{sim}(e_w, e_{\text{king}} - e_{\text{man}} + e_{\text{woman}})$$

$$\text{sim}(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$



Man:Woman as Boy:Girl

Ottawa:Canada as Nairobi:Kenya

Big:Bigger as Tall:Taller

Yen:Japan as Ruble:Russia

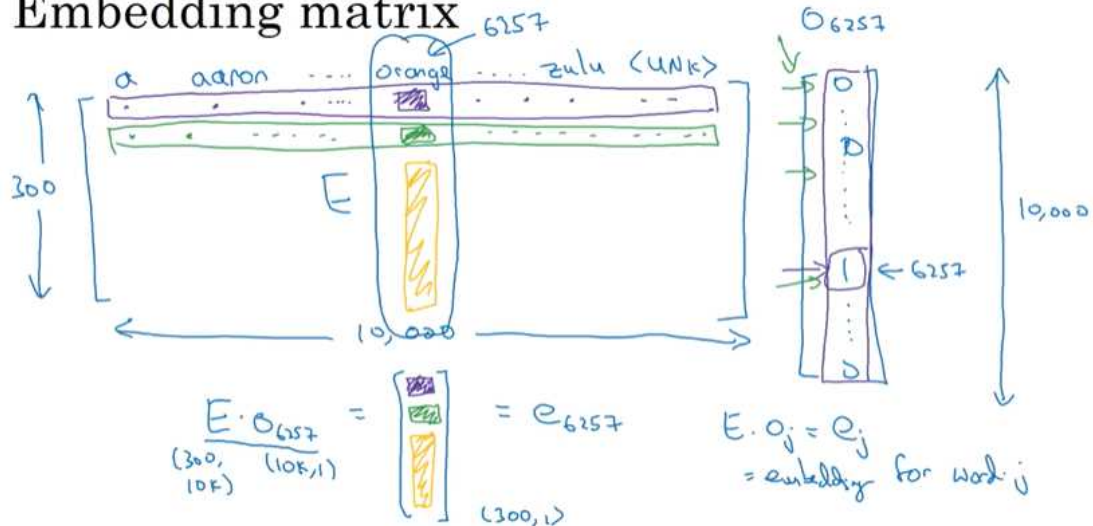
일반적으로 Similarity function으로는 Cosine similarity를 가장 많이 사용한다.

또한 유클리디안 거리를 사용하기도 한다.

$$\text{sim}(u, v) = \|u - v\|^2$$

[Embedding matrix]: 어휘 사전(Vocabulary)에 있는 단어들을 저차원 벡터로 표현하는 행렬
 * embedding은 고차원 One-Hot 벡터를 저차원 밀집 벡터(Dense Vector)로 변환하는 과정

Embedding matrix



단어 임베딩은 Embedding Matrix E 라는 행렬을 학습하는 과정이다.

- 만약 어휘 사전이 1만 개의 단어로 구성되고 300차원의 임베딩 벡터를 사용한다면, matrix(행렬) E 는 $(300, 10k)$ 크기를 가진다.
- 어휘 사전의 각 단어는 임베딩 행렬의 각 열(column)에 해당하는 벡터로 임베딩된다. 예를 들어, 단어 "orange"는 어휘 사전에서 6257번째 단어이며, 임베딩 행렬 E 의 6257번째 열이 "orange"의 임베딩 벡터가 된다. o_{6257}

One-Hot 벡터와 임베딩 행렬의 곱셈:

- 단어를 One-Hot 벡터로 표현하고, 이를 행렬 E 와 곱하면 해당 단어의 임베딩 벡터를 얻을 수 있다. $E \cdot o_{6257} = e_{6257}$
- One-Hot 벡터는 단 하나의 요소만 1이고, 나머지 요소는 모두 0이므로, 임베딩 행렬의 0에 해당하는 열은 모두 0이 되고, 1에 해당하는 열만 남게 된다.
- 따라서 이 곱셈은 임베딩 행렬 E 에서 해당 단어에 대한 특정 열을 선택하는 것과 같다.

임베딩 벡터의 역할:

- 결과적으로 고차원 One-Hot 벡터를 저차원 밀집 벡터(Dense Vector)로 변환하며, 이 벡터는 단어 간의 의미적 관계를 벡터 공간에서 표현한다.
- 이를 통해 단어 간의 의미적 유사성이나 관계를 파악할 수 있다.

일반화하면 다음과 같다.

$$\text{Embedding for word } j = E \cdot o_j = e_j$$

우리가 학습해야 되는 것이 Embedding Matrix E 라는 것이 중요하며, 이 Matrix E 는 초기에 무작위로 초기화된다.

위 공식에서 Matrix E 를 one-hot vector o 와 함께 곱하는 것으로 표현되어있는데, 이것은 꽤 비효율적이다. one-hot vector는 꽤 높은 차원인데, 대부분 0으로 채워져있기 때문에 메모리 낭비가 심하고, 연산량도 많다.

