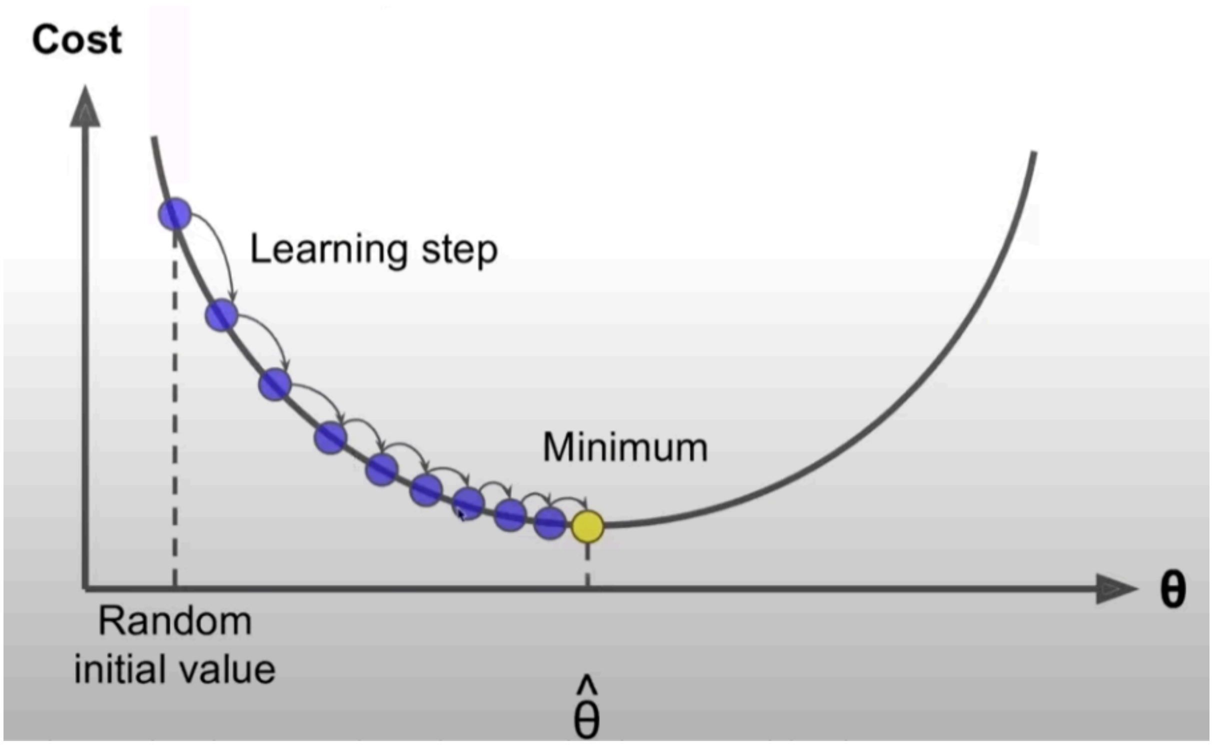
Lecture 2 - Neural Classifiers

Optimization : Gradient Descent

- Algorithm to minimize $J(\theta)$ by changing θ
- Idea : from current value of θ, calculate gradient of $J(\theta)$, then take small step in the direction of negative gradient. repeat



i. step size is too small → long time to minimize the function → a lot of wasted computation
ii. step size is too big → go to worse places

- Update equation (in matrix notation):

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

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

- Update equation (for a single parameter)

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

Stochastic Gradient Descent

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

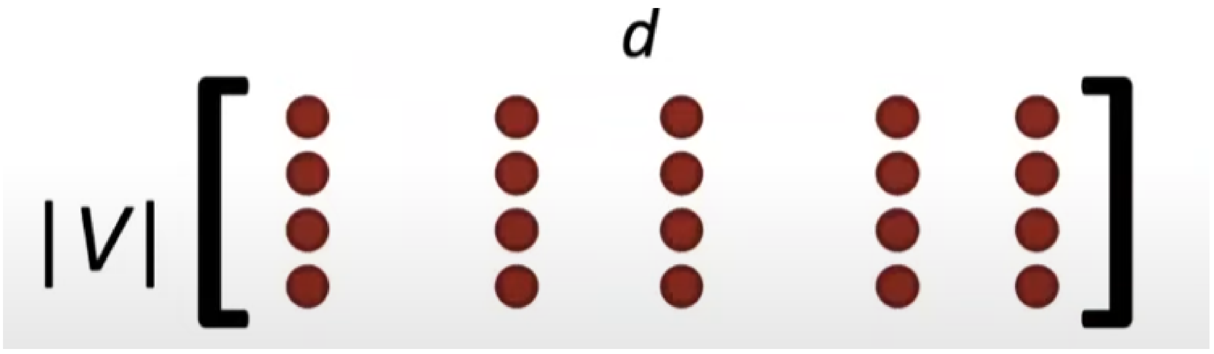
- Solution : Stochastic gradient descent (SGD)
    - Repeatedly sample windows, and update after each one, or each small batch

