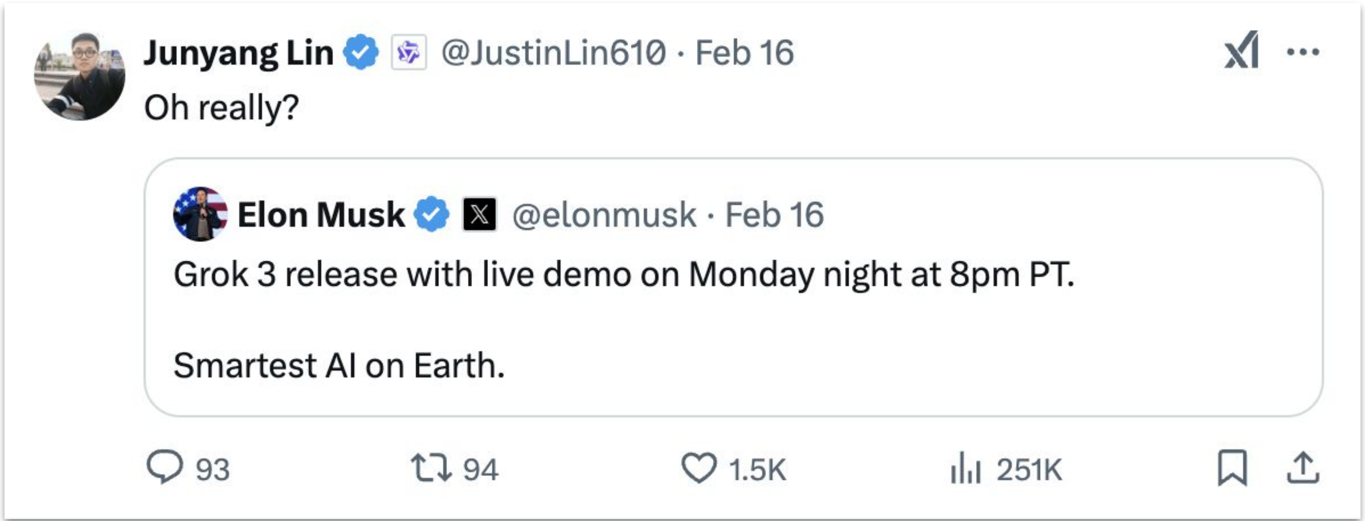
COMP 3361 Natural Language Processing

Lecture 6: Word Embeddings (cont'd)

Spring 2025

Latest AI news



Junyang Lin @JustinLin610 · Feb 16

Oh really?

Elon Musk @elonmusk · Feb 16

Grok 3 release with live demo on Monday night at 8pm PT.

Smartest AI on Earth.

93 94 1.5K 251K

Distributional hypothesis

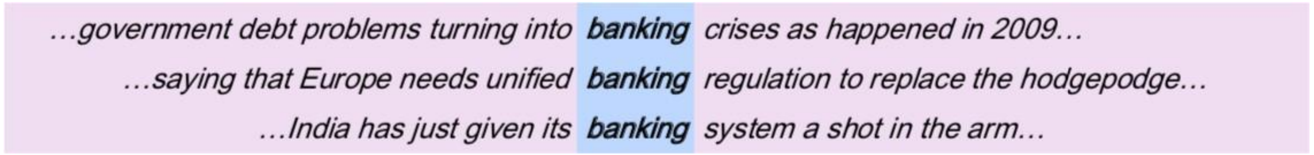
Distributional hypothesis: words that occur in similar **contexts** tend to have similar meaning

J.R.Firth 1957

"You shall know a word by the company it keeps"

One of the most successful ideas of modern statistical NLP!

When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed-size window).



…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent "*banking*".

Word embeddings: word2vec

Word embeddings: the learning problem

- Word embeddings are learned representations from text for representing words
    - Input: a large text corpora, *V*, *d*
        - V: a pre-defined vocabulary
        - d: dimension of word vectors (e.g. 300)
        - Text corpora:
            - Wikipedia + Gigaword 5: 6B tokens
            - Twitter: 27B tokens
            - Common Crawl: 840B tokens
    - **Output:** $f : V \to \mathbb{R}^d$

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \qquad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \qquad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

