CS229N 2023 | WEEK 1-1 박병민

LECTURE 1: INTRODUCTION AND WORD VECTORS

How do we represent the meaning of a word?

Definition: **meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

signifier (symbol) ⟺ signified (idea or thing)

= denotational semantics

LECTURE 1: INTRODUCTION AND WORD VECTORS

Problems with resources like WordNet

- Great as a resource but missing nuance
  - e.g., "proficient" is listed as a synonym for "good"
    This is only correct in some contexts
- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can't compute accurate word similarity →

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

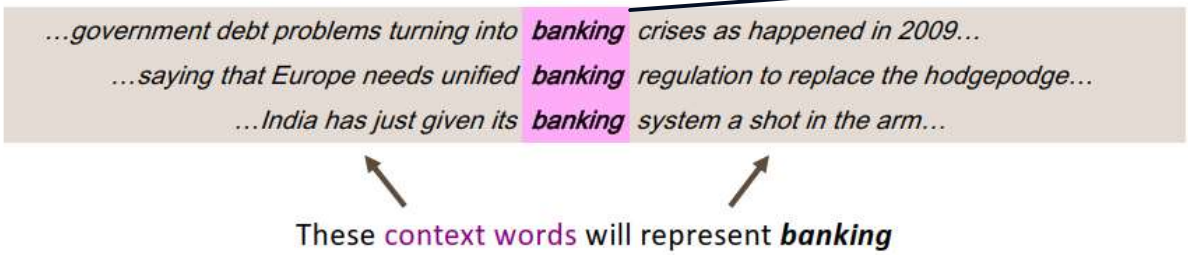
These two vectors are orthogonal
There is no natural notion of **similarity** for one-hot vectors!

denotational semantics는 단어의 의미를 잘 담지 못함!

LECTURE 1: INTRODUCTION AND WORD VECTORS

Distributional semantics: 주변에서 자주 나타나는 문맥에 의해 단어의 의미를 부여!

- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of *w* to build up a representation of *w*



banking = $\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$

word embedding 또는 word representation 이라고도 하며 확률 분포 형태임

LECTURE 1: INTRODUCTION AND WORD VECTORS

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

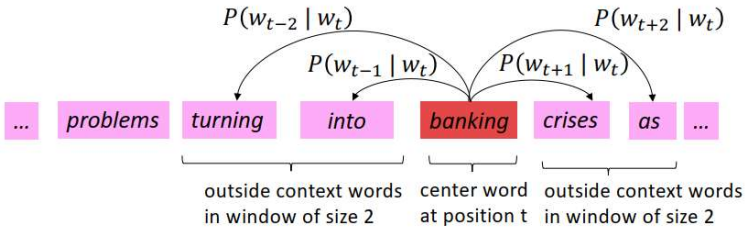
- We have a large corpus ("body") of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Two model variants:

1. Skip-grams (SG)
   Predict context ("outside") words (position independent) given center word
2. Continuous Bag of Words (CBOW)
   Predict center word from (bag of) context words

This lecture so far: **Skip-gram model**

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Keypoint

1. 중심 단어와 주변 단어 벡터의 유사도를 계산할 수 있는가?

2. 유사도를 기반으로 확률 분포로 표현할 수 있나?

LECTURE 1: INTRODUCTION AND WORD VECTORS

Word2vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word w_j. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

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

Minimizing objective function ⟺ Maximizing predictive accuracy

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

② Exponentiation makes anything positive

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

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

③ Normalize over entire vocabulary to give probability distribution

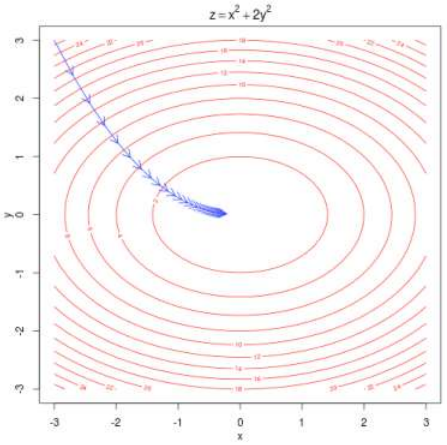
1. 중심 단어와 주변 단어의 유사도를 벡터 내적으로 계산.
2. 내적 값을 지수 함수로 변환하여 양수로 만듦.
3. 전체 어휘에 대해 정규화하여 주변 단어의 등장 확률을 계산.