Stochastic gradients with word vectors!

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

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

- Solution : either you need sparse matrix update operations to only update certain rows of full embedding matrices U and V, or you need to keep around a hash for word vectors



Word2vec algorithm family : More details

- Why two vectors? → Easier optimization. Average both at the end
    - But can implement the algorithm with just one vector per word (slightly better but more complicated)
        - Sometimes you will have the same word type as the center word and the context word
- Two model variants:

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

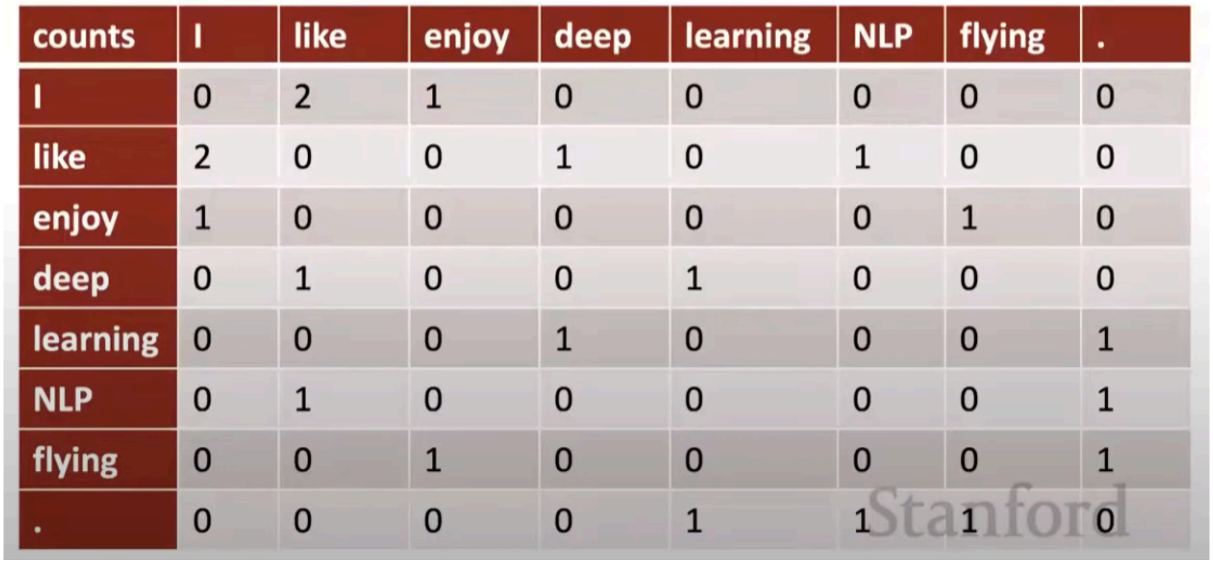
- Additional efficiency in training:
    1. Negative sampling (So far : Focus on naive softmax (simpler, but expensive, training method)
        - Instead of using this softmax, train binary logistic regression models for both the true pair of center word and the context word vs noise pairs where we keep the true center word and we just randomly sample words from the vocabulary
        - 다중분류 → 이진분류
        - Overall objective function
            - $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$, $J_t(\theta) = log\,\sigma(u_o^T v_c)$ + $\sum_{i=1}^{k}\mathbb{E}_{j\sim P(w)}[log\,\sigma(-u_j^T v_c)]$
                - $log\,\sigma(u_o^T v_c)$: 입력 단어 v_c에 대하여 positive sample u_o가 output일 확률 최대화
                - $\sum_{i=1}^{k}\mathbb{E}_{j\sim P(w)}[log\,\sigma(-u_j^T v_c)]$: negative sample u_j이 output이 될 확률 최소화

Why not capture co-occurrence counts directly?