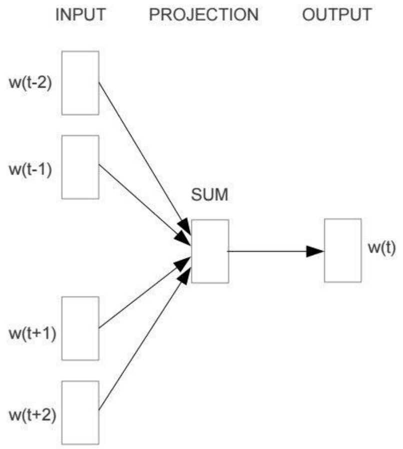
Each word is represented by a low-dimensional (e.g., d = 300), real-valued vector

Each coordinate/dimension of the vector doesn't have a particular interpretation

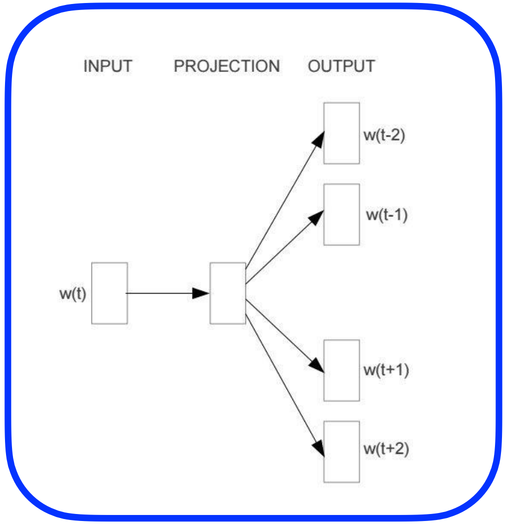
word2vec

- (Mikolov et al 2013a): Efficient Estimation of Word Representations in Vector Space
- (Mikolov et al 2013b): Distributed Representations of Words and Phrases and their Compositionality

Thomas Mikolov



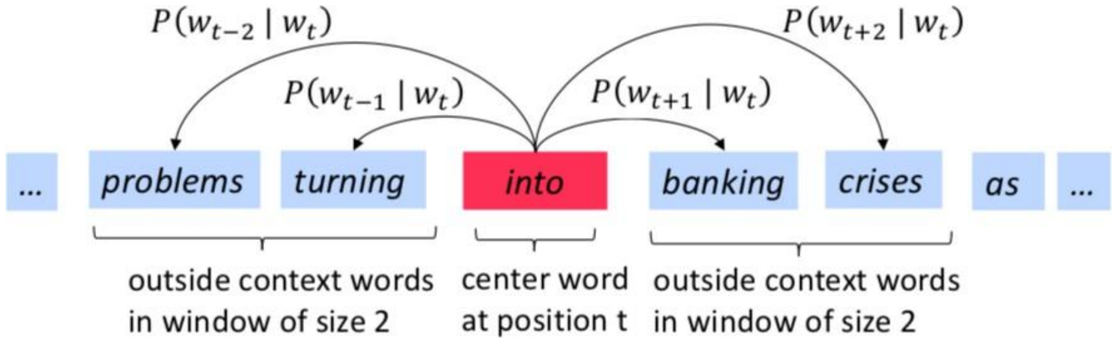
Continuous Bag of Words (CBOW)



Skip-gram

Skip-gram

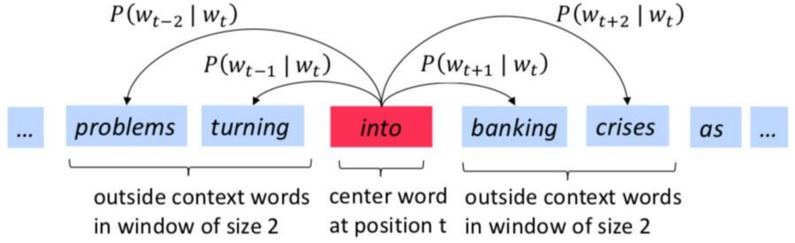
- Assume that we have a large corpus $w_1, w_2, \ldots, w_T \in V$
- **Key idea:** Use each word to **predict** other words in its context ← A classification problem!
- Context: a fixed window of size $2m$ (m = 2 in the example)

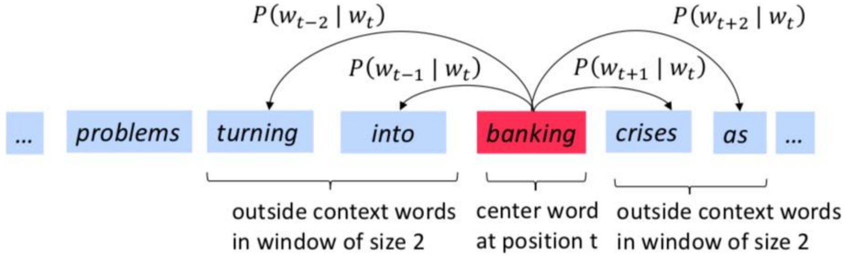


$P(b \mid a)$ = given the center word is a, what is the probability that b is a context word?

