innovate achieve lead

Natural Language Processing

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Session 3

These slides are prepared by the instructor, with grateful acknowledgement of Jurafsky and Martin and many others who made their course materials freely available online.

Session Content

innovate achieve lead

Word Embedding

- Word and Vectors
- Word2Vec
  - Skip gram
  - CBOW
- Glove
- Visualizing Embedding's

Word embeddings

- Vectors for representing words are called embeddings
- Each word = a vector (not just "good" or "w_{45}")
- Defining meaning as a point in space based on distribution
- Similar words are "**nearby in semantic space**"
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**

two-dimensional (t-SNE) projection of embeddings

We define meaning of a word as a vector

- Called an "embedding" because it's embedded into a space
- The standard way to represent meaning in NLP
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**
- Fine-grained model of meaning for similarity

Intuition: why vectors?

- Consider sentiment analysis:
  - With **words**, a feature is a word identity
    - Feature 5: 'The previous word was "terrible"'
    - requires **exact same word** to be in training and test
  - With **embeddings**:
    - Feature is a word vector
    - 'The previous word was vector [35,22,17…]
    - Now in the test set we might see a similar vector [34,21,14]
    - We can generalize to **similar but unseen** words!!!

Word embeddings: types

1. Frequency based Embedding
   - A common baseline model
   - **Sparse** long vectors
   - Words are represented by (a simple function of) the **counts** of nearby words
   - **dimensions** corresponding to **words** in the vocabulary or **documents** in a collection
   - **Count Vector**
   - **TF-IDF Vector**
   - **Co-Occurrence Vector**
2. Prediction based Embedding
   - **Dense** vectors
   - Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
   - **Word2vec: Skip-gram and CBOW**
   - **GloVe**

Common methods for getting short dense vectors

- "Neural Language Model"-inspired models
  - Word2vec (skipgram, CBOW), GloVe

- **Alternative to these "static embeddings":**
  - Contextual Embeddings (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
    - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
    - A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
    - No need for human labels

Word2vec

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**

Word2vec provides various options. We'll do:

skip-gram with negative sampling (SGNS)

Simple static embeddings you can download!

- Word2vec (Mikolov et al)
- https://code.google.com/archive/p/word2vec/

- GloVe (Pennington, Socher, Manning)
- http://nlp.stanford.edu/projects/glove/

Basic word representation

V = [a, aaron, …, zulu, <UNK>]

1-hot representation

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
$\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\\vdots\\1\\\vdots\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix}$	$\begin{bmatrix}0\\\vdots\\1\\\vdots\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix}$

I want a glass of orange _____.

I want a glass of apple _____.

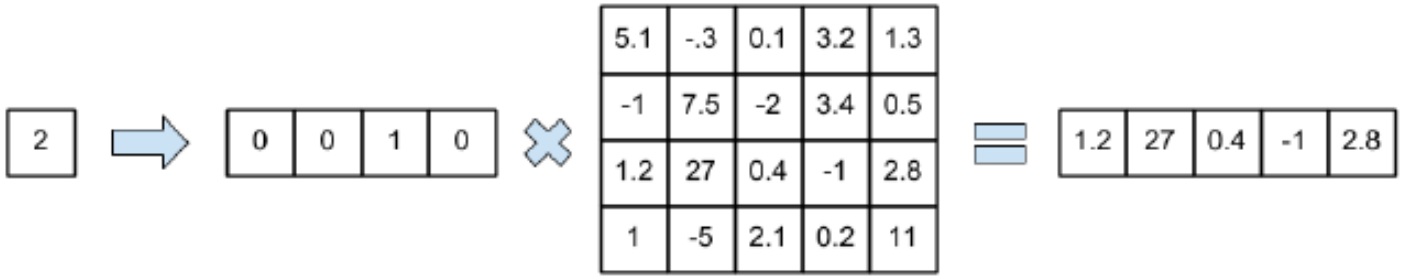
Featurized representation: word embedding