- Building a co-occurrence matrix X

    - 2 options : windows vs. full document
    - Window : Similar to Word2vec, use window around each word → captures some syntactic and semantic information
    - Word-document co-occurrence matrix : Give general topics leading to “Latent Semantic Analysis”

- Example : Window based co-occurrence matix
    - Window length 1 (more common : 5 - 10)
    - Symmetric (irrelevant whether left of 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

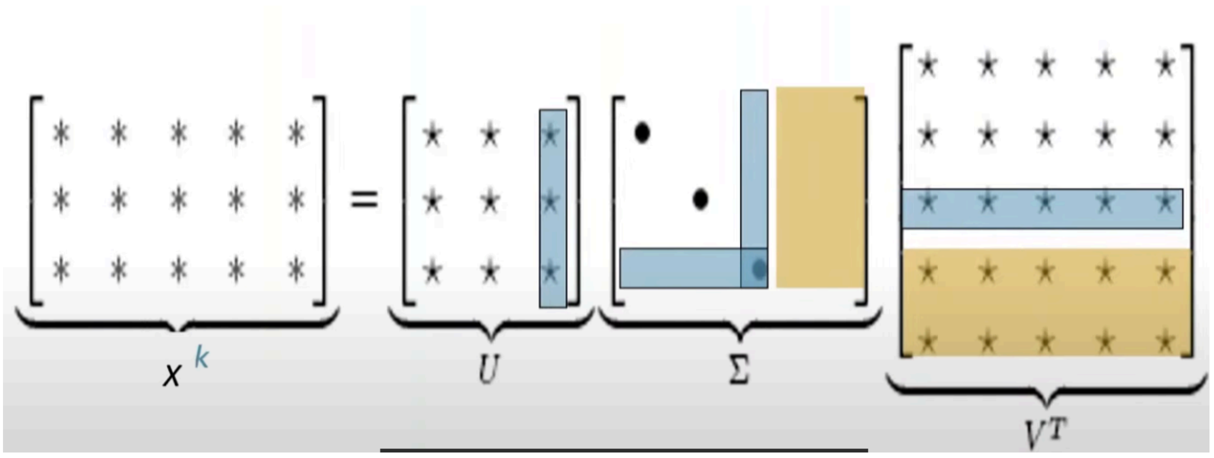
Co-occurrence vectors

- Simple count co-occurrence vectors
    - Vecotrs increase in size with vocabulary
    - Very high dimensional : require a lot of storage (though sparse)
    - Subsequent classification models have sparsity issues → Models are less robust

- Low-dimensional vectors
    - Idea : store "most" of the important information in a fixed, small number of dimensions : a dense vector
    - Usually 25-1000 dimensions, similar to Word2vec
    - How to reduce the dimensionality?