$P(\cdot \mid a)$ is a probability distribution defined over V: $\sum_{w \in V} P(w \mid a) = 1$

Skip-gram





Convert the training data into:
(into, problems)
(into, turning)
(into, banking)
(into, crises)
(banking, turning)
(banking, into)
(banking, crises)
(banking, as)
…

Our goal is to find parameters that can maximize

$\underline{P(\text{problems} \mid \text{into}) \times P(\text{turning} \mid \text{into}) \times P(\text{banking} \mid \text{into}) \times P(\text{crises} \mid \text{into})} \times \underline{P(\text{turning} \mid \text{banking}) \times P(\text{into} \mid \text{banking}) \times P(\text{crises} \mid \text{banking}) \times P(\text{as} \mid \text{banking})}\ldots$

Skip-gram: objective function

- For each position $t = 1, 2, \ldots T$, predict context words within context size m, given center word w_t:

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \ne 0} P(w_{t+j} \mid w_t; \overset{\text{all the parameters to be optimized}}{\theta})$$

- It is equivalent as minimizing the (average) negative log likelihood:

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

How to define $P(w_{t+j} \mid w_t; \theta)$?

- Use two sets of vectors for each word in the vocabulary

$\mathbf{u}_a \in \mathbb{R}^d$: vector for center word a, $\forall a \in V$

$\mathbf{v}_b \in \mathbb{R}^d$: vector for context word b, $\forall b \in V$

- Use inner product $\mathbf{u}_a \cdot \mathbf{v}_b$ to measure how likely word *a* appears with context word *b*

Softmax we have seen in multinomial logistic regression!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Recall that $P(\cdot \mid a)$ is a probability distribution defined over V…

Skip-gram: objective function

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log \frac{\exp(\mathbf{u}_{w_t}\cdot \mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot \mathbf{v}_k)}$$

- In this formulation, we don't care about the classification task itself like we do for the logistic regression model we saw previously.
- The key point is that the parameters used to optimize this training objective— when the training corpus is large enough—can give us very good representations of words (following the principle of distributional hypothesis)!

How many parameters in this model?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of θ)?

(a) $d|V|$
(b) $2d|V|$
(c) $2m|V|$
(d) $2md|V|$

d = dimension of each vector

How many parameters in this model?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of θ)?

(a) $d|V|$

(b) $2d|V|$

(c) $2m|V|$

(d) $2md|V|$

d = dimension of each vector

The answer is (b).
Each word has two d-dimensional vectors, so it is $2\times|V|\times d$.

word2vec formulation

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

Q: Why do we need two vectors for each word instead of one?

Q: Which set of vectors are used as word embeddings?

word2vec formulation

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

Q: Why do we need two vectors for each word instead of one?

A: because one word is not likely to appear in its own context window, e.g., P(dog | dog) should be low. If we use one set of vectors only, it essentially needs to minimize $\mathbf{u}_{\text{dog}}\cdot\mathbf{u}_{\text{dog}}$..

Q: Which set of vectors are used as word embeddings?

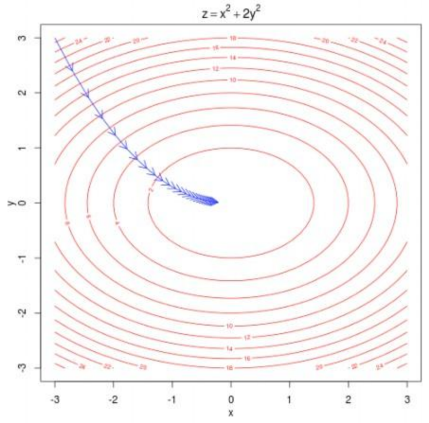
A: This is an empirical question. Typically just $\mathbf{u}_w$ but you can also concatenate the two vectors..

How to train this model?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

- To train such a model, we need to compute the vector gradient $\nabla_\theta J(\theta) = ?$
- Again, θ represents all $2d|V|$ model parameters, in one vector.

