스마트 에너지를 위한

텍스트 처리와 RNN 모델

2024년 1월 22일~1월 23일

텍스트 전처리 개요

텍스트 분석의 개요

- 텍스트 분석의 목적
  - 텍스트의 의미를 알아내는 것
    - 글의 목적
    - 글쓴이의 성향(찬성/반대)
    - 기분(기쁨/슬픔/우울함 등)
    - 제품 피드백 등
- 텍스트 자체는 대표적인 비정형 데이터
- 의미를 추출하려면 비정형 데이터에서 정형화된 정보를 먼저 얻어야 함
- 텍스트 구문을 분석하여 의미를 파악하고 이것을 정량적으로 측정함

텍스트 처리 – 응용

- SNS(트위터, 블로그, 페이스 북 등) 글을 분석
  - 소비자들의 반응, 감성, 트렌드를 파악
  - 개인별 마케팅, 상품 피드백을 분석하는데 사용
- 이메일, 웹사이트 댓글, 신문기사, 콜센터 상담기록, 도서 등을 분석
  - 글의 주요 내용을 파악
  - 문서의 특징을 추출
  - 유사한 글이나 저자를 찾는 작업 등을 수행
- 참고문헌이나 본문 인용의 관계를 통해서 문서간의 연계성, 전문가들의 인적 네트워크 등을 파악하는데도 사용
- 인공지능 스피커, 챗봇 등에서도 기본적으로 텍스트 분석이 필요

텍스트 처리 – 응용

- **챗봇** (Chatbot)
  - **사람과 대화하듯이 음성, 키보드 입력으로 대화를 나누는 인공지능 서비스**
  - **챗봇의 유형**
    - **미리 답을 준비하여 관련 질문이 나오면 해당 답을 하는 간단한 방식 (저수준)**
    - **신경망을 사용하여 최적의 답을 찾아주는 방식 (고수준)**

- QA(Question Answering) **시스템**
  - **질문을 하면 검색을 통해 적절한 답을 찾아주는(대답) 서비스**
    - **대한민국의 수도는?**
    - **오늘 날씨는?**

- **자연어 처리**
  - **언어 모델을 사용**
    - **가장 자연스러운 다음 문장을 완성**
    - **문장을 번역**
    - **문서 요약, 주제 분석, 감성 분석 등을 수행**

텍스트 처리 – 용어

- 사람이 단어나 문장의 의미를 인식하듯이 컴퓨터가 단어 자체 의미를 직접 파악할 수는 없다
- 텍스트 데이터 처리
  - 대표적인 비정형 데이터
  - 먼저 비정형 데이터인 글자로부터 정형화된 데이터인 수치 데이터로 변환
- 토큰화(tokenize) : 텍스트 분석의 첫 단계
  - 주어진 텍스트를 토큰(token)으로 나누는 작업
  - 컴퓨터가 다루는 텍스트의 단위 : 토큰
    - 글자(character), 단어 (word), n-gram, POS(part-of-speech, 품사), syllable(음절: 자음+모음) 등

텍스트 처리 – 용어

- **말뭉치**(corpus)
  - **데이터 분석에 주어진 전체 문서 집합**
- **문서**(document)
  - **코퍼스 내의 한 단위의 텍스트**
  - **예) 하나의 블로그는 문서이고, 분석할 대상 블로그가 1천개이면 이 1천개 블로그 집합이 말뭉치**
- **파싱**(parsing)
  - **코퍼스에서 의미 있는 단어를 추출하는 작업**
- Stop words(**불용어**)
  - **언어**(language)**에서 너무 흔하게 자주 나타나는 단어들**(“the”, “is”, etc)

텍스트 처리 – 도구

- NLTK (Natural Language ToolKit)
  - tokenization, stemming, tagging, parsing 등의 다양한 기능 지원
- KoNLPy (NLP library for Korean Language)
- 주요 함수들
  - Gensim.Word2Vec()
  - tf.Keras.Embedding()
  - CountVectorizer()
  - BeautifulSoup()

문장 토큰화 (Sentence tokenization)

- 보통 갖고 있는 코퍼스가 정제되지 않은 상태라면, 코퍼스는 문장 단위로 구분되어 있지 않을 가능성이 높다.
- 이를 사용하고자 하는 용도에 맞게 하기 위해서는 문장 토큰화가 필요
- 온점(.)을 통해서 문장을 구분할 수도 있겠지만 온점이 항상 문장의 끝에 오는 것은 아니므로 100% 정확도를 얻는 일은 쉬운 일이 아니다.
- 따라서 코퍼스가 어떤 국적의 언어인지, 또는 해당 코퍼스 내에서 특수 문자들이 어떻게 사용되고 있는지에 따라서 직접 규칙들을 정의해볼 수 있다.
- 단어 토큰화에서 사용한 NLTK의 sent_tokenize 함수를 사용해서 영어 문장 토큰화를 진행할 수 있다.
- 한국어에 대한 문장 토큰화 도구가 여럿 있으며 KSS(Korean Sentence Splitter) 또한 그 중 하나

토큰화 (tokenization)

- 자연어 처리에서 웹스크래핑이나 SNS 스크래핑을 통해 얻어낸 코퍼스(corpus) 데이터가 필요에 맞게 전처리되지 않은 상태일 수 있다
- 해당 데이터를 사용하고자 하는 용도에 맞게 토큰화(tokenization) & 정제(cleaning) & 정규화(normalization)하는 일을 수행
- 토큰화
  - 주어진 코퍼스에서 토큰(token)이라 불리는 단위로 나누는 작업
  - 토큰의 단위가 상황에 따라 다르지만, 보통 의미 있는 단위로 토큰을 정의

정제 (cleaning) & 정규화 (normalization)

- 토큰화 작업 전, 후에는 텍스트 데이터를 용도에 맞게 정제(cleaning) 및 정규화(normalization)하는 일이 항상 함께 한다.
- 정제(cleaning)
  - 갖고 있는 Corpus에 대해 noise 데이터를 제거
- 정규화(normalization)
  - 표현 방법이 다른 단어들을 통합시켜서 같은 형태의 단어로 만들어주는 것
- 정제 작업은 토큰화 작업에 방해가 되는 부분들을 배제시키고 토큰화 작업을 수행하기 위해서 토큰화 작업보다 앞서 이루어지기도 하지만, 토큰화 작업 이후에도 여전히 남아있는 노이즈들을 제거하기 위해 지속적으로 이루어지기도 한다.
- 사실 완벽한 정제 작업은 어려운 편이라서, 대부분의 경우 일종의 합의점을 찾기도 한다.

정제 (cleaning) & 정규화 (normalization)

- 규칙에 기반한 표기가 다른 단어들의 통합
  - 같은 의미를 갖고 있음에도, 표기가 다른 단어들을 하나의 단어로 정규화
  - USA & US, 는 uh-huh & uhhuh 등

- 대, 소문자 통합
  - 대, 소문자를 통합 : 단어의 개수를 줄일 수 있다
  - 대문자는 문장의 맨 앞 등과 같은 특정 상황에서만 쓰이고, 대부분의 글은 소문자로 작성되기 때문에 대, 대부분 대문자를 소문자로 변환
  - 물론, 대문자와 소문자를 무작정 통합해서는 안 된다. 대문자와 소문자가 구분되어야 하는 경우도 있기 때문
    - 가령 미국을 뜻하는 단어 US와 우리를 뜻하는 us는 구분되어야 한다. 또 회사 이름(General Motors)나, 사람 이름(Bush) 등은 대문자로 유지되는 것이 옳다.

정제 (cleaning) & 정규화 (normalization)

- 불필요한 단어의 제거
  - 텍스트 데이터에서 너무 적게 등장, 자연어 처리에 도움이 되지 않는 단어
    - 예를 들어 입력된 메일이 정상/스팸 메일 분류기를 설계하는 경우
    - 총 100,000개의 메일을 가지고 정상 메일에서는 어떤 단어들이 주로 등장하고, 스팸 메일에서는 어떤 단어들이 주로 등장하는지를 가지고 설계하고자 한다.
    - 그런데 이때 100,000개의 메일 데이터에서 총 합 5번 밖에 등장하지 않은 단어가 있다면 이 단어는 직관적으로 분류에 거의 도움이 되지 않을 것임을 알 수 있다.
  - 길이가 짧은 단어를 삭제하는 것만으로도 어느정도 자연어 처리에서 크게 의미가 없는 단어들을 제거하는 효과를 볼 수 있다 즉, 영어권 언어에서 길이가 짧은 단어들은 대부분 불용어에 해당
  - 하지만 한국어에서는 길이가 짧은 단어라고 삭제하는 이런 방법이 크게 유효하지 않을 수 있다.
  - 단정적으로 말할 수는 없지만, 영어 단어의 평균 길이는 6~7 정도이며, 한국어 단어의 평균 길이는 2~3 정도로 추정

토큰화 (tokenization)

- 토큰화 단위 (크게 3가지)
  - 단어(word)
    - 사람이 말을 이해할 때, 단어 단위로 인식하기 때문에 많은 연구에서 선호
  - 글자(character)
  - n-gram
- 단어 단위로 정보를 표현하는 과정에서 많은 정보를 잃게 된다
  - “정말”, “정말로”, “정말은” 등 단어
    - 같은 단어로 취급, 아니면 각각 다른 단어로 처리할지에 따라 분석 결과 달라짐
    - 같은 단어로 취급하기 위해 단어를 어근(stem)으로 변환하면 어미 변화를 무시하거나 조사를 무시하게 되어 텍스트에 들어 있던 정보를 잃게 된다
- 일반적으로 단어의 종류는 보통 10만 단어 이상 (언어마다 상이)
  - 신조어, 특수한 단어 포함하면 수 십만개로 확대

토큰화

- 글자 단위로 토큰화를 하면 어근으로 변환할 때 정보를 잃는 문제를 피할 수 있다.
  - “정”, “말 “, “로”, “은 “ 등
- 음절 단위 토큰의 수
  - 영어
    - 음절 단위의 토큰의 수가 매우 적다 : 알파벳이 26글자
  - 한글
    - 음절의 수가 수천 가지 이상

토큰화 – n-gram

- n-gram
  - n개의 연속된 단어를 하나로 취급하는 방법

- 예를 들어 “러시아 월드컵” 이라는 표현을 “러시아” 와 “월드컵” 두 개의 독립된 단어로만 취급하지 않고 두 단어로 구성된 하나의 토큰으로 취급
  - n=2 경우, bi-gram
  - 단어의 수가 매우 크게 증가
    - 실제로는 빈도 수가 최소한 몇 개 이상인 것만 다룬다

토큰화 – n-gram (예)

텍스트: “어제 러시아에 갔다가 러시아 월드컵을 관람했다”

단어토큰: { “어제”, “러시아”, “갔다”, “러시아”, “월드컵”, “관람” }

2-gram 토큰: { “어제 러시아”, “러시아 갔다”, “갔다 러시아”, “러시아 월드컵”, “월드컵 관람” }

정수 인코딩 및 패딩

- 컴퓨터는 텍스트보다는 숫자를 더 잘 처리할 수 있다.
- 자연어 처리에서는 텍스트를 숫자로 바꾸는 여러 가지 기법들이 있다.
  - 그러한 기법들을 본격적으로 적용시키기 위한 첫 단계로
  - 각 단어를 고유한 정수에 맵핑(mapping)시키는 전처리 작업이 필요
- 자연어 처리를 하다 보면 각 문장(또는 문서)은 서로 길이가 다를 수 있다. 그런데 기계는 길이가 전부 동일한 문서들에 대해서는 하나의 행렬로 보고, 한꺼번에 묶어서 처리할 수 있다. 다시 말해 병렬 연산을 위해서 여러 문장의 길이를 임의로 동일하게 맞춰주는 작업이 필요
  - 원하는 길이보다 짧은 문장에는 전부 숫자 0이 뒤로 붙어서 모든 문장의 길이가 동일하게 만들어 준다.
  - 이와 같이 데이터에 특정 값을 채워서 데이터의 크기(shape)를 조정하는 것을 패딩(padding)이라고 함
  - 숫자 0을 사용하고 있다면 제로 패딩(zero padding)

정수 인코딩 및 패딩

- 텍스트에 단어가 5,000개가 있다면, 5,000개의 단어들 각각에 1번부터 5,000번까지 단어와 맵핑되는 고유한 정수(인덱스) 부여
  - 가령, book은 150번, dog는 171번, love는 192번과 같이 숫자가 부여
  - 인덱스를 부여하는 방법으로 랜덤으로 부여하기도 하지만,
  - 단어를 빈도수 순으로 정렬한 단어 집합(vocabulary)을 만들고, 빈도수가 높은 순서대로 차례로 낮은 숫자부터 정수를 부여하는 방법이 있다.