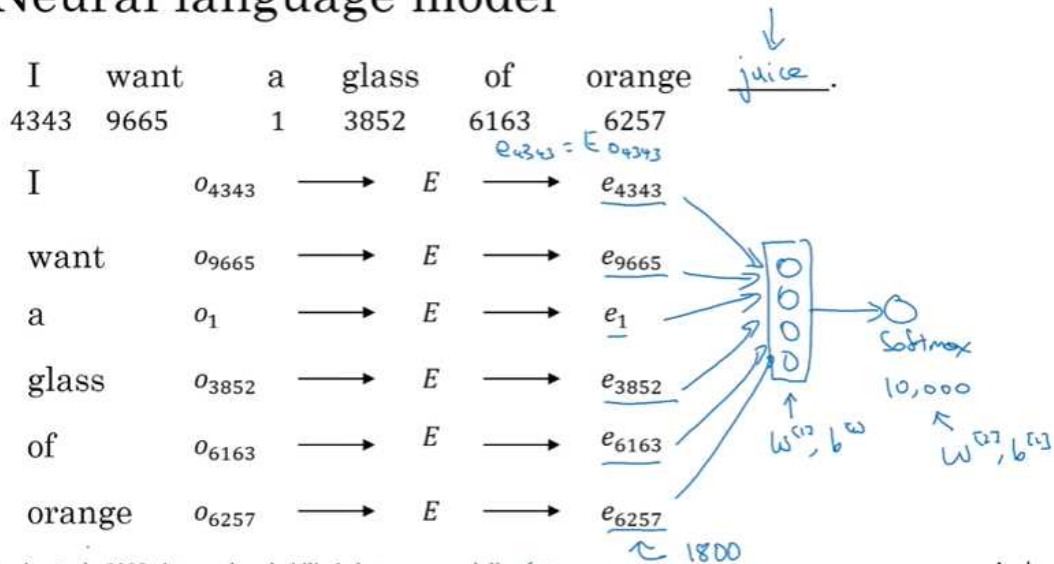
실제로는 one-hot vector를 곱하는 것이 아닌 특화된 함수를 사용한다.

[Learning word embeddings]

단어 임베딩(Word Embedding)은 딥러닝 연구에서 초반에는 비교적 복잡한 알고리즘으로 시작했다. 시간이 지나면서, 훨씬 더 간단하고 단순한 알고리즘도 가능하고, 특히 큰 데이터셋에서 매우 좋은 결과를 얻을 수 있다. 최근 가장 인기 있는 몇몇의 알고리즘들은 매우 단순하다.

단어 임베딩이 어떻게 동작하는지 이해하기 위해서 더 복잡한 알고리즘의 일부를 살펴보자.

Neural language model



[Bengio et. al., 2003, A neural probabilistic language model]

Andrew Ng

Language model을 만들고, I want a glass of orange _____. 에서 orange 다음의 단어를 예측하려고 한다.

Neural language model은 단어 embedding을 학습하고 활용하는 모델이며, 신경망을 통해 연속적인 단어들을 통해서 다음 단어를 예측한다. 첫 번째 단어부터 시작해보자.

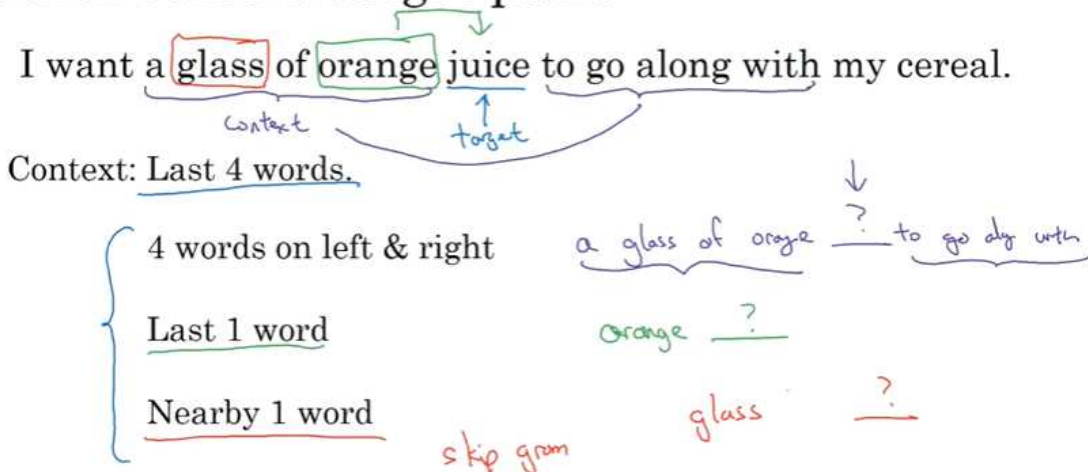
첫 번째 단어 I는 one-hot vector o_{4343} 로 표현된 후, Matrix E를 통해서 300차원의 vector e_{4343} 가 된다. 이 과정을 통해 모든 단어는 300차원 임베딩 벡터로 표현되고 이러한 임베딩 벡터들이 신경망의 입력으로 사용된다. 예측 과정에서는 4개의 단어를 참조하여 다음 단어를 예측하는데, 이를 위해 4개의 임베딩 벡터를 연결하여 1200차원의 입력 벡터를 만든다. 이 벡터는 신경망을 통해 처리되고, Softmax Layer를 통해 어휘 사전(Vocabulary)에 있는 모든 단어에 대한 확률 분포로 변환되어 다음 단어를 예측하게 된다.