Classic Method : Dimensionality Reduction on X

- Singular Value Decomposition of co-occurrence matrix X

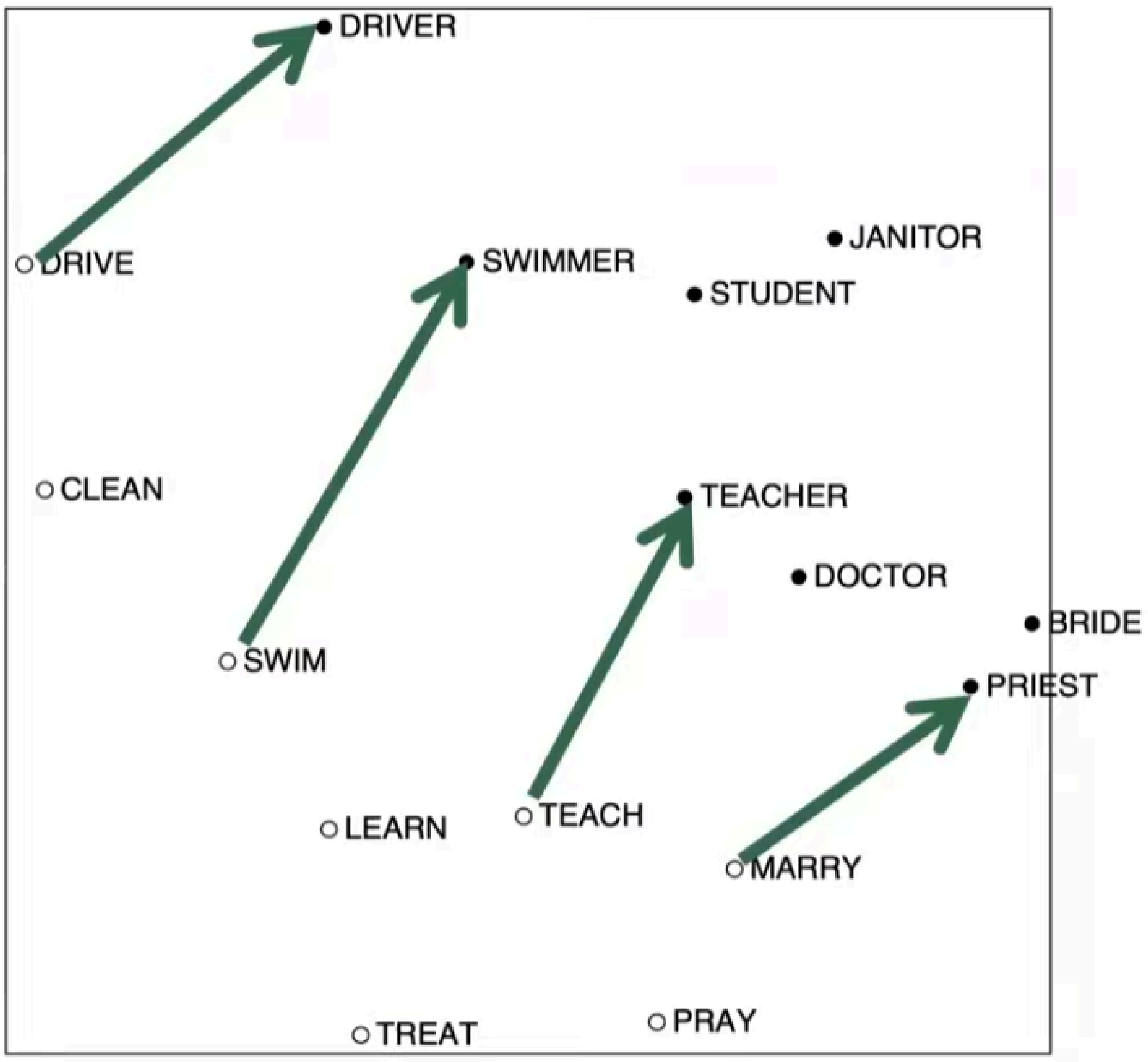


- Yellow : never used
- Blue : delete out more of the matrix (not used)