토큰화

- n-gram을 허용하면 토큰화 대상의 수가 매우 크게 증가
  - 이론적으로는 10만개의 단어를 두 개 붙여서 나올 수 있는 경우의 수는 10만의 자승이 된다.
- 실제로는 빈도수가 최소한 몇 개 이상인 것만을 다룬다.
- 토큰화한 결과를 수치로 만드는 방법
  - 원핫(one-hot) 인코딩
  - BOW(단어모음)
  - 단어벡터(Word Vector) 방법

원 핫 (One-hot) 인코딩

- **원 핫 인코딩**
  - 단어 집합(vocabulary)을 먼저 만들고
  - 각 토큰에 고유 번호를 배정
  - 모든 고유번호 위치의 한 컬럼만 1, 나머지 컬럼은 0인 벡터로 표시
  - 희소 벡터(sparse vector)

텍스트: "어제 러시아에 갔다가 러시아 월드컵을 관람했다"

토큰 사전: { "어제" :0, "러시아" :1, "갔다" :2, "월드컵" :3, "관람" :4}

원핫 인코딩:

어제 = [1, 0, 0, 0, 0]
러시아 = [0, 1, 0, 0, 0]
갔다 = [0, 0, 1, 0, 0]
월드컵 = [0, 0, 0, 1, 0]
관람 = [0, 0, 0, 0, 1]

BOW (Bag of Word, 단어 모음)

- 원핫 인코딩 방식으로 단어(토큰)을 표현하면
  - 단어의 수가 적을 때에는 문제가 안되지만
  - 단어가 모두 10만개이면
    - 모든 단어가 항목이 10만개인 (0과 1로 구성된) 벡터로 표시
  - 주어진 텍스트가 20개의 단어로 구성되어 있다면
    - 20 x 100,000개 크기의 벡터가 필요
- 텍스트 분석은 "문장" 을 단위로 하는 경우가 많다
- 단어 모음(BOW) 방식 : 한 문장을 하나의 벡터로 만드는 방법
  - 한 문장을 단어 사전 크기의 벡터로 표현하고 그 문장에 들어 있는 단어의 컬럼만 1로, 단어가 없는 컬럼은 모두 0으로 표현
- 먼저 단어 사전을 만들고 각 문장에 어떤 단어가 들어 있는지 조사하여 해당 컬럼만 1로, 나머지는 0으로 코딩

BOW

- 단어 사전: { “어제” :0, “오늘” :1, “미국” :2, “러시아” :3,
  “갔다” :4, “축구” :5, “월드컵” :6, “올림픽” :7,
  “관람” :8, “나는” :9, …, “중국” :4999 }
- Text_1: “어제 러시아에 갔다가 러시아 월드컵을 관람했다” 를 BOW로 표현하면

문장번호	0	1	2	3	4	5	6	7	8	9	10	…	4998	4999
Text_1	1	0	0	1	1	0	1	0	1	0	0	0	0	0
Text_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Text_3	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Text_4	0	0	0	0	0	0	0	1	0	0	0	0	1	0
…														
Text_50	0	0	0	0	0	0	1	0	0	0	0	0	0	0

문서-단어 행렬

- **문서-단어(document-term) 행렬**
  - 문장 단위로 어떤 단어들이 있는지를 나타내는 BOW의 확장
  - 문서(document) 단위로 어떤 단어들이 있는지를 표현
  - 같은 단어가 여러번 등장하면 1 이상의 값을 갖는다
- CountVectorizer() 함수

문서번호	0	1	2	3	4	5	6	7	8	9	10	...	4998	4999
Doc_1	1	2	3	1	4	0	2	0	1	3	0	0	0	0
Doc_2	0	0	0	0	2	0	0	0	0	0	0	0	2	0
Doc_3	0	0	0	1	0	0	0	0	3	0	0	0	0	1
Doc_4	4	0	0	0	0	0	0	1	0	0	4	0	1	0
...														
Doc_100	0	2	0	0	0	0	1	4	0	1	0	0	0	0

tf-idf

- **좀 더 효과적으로 정보를 추출하기 위해서는**
  - **어떤 대상의 특징을 잘 표현해야 한다.**
- Term frequency-inverse document frequency
- Tf : **단어가 각 문서에서 발생한 빈도**
- Df(document frequency) : **그 단어가 등장한 '문서'의 빈도**
- **적은 문서에서 발견될수록 가치 있는 정보**
- **많은 문서에 등장하는 단어일수록 : 일반적인 단어**
  - **이러한 공통적인 단어는 tf가 크다고 하여도 비중을 낮추어야 분석이 제대로 이루어질 수 있다.**
- **따라서 단어가 특정 문서에만 나타나는 희소성을 반영하기 위해서 idf(df의 역수)를 tf에 곱한 값을 tf 대신 사용**

td-idf

$$\mathrm{TF-IDF}(w,d) = \frac{\mathrm{TF}(w,d)}{\mathrm{DF}(w)}$$

$$\mathrm{IDF}(w,d) = \log\left(\frac{n}{1+\mathrm{DF}(w)}\right)$$

- IDF는 DF의 역수
- 위 식에서 log와 1을 분모에서 더한 이유
  - n은 문서의 수를 의미
  - n이 커질수록 IDF의 값이 기하급수적으로 커지기 때문에 그것을 방지하기 위해 log를 사용
- TfIdfVectorizer() 함수

td-idf

$$X = \begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \cdots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

the is of goal lecture bag words via text approach

Document 1
Document 2
Document 3

$$X = \begin{bmatrix} 0.8 & 0.4 & 0 & 1.1 \\ 0.4 & 0.4 & 0 & 0 \cdots \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

the is of goal

$$\text{idf}_{\text{of}} = \log\left(\frac{3}{3}\right) = 0$$

$$\text{idf}_{\text{is}} = \log\left(\frac{3}{2}\right) = 0.405$$

$$\text{idf}_{\text{goal}} = \log\left(\frac{3}{1}\right) = 1.098$$

[note] In TfidfVectorizer(), if smooth_idf=true (default), the constant "1" is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents zero divisions: idf(t) = log [(1 + n) / (1 + df(t))] + 1

Text Processing – 단계

- **Data preprocessing:**
  - Tokenization: convert sentences to words
  - Removing punctuations and tags
  - Removing stop words
  - Stemming: words are reduced to a root (ex: formalize -> formal, allowance -> allow)
  - Lemmatization (ex: is, are, am -> be, having -> have)
- **Feature Extraction**
  - One-hot encoding
  - BOW (Bag of Words): each sentence is represented with counts (or presence) of words (discards word order, hence ignoring the context and meaning in the document), sparse
    - CountVectorizer(), TfIdfVectorizer()
  - Word Embedding: represent each word with a meaningful n-dimensional vector, dense
    - Word2vec(): put words of similar meaning (or context) to closer places
    - tf.keras.Embedding(): vector representation is trained (only used as the first layer in deep learning)
- **Choose ML algorithms**
  - Classical ML (simple text classification) or Deep Learning (sentiment analysis or language translation)

RNN

순환 신경망 (RNN) – CNN

- 합성곱 신경망(CNN)
  - 2차원(공간) 영상에서 좋은 성과 입증, 이미지 학습 분야에서 널리 사용

- 그러나 CNN은 시간적인 차이를 두고 연속적으로 발생하는 데이터 분석에는 성능이 부족
  - 연속된 단어로 구성되는 문장 분석, 자연어 처리, 자동 번역, 시계열 데이터 분석 등

- 이런 데이터에는 순환신경망(Recurrent Neural Network)이 잘 동작

순환 신경망 (RNN) – 텍스트 처리

- 기존 텍스트 분석에서 BOW를 이용
  - 문장 내에 어떤 단어들이 존재하는지는 파악하지만
  - 단어의 배열 정보는 사라지고 이용하지 못함

- 단어의 순서 정보도 분석하려면
  - BoW, 단어 벡터의 도입만으로는 부족
  - 단어의 발생 순서 정보를 활용할 수 있는 신경망 모델을 사용해야 한다
  - 이를 위해서 순환신경망(RNN)이 널리 사용

순환 신경망 (RNN) – 과거 정보 기억

- MLP, CNN에서는 입력 데이터를 한 번 신경망에 통과시키면 데이터가 한 방향으로만 흘러가는 구조

- RNN은 과거의 데이터에서 추출한 정보를 지속적으로 모델 내에서 저장하고 이를 학습에 재사용
  - 사람이 대화를 나누거나 책을 읽을 때 단어를 순차적으로 듣게 되고 최신의 입력 단어들 뿐 아니라 과거의 정보를 계속 내부적으로 사용해야 하는 것과 같은 원리

- 즉, CNN에서는 과거의 기억(상태정보)을 사용하지 않지만 RNN은 과거의 기억을 사용하는 구조를 제공

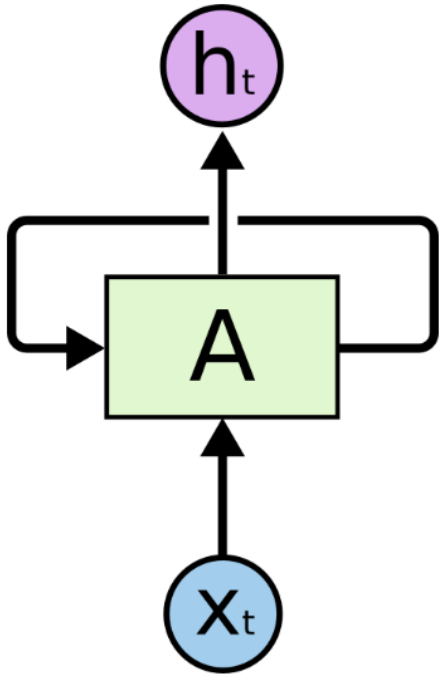
순환 신경망 (RNN) – 응용

- 주요 응용
  - 음성인식
  - 텍스트 분석
  - 자연어 처리
  - 챗봇
  - 감성 분석
  - 언어 모델링(language modeling)