LECTURE 1: INTRODUCTION AND WORD VECTORS

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with *d*-dimensional vectors and *V*-many words, we have:
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



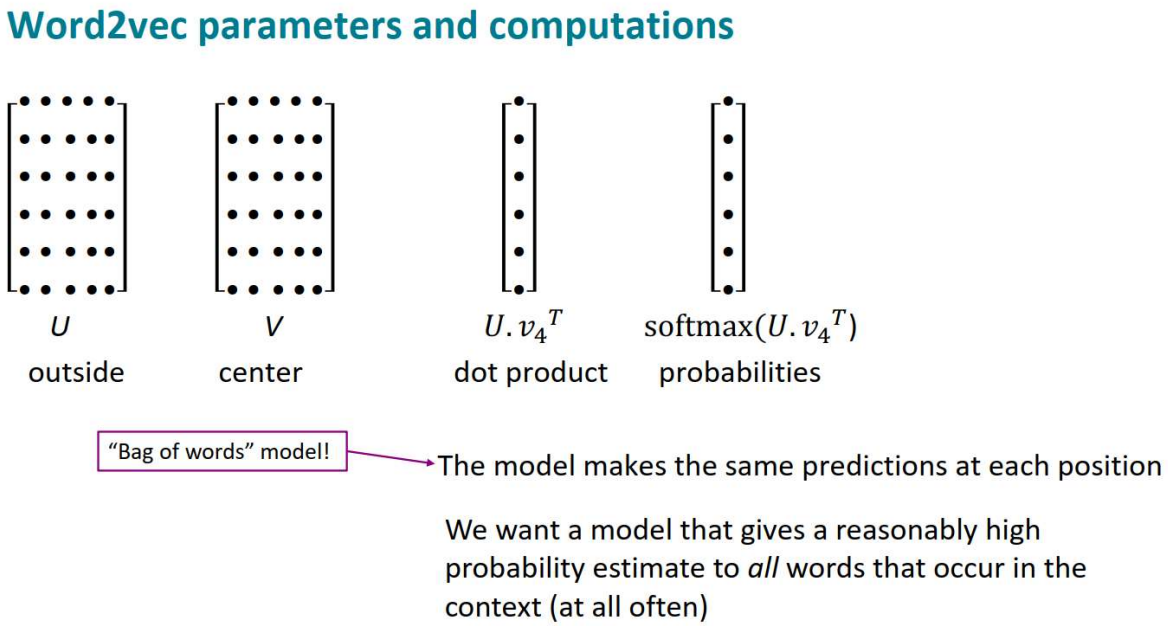
파라미터의 크기가 왜 $2dV$인가?

- **전체 어휘 V**
- **단어 벡터 크기 d**
- **중심 단어일 경우 & 주변 단어일 경우를 고려 $\times 2$**

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

U 벡터는 outside words (문맥 단어)에 해당하는 벡터

V 벡터는 center words (중심 단어)에 해당하는 벡터



Bag of Words 모델:

특정 위치에서 동일한 예측을 반복적으로 수행.

문맥에서 자주 나타나는 모든 단어에 대해 높은 확률을 할당하는 모델을 만들도록 함.

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

Stochastic Gradient Descent

- **Problem**: $J(\theta)$ is a function of **all** windows in the corpus (often, billions!)
  - So $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!

- **Very** bad idea for pretty much all neural nets!
- **Solution: Stochastic gradient descent (SGD)**
  - Repeatedly sample windows, and update after each one, or each small batch

- Iteratively take gradients at each such window for SGD
- But in each window, we only have at most $2m + 1$ words, so $\nabla_\theta J_t(\theta)$ is very sparse!

