

강의 : [https://www.youtube.com/watch?v=QQQ-W\\_63UgQ&list=PL3FW7Lu3i5Jsnh1rnUwq\\_TcyINr7EkRe6](https://www.youtube.com/watch?v=QQQ-W_63UgQ&list=PL3FW7Lu3i5Jsnh1rnUwq_TcyINr7EkRe6)

강의자료 : <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/syllabus.html>

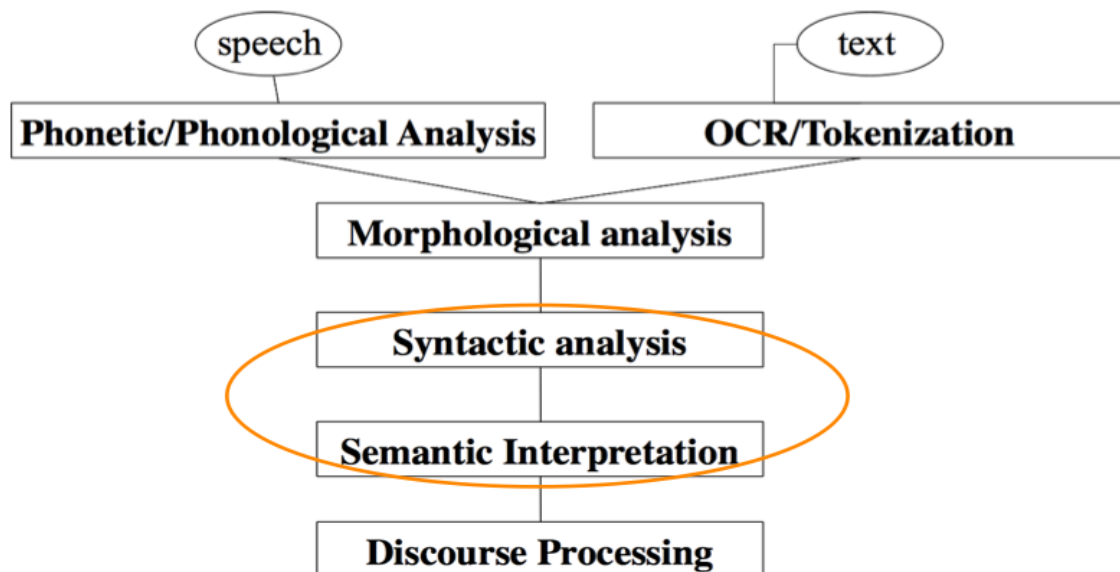
강의 정리 참고 자료:

<https://m.blog.naver.com/PostView.nhn?blogId=jujbob&logNo=221147988848&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F>

## Lecture 1. Introduction

### 1. What is Natural Language Processing (NLP)?

자연어처리는 컴퓨터과학, 인공지능, 언어학에 속하는 분야이다. 자연어처리의 목표는 컴퓨터가 자연어를 이해하거나 처리하여 유용한 작업을 수행할 수 있도록 하는 것이다. 그 예로는 물건을 사게 하는 등의 작업을 하거나, 언어 번역, 질의 응답(Siri etc) 등이 있다. 언어의 의미를 완전히 이해하거나 표현하게 하는 것은 매우 어려운 목표이기 때문에 언어를 완벽하게 이해한다면 인공지능이 완성되었다고 볼 수 있다.



자연어처리는 위 그림과 같이 구분할 수 있다.

- **Phonetic/Phonological Analysis** : 음성 분석
- **Morphological analysis** : 형태론적 분석 (**incomprehensible** 이라는 단어는 **in** 이 앞에 붙고 **ible** 이 뒤에 붙어 단어를 만들고 있다)
- **Syntactic analysis** : 통사론적 분석 (문장의 구조를 이해하는 과정이다. I sat on the bench. 라는 문장에서 I 는 주어, sat 은 동사, on the bench 는 위치이다.)
- **Semantic Interpretation** : 의미론적 이해 (각각 단어의 뜻만 안다고 해서 사람의 말을 이해할 수 있는 것은 아니다. 문맥을 이용해 문장의 의미를 이해하는 과정이다.)
- **Discourse Processing** : 담화 처리 (문맥 이해)

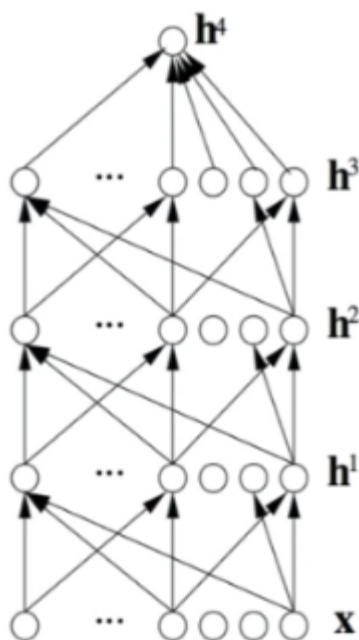
자연어처리가 활용되는 분야는 다음과 같다. 맞춤법 체크, 키워드 검색, 유사어 찾기, 웹사이트로부터 정보를 추출하기, 문서를 읽고 긍정/부정 판별하기 (댓글을 보고 사람들이 어떤 반응을 보이는지 알 수 있다.), 기계 번역, 대화 시스템, 질의 응답 등이 있다.

### What's special about human language?

인간의 언어는 말하고자 하는 사람의 의도를 전달하기 위해 만들어진 시스템이다. 사람은 언어를 빠르게 인코딩하여 이해할 수 있다. 보통 사람의 언어는 **discrete/symbolic/categorical 한 신호 시스템**이다. 음성, 손동작, 문자로 분류되는 상징은 소통의 신호로 해석할 수 있다. (여러 방법으로 인코딩된다고 표현)

인간의 언어는 상징적이고 분류할 수 있는 신호 시스템이다. 하지만 인간의 뇌는 연속적인 패턴으로 인코딩한다. 즉 뇌는 이산적인 상징을 연속적인 신호로 변환한다.

### 2. What's Deep Learning (DL)?



딥러닝은 기계학습의 하위분야이다. 기계학습은 사람이 디자인한 표현과 입력 feature 덕분에 잘 작동한다. 즉, 기계학습은 인간이 직접 문제 해결을 하기 위해 feature를 선택하여 기계를 학습시킨다. input과 feature을 인간이 미리 다 정해 놓은 후 기계를 학습시키는데 이때 기계의 역할은 weight를 최적화 시키기 위한 계산을 하는 것이다. 따라서 이와 같은 **기계학습**은 기계가 정말 "학습" 한다고 할 수 없다.

기계가 인간처럼 raw data로 부터 자동으로 feature를 선택하게 하고 또는 새롭게 feature를 정의하는 것이 **Deep Learning** 이다. 이렇게 **input data** 로 부터 특징을 자동으로 추출하는 것을 **표현학습 (Representation Learning)**이라고 한다.