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix}$$



Let's compute gradients for word2vec

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\ne 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

Consider one pair of center/context words (t, c):

$$y = -\log\left(\frac{\exp(\mathbf{u}_t\cdot\mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_t\cdot\mathbf{v}_k)}\right)$$

We need to compute the gradient of y with respect to

$\mathbf{u}_t$ and $\mathbf{v}_k, \forall k \in V$

Let's compute gradients for word2vec

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

$$y = -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log(\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$
$$= -\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$

Recall that

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

$$\frac{\partial y}{\partial \mathbf{u}_t} = \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t}$$

$$= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

$$= -\mathbf{v}_c + \frac{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)\cdot\mathbf{v}_k}{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

$$\boxed{= -\mathbf{v}_c + \sum_{k\in V}\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\sum_{k'\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_{k'})}\mathbf{v}_k}$$

$$= -\mathbf{v}_c + \sum_{k\in V} P(k \mid t)\mathbf{v}_k$$

Let’s compute gradients for word2vec

What about context vectors?

$$\frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\,\mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$$

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

Overall algorithm

- Input: text corpus, embedding size d, vocabulary V, **context size m**
- Initialize $\mathbf{u}_i, \mathbf{v}_i$ randomly $\forall i \in V$
- Run through the training corpus and for each training instance (*t*, *c*):
  - Update $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u_t}}$ $\quad \frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid t)\mathbf{v}_k$
  - Update $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v_k}}, \forall k \in V$ $\quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\, \mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$

Q: Can you think of any issues with this algorithm?

Skip-gram with negative sampling (SGNS)

Problem: every time you get one pair of (t, c), you need to update $\mathbf{v}_k$ with all the words in the vocabulary! This is very expensive computationally.

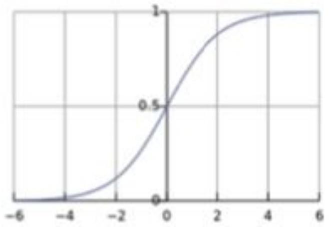
$$\frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid t)\mathbf{v}_k \qquad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\,\mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$$

Negative sampling: instead of considering all the words in V, let's randomly sample K (5-20) negative examples.

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

softmax: $$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

Negative sampling: $$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$



Skip-gram with negative sampling (SGNS)

Key idea: Convert the $|V|$-way classification into a set of binary classification tasks.

Every time we get a pair of words (*t*, *c*), we don't predict *c* among all the words in the vocabulary. Instead, we predict (*t*, *c*) is a positive pair, and (*t*, *c'*) is a negative pair for a small number of sampled *c'*.

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

P(w): sampling according to the frequency of words

Similar to **binary logistic regression**, but we need to optimize $\mathbf{u}$ and $\mathbf{v}$ together.

$$P(y = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c) \qquad p(y = 0 \mid t, c') = 1 - \sigma(\mathbf{u}_t \cdot \mathbf{v}_{c'}) = \sigma(-\mathbf{u}_t \cdot \mathbf{v}_{c'})$$

Understanding SGNS

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in θ for every (t, c) pair?

(a) Kd

(b) $2Kd$

(c) $(K+1)d$

(d) $(K+2)d$

Understanding SGNS

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in θ for every (t, c) pair?

(a) Kd

(b) $2Kd$

(c) $(K+1)d$

(d) $(K+2)d$

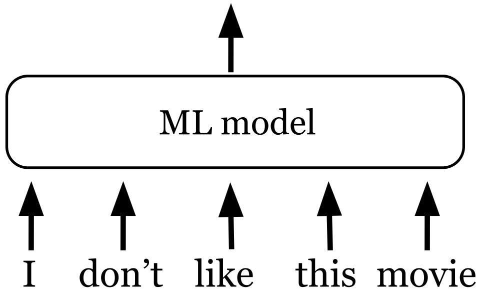
The answer is (d).
We need to calculate gradients with respect to $\mathbf{u}_t$ and (K + 1) $\mathbf{v}_i$ (one positive and K negatives).

Evaluating word embeddings

Extrinsic vs intrinsic evaluation

Extrinsic evaluation

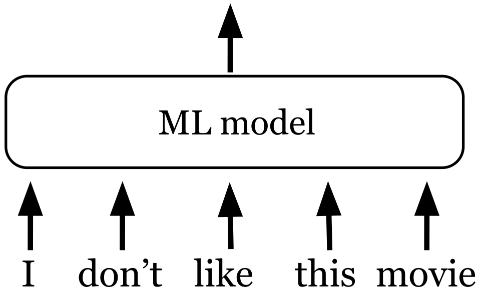
- Let’s plug these word embeddings into a real NLP system and see whether this improves performance
- Could take a long time but still the most important evaluation metric



Intrinsic evaluation

- Evaluate on a specific/intermediate subtask
- Fast to compute
- Not clear if it really helps downstream tasks

Extrinsic evaluation



A straightforward solution: given an input sentence $x_1, x_2, \ldots, x_n$

Instead of using a bag-of-words model, we can compute $vec(x) = \mathbf{e}(x_1) + \mathbf{e}(x_2) + \ldots + \mathbf{e}(x_n)$

And then train a logistic regression classifier on as we did before!