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

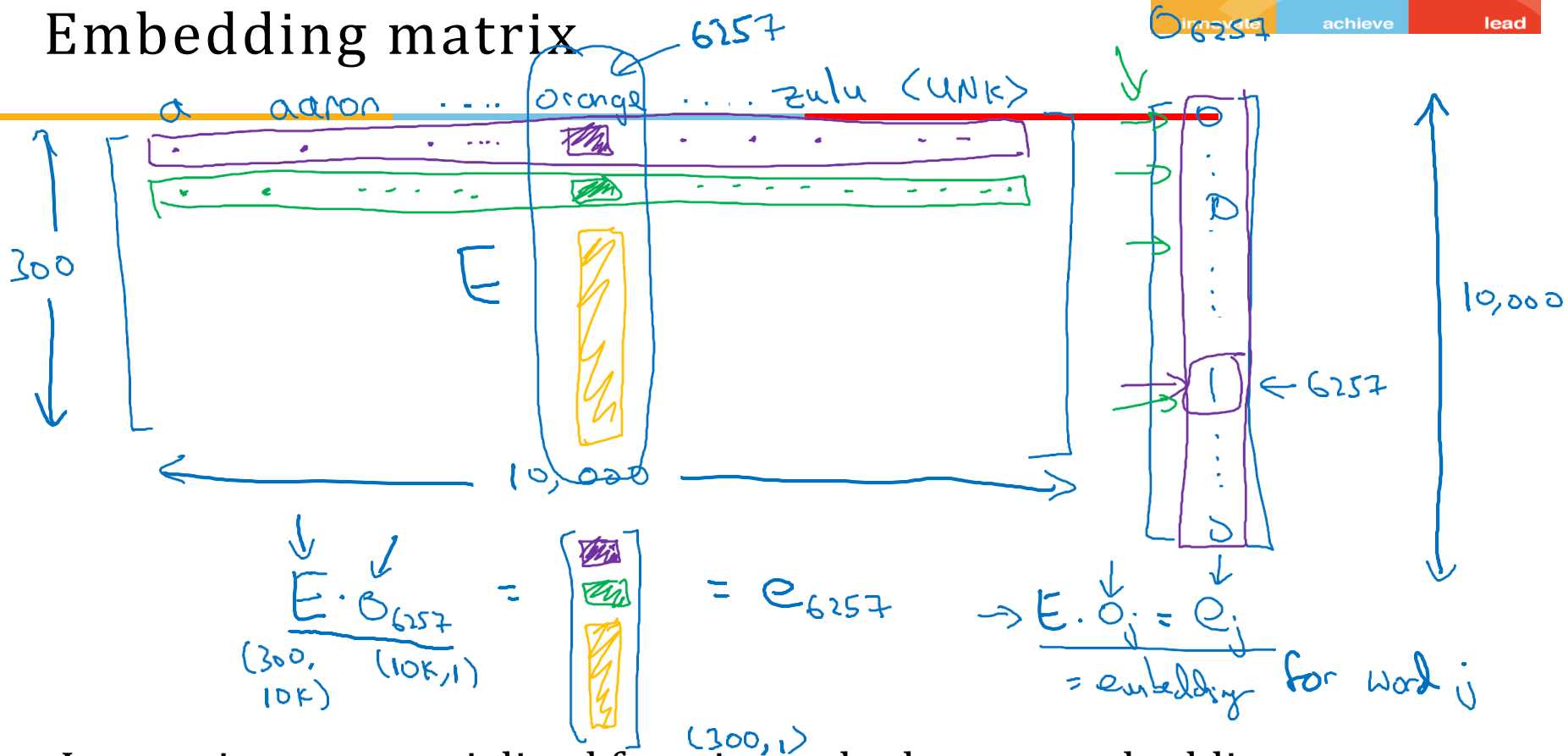
I want a glass of orange I want a glass of apple

[Mikolov et. al., 2013, Linguistic regularities in continuous space word representations]

Word embeddings



Embedding matrix



In practice, use specialized function to look up an embedding.

Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
    - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
    - A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
    - No need for human labels
    - Bengio et al. (2003); Collobert et al. (2011)

Word2vec

- An unsupervised NLP method developed by Google in 2013 (Mikolov et al. 2013)
- Quantify the relationship between words
- Framework for learning word vectors Idea:
  - We have a large corpus (“body”) of text: a long list of words
  - 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

Approach: predict if candidate word *c* is a "neighbor"

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skip-Gram Training Data

- Assume a +/- 2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

[target]

- c1 c2 c3 c4

Skip-Gram Classifier

- (assuming a +/- 2 word window)

…lemon, a [tablespoon of apricot jam, a] pinch…
c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair
(apricot, jam)
(apricot, aardvark)
…
And assigns each pair a probability:

$P(+|w, c)$

$P(-|w, c) = 1 - P(+|w, c)$

Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product
- So:
  - Similarity(w,c) $\propto w \cdot c$
- We'll need to normalize to get a probability
  - (cosine isn't a probability either)