  등에 널리 사용

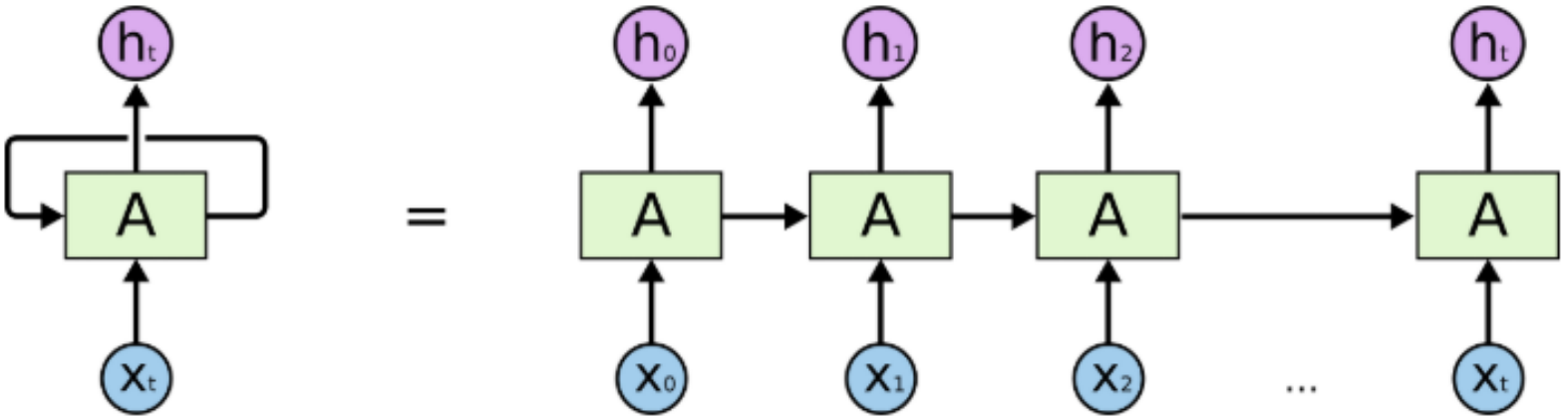
RNN 기본 구조

- RNN은 loop가 들어 있고
- 과거의 데이터가 미래에 영향을 줄 수 있는 구조

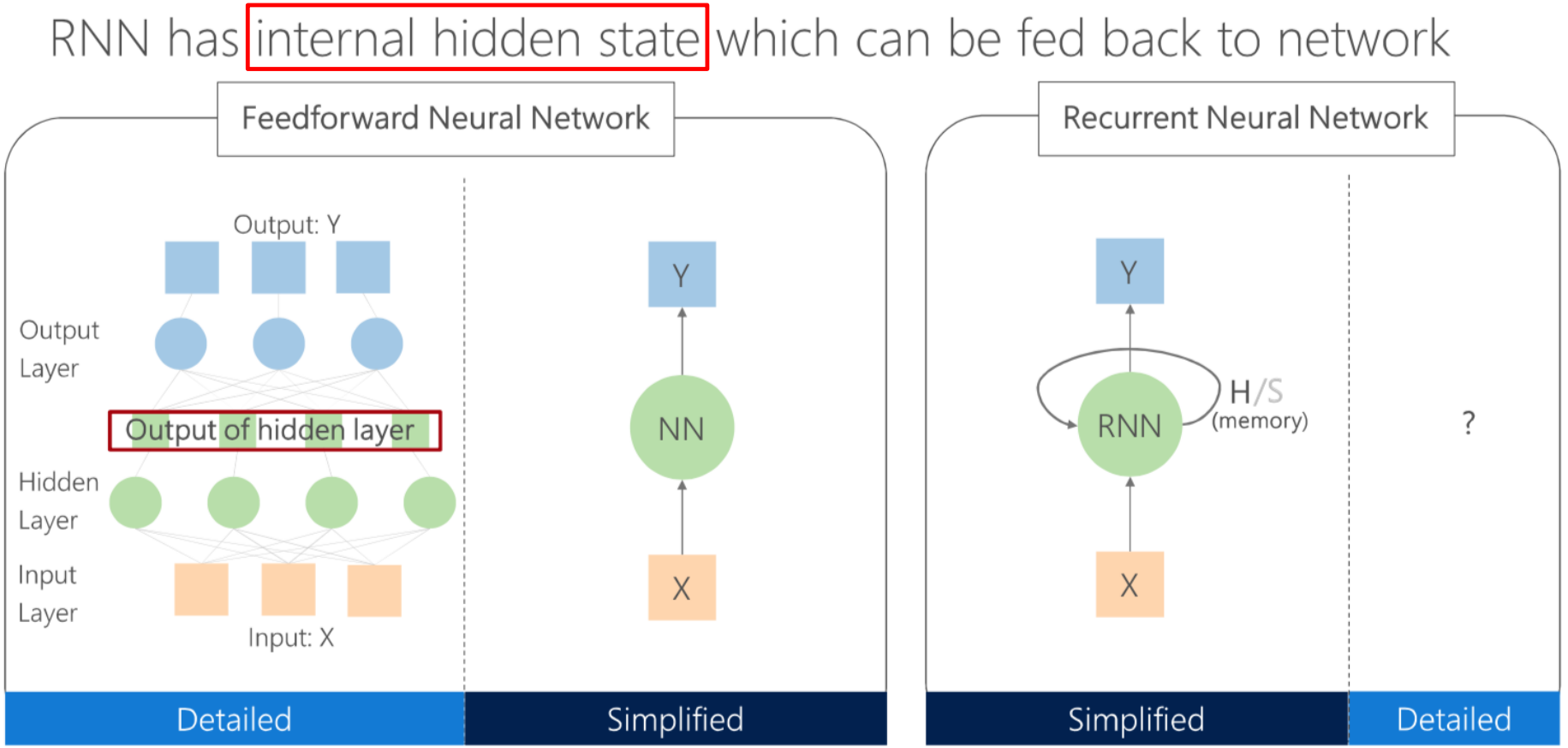


RNN unrolled

- 좌측
  - ➢ 순환신경망 하나의 유닛(셀)을 그린 것
  - ➢ 현재의 상태값이 다음번 상태값을 구하는데 순환적으로 사용
- 우측
  - ➢ 이해하게 쉽게 시간축으로 나누어 그린 것

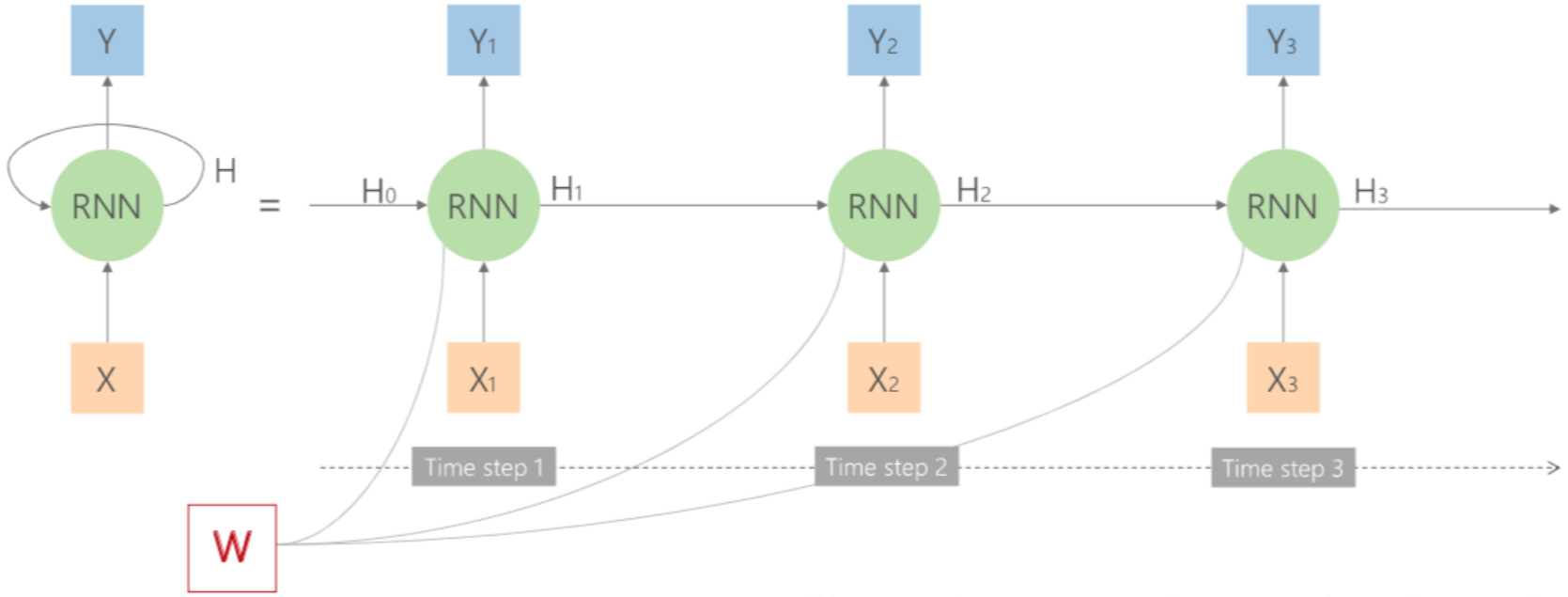


RNN vs FF 신경망

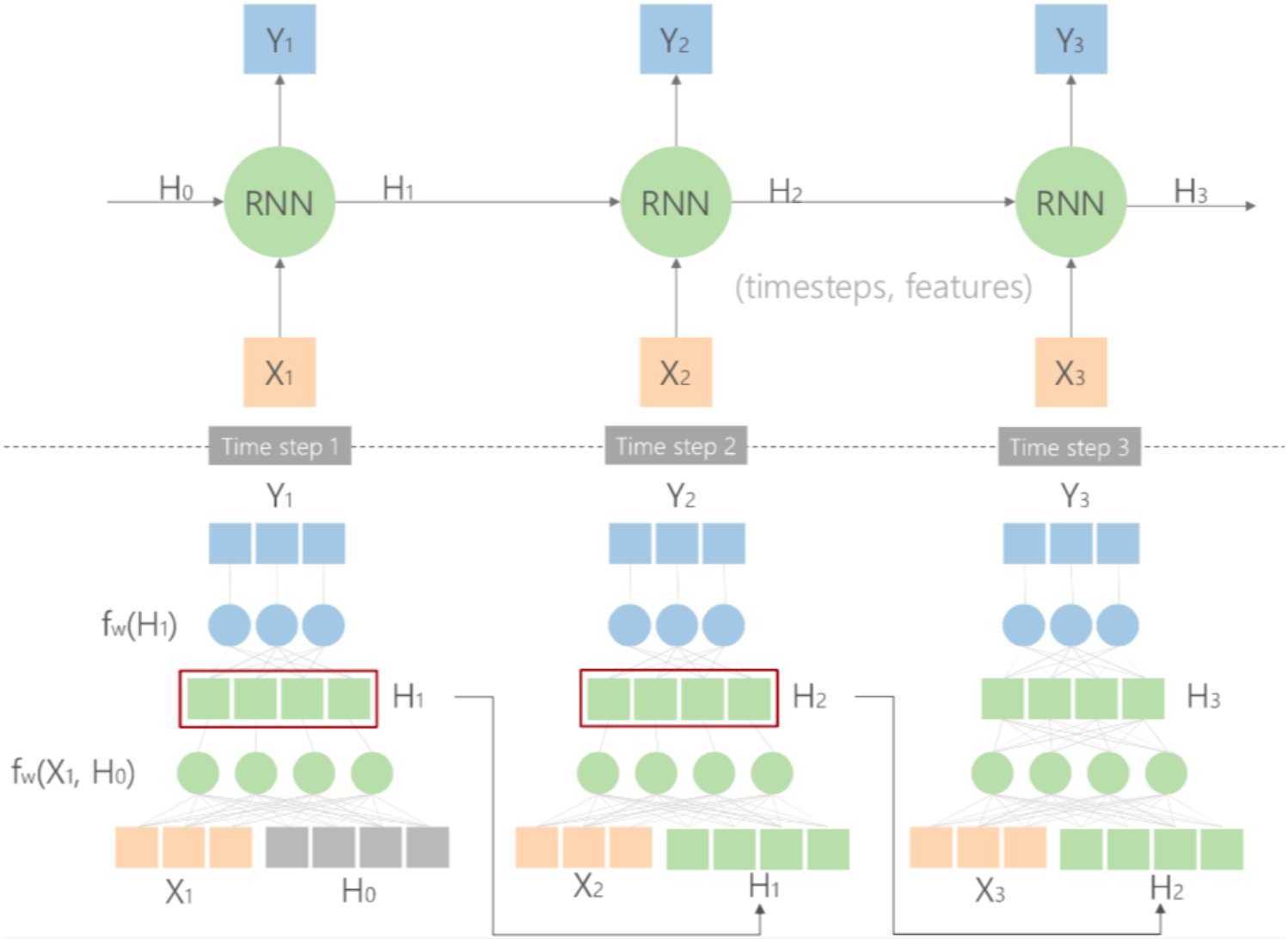

RNN has internal hidden state which can be fed back to network
Feedforward Neural Network
Output: Y
Output Layer
Output of hidden layer
Hidden Layer
Input Layer
Input: X
Y
NN
X
Detailed
Simplified
Recurrent Neural Network
Y
RNN
H/S
(memory)
X
?
Simplified
Detailed

RNN unrolled

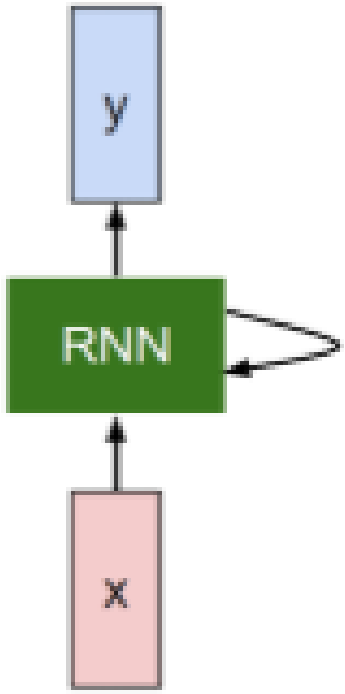
- 매 timesteps에 동일한 가중치(weight)를 사용



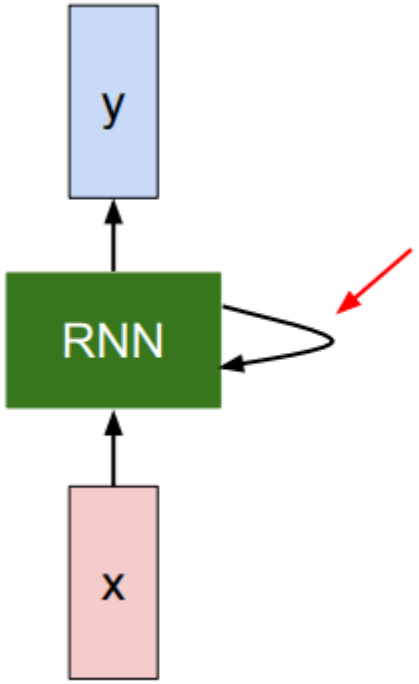
상세 동작


Y1
Y2
Y3
H0
RNN
H1
RNN
H2
RNN
H3
(timesteps, features)
X1
X2
X3
Time step 1
Time step 2
Time step 3
Y1
Y2
Y3
fw(H1)
H1
H2
H3
fw(X1, H0)
X1
H0
X2
H1
X3
H2