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

- 각 window에서 모델이 처리하는 단어의 수는 최대 $2m + 1$로 매우 적음.
- $\nabla_\theta J_t(\theta)$ 는 대부분의 요소가 0인 매우 희소(sparse)한 형태를 가지는 문제 발생
- 해결책으로, 임베딩 행렬 U와 V의 특정 행만 업데이트하는 희소 행렬(sparse matrix) 업데이트 연산으로 해결

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

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

The normalization term is computationally expensive

- Hence, in standard word2vec and HW2 you implement the skip-gram model with **negative sampling**

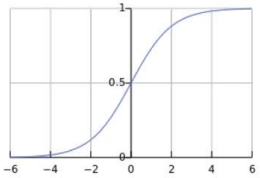
$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

Main idea: train binary logistic regressions for

a true pair (center word and a word in its context window) versus

several noise pairs (the center word paired with a random word)

- The logistic/sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$ (we'll become good friends soon)
- We maximize the probability of two words co-occurring in first log and minimize probability of noise words



주어진 중심 단어에 대해 실제 문맥 단어가 나타날 확률을 최대화하고, 무작위 단어가 나타날 확률을 최소화함.

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

$$J_{neg-sample}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)$$

log 함수와 시그모이드 함수(σ)가 결합된 형태는 로지스틱 손실(Logistic Loss)

- 첫 번째 항은 실제 문맥 단어 u_o와 중심 단어 v_c 의 내적을 기반으로 한 로지스틱 손실을 최소화함.
- 두 번째 항은 무작위로 선택된 k개의 '부정적' 단어들과 중심 단어 간의 내적에 대한 로지스틱 손실을 최소화함.

- 부정적 샘플은 주어진 단어 확률에 따라 샘플링되며, 이때 단어의 빈도수에 따라 가중치가 조정됨.
  일반적으로, **단어의 빈도 $U(w)$의 3/4 제곱으로 샘플링하여, 드문 단어가 더 자주 선택될 가능성을 높임.**

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

- **Co-occurrence matrix는 전체 문서나 특정 윈도우 내에서 단어의 동시 발생 빈도를 캡처**하지만,
  차원의 저주 문제로 인해 문서와 어휘의 크기가 커질수록 행렬의 크기가 기하급수적으로 증가하여 계산이 복잡해짐.

- **Word2Vec은 co-occurrence matrix 대신 negative sampling을 사용**하여 차원이 커지는 문제를 해결하고,
  무작위 부정 단어와의 관계를 최소화함.

Simple count co-occurrence vectors

- 어휘가 증가할수록 벡터 크기가 커지며, 매우 높은 차원으로 인해 많은 저장 공간이 필요
- 희소성 문제를 일으켜 모델의 안정성이 떨어짐.

중요한 정보를 고정된 작은 차원(25~1000 차원)의 밀집 벡터에 저장하여 해결

- SVD(Singular Value Decomposition)와 같은 차원 축소 기법 및 "Hacks to X" 기법이 제시됨.

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning
  - I like NLP
  - I enjoy flying

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = fashion
$P(x\|\text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x\|\text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

기체 → 고체로의 의미요소 분석?
⇒ 동시 발생 확률!

임베딩된 두 단어벡터의 내적이 코퍼스 전체에서의
co-occurrence probability의 로그값이 되도록 정의

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

단어 벡터 공간에서 선형 의미 구성 요소로 co-occurrence probability의 비율로 표현할 수 있는가?

Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

with vector differences $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

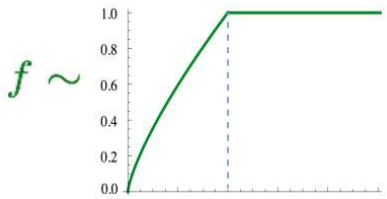
- Log-bilinear 모델: **두 단어 사이의 공기 확률을 단어 벡터 공간에서 선형적으로 표현**하기 위해 두 단어 벡터 w_i와 w_j의 내적이 $P(i \mid j)$ 의 로그 값으로 정의
- 벡터 차이를 통한 비율: **특정 단어(x)가 두 다른 단어(a & b)와의 관계에서 어떤 단어에 더 가깝다는 것을 표현하기 위해**, w_x와 $w_a - w_b$의 내적를 $\log \frac{P(x \mid a)}{P(x \mid b)}$ 으로 표현

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

$$w_i \cdot w_j = \log P(i|j)$$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors



Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

- Most words have lots of meanings!
  - Especially common words
  - Especially words that have existed for a long time

- 단어 임베딩에서는 **단어의 다양한 의미가 선형 중첩으로 표현**되며, 단어 벡터는 여러 의미 벡터의 가중합으로 나타남. **각 의미의 빈도에 따라 가중치가 부여됨.**
- **가중치는 해당 의미의 빈도에 비례**하며, 자주 사용되는 의미일수록 단어 벡터에 더 큰 영향을 미침.
- **희소 코딩 접근법을 통해 단어의 서로 다른 의미를 독립적으로 분리할 수 있음.**
- 예를 들어, "tie"라는 단어는 서로 다른 맥락에서 다양한 의미로 사용될 수 있음.

Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora, …, Ma, …, TACL 2018)

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like word2vec
- $v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$
- Where $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$, etc., for frequency f
- Surprising result:
  - Because of ideas from *sparse coding* you can actually separate out the senses (providing they are relatively common)!

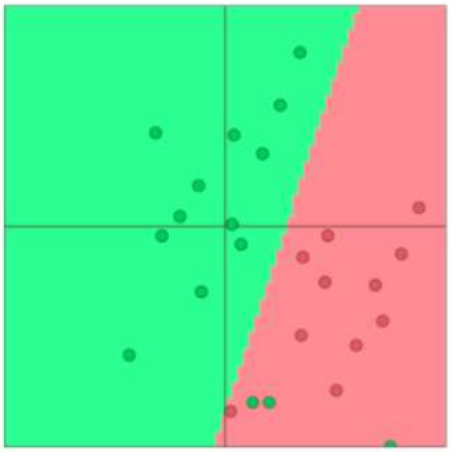
tie				
trousers	season	scoreline	wires	operatic
blouse	teams	goalless	cables	soprano
waistcoat	winning	equaliser	wiring	mezzo
skirt	league	clinching	electrical	contralto
sleeved	finished	scoreless	wire	baritone
pants	championship	replay	cable	coloratura

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

- Generally, we have a training dataset consisting of samples

$$\{x_i, y_i\}_{i=1}^{N}$$

- x_i are inputs, e.g., words (indices or vectors!), sentences, documents, etc.
  - Dimension d
- y_i are labels (one of C classes) we try to predict, for example:
  - classes: sentiment (+/–), named entities, buy/sell decision
  - other words
  - later: multi-word sequences



Word vector를 classify하는 방법?

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

Training with "cross entropy loss"

- Concept of "cross entropy" is from information theory
- Let the true probability distribution be p
- Let our computed model probability be q

$$H(p,q) = -\sum_{c=1}^{C} p(c)\log q(c)$$

분류 문제에서 예측 확률 분포와 실제 라벨(참값) 분포 간의 차이를 측정하는 방법!

문맥 단어	참 라벨 (p)	모델 예측 확률 (q)	크로스 엔트로피 손실
fruit	1	0.8	$-\log(0.8)$
banana	0	0.15	0
technology	0	0.05	0

- Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else:
  p = [0,...,0,1,0,...0] then:
- **Because of one-hot p, the only term left is the negative log probability of the true class:** $-\log p(y_i|x_i)$

계산 과정:

1. **fruit**: $p(fruit) = 1$, $q(fruit) = 0.8$

$$-1 \times \log(0.8) \approx 0.22$$

2. **banana**: $p(banana) = 0$, $q(banana) = 0.15$

$$-0 \times \log(0.15) = 0$$

3. **technology**: $p(technology) = 0$, $q(technology) = 0.05$

$$-0 \times \log(0.05) = 0$$

최종 크로스 엔트로피 손실:

$$H(p,q) = 0.22 + 0 + 0 = 0.22$$

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

How to calculate $\mathbf{q}(c)$?

Softmax classifier

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

Again, we can tease apart the prediction function into three steps:

1. For each row *y* row of *W*, calculate dot product with *x*: $W_{y\cdot}x = \sum_{i=1}^{d} W_{yi}x_i = f_y$
2. Apply softmax function to get normalized probability:

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

3. Choose the *y* with maximum probability

- For each training example (*x*,*y*), our objective is to maximize the probability of the correct class *y* or we can minimize the negative log probability of that class:

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

클래스	내적 점수	소프트맥스 확률 계산	소프트맥스 확률 $p(y \mid x)$
과일 (0)	2.0	$\frac{\exp(2.0)}{\exp(2.0)+\exp(1.0)+\exp(0.5)}$	0.64
기술 (1)	1.0	$\frac{\exp(1.0)}{\exp(2.0)+\exp(1.0)+\exp(0.5)}$	0.24
기타 (2)	0.5	$\frac{\exp(0.5)}{\exp(2.0)+\exp(1.0)+\exp(0.5)}$	0.12