Turning dot products into probabilities

- Sim(w,c) ≈ w · c
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$

$$= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}$$

How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words.
We'll assume independence and just multiply them:

$$P(+|w,c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

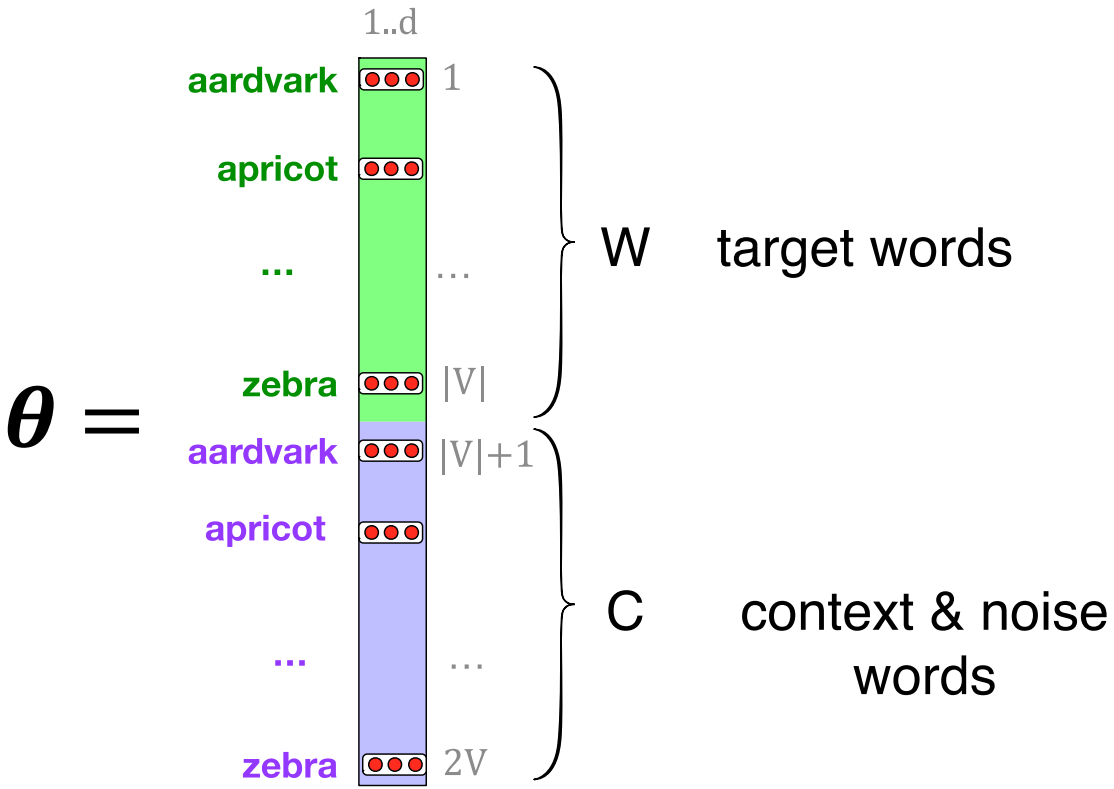
$$\log P(+|w,c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Skip-gram classifier: summary

- A probabilistic classifier, given
  - a test target word *w*
  - its context window of *L* words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

- To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w, a set for c



Skip-Gram Training data

- For every + example select k negative examples

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

The paper (Mikolov et al., 2013) says that K=2 ~ 5 works for large data sets, and K=5 ~ 20 for small data sets.

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

For each positive example we'll grab k negative examples, sampling by frequency

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

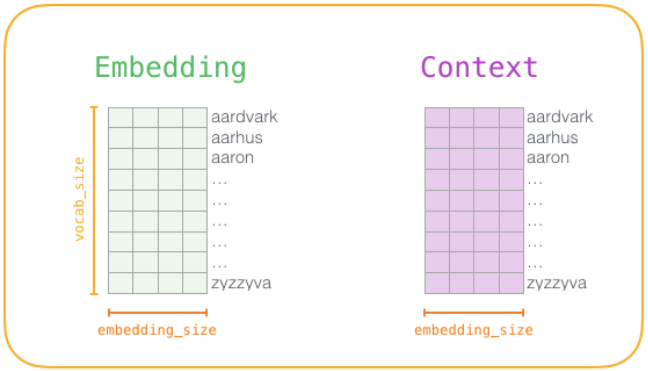
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