- NLM은 일반적으로 fixed historical window(예: 4개의 단어)를 사용하여 다음 단어를 예측한다. 따라서 긴 문장에서도 바로 앞의 4개의 단어만을 참조한다. 문맥 창이 고정되어 있기 때문에 모델이 처리하는 입력 벡터의 크기는 항상 일정하기 때문에 긴 문장까지 처리할 수 있다.
- 이 모델의 파라미터는 임베딩 매트릭스 Matrix E와 $W^{[1]}, b^{[1]}$, $W^{[2]}, b^{[2]}$ 로 구성된다. 학습 과정에서 이 파라미터들을 조정하여 입력 단어들의 임베딩과 문맥을 기반으로 다음 단어를 정확히 예측하도록 한다. 결과적으로, 이 모델은 단어 임베딩을 학습하고, 문맥에 따라 다음에 나올 단어를 예측하는 과정을 통해 임베딩 벡터가 자연스럽게 단어 간의 의미적 유사성과 관계를 학습하게 된다. 만약 orange가 아닌 apple이나 durian이 오더라도 유사한 단어라면 juice를 예측할 수 있게 되는 것이다.

중요한 것은 Matrix E를 학습한다는 데 있으며, 이후 이 알고리즘을 어떻게 일반화해서 어떻게 더 단순한 알고리즘을 도출할 수 있는지 살펴보자.

언어 모델이 다음 단어를 예측할 때 문맥(Context)과 타겟(target) 단어의 관계를 알아보자

Other context/target pairs



I want a glass of orange juice to go along with my cereal"이라는 문장에서 "juice"를 예측한다고 해보자. 이때 예측하려는 단어인 "juice"가 target이 된다.

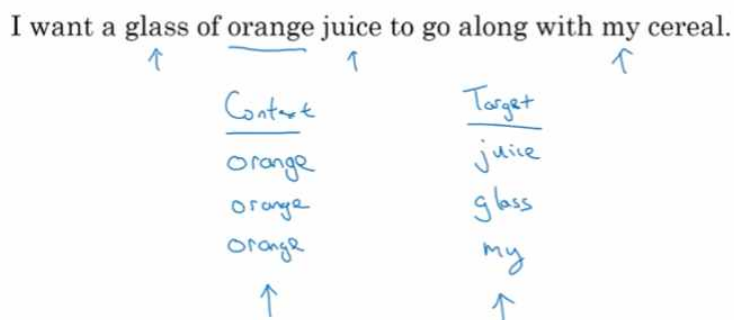
언어 모델은 이 target 단어를 예측하기 위해 문맥(Context)을 사용한다.- 바로 앞 4개의 단어, 왼쪽과 오른쪽 4개의 단어, 다른 Context를 선택할 수도 있다. 조금 더 단순하게 사용한다면 마지막 한 단어만 선택할 수도 있고, 가장 가까운 단어 하나를 사용할 수도 있다.

다음 영상에서 더 간단한 Context와 더 간단한 알고리즘을 통해서 어떻게 target word를 예측하게 되는지, 어떻게 좋은 단어 임베딩을 학습할 수 있는지 살펴보자.

[Word2Vec]

Word2Vec: 단어를 벡터로 바꾸어주는 알고리즘, 그 방법 중 하나인 Skip gram에 대해 알아보자.

Skip-grams

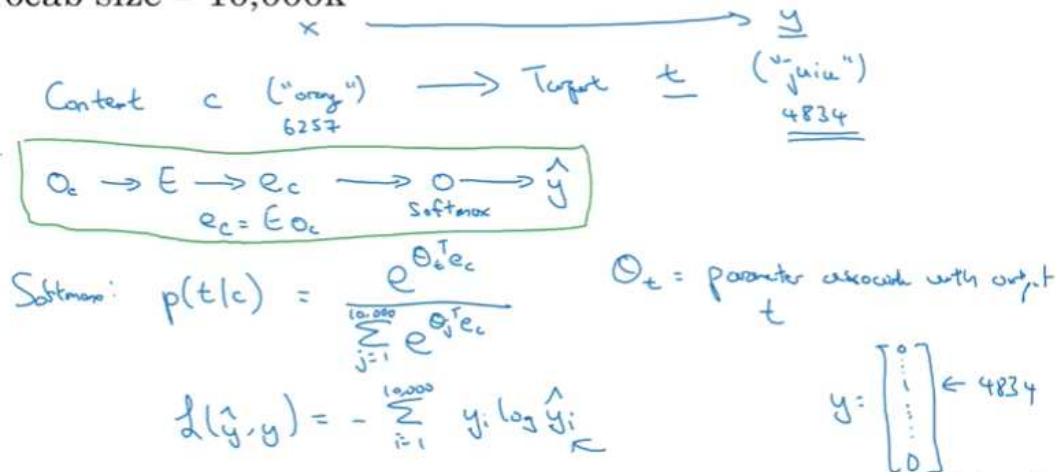


Skip gram은 중심이 되는 단어를 무작위로 선택하고, 중심 단어를 기준으로 주변 단어를 예측하는 방법이다. 중심 단어를 무작위로 선택하고(Context(입력)) 주변 단어를 Target으로 설정하고 예측하는 supervised learning이다. (주변 단어는 여러 개를 선택할 수 있다) - 단어 간의 의미적 유사성과 문맥적 관계를 효과적으로 파악

다음으로 Skip gram의 모델이다.

Model

Vocab size = 10,000k



Vocabulary 크기가 10,000개 단어라고 가정하자. 이 단어들을 통해서 Context c 에서 Target t 를 예측하기 위해 Skip-Gram 모델이 사용된다. 입력과 결과의 예를 들면, orange와 juice가 될 수 있다.