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

Classification over a full dataset

- Cross entropy loss function over full dataset $\{x_i, y_i\}_{i=1}^N$

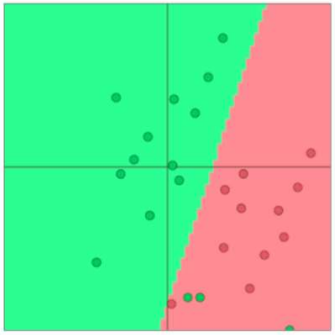
$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

- Instead of

$$f_y = f_y(x) = W_{y\cdot}x = \sum_{j=1}^{d} W_{yj}x_j$$

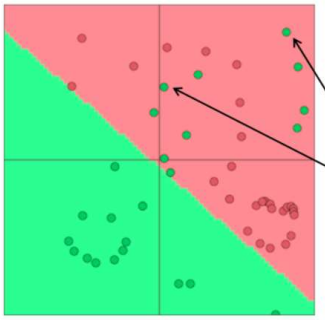
We will write f in matrix notation:

$$f = Wx$$



Neural Network Classifiers

- Softmax (≈ logistic regression) alone is not very powerful
- Softmax classifier only gives linear decision boundaries



This can be quite limiting

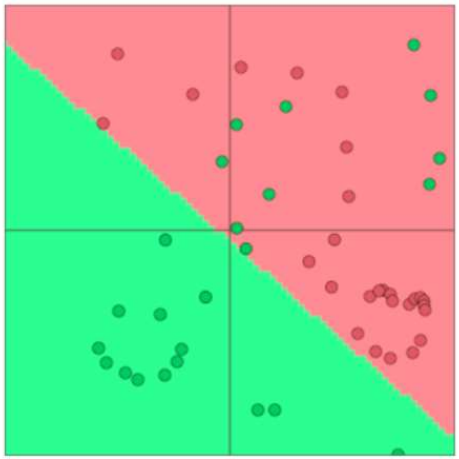
→ Unhelpful when a problem is complex

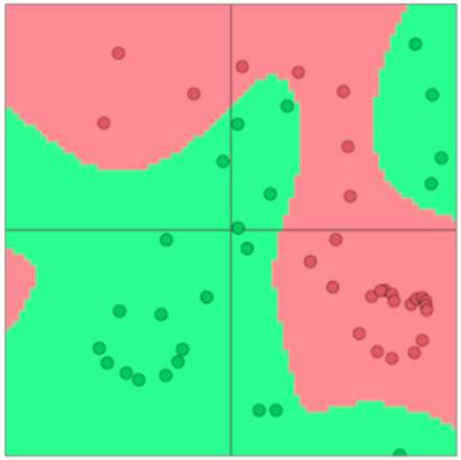
Wouldn't it be cool to get these correct?

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

Neural Nets for the Win!

- Neural networks can learn much more complex functions with nonlinear decision boundaries!
  - Non-linear in the original space





LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

NLP 딥러닝 모델에서의 학습 과정

- **가중치 W와 단어 벡터 x를 모두 학습.**
- 단어 벡터는 one-hot vector를 재표현 → embedding layer를 통해 벡터 공간에서 쉽게 분류할 수 있도록 학습.
- 문제는 **모델의 매개변수 수는 매우 큼**, 특히 V_d 항이 큼.

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix} \in \mathbb{R}^{Cd+\boxed{Vd}}$$

Very large number of parameters!

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

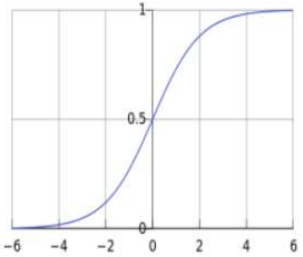
A neuron can be modeled as a binary logistic regression unit

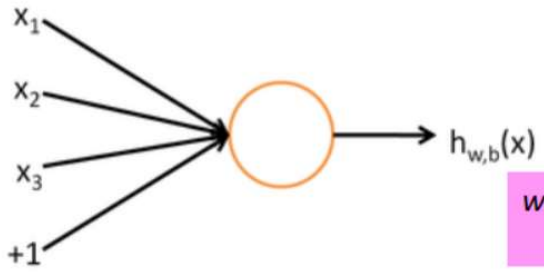
f = nonlinear activation function (e.g. sigmoid), *w* = weights, *b* = bias, *h* = hidden, *x* = inputs