일반적인 기계학습과 달리 딥러닝에서는 raw signal (visual signals or language signals) 을 컴퓨터에게 주면 컴퓨터는 어떠한 작업을 잘 수행하도록 그 signal을 표현한다. 표현학습은 좋은 feature, 표현(representation)을 자동으로 만들게 한다. **딥러닝** 알고리즘은 **input x**를 보고 여러 레벨로 **표현(h1, h2, h3)**하며 **output h4**를 만들어낸다.

### Reasons for Exploring Deep Learning

사람이 직접 만든 feature은 보통 너무 구체적이거나 완전하지 않고 디자인하고 검증하는데 시간이 오래 걸린다. 하지만 표현학습을 통해서 만들어진 learned feature 는 학습하기 쉽고 적용하기 쉽다. 딥러닝은 시각적, 언어적 정보를 **표현**하는 아주 유연하고 보편적이고 학습가능한 framework를 제공한다. 딥러닝은 unsupervised, supervised 두 방법 모두 적용 가능하다.

딥러닝은 80~90년대에 만들어진 기술이지만 별로 각광받지 못하다가 요즘들어 크게 관심이 쏠리고 있다. 그 이유는 딥러닝에 필요한 방대한 양의 training data가 있고 하드웨어의 발전으로 멀티코어 CPU/GPU 가 생겼기 때문에 연산을 빠르게할 수 있어졌기 때문이다. 또한 더 좋은 알고리즘, 모델 등이 만들어졌기 때문에 딥러닝의 성능이 크게 향상되었다.

## Deep Learning 분야

Toronto에서 딥러닝을 사용하여 **음성인식**을 하는데 큰 성과를 보이면서 딥러닝의 획기적인 발전이 시작되었다. 또한 ImageNet 으로 이미지를 인기하는데 큰 발전이 있었다. (CNN)

### 3. Why is NLP hard?

사람의 언어는 느리지만 효과적이다. 말하고자 하는 것을 모두 전달하지 않고 문맥에 의존하여 최소한의 단어를 이용해 소통하기 때문이다. 따라서 사람의 언어는 매우 모호하고 문맥에 매우 많이 의존한다. 완전한 문장도 중의적인 의미를 가질 수 있기 때문에 문맥을 알지 못하면 잘못 해석할 수 있다.

### 4. Deep NLP = Deep Learning + NLP

자연어처리와 표현학습(딥러닝)을 합쳐서 여러 문제를 해결하였다. 딥러닝을 사용하여 질의응답, 대화시스템, 기계 번역 등의 분야에서 크게 성능이 향상되었다.

### Word meaning as a neural word vector - visualization

단어의 의미를 벡터로 표현한다. 유사한 의미의 벡터가 공간상에서 비슷한 곳에 위치한다. 벡터는 300~1000 등의 차원이 될 수 있다. 이때 각각의 축은 특별한 의미가 없다.

### Representations of NLP Levels: Morphology

Morphology는 **형태론**을 의미한다. 단어를 **prefix, stem, suffix** 로 구분하는 것이다. 딥러닝에서는 각각의 morpheme이 하나의 **벡터**이다. 뉴럴 네트워크가 두개의 벡터를 하나의 벡터로 만들어가는 과정을 반복하면서 각각의 morpheme 벡터가 합쳐져 하나의 **단어 벡터**로 만든다.

### NLP Tools: Parsing for sentence structure

Parsing은 문법적으로 구문을 분석하는 것이다. NN은 문장의 문법적인 구조를 정확하게 구분하고 이것은 문장의 모호성을 없애고 해석을 도와준다.

### Representations of NLP Levels: Semantics

Semantic은 **의미적**으로 분석하는 것이다. 모든 **단어**와 모든 **구문**, 모든 **논리적 표현**은 **벡터**로 표현된다. NN은 두 개의 벡터를 하나의 벡터로 합치는 과정을 반복하면서 하나의 **문장 벡터**로 만든다.

### NLP Applications: Sentiment Analysis

Sentiment Analysis는 **감정 분석**을 의미한다. 문장에서 단어의 순서를 무시하고 문장을 단어의 집합으로 보는 것이다. 딥러닝 모델인 RNN(TreeRNNs) 을 사용하여 형태론, 문법, 논리적인 정보를 이용하여 감정을 분석할 수 있다.

### Question Answering

## Dialogue agents / Response Generation

### Machine Translation

### Neural Machine Translation

## Conclusion: Representation for all levels? Vectors

NLP에서 단어를 거의 벡터로 표현한다. 단어의 어떤 것을 표현하는지 Lecture2 에서 배우자.

## Lecture 2. Word Vector Representations: word2vec

단어를 symbol에 따라 vector로 만드는 것(one hot vector)이 아니라 문법적, 개념적 의미로 표현(word embedding)한다.

### Part 1

#### How do we represent the meaning of a word?

meaning 이라는 것은 단어, 구로 표현되는 아이디어이다. 아이디어나 물체는 상징에 의해 표현될 수 있다. 이를 “denotation” 이라 한다.

#### How do we have usable meaning in a computer?

일반적으로 WordNet이라는 것을 사용하는데 이는 단어의 상위어나 유사어 목록을 가지고 있는 데이터이다.

#### Problems with resources like WordNet

WordNet과 같은 리소스에서는 여러가지 문제점이 있다. 유사어로 분류된 단어가 모든 문맥에서 유사하지는 않다. 또한 단어 사이의 유사성을 정확히 계산하기 어렵다.

#### Representing words as discrete symbols : **Denotational**, **Localist representation** (과거)

단어를 컴퓨터에 표현하기 위해서 전통적인 NLP에서는 “**one-hot**” **vector** 를 사용하였다. 하지만 이는 사용되는 단어의 수가 늘어날 수록 벡터의 차원이 무한대로 커져야한다는 문제점과, 단어 간의 유사성을 표현하지 못한다는 문제점이 있다.

#### Representing words by their context : **Distributional**, **Distributed representation** (현재)

한 단어의 주변에 나타나는 단어들을 이용해 해당 단어를 벡터로 표현하는 방법이다. 문맥을 이용해 표현함으로써 표현하고자 하는 단어가 어떤 단어들과 함께 등장하는지 표현할 수 있고, 유사성 또한 표현할 수 있다. 이 방법을 통해 단어의 상징이 아니라 단어가 가진 **문법적, 개념적 의미**를 표현할 수 있다. 이렇게 표현된 벡터를 **dense vector** 이라고 한다. 유사한 단어 벡터는 공간에서 cluster를 형성한다. 이러한 벡터를 **word embedding** 이라고도 한다.

## Word vectors

각 단어에 대해 dense vector 을 만들어 보자. 유사한 문맥에 등장하는 단어끼리는 벡터도 비슷할 것이다.