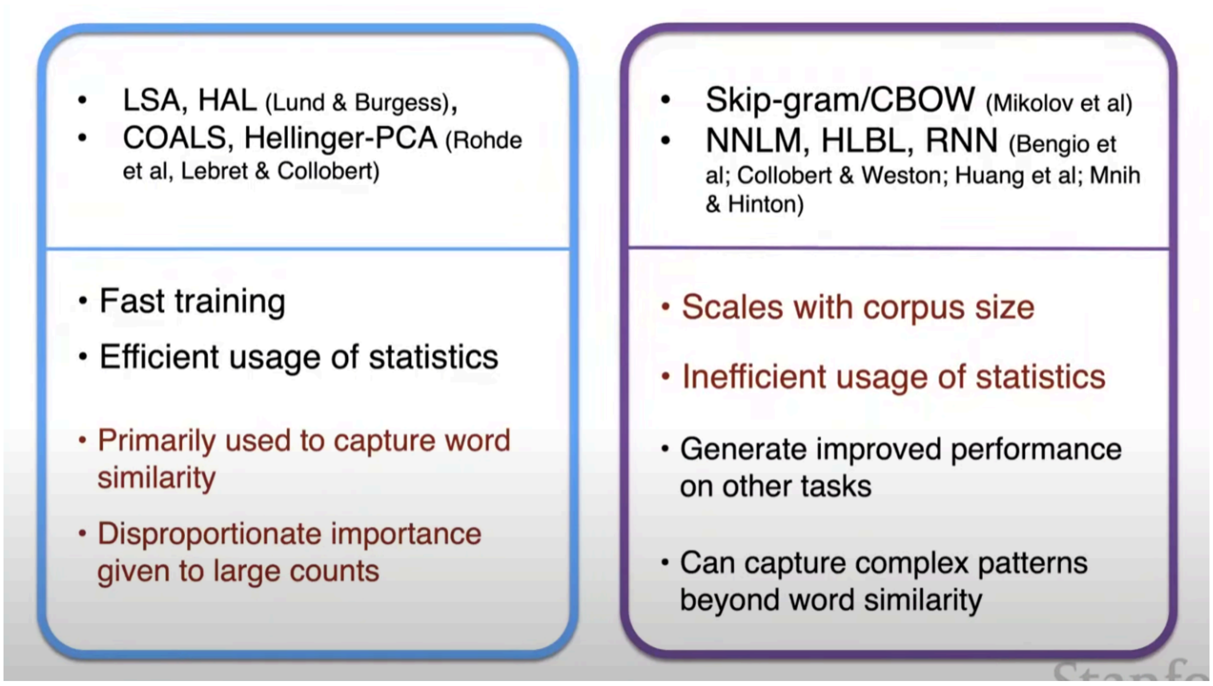
Hacks to X

- Running an SVD on raw counts doesn't work well
- Scaling the counts in the cells can help **a lot**
    - Problem : function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
        - log the frequencies
        - $min(X, t)$, with $t \approx 100$
        - Ignore the function words

Interesting semantic patterns emerge in the scaled vectors (COALS Model)



Towards GloVe : Count based vs. direct prediction



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

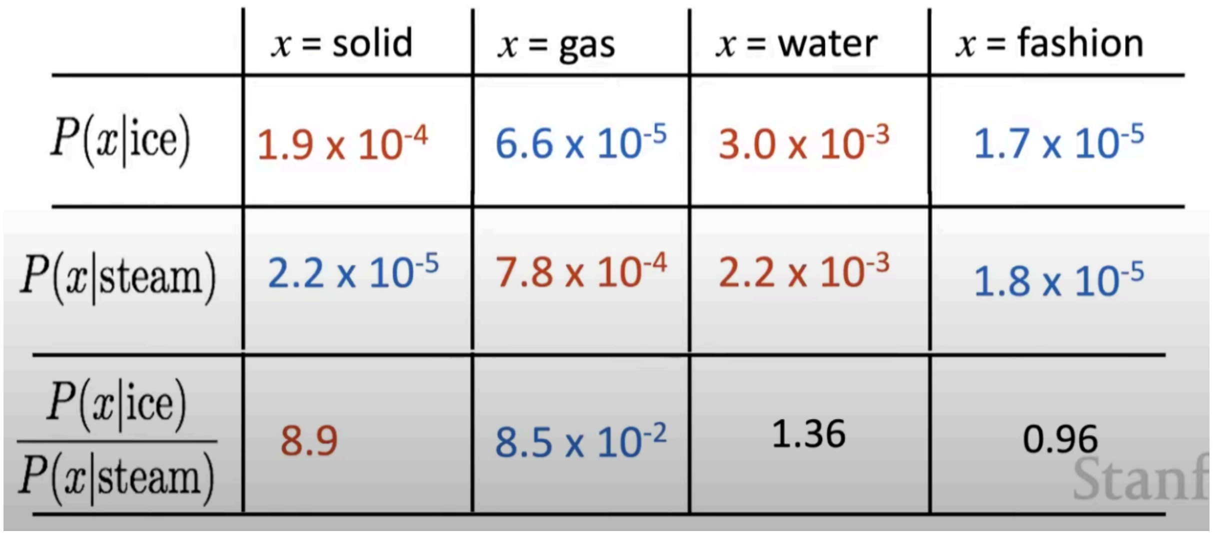
- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

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

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

GloVe algorithm (Encoding meaning components in vector differences)

- 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

- Q : How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?
- A : Log - bilinear model with vector differences
  - $w_i \cdot w_j = log\ P(i|j)$, $w_x \cdot (w_a - w_b) = log \frac{P(x|a)}{P(x|b)}$

    - 두 단어 임베딩 벡터의 내적이 co-occurrence matrix에서 동시 등장확률 로그값이 되도록 목적함수 정의
- $J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w_j} + b_i + \tilde{b_j} - logX_{ij})^2$
- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors

How to evaluate word vectors?

- Related to general evaluation in NLP : Intrinsic vs. extrinsic
- Intrinsic
    - Evalutation on a specific / intermediate subtask
    - Fast to compute
    - Help to understand that system
    - Not clear if really helpful unless correlation to real task is established
- Extrinsic
    - Evaluation on a real task
    - Can take a long time to compute accuracy
    - Unclear if the subsystem is the problem or its interaction or other subsystems
    - If replacing exactly one subsystem with another improves accuracy