There are much better ways to do this e.g., take word embeddings as input of neural networks

Intrinsic evaluation: word similarity

Word similarity

Example dataset: wordsim-353
353 pairs of words with human judgement
http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

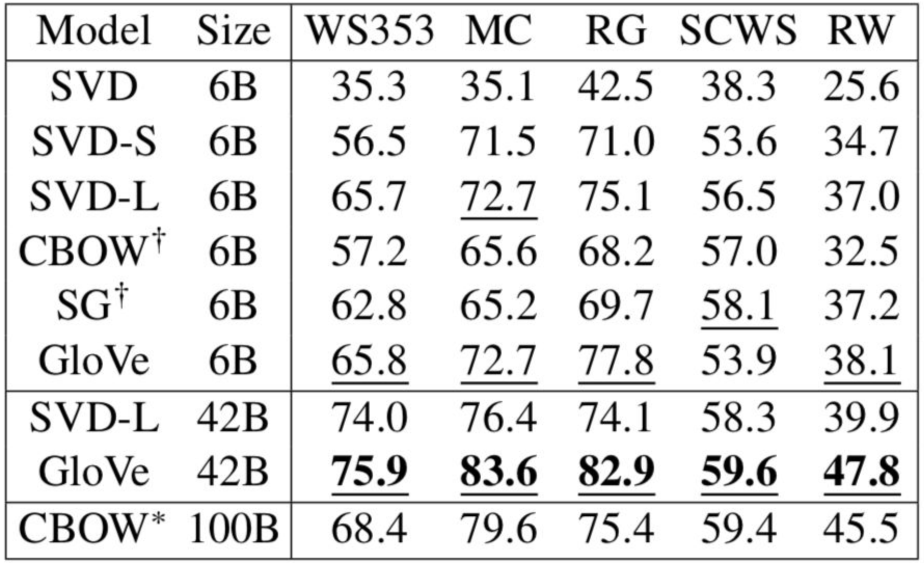
Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Cosine similarity:

$$\cos(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{\boldsymbol{u}_i \cdot \boldsymbol{u}_j}{||\boldsymbol{u}_i||_2 \times ||\boldsymbol{u}_j||_2}.$$

Metric: Spearman rank correlation

Intrinsic evaluation: word similarity



Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

SG: Skip-gram

Intrinsic evaluation: word analogy

Word analogy test: $a : a* :: b : b*$

$$b^* = \arg\max_{w \in V} \cos(e(w), e(a^*) - e(a) + e(b))$$

semantic

syntactic

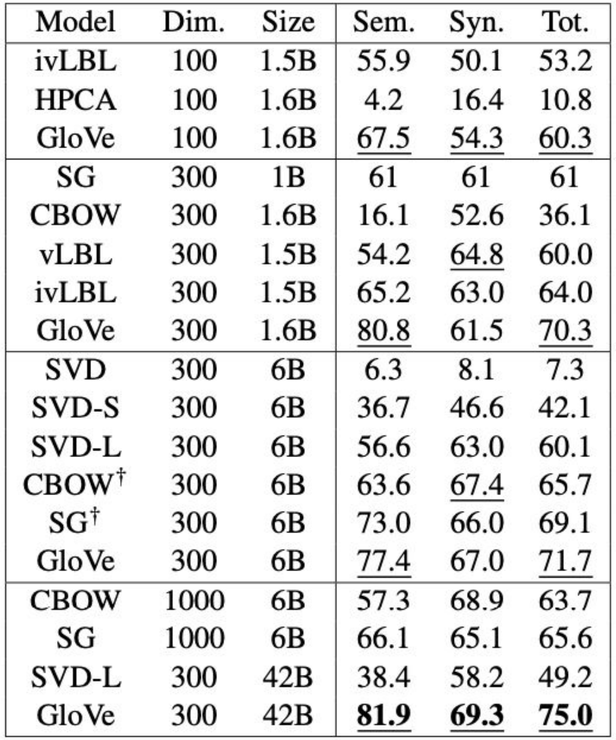
Chicago:Illinois ~ Philadelphia: ?

bad:worst ~ cool: ?

More examples at http://download.tensorflow.org/data/questions-words.txt

Metric: accuracy

Intrinsic evaluation: word analogy



Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**