## Part 2 What is Word2Vec?

### Basic idea of learning neural network word embeddings

center word  $w_t$  와 이 단어의 context words  $w_{-t}$  를 word vector 를 이용하여 예측하는 모델을 정의한다.

$p(context|w_t) = \dots$  center word가 주어졌을 때 context word가 나올 확률이다.

$J = 1 - p(w_{-t}|w_t)$  잘 예측했는지 확인하는 loss function 이다.

모델에서 loss를 최소화하는 방향으로 벡터 표현을 계속 업데이트한다. loss 가 줄어든다는 것은

**어떤 단어를 보고 그 단어 주변에 올 법한 단어를 잘 예측할 수 있도록 단어를 벡터로 표현하는 것을 의미한다.** 이 방법은 단어의 의미를 표현하는데 아주 유용하다.

### Word2vec: Overview

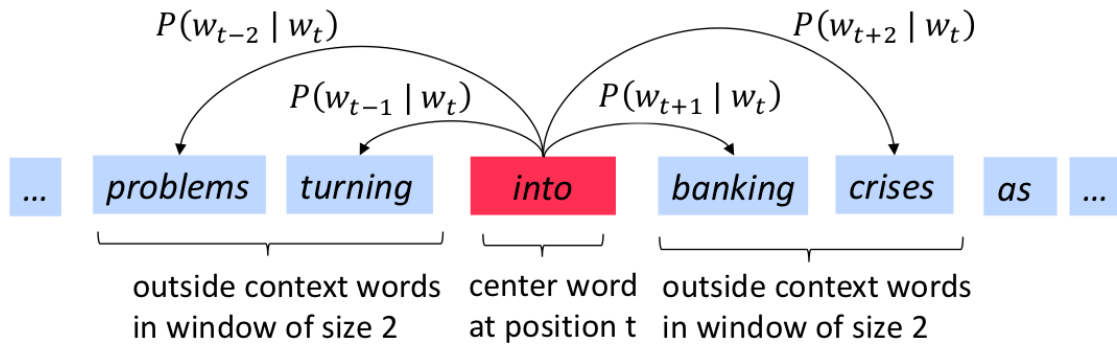
**Word2vec**은 theory of meaning을 이용하여 각 단어와 해당 단어의 문맥 단어를 예측하는 아이디어를 기반으로 하여 word vector를 학습하는 framework이다. word vector를 만드는 두가지 알고리즘이 있다.

1. Skip-gram (SG)
2. Continuous Bag of Words (CBOW)

Word2vec 의 기본적인 아이디어는 다음과 같다.

- 큰 corpus가 있다.
- 각각의 단어는 벡터로 표현된다.
- 텍스트에서 모든 position  $t$  에 대하여 진행하는데, 이때 center word를  $c$ , context(outside) word를  $o$  라고 한다.
- $c$  와  $o$  의 단어 벡터의 유사성(dot product)을 사용하여  $p(o|c)$  또는  $p(c|o)$  를 계산한다.
- 확률을 최대화 하는 방향으로 단어 벡터를 업데이트 한다. (loss를 최소화하는 방향과 같은 말)

### Skip-gram (SG)



Skip-gram 은 word vector를 학습하는 방법 중 하나이다. SG의 아이디어는 각 단계에서 하나의 단어를  $c$ 로 정하고  $c$ 를 중심으로 window size안에 있는  $o$ 를 예측하는 것이다. 따라서 이 모델에는 확률 분포가 필요하고 이 확률 분포 안에서 확률을 최대화 하는 word vector를 찾는 것이다.

위 그림을 보자.  $t$  position에 있는 center word "into" 가 정해졌을 때 window size  $m = 2$  라고 하자.  $t-2, t-1, t+1, t+2$  위치에 있는 단어는  $t$  위치에 있는 "into"를 center word로 하는 window 안에 있는 context word이다. 이때 우리는  $w_t$  가 center word로 주어졌을 때  $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$  가 나올 확률을 계산한다. corpus에 단어가 총  $T$ 개 있다고 하면  $1 \sim T$  위치의 단어가 각각 center word 인 경우에 대하여  $p(\text{context} | w_t)$  를 계산하여 모두 곱한다. 이때 이 확률을 최대화 하는 벡터 표현 방법을 구하는 것이 목표이다. 이때 벡터 표현 방법은  $\theta$  로 표시한다. 확률을 최대화하는 대신 이 확률에 negative log likelihood 를 취해 loss function으로 만든 뒤 이 loss function을 최소화 하는 방향으로 계산한다. SG에서 context word가 center word로 부터 어떤 위치에 있는지는 신경쓰지 않는다.

< loss function > : Maximize the probability of any context word given the current center word.

$J'(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$   
 for each word in T  
 for each word in range  $(-m, m)$   
 multiply  $P(w_{t+j} | w_t)$

$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$   
 maximize  $\downarrow$  minimize  
 normalization  $\Rightarrow J(\theta)$  is loss function per word

$\theta$  : vector representation  
 $T$  : # of words from long sequence of words  
 $m$  : half of window size  
 $w_t$  : word of word index  $t$

Goal : set  $\theta$  that maximize  $J'(\theta)$

Goal : set  $\theta$  that minimize  $J(\theta)$

따라서 나온 loss function은 다음과 같다.

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

이때  $\frac{1}{T}$  는  $J$ 를 각 단어에 대한 loss로 표현하기 위해 normalize한 것이고,  $\log$ 를 씌운 것은 분포를 좀 더 regular하게 만들기 위한 것이다.  $\rightarrow$  를 붙임으로써 최대화해야 했던 것을 최소화할 수 있도록 하여 계산을 좀 더 편하게 하였다.

loss function = cost function = objective function 은 모두 같은 의미이다.

$P(w_{t+j} | w_t; \theta)$  는 어떻게 계산할까?

단어  $w$  를 표현하는 벡터가 2개 있다.  $v_w$  는  $w$ 가 center word일 때  $w$ 를 표현하는 벡터이다.  $u_w$  는  $w$ 가 context word 일 때  $w$ 를 표현하는 벡터이다. 즉, 한 개의 단어는 center word로서, context word로서 두 가지의 벡터로 표현된다. center word 의 index를  $c$ 라고 표현하고 context word 의 index를  $o$ 라고 표현하자.

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product compares similarity of  $o$  and  $c$ .  
Larger dot product = larger probability

After taking exponent,  
normalize over entire vocabulary

$c$ 에 대하여 window 안에 있는 모든 context word의  $\exp(u_o^T v_c)$  를 곱한 것이 분모, 그 중 어느 한 context word  $o$ 의  $\exp(u_o^T v_c)$  가 분자이다.

Dot product

$$u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$$

Bigger if  $u$  and  $v$  are more similar!