이전 강의에서 본 것처럼, Context 단어 c 는 Embedding Matrix E 를 통해서 Embedding vector로 변환되고, 이는 신경망을 통해 softmax layer를 통과해서 다음 단어에 대한 확률분포인 output \hat{y} 를 구할 수 있다.

Softmax를 통해 계산되는 확률(output)은 아래와 같이 나타낼 수 있다.

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

여기서 θ_t 는 각 단어의 weight와 관련된 parameter고, bias는 생략했다.

이 확률분포를 통해 Skip-Gram 모델은 특정 Context 단어에 대한 Target 단어를 예측하고, 이를 통해 단어 간의 관계를 학습한다.

Loss는 Negative Log Likelihood Loss를 사용해서 아래와 같이 구할 수 있다.

$$\mathcal{L}(\hat{y}, y) = - \sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

\hat{y} 는 Softmax 결과로 나온 단어의 확률 분포를 의미하고, y 는 실제 타겟 단어에 대한 원-핫 벡터를 의미한다.

output \hat{y} 는 1만 차원을 가지고, 가능한 1만개의 단어의 확률들을 포함한다.

Problems with softmax classification

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

Handwritten notes and diagrams:

- Hierarchical softmax*
- $\log |V|$
- A tree diagram showing a path from root to leaf, with nodes labeled sk and $2sa$. An arrow points to a leaf node labeled the .
- A second tree diagram showing a path from root to leaf, with nodes labeled the and at . An arrow points to a leaf node labeled $denim$.

하지만, 이 skip gram model에는 몇 가지 문제가 존재한다. 주된 문제는 계산 속도이다.

- 특히 softmax를 통해 확률을 계산할 때 Vocabulary 크기가 크면(예: 10만, 100만 단어) 매번 모든 단어에 대한 확률을 계산하는 것은 굉장히 느리다. ($p(t|c)$ 의 분모항)
- 이러한 문제를 해결하기 위해 Hierarchical Softmax나 Negative Sampling 방법이 사용된다.

Hierarchical Softmax

트리 구조를 통해 단어를 탐색하는데, 자주 사용되는 단어일수록 트리의 위쪽에 위치하고, 그렇지 않은 단어일수록 트리의 아래쪽에 위치한다.

따라서 어느 한 단어를 예측할 때 트리의 경로 탐색을 통해 계산하기 때문에 선형 탐색이 아닌, 트리의 로그(log) 크기만큼의 시간 복잡도로 확률을 계산할 수 있어 softmax보다 빠르다. 자세한 내용은 논문을 참조하면 된다.

Negative Sampling

모든 Vocabulary에 대한 확률을 계산하는 대신, 일부 단어만 선택해서 확률을 계산함으로써 효율적으로 학습한다.

Negative Sampling에 대해 자세히 설명하기 전에 어떻게 Context c 를 샘플링하는지에 대해서 이야기를 해보자.

Skip-Gram에서 Context를 선택할 때는 중심 단어의 좌우 10개 단어에서 무작위로 선택한다. 하지만 무작위로 균일하게 샘플링하면, "the", "of", "a"와 같은 빈번한 단어가 많이 선택되는 문제가 있다. 따라서 샘플링할 때는 이러한 빈번한 단어와 그렇지 않은 단어 사이의 균형을 맞추기 위해 샘플링 방법을 조정해야 한다.

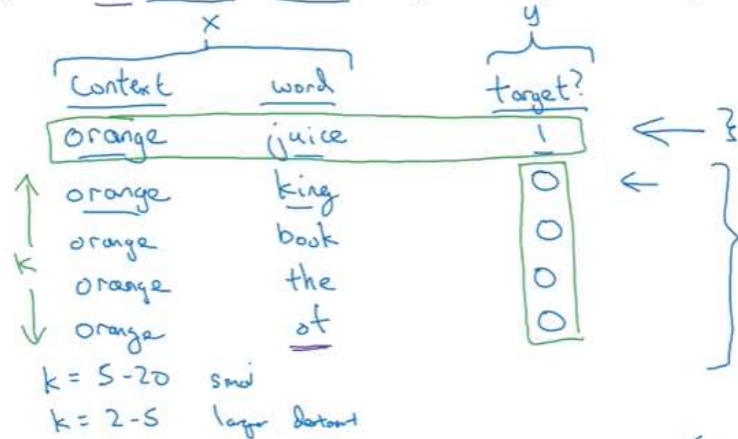
[Negative Sampling]

Negative sampling은 skip-gram 모델의 느린 연산 속도를 개선한 알고리즘

(Skip-gram의 경우 softmax 연산 때문에 전체 vocabulary에 대한 확률을 계산해야 해 느림)

Defining a new learning problem

I want a glass of orange juice to go along with my cereal.



[Mikolov et. al., 2013. Distributed representation of words and phrases and their compositionality]

Andrew Ng

Negative sampling은 특정 context와 target 쌍을 positive sample로 정의하고, 이에 대한 반대의 의미인 negative samples를 랜덤하게 골라 학습하는 방법이다. 예를 들어, 문장 "I want a glass of orange juice"에서 context가 "orange"일 때 target은 "juice"가 될 수 있다. 이처럼 주어진 positive 쌍(orange-juice와 같은 positive training set) 외에 무작위로 vocabulary에서 다른 단어들을 뽑아서 negative samples로 만들어 학습한다.(context와 관련 없는 단어를 무작위로 고르는 것) 이렇게 K개의 negative samples를 만들고, positive와 함께 학습을 진행하면 softmax 연산을 대신하는 간단한 logistic regression model 형태로 바뀌어 연산량이 줄어들게 된다.