RNN 기본 구조



RNN 기본 구조 – 내부 상태 보유



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

RNN 기본 구조 – 내부 상태 보유

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

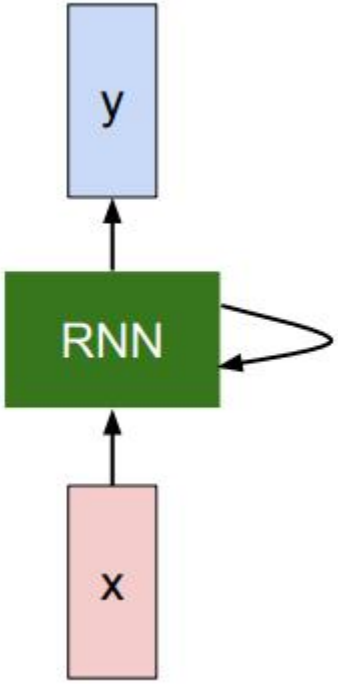
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

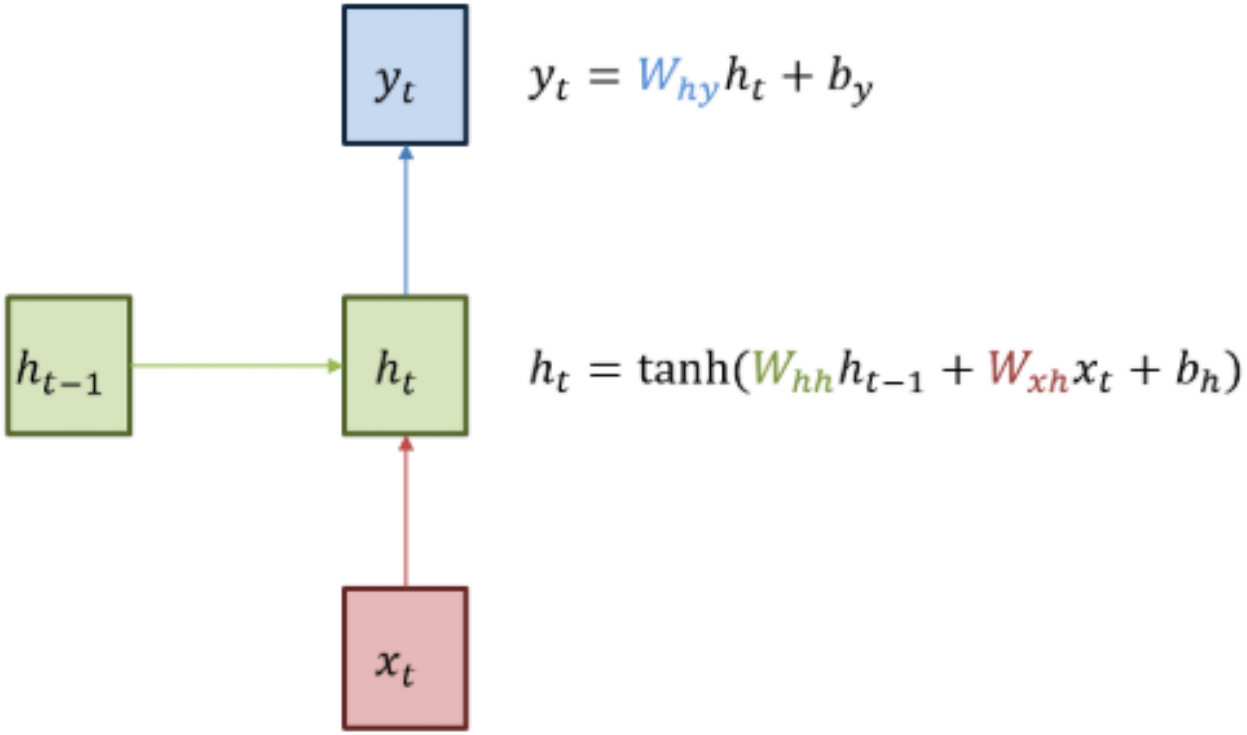
old state

input vector at some time step



기본 순환 신경망

- **현재의 출력**
  - 현재의 상태값으로부터 구한다



기본 순환 신경망 – 과거 정보 사용

- 모델의 **현재 상태값**을 얻기 위해서
  - **현재의 입력정보** 뿐 아니라 모델의 **이전의 상태 정보**도 동시에 사용

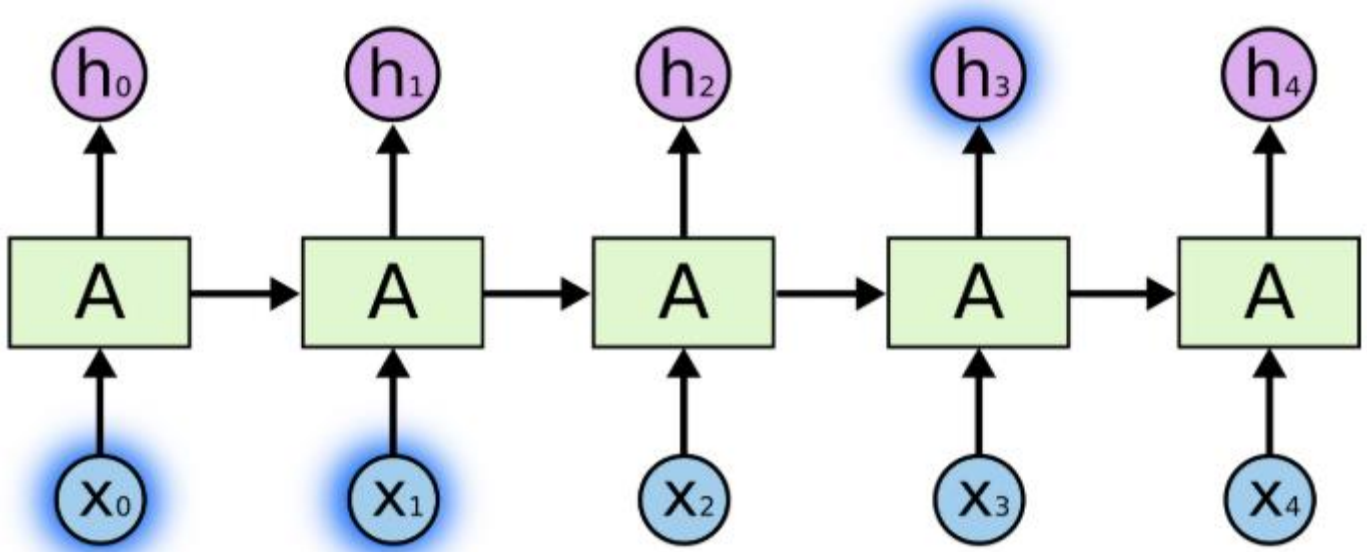
- 수식으로 표현

$$H_t = UX_t + WH_{t-1}$$

  - X(t) : 현재 입력 정보
  - H(t-1) : 이전 상태 정보
  - U, V : 가중치(계수) 매트릭스

- 이로써 **과거의 정보**를 내부적으로 재사용

기본 순환 신경망 – 과거 정보 사용



기본 순환 신경망 – 상태 , 출력

- 현재 상태 상태 활성화 함수
  - Tanh함수를 주로 사용 (참고로 CNN에서는 ReLU)
  - W : 과거의 상태 정보를 얼마나 많이 활용하는지를 결정

$$H_t = \tanh(WH_{t-1} + UX_t)$$

- 현재의 출력
  - 현재의 상태값으로부터 구한다

$$y_t = VH_t + c$$

기본 순환 신경망 – 동일한 가중치 사용

- 모든 step에서 동일한 필터(가중치)를 사용
  - 계수 U, V, W는 시각(t)의 함수가 아니라, 모든 step에서 동일

- 순환신경망을 학습
  - 가중치 매트릭스 U, V, W의 값이 최적화되도록 학습하는 것

기본 순환 신경망 – 상태 , 출력

- 우리가 다른 사람의 말을 듣고 대답을 한다면,
  - 먼저 머릿속에 상대방의 말을 이해하는 상태를 만들어야 하고(즉, H_t)
  - 이 상태에 기반하여 출력(y_t)를 만들어내는 것과 유사한 개념

- 입력(X)의 단위
  - 한 단어일 수도 있고
  - 10개의 단어일 수도 있고, 한 문장일 수도

- 현재 출력을 구하는데 이전의 상태 정보를 순환적으로 사용
  - 결국 최초의 상태값을 포함해서 오래된 상태 정보를 조금이라도 활용한다고 볼 수 있다

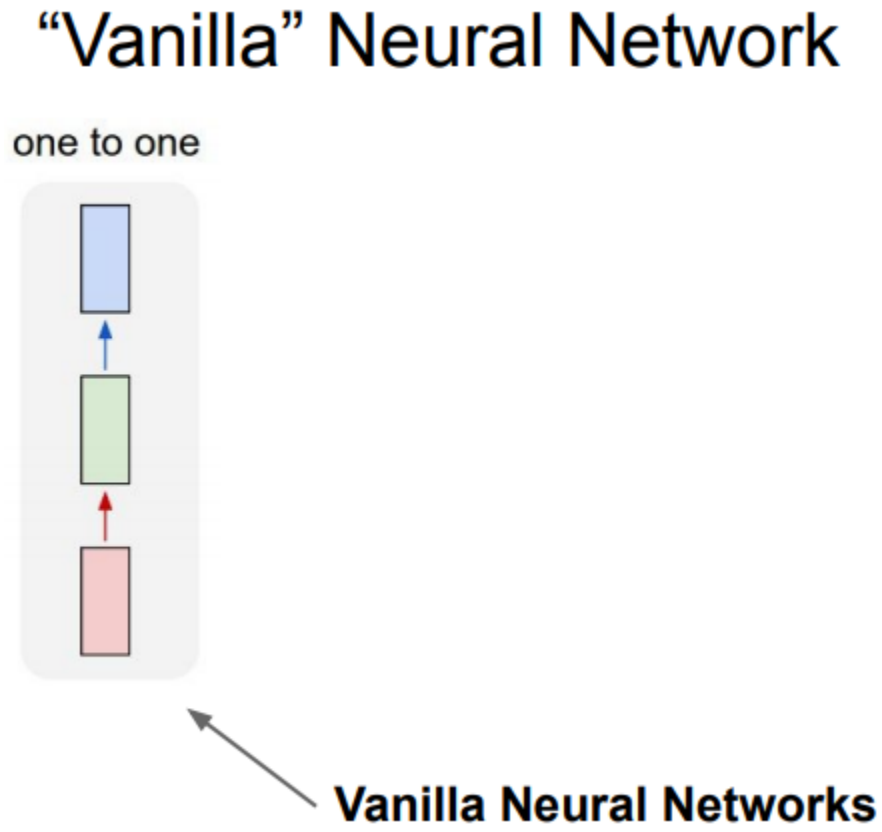
기본 순환 신경망 – 기울기 소실 문제

- 과거의 오래된 신호를 적은 비중으로 일괄적으로 반영하면 신호의 업데이트가 너무 적어져서 기울기 소실 문제가 발생

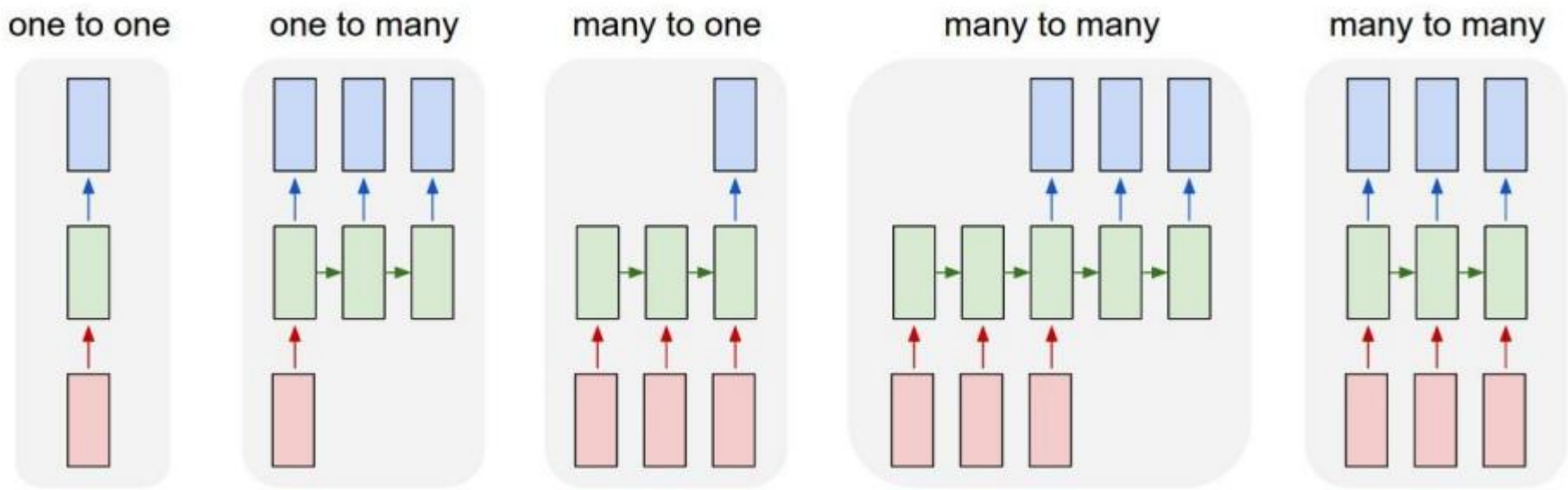
- 과거의 신호를 너무 많이 반영하면 신호가 발산하는 문제가 발생

- 해결책
  - LSTM(Long-Short Term Memory) 기법이 널리 사용
  - 오랜 기간 중요도를 유지할 정보와 그럴 필요 없이 망각해야 할 신호를 구분하여 따로 처리