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

b: We can have an "always on" bias feature, which gives a class prior, or separate it out, as a bias term

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



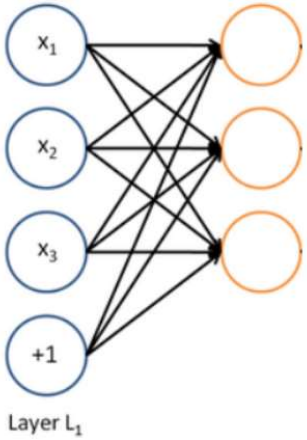


w, *b* are the parameters of this neuron i.e., this logistic regression model

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

Difference #2: A neural network
= running several logistic regressions at the same time

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …

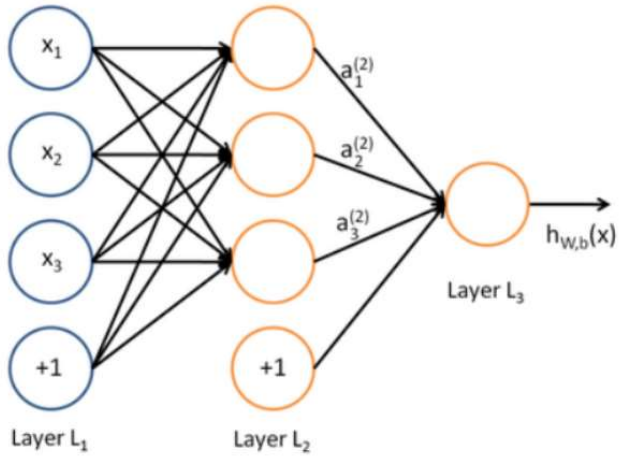


But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

이 과정에서 각 로지스틱 회귀가 무엇을 예측하는지를
사전에 결정할 필요가 없으며,
학습을 통해 자동으로 최적의 예측을 수행

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS

… which we can feed into another logistic regression function



It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.

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.

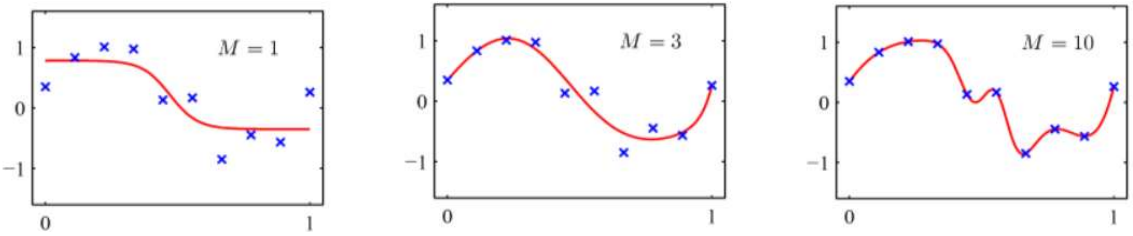
In matrix notation

$$z = Wx + b$$
$$a = f(z)$$

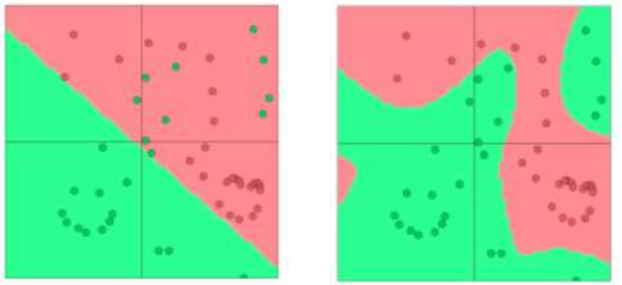
Activation *f* is applied element-wise:

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

LECTURE 2: WORD VECTORS, WORD SENSES, AND NEURAL CLASSIFIERS



- With more layers, they can approximate more complex functions!



- **비선형성이 없으면 딥러닝 신경망은 선형 변환 이상의 작업을 수행할 수 없음.**
  여러 층이 있어도 단순히 하나의 선형 변환으로 축약.
- **비선형성을 도입하면 신경망이 더 복잡한 함수들을 근사할 수 있게 되어, 더 복잡한 패턴을 학습할 수 있음.**