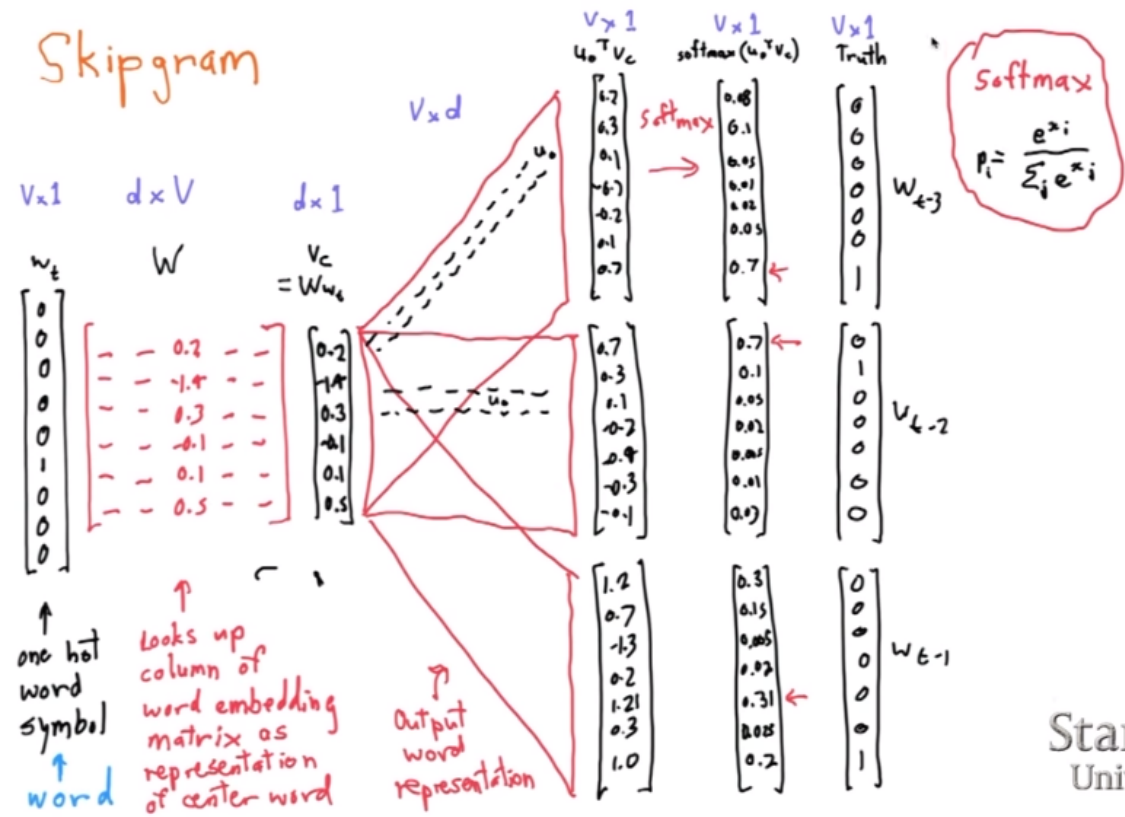
Iterate over  $w=1 \dots W$ :  $u_w^T v$  means:  
Work out how similar each word is to  $v$

$o$ 와  $c$ 의 유사도를 비교하기 위하여 dot product 가 사용된다. 값이 클 수록  $o$ 와  $c$ 가 유사하다. exp 로 값을 positive하게 만들고 차이가 큰 것의 차이를 극대화한다.

이렇게 하는 것은 **softmax function**의 사용 예시 중 하나이다. softmax function은 숫자를 확률 분포의 값으로 나타내는 전형적인 방법 중 하나이다. 임의의 값  $x_i$  를 확률 분포  $p_i$ 로 매핑시킨다.

(probability 와 likelihood의 차이)

skip-gram 이 구현되는 과정을 그림으로 그려보면 다음과 같다.



가장 왼쪽 부터 순서대로 해석해보자.

$w_t$ : center word를 one hot vector로 표현 한 것.

$W$ : 각 단어가 center word로 표현될 때의 벡터를 모아 만들어진 행렬. matrix of representation of center word.

$W_t = v_c$ :  $W * w_t$  하여  $W$ 에서  $w_t$ 의 열만 추출한 것.  $w_t$ 의 word vector

$W'$ : window 안에 있는(?) 각 단어가 context word로 표현될 때의 벡터를 모아 만들어진 행렬. matrix of representation of context word.

$u_o^T v_c$ : score. context word와 center word의 유사도를 의미하는 내적값.

$\text{softmax}(u_o^T v_c)$ : score에 softmax를 취하여 확률분포 값으로 나타낸다.

Truth: 실제 context word의 one-hot vector

$\text{softmax}(u_o^T v_c)$ 와 Truth를 비교한다. 틀리게 예측했다면 loss가 발생할 것. 이 loss를 최소화 하는 방향으로  $W$ 와  $W'$ 를 업데이트한다.

**To train the model: Compute all vector gradients!**

d-dimensional vectors: 각각의 단어는 d 차원의 벡터로 표현된다.

V-many words: V 개의 단어가 있다.

$\theta$ 는 모든 단어에 대한  $u, v$ 를 concatenate한 하나의 큰 벡터이다. 이는  $2dV$  차원에 속한다.

## Part3



## Derivations of gradient

$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{m \leq j \leq m} \log P(w_{t+j} | w_t : \theta)$

$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$

We want to find  $\theta$  that minimize  $J(\theta)$ .  
 How can we change parameters?  $\Rightarrow$  use Gradient!

$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$   
 $\hookrightarrow J(\theta)$ 를 minimize 하는 방법으로  $v_c$ 를 어떻게 바꾸면 좋을까?  $u_o$ 에 대해서도  $u_o \rightarrow b_5$

First thing to make this simpler: Subtraction

$\frac{\partial}{\partial v_c} \left\{ \log \exp(u_o^T v_c) - \log \sum_{w=1}^V \exp(u_w^T v_c) \right\}$

①  $\log \exp(u_o^T v_c) \therefore \frac{\partial}{\partial v_c} u_o^T v_c = u_o$   
not a single number: whole vector  
 $* \frac{\partial}{\partial (v_c)_k} \sum_{k=1}^d (u_o)_k (v_c)_k = (u_o)_k$

②  $\frac{\partial}{\partial v_c} \log \sum_{w=1}^V \exp(u_w^T v_c)$  Chain rule  
 $f \quad z \quad f(g(u))$

$= \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^V \exp(u_x^T v_c)$

$= \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \left[ \sum_{x=1}^V \frac{\partial}{\partial v_c} \exp(u_x^T v_c) \right]$

$= \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \left[ \sum_{x=1}^V \exp(u_x^T v_c) \cdot \frac{\partial}{\partial v_c} u_x^T v_c \right]$

$= \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \left[ \sum_{x=1}^V \exp(u_x^T v_c) \cdot u_x \right]$

$= \sum_{x=1}^V \frac{\exp(u_x^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)} \cdot u_x$

$= \sum_{x=1}^V P(x|c) \cdot u_x$

**Final form**  
 $u_o - \sum_{x=1}^V P(x|c) \cdot u_x$   
↑ observed (actual output)    ↑ expectation (target)    ↑ likelihood of  $u_x$  occurrence    ↑ average over all the possible context vectors

$$P(o|c) = \text{softmax}(u_o^T v_c) \text{이다.}$$

loss function  $J$ 에 대해  $v_c$ 로 편미분값을 구하고,  $u_o$ 로 편미분한값을 구해 Gradient Decent 방법을 사용하여  $\theta$ 를 업데이트해야 한다. 이 과정을 반복하여 최적의 파라미터를 구하면 skip gram의 학습이 끝나게 된다.