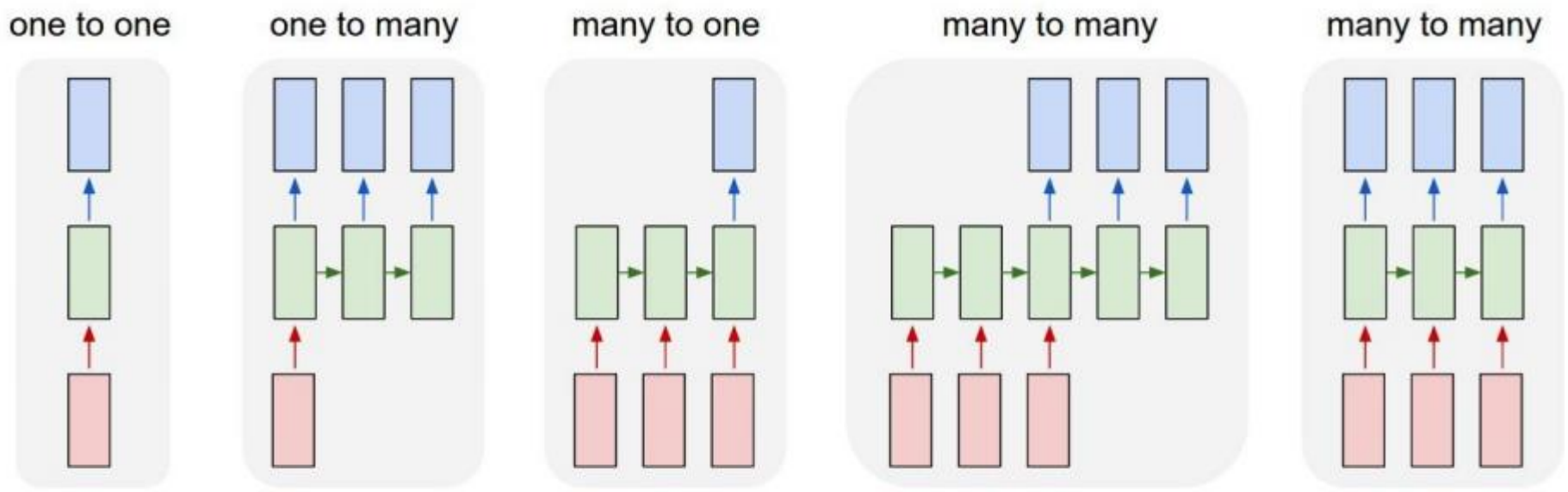
RNN 기본 모델 유형



RNN 기본 모델 유형

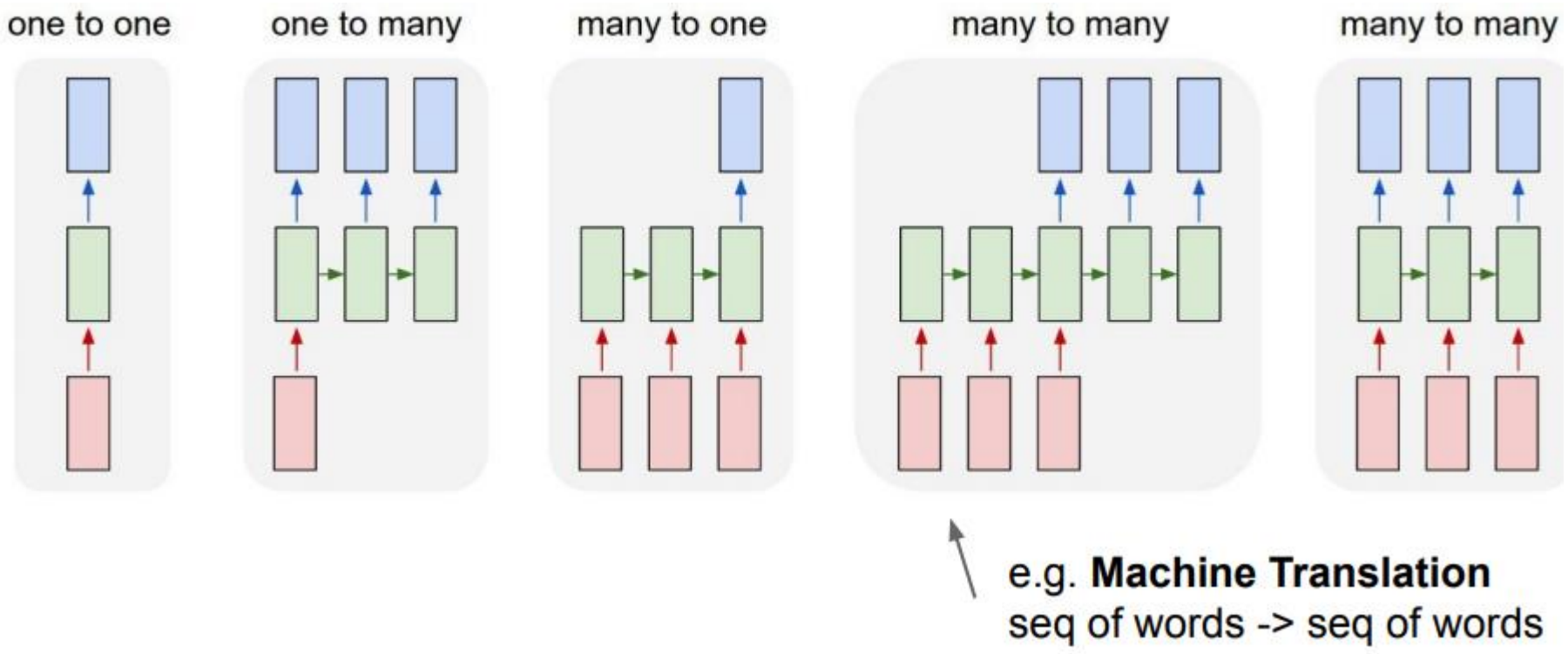

one to one
one to many
many to one
many to many
many to many
e.g. **Image Captioning**
image -> sequence of words

RNN 기본 모델 유형

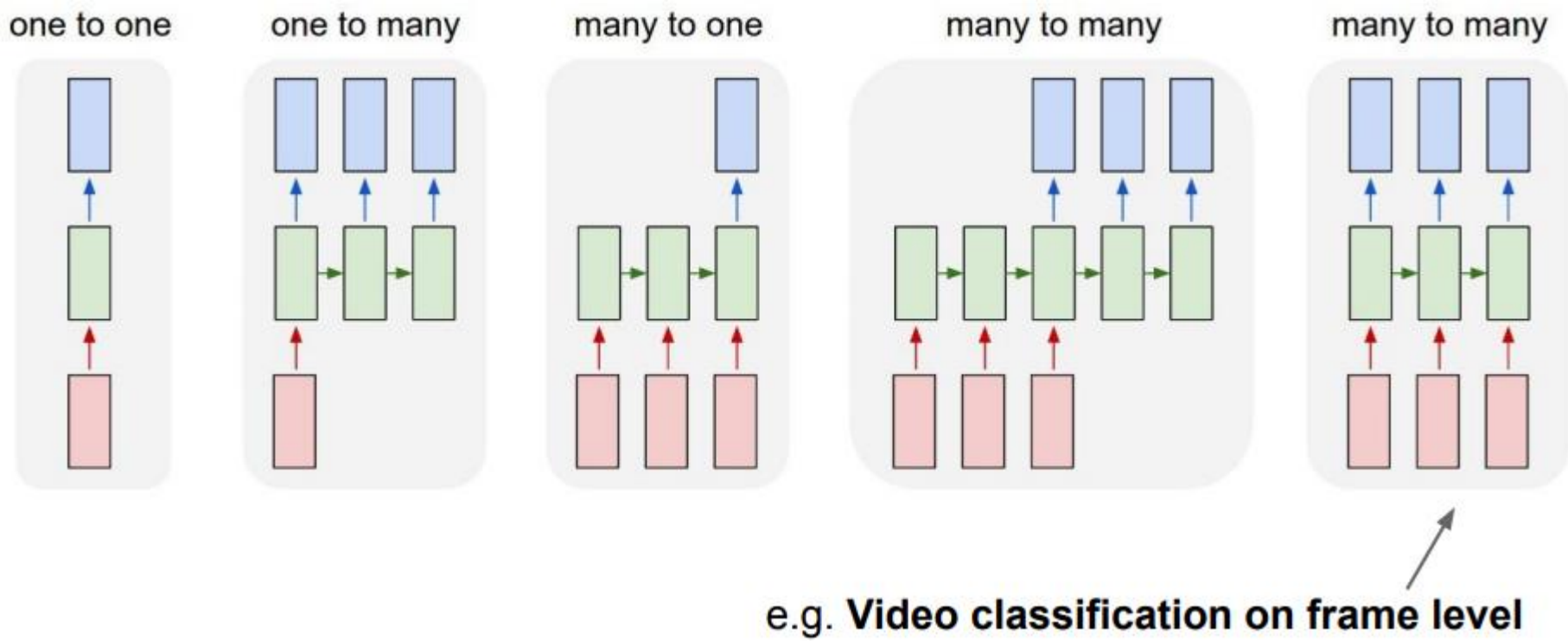

one to one
one to many
many to one
many to many
many to many
e.g. **Sentiment Classification**
sequence of words -> sentiment

RNN 기본 모델 유형

- 언어 모델(language model)
  - 일정 크기의 문장을 입력하면 이에 상응하여 한 글자씩 다음 위치에 있는 글자나 단어를 출력
  - 매 입력마다 출력이 발생



RNN 기본 모델 유형


one to one
one to many
many to one
many to many
many to many
e.g. **Video classification on frame level**

RNN 모델 – 인코더-디코더 (many to many)