- 작은 데이터셋의 경우 k를 5-20 정도로, 큰 데이터셋의 경우 2-5로 설정한다. 위 슬라이드에서는 K=4로 설정했다.

다음은 x->y mapping model이다.

Model

Softmax:
$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

$$P(y=1 | c, t) = \sigma(\theta_t^T e_c) \leftarrow$$

x		y
context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

Diagram showing the mapping from context (c) and word (t) to target (y). Arrows indicate the flow of information from the input variables to the target variable.

Negative Sampling에서는 Logistic Regression을 사용해서 이진 분류 문제로 바꾼다. 즉, 주어진 context와 target이 의미적으로 연관되는지를 판단하는 문제로 단순화한다.

$P(y=1 | \text{context}, \text{target})$ 의 확률을 계산하는데, $y=1$ 은 해당 쌍이 positive (연관이 있음)를 나타내고 $y=0$ 은 negative (연관이 없음)를 나타낸다.

학습 과정에서 한 개의 positive sample (orange-juice처럼 실제로 의미적으로 연관된 쌍)과 K개의 negative sample (임의로 선택된 의미적으로 연관되지 않은 단어 쌍)을 학습한다.

Negative Sampling을 통해서는 한 쌍의 positive sample과 몇 개의 (K개의) negative sample에 대해서만 계산하면 된다.

이렇게 하면 기존의 10,000개 이상의 단어에 대한 Softmax 계산이 아닌 K개의 샘플에 대한 이진 분류만 수행하면 되어서 계산량이 크게 줄어들게 된다.

그리고 모든 iteration과정에서 한개의 positive와 K개의 negative 샘플만 학습하고 있다.

이 알고리즘에서 가장 중요한 점은 어떻게 negative sample을 선택하느냐인데, 가장 많이 사용하는 방법은 말뭉치(corpus)에서의 경험적 빈도(empirical frequency)에 기반해 샘플링하는 것이다. 그래서 얼마나 자주 특정 단어들이 나타나는지에 따라서 negative sample을 고른다. 하지만 이로 인해 의미 없는 단어들이 지나치게 많이 선택될 수 있는 문제가 있다. 예를 들어 the, a, of 같은 단어는 굉장히 자주 등장하기 때문에 무작위로 선택시 negative sample에 많이 포함될 수 있다.

또 다른 극단적인 방법은 $1/\text{voca size}$ 만큼 무작위로 단어를 선택하는 방식. 이 방법은 영어 단어의 분포를 생각하지 않는다.

논문의 저자는 경험적으로 가장 좋은 방법을 직접 찾아서 사용하는 것이 좋다고 한다. 논문에서는 단어 빈도의 $3/4$ 제곱에 비례하는 샘플링을 진행했다.

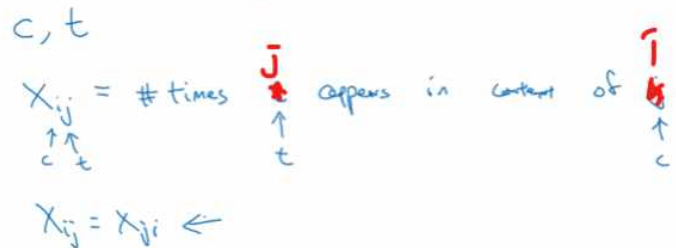
$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10,000} f(w_j)^{3/4}}$$

[GloVe(Global Vectors for Word Representation) word vectors]

단어 임베딩의 또 다른 알고리즘은 GloVe 알고리즘이다. Word2Vec or skip gram model만큼 사용되지는 않지만, 꽤 단순하기 때문에 사용된다.

GloVe (global vectors for word representation)

I want a glass of orange juice to go along with my cereal.



GloVe 알고리즘은 말뭉치(corpus)에서 각 단어 쌍(context와 target)이 함께 등장하는 빈도 수 즉, context i 에서 target j 가 몇 번 나타내는지(즉, 단어 i 와 j 가 함께 등장한 횟수 : X_{ij})를 이용해 단어 임베딩을 만드는 알고리즘이다. (단어 사이의 관계를 파악하는 데 활용)

- context와 target의 범위를 어떻게 지정하느냐(어떤 단어를 주변 맥락으로 포함하는지)에 따라서 X_{ij} 와 X_{ji} 의 값이 동일할 수도 있고, 다를 수도 있다. (양방향 context를 사용하는 경우 두 값은 동일, 제한된 범위의 context를 사용하는 경우 달라질 수 있음. 즉 context의 방향성과 범위 설정에 따라 단어 쌍의 빈도 수가 대칭적이거나 비대칭적으로 나타나게 된다.)