## Gradient Descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

$\alpha = \text{step size or learning rate}$

- Update equation (for single parameter):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- Algorithm:

```
while True:
    theta_grad = evaluate_gradient(J, corpus, theta)
    theta = theta - alpha * theta_grad
```

### Stochastic Gradient Descent (SGD)

J는 corpus에서 모든 window에 대해 적용하는 함수이므로 큰 corpus에서 방대한 양의 계산을 필요로 하고, 하나의 요소를 update하기 위해 긴 시간이 소요되는 문제점이 있다. SGD가 이 문제에 대한 해결책이 될 수 있다.

모든 window에 대해 J를 계산하여 update하는 것이 아니라 임의로 하나의 window를 선택하여 update에 사용하는 기법이다.

- Algorithm:

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J, window, theta)
    theta = theta - alpha * theta_grad
```

## Lecture 3. GloVe: Global Vectors for Word Representation

### Review: Main ideas of word2vec

Skip Gram의 아이디어를 다시 한번 떠올려보자. 전체 corpus에 있는 각각의 단어에 대하여 진행한다. 각각의 단어에 대하여 주변에 위치할 단어를 예측한다. 이때 주어진 확률 분포는

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

이다. 그리고 각각의 window에 대하여 SGD를 적용한다. (?)

SGD란 Stochastic Gradient Descent로 **확률그라디언트하강**이다. 하나의 파라미터를 업데이트하기 위해 전체 window에 대한 loss와 gradient를 계산하는 대신 하나의 window에 대한 loss와 gradient를 계산하여 파라미터를 업데이트하는 방법이다.

하지만 각각의 window에는  $2m+1$  개의 단어만 존재하기 때문에 그에 비해  $\nabla_{\theta} J_t(\theta)$  가 너무 크다.

$$\nabla_{\theta} J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

우리는 전체 word vector 중 corpus에 실제로 등장하는 단어에 대한 벡터만 업데이트할 것인데

$\nabla_{\theta} J_t(\theta)$  의 차원인  $2dV$  에 대해 모두 고려할 필요는 없다.

이에 대한 해결책은 전체  $U$  와  $V$  중 특정한 열만 업데이트하기 위해 sparse matrix update operation 이 필요하거나 전체 word vector에 대한 hash 가 필요하다.

### Approximations: Assignment 1

normalization factor  $J(\theta) = \frac{1}{T} \sum_{t=1}^T J_t(\theta)$  는 너무 낭비이다. 이에 대한 해결책으로 **negative sampling** 이 있다. negative sampling 의 핵심 아이디어는 train binary logistic regressions for a true pair (center word and word in its context window) versus a couple of noise pairs (the center word paired with a random word)

**ASS 1: The skip-gram model and negative sampling**

- Clear notation:  $J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{j \sim p(u_o)} [\log \sigma(-u_j^T v_c)]$ 

*all the U vectors and all the V vectors*

*cost function*
- We can take  $k$  negative samples
- Maximize probability that real outside word appears, minimize probability that random words appear around center word.
- $P(w) = U(w)^{\frac{3}{4}} / Z$  (choice)
 

the unigram distribution  $U(w)$  raised to the  $\frac{3}{4}$  power
- The power make less frequent words be sampled more often.

THE, 같은 out ↓

**negative sampling** 은 'softmax에서 너무 많은 단어들에 대해 계산을 해야하니, 여기서 몇 개만 샘플링해서 계산하면 안될까?' 라는 생각에서 시작한다. 전체 단어들 중에 일부만 뽑아서 softmax 계산을 하고 normalization 해주는 것이다. 이때 어떻게 일부만 뽑을 것인가가 중요한데, 정해진 규칙 안에서 랜덤으로 선택하는 방법이 사용된다. **고의적으로 여러 개의 오답이 될 만한 후보를 랜덤하게 선택한 후 확률값에 negative를 취하는 방법을 negative sampling**이라고 한다.

skipgram 에 negative sampling 을 적용하는 방법은 손실함수에 연산을 덜 하도록 바꾸면 되는 것이다. 위 그림의 하이라이트한 식을 보자. 자세한 설명은 [이곳](#)을 참고하자. (이해가 안됨)

추가적인 방법으로 the, a, in 과 같이 자주 등장하는 단어들을 확률적으로 제외하여 학습 속도를 향상시키기도 한다.

### Word2vec improves objective function by putting similar words nearby in space

비슷한 문맥에서 등장하는 즉, 유사한 의미를 가진 단어는 벡터 공간 상에서 cluster를 형성하며 모여있다.

### Summary of word2vec

word2vec 은 결국 center word와 context word 가 얼마나 자주 동시에 등장하는가를 파악한다. 그렇다면 왜 한 쌍의 단어에 대한 co-occurrence 빈도수를 직접 세지 않을까? word2vec 이전에는 이 방법을 사용했다.

### co-occurrence matrix

co-occurrence matrix  $X$ 가 있다고 하자. window 기반 또는 full document 기반으로 행렬을 만들 수 있다.

window 기반으로 행렬을 만든다고 하면 word2vec과 유사하게 각각의 단어에 대해 window를 정하고 syntactic (문법적), semantic (의미적) 정보를 파악한다. 이때 행렬은 그냥 동시에 등장하는 빈도수를 측정함으로써 만든다. skip gram과 그나마 유사하다.

full document 기반으로 행렬을 만든다고 하면 행렬은 어떤 일반적인 topic을 줄 것이다. "Latent Semantic Analysis"라고 불린다.

### Problems & Solution for simple co-occurrence vectors

co-occurrence vectors 의 문제점은 단어의 수가 많아질 수록 크기가 커진다는 것이다. 그래서 모델이 견고하지 않다. 이에 대한 해결 방법은 벡터의 차원을 낮추는 것이다. 가장 중요한 정보를 포함하게 하여 dense vector를 만든다. 그렇다면 차원을 어떻게 낮출 수 있을까?

### Method 1: Dimensionality Reduction on $X$

**SVD (Singular Value Decomposition)** 은 특이값 분해 이다. 특이값 분해는 고유값 분해(eigendecomposition)와 같이 행렬을 대각화하는 방법 중 하나이다. (이해가 가지 않아 자세한 설명은 생략)

## Count based vs direct prediction

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebert & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

- Scale with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

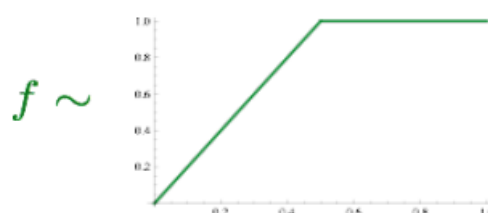
왼쪽이 count-based이고 오른쪽이 window-based 에 대한 설명이다. 각각의 장, 단점 설명이다.

## Combining the best of both worlds: GloVe

**GloVe(Global Vectors)** 모델은 Global co-occurrence statistics 정보를 포함한 direct prediction word embedding 이라고 생각할 수 있다. 이는 위에서 이야기한 count based 모델과 window based 모델을 합친 것이라고 할 수 있다.

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

$\theta$  는 모든 U 벡터와 V 벡터를 의미한다. 함께 등장할 수 있는 모든 단어 쌍에 대하여 진행한다.  $P_{ij}$  가 한 쌍을 의미한다. 우리는 두 단어의 inner product 값과 co-occurrence 빈도수에 log 취한 값 사이의 거리를 줄이고자 한다. 이때 co-occurrence 빈도수는 co-occurrence matrix 를 사용하여 구한다. f 항은 너무 많이 등장하는 단어에 대하여 가중치를 제한하는 역할을 한다. f는 0~1 사이의 값으로 일정 빈도수 이상이면 1로 고정된다.



이를 통해 'the' 와 같이 자주 등장하는 단어에 대한 가중치값을 조절한다. 이 방법은 작은 corpus에서 잘 작동한다.