- 인코더-디코더 모델
  - 일종의 시퀀스-시퀀스 모델
  - 하나의 문장을 모두 듣고 이를 번역 또는 다른 대답을 하는 모델 등에 사용

- 시퀀스-시퀀스 모델
  - 하나의 시퀀스를 모두 입력하면 그 이후에 출력이 시퀀스 형태로 나오는 것

- 인코딩 된 결과는 디코더로 전달
  - context vector 또는 machine thought 벡터 형태로 전달

- teacher forcing
  - 예측한 출력이 아니라 실제 레이블 값으로 학습시키는 방법

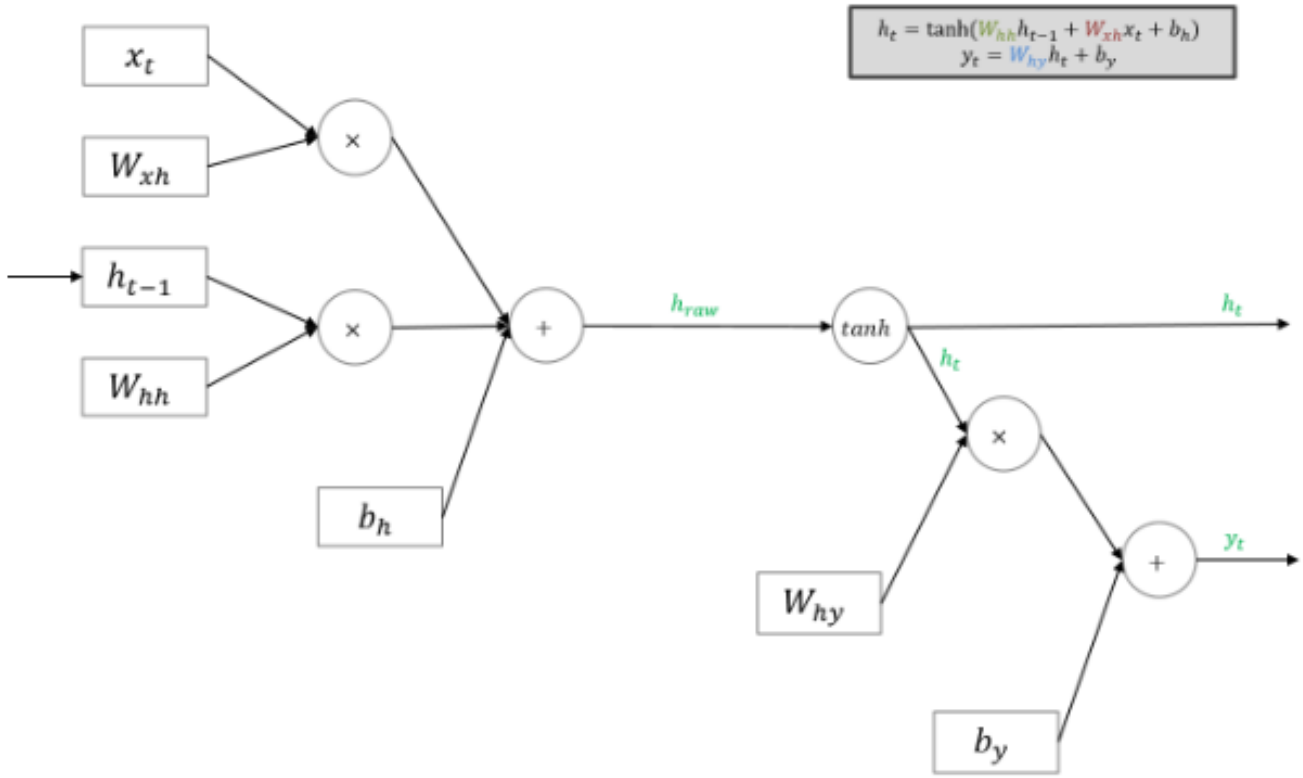
RNN 학습 – BPTT

- 경사하강(gradient descent) 방식의 학습을 주로 사용
  - 오류역전파 방식을 이용
  - 모델의 출력과 레이블 사이에 오류가 발생하며 이로부터 계산된 손실의 경사 값을 앞단쪽의 계층으로 전달하여 가중치를 조절

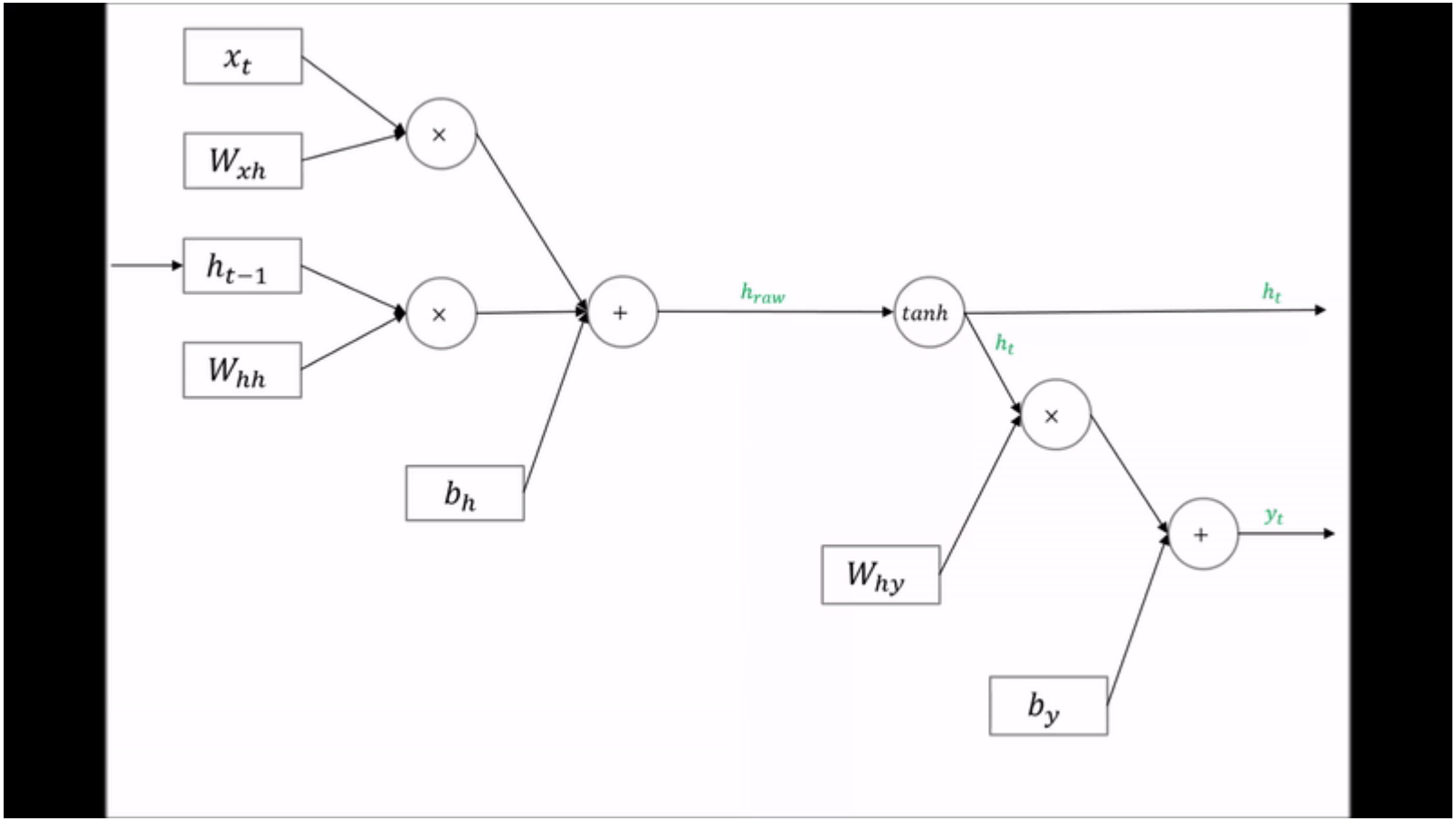
- 학습에 의해서 가중치 매트릭스 W, U, V가 업데이트 되는데 RNN에서는 모든 타임 스텝에서 동일한 가중치를 사용한다는 것을 주의해야 한다.

- 즉, 현재의 경사(gradient) 변화는 과거 스텝의 계산에도 영향을 미친다. 이러한 현상을 BPTT(backpropagation through time)이라고 함

RNN – 순전파


$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$
$y_t = W_{hy}h_t + b_y$
x_t
W_{xh}
h_{t-1}
W_{hh}
b_h
×
×
+
h_{raw}
tanh
h_t
h_t
×
W_{hy}
+
y_t
b_y

RNN – 역전파



RNN의 문제점

- RNN에서 여러 계층을 거치는 경우, 예를 들어 단어가 수십 개로 된 문장을 해석하는 경우, 오차 역전파를 하면서 경사값이 거의 사라지거나 또는 너무 큰 값으로 발산하여 RNN이 제대로 동작하지 못하기 쉽다
  - 오래된 정보를 모두 중요시하면 정보가 너무 많이 축적되어 발산할 우려
  - 오래된 정보를 약하게 반영하면 오래되었지만 중요한 정보를 캐치하지 못하는 즉, 소실되는 우려

- 이를 해결하기 위해서
  - LSTM(Long-Short Term Memory), GRU 기법이 제안
  - LSTM에서는 오랜기간 중요도를 유지할 정보와 그럴 필요 없이 망각해야 할 신호를 구분하여 따로 처리

단어 임베딩

단어 임베딩의 정의

- 앞에서 소개한 세 가지 텍스트 코딩 방식인 원핫 인코딩, BOW(단어모음), 문서-단어 행렬방식은 단어마다 고유번호를 배정하여 사용
- 그러나 이 고유 번호 숫자에는 아무런 의미가 들어 있지 못하며 단지 인덱스의 성격만 갖는다.
- 단어를 인덱싱이 아니라, 의미 있는 숫자 들의 집합, 즉, 벡터로 표현하는 방법이 단어 임베딩 (Word Embeding)이다.

단어 임베딩

- 원-핫 인코딩
  - 근본적으로 문장내에서 단어의 존재 여부만 코딩
  - 문장내에서 단어가 가지고 있는 정보(문장 구조 정보)를 모두 잃어버린다.
- 문서 단어 행렬
  - 문서 단위로 원-핫 인코딩을 사용 & 단어사전 크기의 벡터를 사용
  - 이 벡터는 크기가 매우 크고 희소한 구조를 가지고 있다.
- 단어 임베딩
  - 단어를 표현하는 방법을 개선
  - 단어를 물리적으로 의미가 있는 고차원 공간상의 위치(점)으로 표현

단어 임베딩

- 단어 임베딩
  - 단어를 표현하는 방법을 개선
  - 물리적으로 의미가 있는 고차원 공간상의 위치(점)으로 단어를 표현
  - 단어 벡터를 사용함으로써 희소 벡터대신 밀집 벡터를 사용
  - id를 구분하는 용도가 아니라 공간상의 점을 매핑할 수 있게 됨
  - 이로 인해 단어 사이의 거리를 계산할 수 있고 단어 간의 관계를 벡터로 표현 가능
- 단어 임베딩은 텍스트 분석에서 널리 사용되며 자연어 처리, 문서 분류, 감성분석, 저작 식별, QA, 챗봇 등에서 BoW나 문서 단어 행렬에 비해 높은 성능을 냄

단어 벡터 (Word Vector)

- 단어 벡터 (Word Vector)
  - 밀집 표현
    - 벡터의 차원을 단어 집합의 크기가 아니라 사용자가 설정한 값으로 모든 단어의 벡터 표현의 차원을 맞추는 방식
    - 더 이상 0/1 만의 값이 아니라 실수 값을 가지게 됨
  - 각 단어를 50~300개 정도의 차원으로 구성된 벡터로 표현

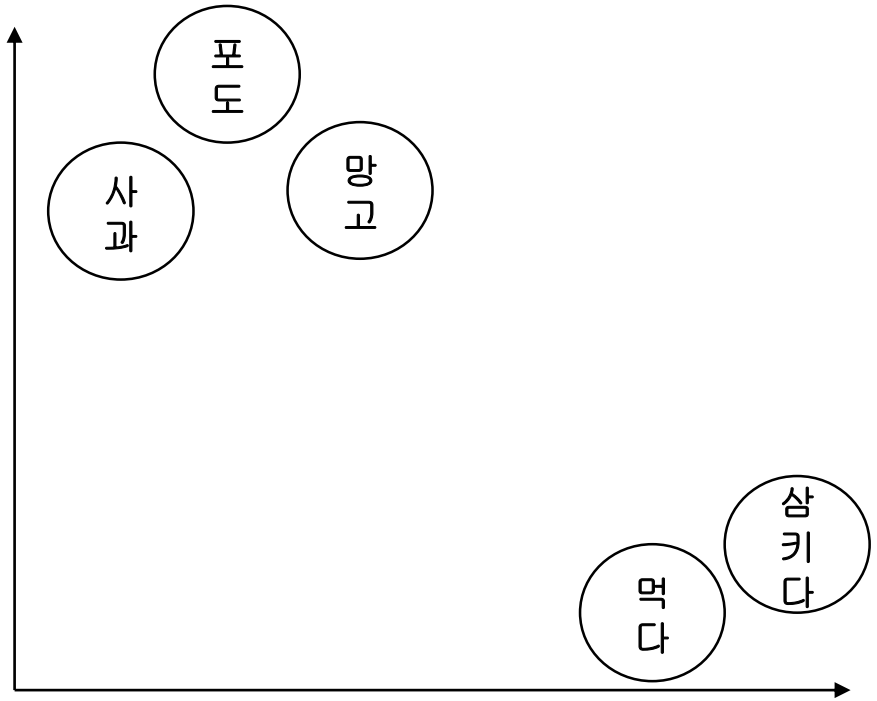
    학교 = [0.23, 0.58, 0.97, … , 0.87, 0.95]
    바다 = [0.45, 0.37, 0.81, … , 0.22, 0.64]

- 단어 벡터를 사용하면
  - 각 단어들 사이의 "거리" 를 계산이 가능
  - 거리를 기반으로 유의어/반대어 등을 찾아낼 수 있다
  - 동물의 성별, 단수/복수, 동사/명사를 구분할 수도 있다
  - 그러나 각 벡터 값의 의미는 알 수 없다