Model

$$\text{Minimize } \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) (\theta_i^T e_j + b_i + b'_j - \log X_{ij})^2$$

Handwritten notes and annotations:

- $\theta_i^T e_j$ is labeled as the "weighting term".
- $f(X_{ij}) = 0$ if $X_{ij} = 0$.
- $\log 0 = 0$.
- Example: $\theta_{\text{this}}^T e_{\text{is}}$ for the pair "this, is".
- θ_i, e_j are symmetric.
- $e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}$

GloVe 모델이 하는 것은 아래 식을 최적화하는 것이다.

$$\text{minimize } \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) (\theta_i^T e_j + b_i + b'_j - \log X_{ij})^2$$

위 식에서 $f(X_{ij})$ 를 설정하는데, 이는 weighting term이다. 등장횟수 X_{ij} 가 높을수록 가중치가 높아지고 낮을수록 가중치가 낮아지는 특징을 가진다. 지나치게 빈도가 높거나 낮은 단어로 인해서 X_{ij} 값이 특정값 이상으로 튀는 것을 방지하는 역할을 한다.(영향력 조절)

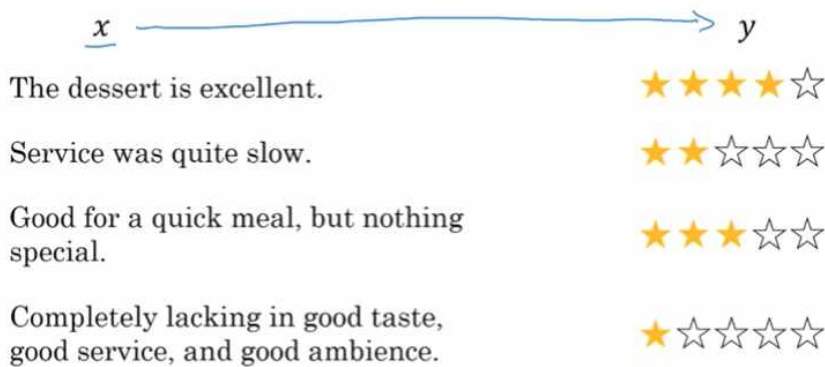
마지막으로 이 알고리즘에서 흥미로운 점은 임베딩 벡터 θ_i 와 e_j 가 완전히 대칭적이라는 것이다.

- 예를 들어, θ_i 가 특정 단어의 의미적 특성을 표현한다면, e_j 도 마찬가지로 의미적 특성을 표현한다. 즉, 두 벡터는 같은 구조와 역할을 갖는다는 의미에서 대칭적인 것이다.
- 이렇게 대칭적 구조를 가지면, 어떤 단어 i 가 다른 단어 j 와 어떤 관계를 맺고 있는지를 쉽게 파악할 수 있다. (두 벡터 간의 내적을 통해 두 단어 사이의 관계를 계산할 수 있고, 이를 통해 모델은 단어 간의 의미적 유사성이나 관련성을 잘 학습할 수 있음)

[Sentiment classification]

Sentiment classification(감정 분석)은 NLP에서 중요한 구성요소 중의 하나이다. 감정 분석은 많은 dataset이 아니더라도 단어 임베딩을 사용해서 좋은 성능의 분류기를 만들 수 있다.

Sentiment classification problem



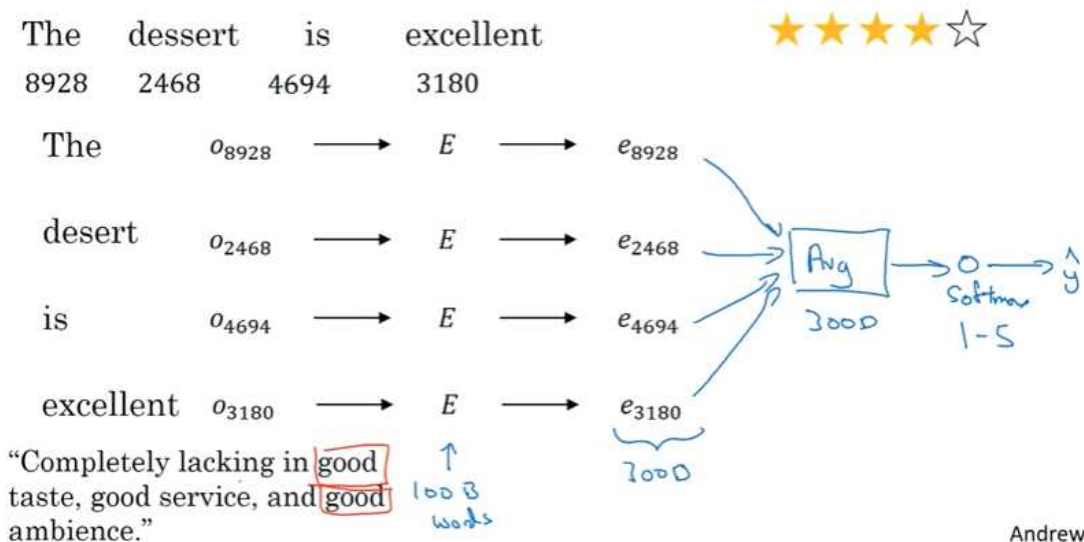
10,000 → 100,000 words

Sentiment classification problem은 위와 같은 매핑 문제이다.

- 감정 분석 문제는 입력된 문장의 각 단어를 모델에 매핑하고, 이 매핑을 통해 전체 문장의 긍정/부정 감정을 분류하는 문제로 설명할 수 있다. 이때 단어 임베딩이 사용돼서 각 단어는 고차원 벡터(보통 300차원)로 변환되며, 이를 통해 단어의 의미적 유사성이나 관계를 파악할 수 있다.

간단한 모델은 아래와 같이 구성할 수 있다.