## What to do with the two sets of vectors?

inside vector ( $v$ ) 와 outside vector ( $u$ ) 가 있었는데 어차피 동시에 등장하는 빈도수를 파악할 것이기 때문에 이 둘을 합쳐서 하나의 벡터로 생각해보자. 합치는 방법으로는 concat 등 다양한 방법이 있지만 여기서는  $U$  와  $V$  를 더해  $X$  벡터로 만든다. 이제 하나의 단어에 하나의 벡터 표현이 존재하는 것이다.

$$X_{\text{final}} = U + V$$

### What do we do about polysemy? (TA 강의)

단어의 의미에 따라 벡터가 정해진다고 하였는데 다의어에 대한 표현은 어떻게 해야 할까?

1. Polysemous vectors are superpositioned.

tie라는 단어가 3개의 의미를 지닌다고 하면 tie1, tie2, tie3 으로 표현하여 벡터를 구한 후 3개의 벡터를 combine 하여 tie에 대한 벡터를 만든다.

2. Senses can be recovered by sparse coding
3. Senses recovered are non-native English level

### How to evaluate word vectors?

word embedding 을 학습시킨 이후에 평가하는 방법은 intrinsic(내적) vs extrinsic(외적) 평가로 나눌 수 있다.

**Intrinsic(내적)** : word embedding 만을 이용해 내부 단어간의 유사도를 통해 평가함. 사람이 직접 평가 데이터를 만들고 두 단어의 유사도를 직접 점수 매기는 방법이 있다. word2vec 논문에서 소개된 데이터 셋을 기준으로 코사인 유사도를 평가하여 의미적, 문법적으로 평가하는 방법이 있다.

다음 그림은 cosine distance 를 이용해 의미적, 문법적 분석을 하는 방법이다.

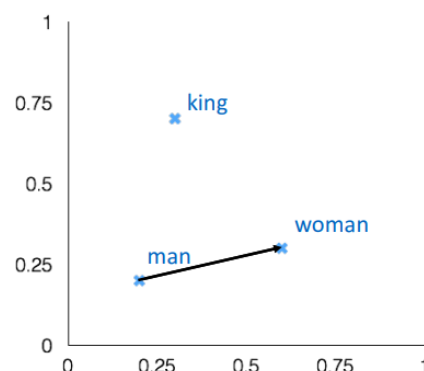
### Intrinsic word vector evaluation

- Word Vector Analogies

a:b :: c:?  
man:woman :: king:?

$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



**Extrinsic(외적)** : 실제 문제에 직접 적용하여 성능을 평가한다. 각종 NLP 시스템에 embedding을 직접 적용하여 시스템의 성능을 측정한다. 개체명 인식기(NER, Named Entity Recognition)를 평가한다. 시간이 오래 걸리며 어떤 부분이 improvement를 가져왔는지 파악하기 어렵다.

\* 논문, 보고서에 도표 및 그래프를 삽입하는 경우 : 특정 값으로 수렴할 때 까지 보여주는 것이 좋다. 예를 들어 증가하다가 수렴하는 함수일 경우, 증가하는 부분만 첨부한다면 그 이후에는 계속 증가만 할 것이라고 생각할 수 있기 때문이다.

## Lecture 4. Word Window Classification and Neural Networks

### Classification setup and notation

word embedding은 주변 context를 기반으로 하여 한 단어의 여러 의미를 내포하고 있다.(다의어) 이 경우 단어 하나만 가지고 해당 단어가 어떤 의미로 쓰였는지 알 수 없다. 중심 단어와 context 단어를 함께 분류 문제에 활용하는 방법을 **word window classification** 이라고 부른다.

training dataset : 훈련 데이터로 input  $x$ 와 output  $y$  를 포함한다.  $x$  는  $d$  차원 인풋으로 단어의 벡터, 인덱스가 될 수 있고 context window, sentence, document 가 될 수 있다.  $y$  는 label로 예측하고자 하는 값이다. 클래스 등이 예시이다.

### Classification intuition

일반적인 기계학습 관점에서 보자면  $x$ 는 고정되어 있고 학습과정에서 logistic regression 의 가중치를 변화시키며 분류기를 학습시킨다.

### Details of the softmax classifier

각각의  $x$ 에 대하여 다음을 예측하는 것이 logistic regression의 목적이다.

$$p(y|x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^C \exp(W_c \cdot x)}$$

위 식은 softmax 식이고 이 확률을 이용하여 분류하는 모델을 softmax classifier라고 한다.

이 식을 두 단계로 구분할 수 있다.



1. Take the  $y$ 'th row of  $W$  and multiply that row with  $x$ :

$$W_{y \cdot} x = \sum_{i=1}^d W_{yi} x_i = f_y$$

Compute all  $f_c$  for  $c=1, \dots, C$

2. Apply softmax function to get normalized probability:

$$p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^C \exp(f_c)} = \text{softmax}(f)_y$$

1번에서  $W_{y \cdot}$  은  $W$ 에서  $y$  번째 행을 의미한다.  $y$  번째 행과 벡터인  $x$ 를 한 값이  $f_y$  이다.

이렇게 구한  $f_y$  에 대해 2번에서 softmax를 취한다. score를 계산하여 나온 scalar 값을 확률 분포에 대응하는 방법이다.

### Training with softmax and cross-entropy error

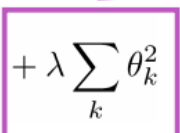
학습하는 과정에서  $p(y|x)$  를 최대화하고자 하는 것이 목적인데,  $p(y|x)$  에서 negative log likelihood를 취하여 만든 objective function을 최소화하는 것으로 바꾸어 말할 수 있다. 이렇게 negative log likelihood를 취하는 것을 **cross entropy error** 이라고 한다.

$$-\log p(y|x) = -\log \left( \frac{\exp(f_y)}{\sum_{c=1}^C \exp(f_c)} \right)$$

### Classification: Regularization

실제로 손실 함수에는 regularization을 사용한다. 이는 일반화함으로써 overfitting을 방지하는 역할을 한다.

- Really full loss function in practice includes **regularization** over all parameters  $\theta$ :

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N -\log \left( \frac{e^{f_{y_i}}}{\sum_{c=1}^C e^{f_c}} \right) + \lambda \sum_k \theta_k^2$$


### Classification difference with word vectors

일반적인 기계학습에서와 달리 deep learning에서는  $W$ 와  $x$ 를 모두 훈련시킨다. 하지만 이렇게 word vector 인  $x$  를 update 한다면 training 에 사용된 단어만 제대로 update되고 사용되지 않은 단어는 update되지 않아 실제로 test 할 때 제대로 분류하기 어려워진다.

따라서 작은 training data 에서는 word vector를 train 하지 않고 충분한 양의 training data 가 있을 때 word vector를 train 해야 한다.

### Window classification



하나의 단어만 분류하는 일은 극히 드물다. 문맥 속에 있는 단어를 분류하는 경우가 많은데 이때 하나의 단어가 여러 의미를 가질 수 있기 때문에 이 모호성을 줄일 수 있는 방법이 필요하다.

window classification 은 하나의 단어를 context window 안에서 분류하는 아이디어를 기반으로 한다. 예를 들어 **Named Entity Recognition** 을 살펴보자. 단어를 보고 Person, Location, Organization, None 클래스로 구분하는 분류기이다.

softmax classifier를 학습시키는데 이때 분류하고자 하는 중심 단어 주위의 단어를 모두 concat 한 것을 보고 중심 단어에 label을 할당한다.

- Example: Classify “Paris” in the context of this sentence with window length 2:

$$\begin{array}{ccccccc}
 \dots & \text{museums} & \text{in} & \text{Paris} & \text{are} & \text{amazing} & \dots
 \\
 \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet
 \\
 X_{\text{window}} = [ & x_{\text{museums}} & x_{\text{in}} & x_{\text{Paris}} & x_{\text{are}} & x_{\text{amazing}} & ]^T
 \end{array}$$

- Resulting vector  $x_{\text{window}} = \boxed{x \in \mathbb{R}^{5d}}$  , a column vector!

이  $x$  가 window 안에 있는 모든 단어 벡터를 concat 하여 만든 벡터이고 이  $x$ 를 인풋으로 사용하고 위 설명에서와 같이 softmax를 취해  $p(y|x)$ 를 계산하고 cross entropy error 를 구한다.

이제 word vector를 update해야 하는데 어떤 방법을 사용하면 좋을까?  
 답은 이전과 같이 미분을 사용하는 것이다.

## UPDATING...

### A note on matrix implementations

softmax classifier에는 두 개의 큰 연산이 있다. 행렬 곱셈과 지수 연산이다. 큰 행렬의 곱셈을 하는 것이 for 반복문을 사용하는 것 보다 훨씬 효과적이다. 다음은 반복문을 사용했을 때와 여러 개의 word vector를 하나의 큰 행렬로 합쳐서 계산할 때 걸리는 시간을 비교한 코드이다.