단어 벡터

- 단어 벡터는 대형 말뭉치로부터 학습
  - 말뭉치의 문장들을 계속 입력하여 학습을 시키면 단어 벡터를 얻을 수 있다
  - 예를 들어 음식과 관련된 다음과 같은 문장들로 학습을 시키면 다음과 같은 단어 벡터를 얻을 수 있을 것이다.
  - 학습에 사용된 문장 예:

"나는 어제 바나나를 맛있게 먹었다"
"이 망고는 먹기가 힘들다"
"이 사과는 씹는 맛이 아주 좋다"
"바나나가 사과보다 맛있다"
"잘 씹어야 맛있게 먹을 수 있다"
…



단어 벡터

- **이미 만들어져 있는 단어 벡터를 가져다 사용할 수도 있다.**
- **GloVe**(Global Vectors for Word Representation)
  - **2014년 스탠포드에서 개발**
  - **카운트 기반과 예측 기반을 모두 사용하는 방법론**
    - **실제 구현할 때는 hyper parameter의 튜닝 여부에 따라 Word2Vec와 큰 성능 차이는 없다고 알려짐**
  - **위키피디아 데이터로부터 학습**
  - **40만개 단어를 100차원으로 임베딩**
  - npl.stanford.edu/projects/glove **에서 다운로드**

단어 임베딩 방법

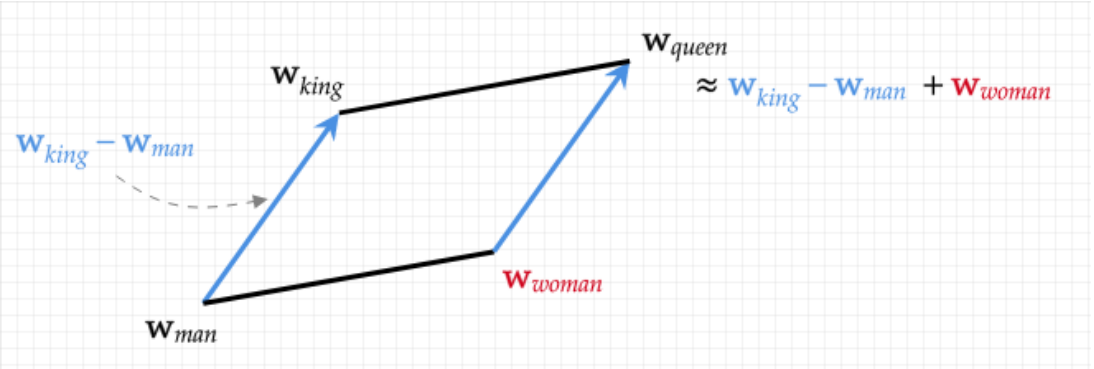
- 단어 임베딩 방법론
  - LSA
  - Word2Vec
  - FastText
  - Glove 등
- Keras 에서 제공하는 도구인 keras.Embedding() 레이어
  - 단어를 랜덤한 값을 가지는 밀집 벡터로 변환한 뒤에, 인공 신경망의 가중치를 학습하는 것과 같은 방식으로 단어 벡터를 학습하는 방법을 사용
- 원-핫 벡터과 임베딩 벡터의 차이

	원-핫 벡터	임베딩 벡터
차원	고차원 (단어 집합의 크기)	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련데이터에서 학습
값의 타입	1, 0	실수

단어 벡터 생성

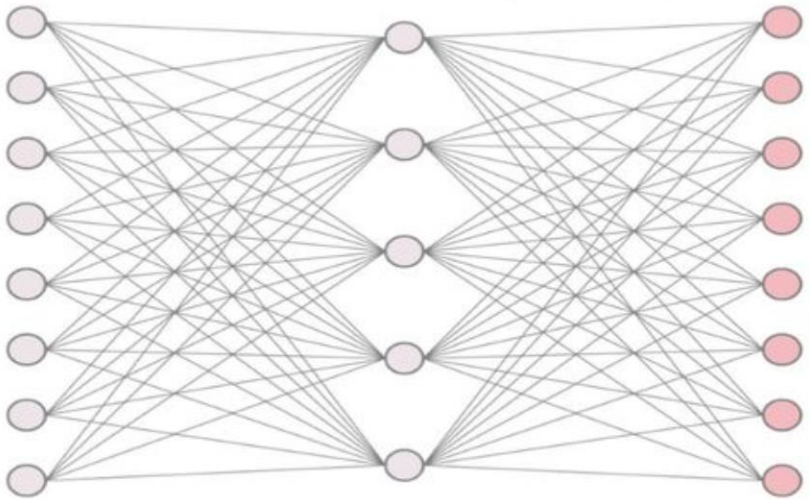
- Word2Vec
  - 2013년 구글에서 제안
  - 종래의 NNLM(Neural Net Language Model)을 기반으로 대량의 문서 코퍼스를 처리 가능
  - vector(“king”)-vector(“man”+vector(“woman”)=vector(“queen”) 이라는 예제를 이용해 관계를 연산으로 설명할 수 있다는 벡터 표현이 가능하다는 것을 보여주면서 관심을 받았음

http://word2vec.kr



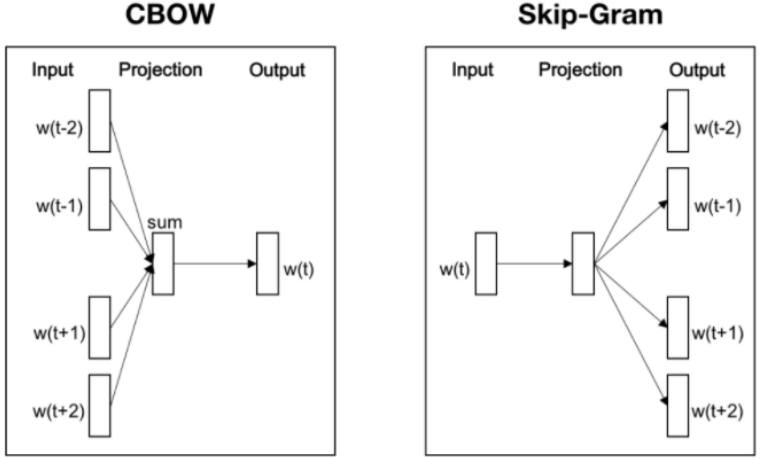
단어 벡터 생성 – Word2Vec

- Predict words using context
  - Concept: '비슷한 위치 (근처)에서 등장하는 단어들은 비슷한 의미를 가진다'
- Word2Vec Neural Network model
  - NN with single hidden layer
  - Often used for auto-encoder to compress input vector in hidden layer



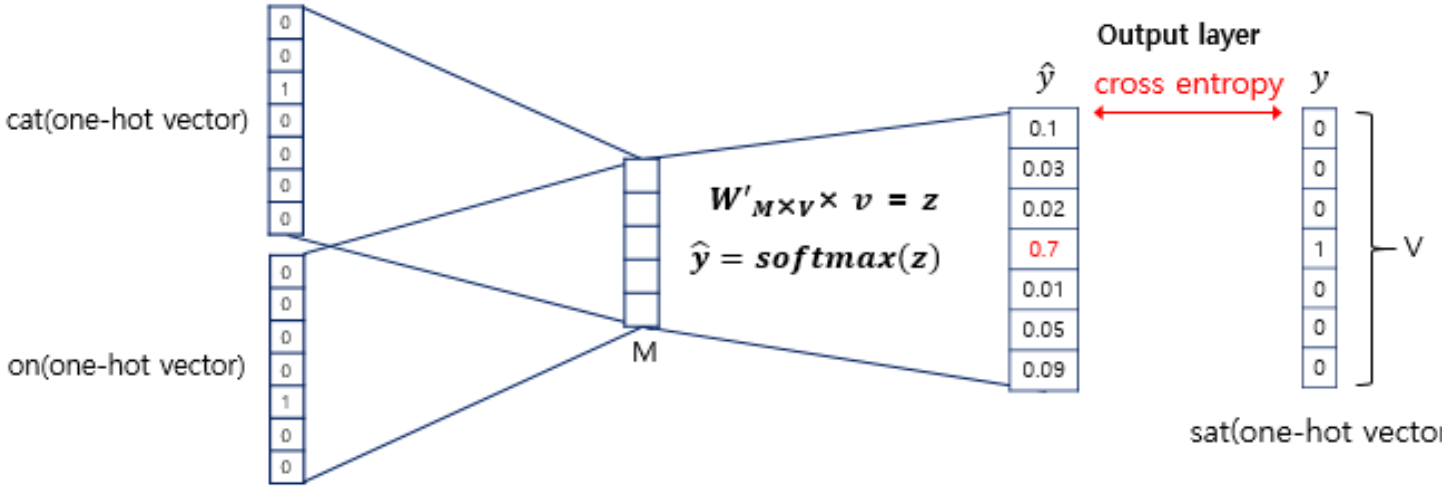
단어 벡터 생성

- Word2Vec
  - CBOW & Skip-Gram 두 가지 방식이 있음
  - CBOW (Continuous Bag of Words)
    - 주변에 있는 단어들을 가지고, 중간에 있는 단어들을 예측하는 방법
    - 즉, 문맥 단어를 보고 기준 단어가 무엇인지 예측하는 모델
  - Skip-Gram
    - 중간에 있는 단어로 주변 단어들을 예측하는 방법
    - 즉, 기준 단어를 보고 어떤 문맥 단어가 등장할지 예측하는 모델
  - 메커니즘 자체는 거의 동일 & 많은 경우에 Skip-Gram이 더 우수



단어 벡터 생성 – Word2Vec

- CBOW
  - 예문 "The fat cat sat on the mat"에서 가운데 단어를 예측
  - 즉, {"The", "fat", "cat", "on", "the", "mat"}으로부터 sat을 예측
  - 예측해야 하는 단어 sat을 중심 단어(center word)라고 하고, 예측에 사용되는 단어들을 주변 단어(context word)



단어 벡터 생성 – Word2Vec

- (ex) “I like playing football with my friends” with window = 2

center word	context words
[1,0,0,0,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0]
[0,1,0,0,0,0,0]	[1,0,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,1,0,0,0]
[0,0,1,0,0,0,0]	[1,0,0,0,0,0,0] [0,1,0,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,1,0,0]
[0,0,0,1,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,0,1,0,0] [0,0,0,0,0,1,0]
[0,0,0,0,1,0,0]	[0,0,1,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,0,1,0] [0,0,0,0,0,0,1]
[0,0,0,0,0,1,0]	[1,0,0,1,0,0,0] [0,0,0,0,1,0,0] [0,0,0,0,0,0,1]
[0,0,0,0,0,0,1]	[0,0,0,0,1,0,0] [0,0,0,0,0,1,0]

Training samples

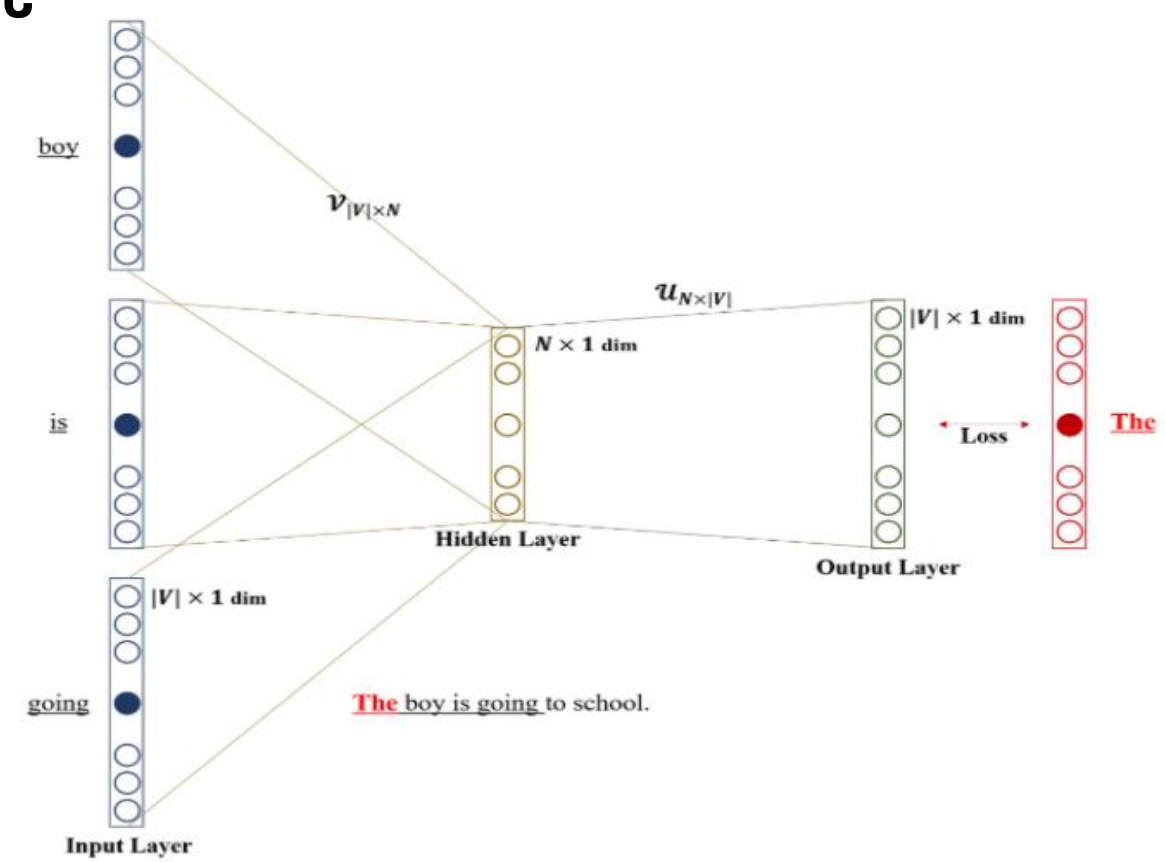
(I, like)
(I, playing)

(like, I)
(like, playing)
(like, football)

(playing, I)
(playing, like)
(playing, football)
(playing, with)
.
.
.
.
.
.
.
.
.
.
.
.
.