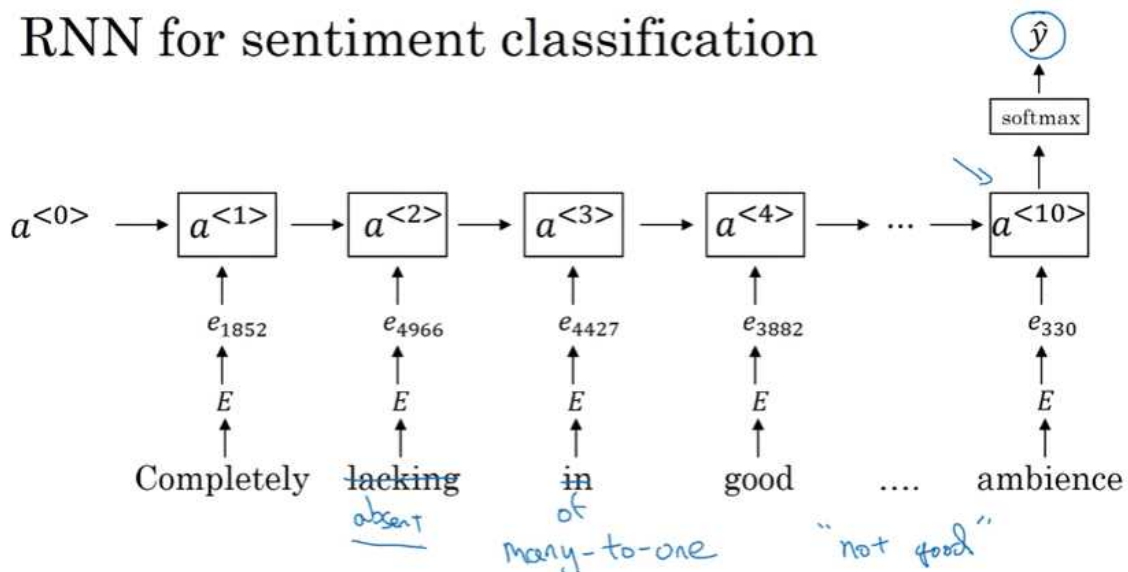
Simple sentiment classification model



- 'The dessert is excellent'라는 입력이 있을 때, 각 단어들을 임베딩 vector(300D)로 변환하고 모든 단어의 벡터 값을 평균해서 하나의 벡터를 만들고, softmax를 사용해 결과를 예측하는데 활용한다.
- 장점: 작은 데이터셋이나 낱선 단어에 대응 가능: 단어 임베딩을 사용하기 때문에, 작데이터셋이 작거나 자주 등장하지 않는 단어들(심지어 학습에 사용되지 않은 단어)가 입력되더라도 해당 모델에 적용이 가능하다. (해당 단어와 유사한 임베딩을 가진 단어로 학습한 패턴을 통해 어느 정도 대응 - ∵ 단어 임베딩은 각 단어를 고차원 공간에서 의미적으로 유사한 단어들이 가까이 위치하도록 학습)
- 한계: 단어의 순서를 무시한다. 단순 나열된 형태로 입력을 처리하고, 단어의 임베딩 벡터의 평균을 사용해 전체 문장을 표현하므로 별로 좋은 모델은 아니다.
- 예를 들어 'Completely lacking in good taste, good service, and good ambience'라는 리뷰가 있다면, good이라는 단어가 많이 나왔기 때문에 positive로 예측할 수도 있다는 것이다.

그래서 해당 문제점을 해결하기 위해서 RNN을 사용할 수 있다.

RNN for sentiment classification



방금 부정적인 예시를 입력으로 사용하더라도 시퀀스의 순서를 고려하기 때문에 해당 리뷰가 부정적이라는 것을 알 수 있다. 따라서 이 모델로 학습하면 상당히 괜찮은 분류기가 된다. 또한 absent라는 단어가 training set에 존재하지 않았더라도 단어 임베딩 벡터에 포함되어 있는 의미적 유사성을 활용해 예측할 수 있다.

[Debiasing word embeddings]

Machine Learning과 AI 알고리즘은 매우 중요한 결정을 하는데 도움이 되거나, 많은 도움을 줄 수 있다고 신뢰받는다. 따라서 불공정하거나 편향된 결과를 초래하지 않고 bias(편향)에서 자유롭다는 것을 확인하고자 한다.(ex. 성이나 인종에 대한 편향을 가지지 않는 알고리즘이기를 원한다.) 단어 임베딩에서 이러한 편향들이 있을 때, 이런 편향을 없애거나 최소화하기 위해 편향을 제거하는 "debiasing" 기법이 필요하다.