```

from numpy import random
N = 500 # number of window to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C, d) # weight

wordvectors_list = [random.rand(d, 1) for i in range(N)]
wordvectors_one_matrix = random.rand(d, N) # d for number of row, N for number of window

'''to check running time'''
%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)

1000 loops, best of 3: 425 µs per loop
The slowest run took 48.92 times longer than the fastest. This could mean that an intermediate re
sult is being cached.
10000 loops, best of 3: 53.3 µs per loop

```

행렬 곱셈으로 연산할 때가 훨씬 빠르다는 것을 알 수 있다. 따라서 구현할 때 여러개의 벡터를 반복문을 사용하여 곱하지 말고 하나의 행렬로 만들어 계산하는 방법을 선택하자.

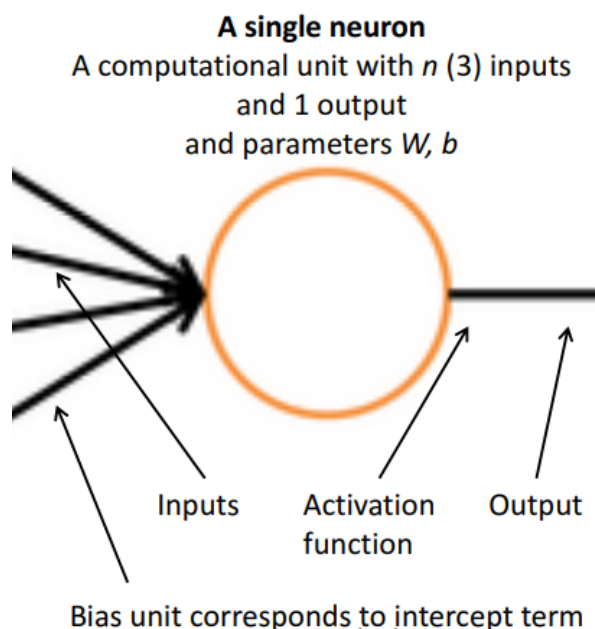
### Softmax (= logistic regression) alone not very powerful

softmax 는 linear decision boundary 만 제공하기 때문에 data를 분류하는데 효과적이지 않다. 데이터의 양이 적다면 괜찮은 regularizer로 작동하겠지만 데이터의 양이 많다면 매우 제한적이다.

### Neural Nets for the Win!

linear decision boundary 를 해결할 수 있는 것이 nonlinear decision boundary 이다. Neural Net 은 nonlinear decision boundary를 갖고 있기 때문에 softmax classifier가 해결하지 못하는 것을 해결할 수 있다.

### Demystifying neural networks



하나의 뉴런은 위 그림과 같이 생겼다. input, bias가 입력으로 들어가고 activation function을 거쳐 output이 나온다. 이때 이 뉴런의 parameter는 입력과 곱해지는 가중치인  $W$ 와 입력과 더해지는  $b$ 이다.

### A neuron is essentially a binary logistic regression unit

뉴런은 2개의 클래스 (Y/N 등)를 구분할 수 있는 logistic regression unit이다. 즉, *뉴런 한 개는 linear decision boundary를 가진다고 할 수 있음.*

다음은 single neuron 에 대한 정의이다.

$f$  = nonlinear activation function (e.g. sigmoid) (element wise function)

$w$  = weights

$b$  = bias (class prior를 줄 수 있는 것으로 weight와 구분한 항으로 표기)

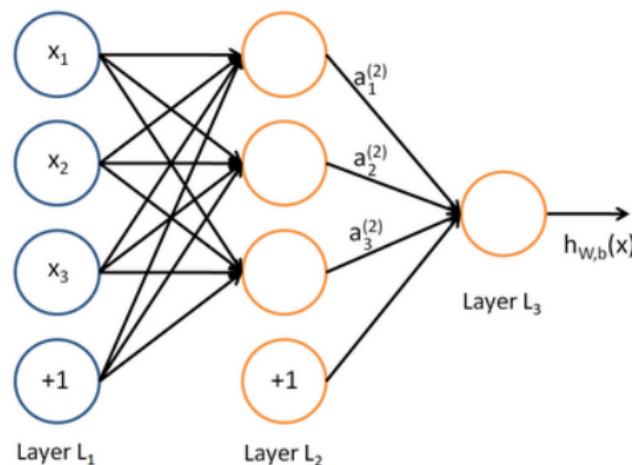
$h$  = hidden

$x$  = inputs

$$h_{w,b}(x) = f(w^T x + b)$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

A neural network = running several logistic regressions at the same time



NN 은 위와 같이 여러 개의 뉴런으로 이루어진 구조이다. 즉, 여러 개의 logistic regression 을 동시에 작동시키는 것이다. 뉴런 하나의 output이 다른 뉴런의 input이 된다. 하지만 각각의 logistic regression이 어떤 것을 예측하려고 하는가에 대해 이해할 필요는 없다. 모델은 스스로 최종 output에 error를 최소화하는 방향으로 input의 모양을 바꾼다.

이러한 hidden layer를 쭉 쌓으면서 여러 층이 있는 NN을 만들 수 있다.

### Matrix notation for a layer

We have

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

etc.

In matrix notation

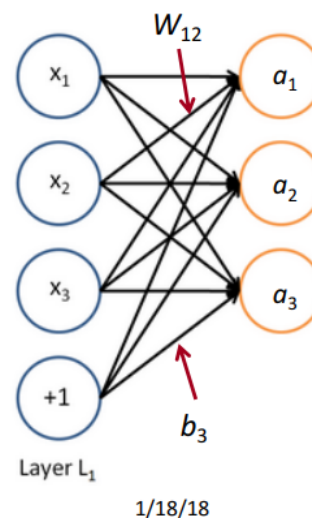
$$z = Wx + b$$

$$a = f(z)$$