단어 벡터 생성 – Word2Vec

- CBOW architecture



단어 벡터 생성 – Word2Vec

- **CBOW Training**

1. (Input Layer) context word 각각의 one-hot vector를 입력

$$x^{(c-m)}, \cdots, x^{(c-1)}, x^{(c+1)}, \cdots, x^{(c+m)} \in \mathbb{R}^{|V|}$$

2. (Input Layer -> Hidden Layer) v = V * input (N-dim vector)

$$v_{c-m} = \mathcal{V}x^{(c-m)}, \cdots, v_{c+m} = \mathcal{V}x^{(c+m)} \in \mathbb{R}^{n}$$

3. (Hidden Layer) averaging v's of Step.2 -> v_hat

$$\hat{v} = \frac{v_{c-m} + \cdots + v_{c+m}}{2m} \in \mathbb{R}^{n}$$

4. (Hidden Layer -> Output Layer) z = U * v_hat

$$z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$$

5. (Output Layer) y_hat = softmax(z)

$$\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$$

6. (Loss) error = y_hat – y(center word)

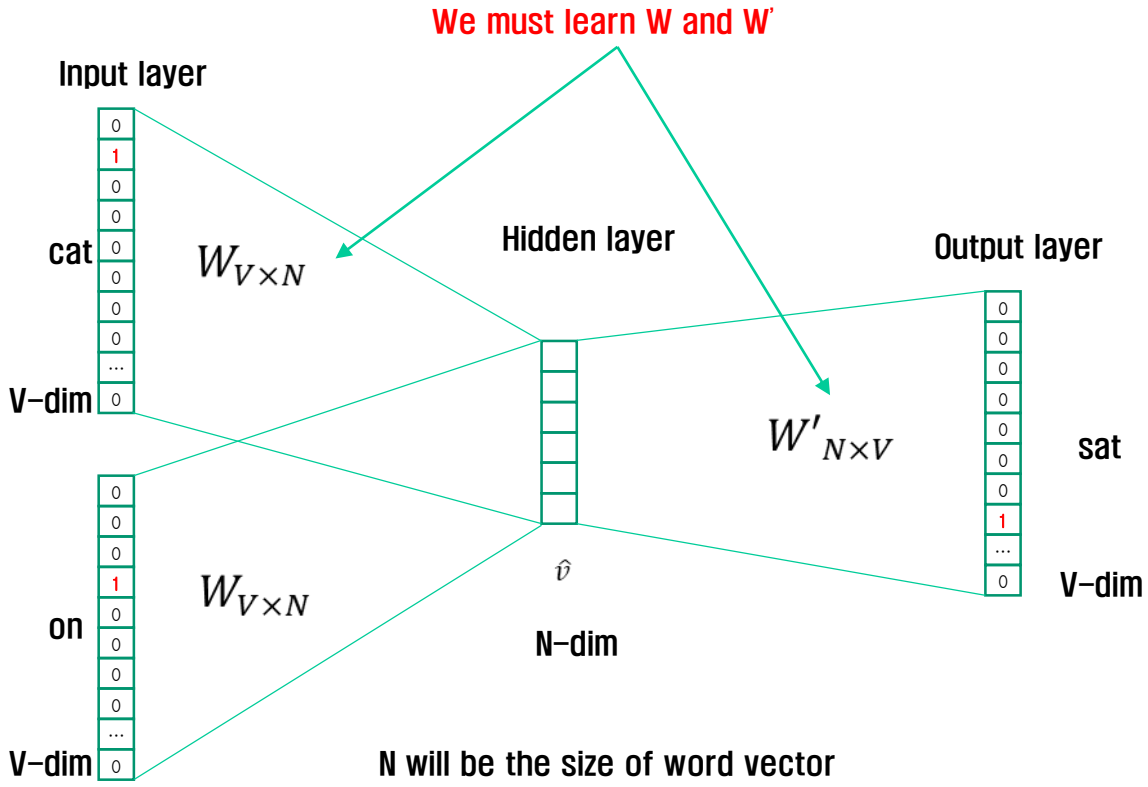
$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log \hat{y}_j = -y_i \log \hat{y}_i$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\hat{y}, y)$$

https://wooono.tistory.com/244

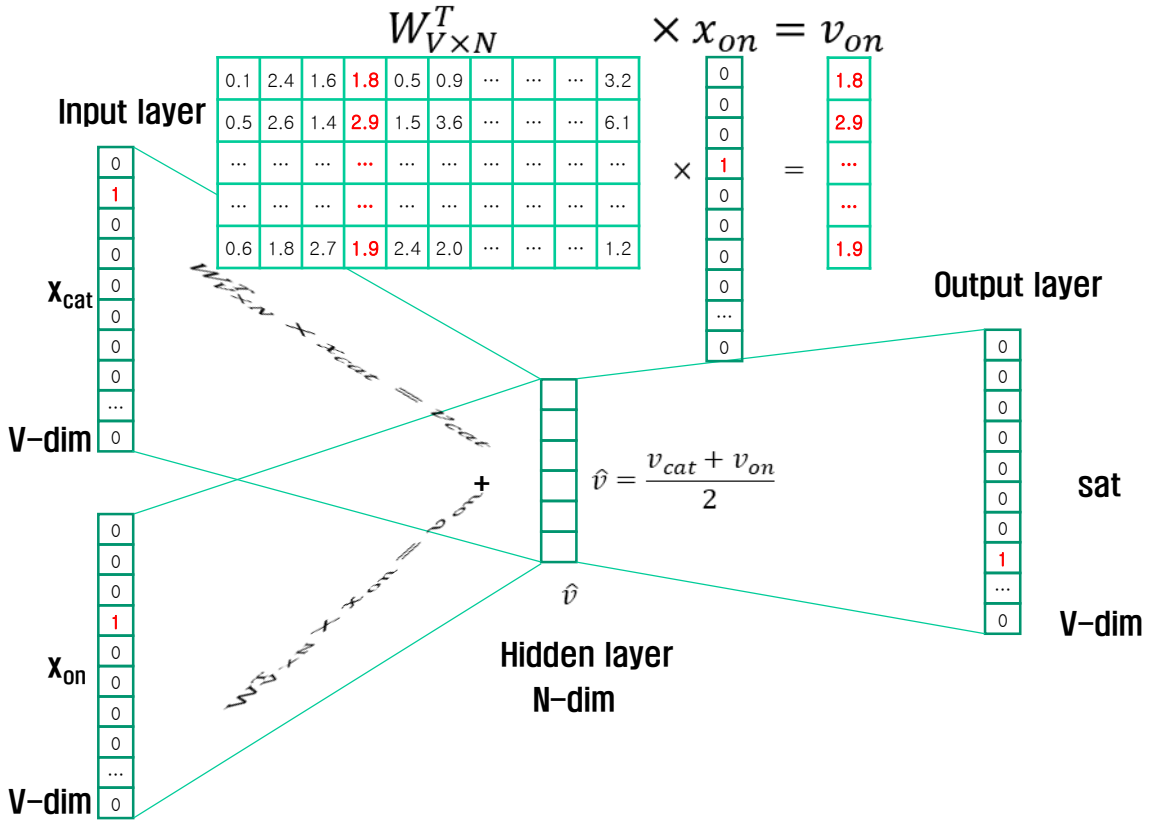
단어 벡터 생성 – Word2Vec

- (ex) “The cat sat on floor” with window_size = 1



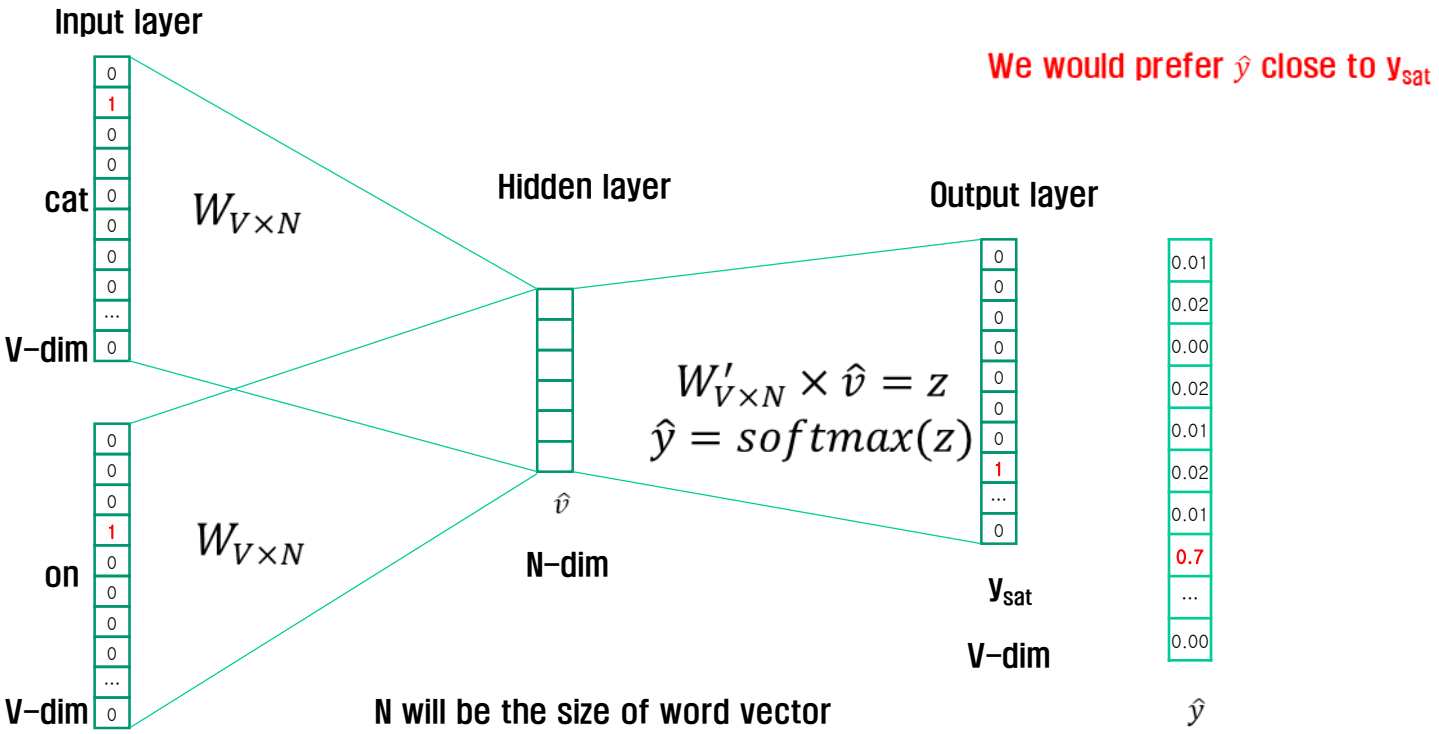
단어 벡터 생성 – Word2Vec

- (ex) “The cat sat on floor” with window_size = 1



단어 벡터 생성 – Word2Vec

- (ex) “The cat sat on floor” with window_size = 1



단어 벡터 생성 – Word2Vec

- **단어 벡터 만드는 과정을 소개**
  - **가장 널리 사용되는 라이브러리 : Gensim**
    - pip install gensim

```
from gensim.models.word2vec import Word2Vec
model = Word2Vec(sentense_list, min_count=1)
model.most_similar(positive="조선")

##
[('일본', 0.9953970909118652),
 ('관련', 0.9941188097000122),
 ('인물', 0.9938454031944275),
 ('러시아', 0.9931197166442871),
 ('주요', 0.9918481111526489),
 ('대원군', 0.9915156960487366),
...
```