The problem of bias in word embeddings

Man:Woman as King:Queen

Man:Computer_Programmer as Woman:Homemaker ✕

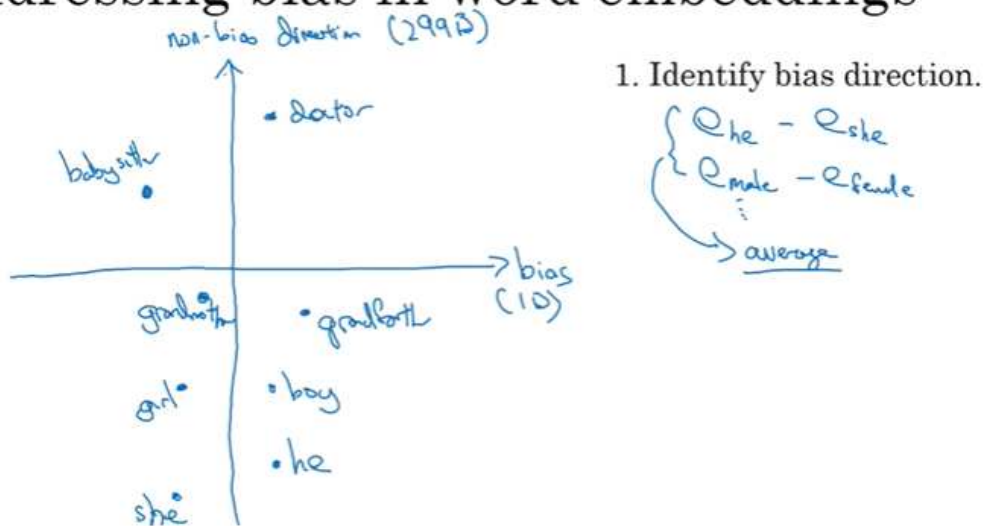
Father:Doctor as Mother:Nurse ✕

Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model.

[Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings]  Andrew Ng

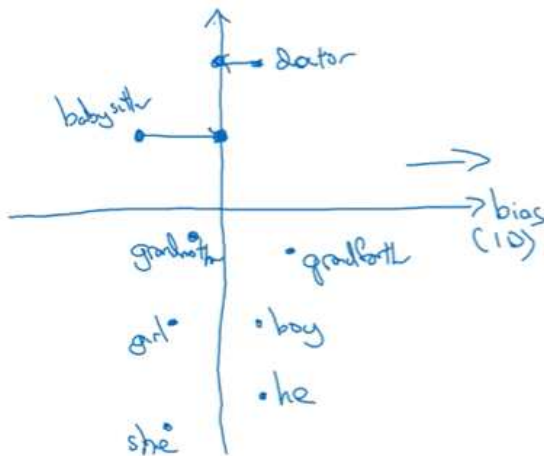
단어 임베딩에서 편향은 다음과 같은 과정을 거쳐서 제거할 수 있다.

Addressing bias in word embeddings



1. 먼저 bias의 direction을 구한다.

성별과 같은 편향이 있을 수 있는 단어들의 차이 벡터를 평균화하여 성별 bias의 방향을 계산한다. 만약 성별의 direction을 구한다면, $e_{he} - e_{she}$, $e_{male} - e_{female}$ 등의 성별을 나타낼 수 있는 단어들의 벡터 차이를 구한다. 이 방향은 1차원 벡터(bias direction)로 표현될 수 있고, 나머지 299차원은 bias와 무관한 방향(non-bias direction)이 된다. (실제로는 더 복잡한 SVD-특이값분해라는 알고리즘을 통해서 bias direction을 구한다)



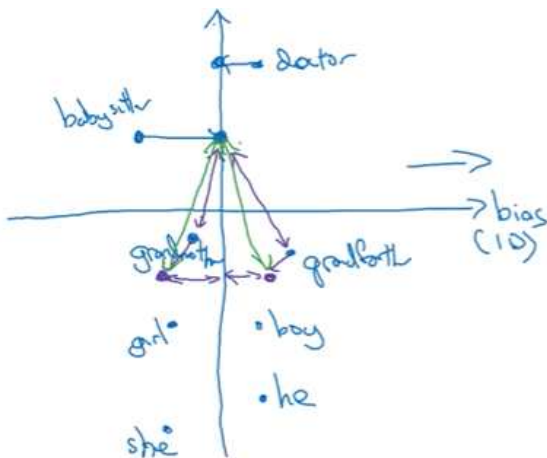
1. Identify bias direction.

$$\begin{cases} e_{he} - e_{she} \\ e_{male} - e_{female} \\ \vdots \end{cases} \rightarrow \text{average}$$

2. Neutralize: For every word that is not definitional, project to get rid of bias.

2. 다음으로는 Neutralize(중성화) 작업을 수행한다.

bias가 없어야 하는 단어들에 대해서 bias 요소를 제거하는 neutralize 작업을 수행한다. 예를 들어, doctor나 babysitter 처럼 편향이 없어야 할 단어들의 임베딩 벡터에서 bias direction을 제거한다.



1. Identify bias direction.

$$\begin{cases} e_{he} - e_{she} \\ e_{male} - e_{female} \\ \vdots \end{cases} \rightarrow \text{average}$$

2. Neutralize: For every word that is not definitional, project to get rid of bias.

3. Equalize pairs.

$$\begin{matrix} \text{grandmother} & - & \text{grandfather} \\ \text{girl} & & \text{boy} \end{matrix}$$

3. 마지막으로 Equalize pairs 작업을 수행한다.

성별 요소가 있는 단어 쌍들, 즉 'boy-girl', 'grandfather-grandmother'처럼 성별이 포함된 단어들에 대해 동일한 편향 방향을 갖도록 만드는 equalize pairs 작업을 수행한다. 각 단어의 성별 요소가 편향 방향에 따라 동일한 거리를 가지도록 위치를 조정한다. 이렇게 함으로써 성별 관련 편향이 균형을 이루게 된다.

예를 들어, boy-girl / grandfather-grandmother과 같은 단어는 각 단어가 성별 요소가 있기 때문에, 이러한 단어들이 bias direction을 기준으로 같은 거리에 있도록 한다.