where  $f$  is applied element-wise:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

43



### Binary classification with unnormalized scores

softmax classifier에서 단어 벡터인  $x$ 에 softmax를 취했던 것과 다르게 NN에서는 단어 벡터와 output 사이에 hidden layer가 있다.

museums in Paris are amazing 이라는 예시로 돌아가보자. NER에서 center word가 location인지 구분하고자 한다고 가정하자.

word2vec와 유사하게 corpus 내의 모든 position에 대해 적용하지만 supervised되어 있고 몇몇의 위치의 단어만 높은 score를 가질 수 있도록 할 것이다. 실제 NER location인 position을 "true" position 이라고 부르도록 하겠다.

### Binary classification for NER

Not all museums in Paris are amazing.

위 문장을 예시로 하여 location 을 구분하는 과정을 살펴보자. NER location 인 Paris 의 position 만 true position이 되고 그 이외의 단어들은 모두 "corrupt" 되었다고 표현한다.

### A Single Layer Neural Network

기계학습 시간에 배운 퍼셉트론과 같은 것이라고 볼 수 있겠다.

single layer는 linear layer와 nonlinearity를 합친 것이다.

$$z = Wx + b \quad \# \text{ linear layer}$$

$$a = f(z) \quad \# \text{ nonlinearity}$$

neural activation  $a$  는 output을 계산하는데 사용될 수 있다.

- For instance, a probability via softmax:

$$p(y|x) = \text{softmax}(Wa)$$

- Or an unnormalized score (even simpler):

$$\text{score}(x) = U^T a \in \mathbb{R}$$

위 예시로 classify를 하기 전에 feed-forward 가 돌아가는 과정에 대해 간단히 살펴보자.

### Summary: Feed-forward Computation

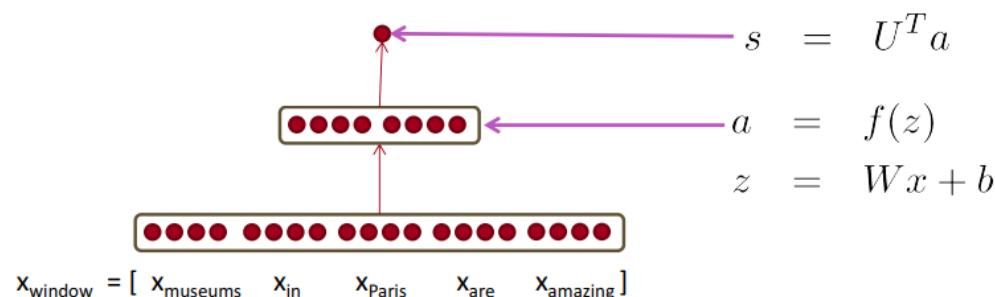
feed forward computation은 말 그대로 모델을 순서대로 동작하는 것(backpropagation과 반대 방향)을 의미한다. 모델을 테스트할 때 사용되거나 훈련과정 중에 미분을 하기 전 순서대로 동작시키는 과정에 사용된다.

3-layer neural net에서 window의 score를 계산해보자.

- $s = \text{score}(\text{"museums in Paris are amazing"})$

$$s = U^T f(Wx + b)$$

$$x \in \mathbb{R}^{20 \times 1}, W \in \mathbb{R}^{8 \times 20}, U \in \mathbb{R}^{8 \times 1}$$



이때 각 층은 input word vector 사이에 non-linear interaction 을 학습한다.

### The max-margin loss

이제 모델에 대한 loss function을 정해야 한다. 이 모델의 training idea는 true window의 score를 높이고 corrupt window의 score를 낮추는 것이다.

이때 true window란 true position 이 center word인 경우이다.

$$s = \text{score}(\text{museums in Paris are amazing})$$

$$s_c = \text{score}(\text{Not all museums in Paris})$$

single window에 대한 objective function 은 다음과 같다.

$$J = \max(0, 1 - s + s_c)$$

NER location 을 center에 포함하는 각각의 window는 corrupted window보다 score가 1 높아야 한다.

전체에 대한 objective function은 true window에 대해 corrupt window 여러 개를 선택하여 training 하고 모두 더한다. 이는 word2vec에서 negative sampling 과 유사하다.

이후 Derivation에 대한 수식이 나와있다..! 완전히 이해가 안되니 강의자료를 참고하도록..!