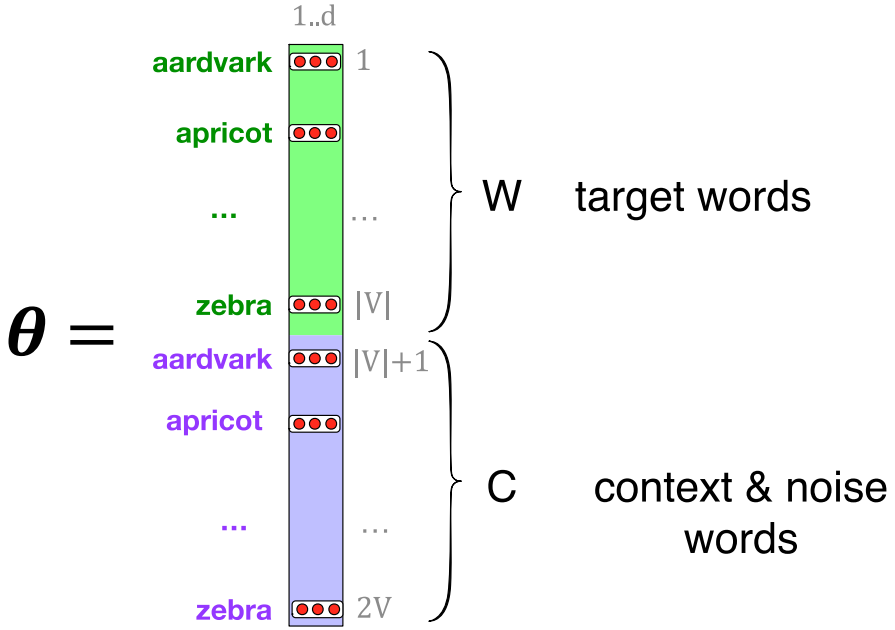
negative examples -

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

Skip-gram parameters

- SGNS learns two sets of embeddings
  - Target embeddings matrix W
  - Context embedding matrix C
- It's common to just add them together, representing word i as the vector $w_i + c_i$
- Thus the parameters we need to learn are two matrices W and C, each containing an embedding for every one of the |V| words in the vocabulary |V|





Word2vec training : how to learn vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
  - **Maximize** the similarity of the target word, context word pairs (w , c_{pos}) drawn from the positive data
  - **Minimize** the similarity of the (w , c_{neg}) pairs drawn from the negative data.

Log Loss function

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Loss function for one *w with* c_{pos} , c_{neg1} ...c_{negk}

- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the *k* negative sampled non-neighbor words.

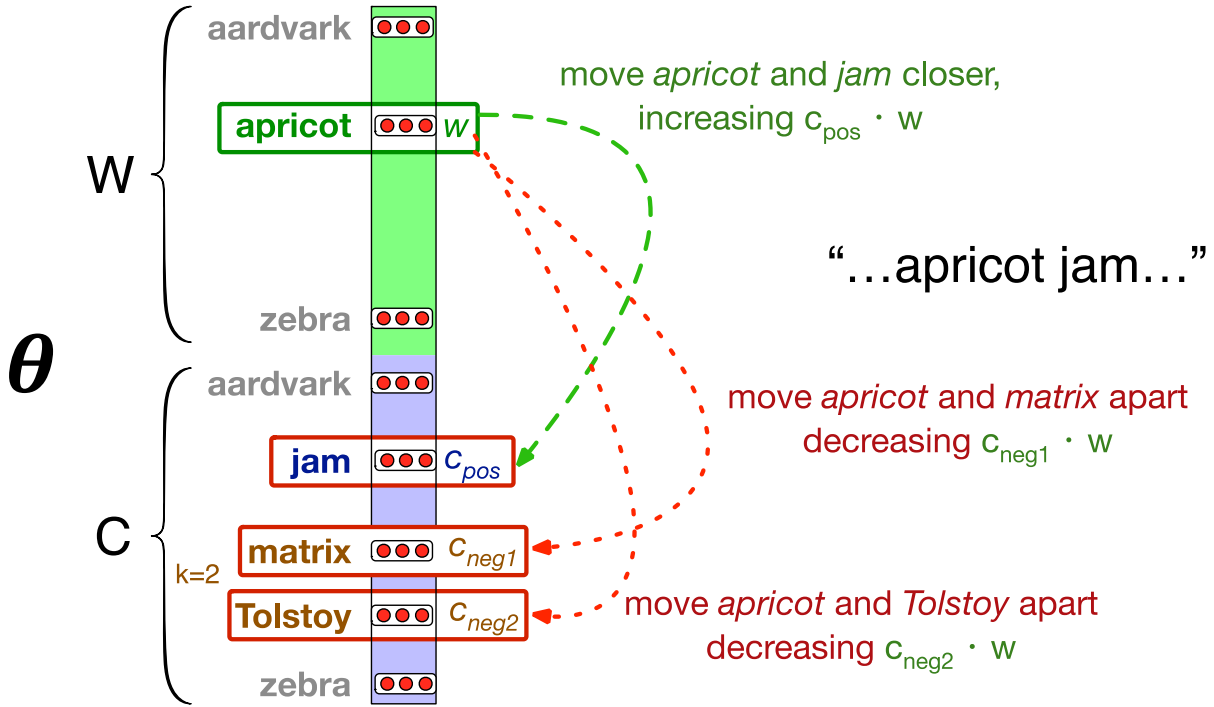
$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Learning the classifier

- How to learn?
  - Stochastic gradient descent!

- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.

Intuition of one step of gradient descent



- Tries to shift embeddings
- So the target embeddings (here for apricot) are closer to (have a higher dot product with) context embeddings for nearby words (here jam)
- And further from (lower dot product with) context embeddings for noise words that don't occur nearby (here Tolstoy and matrix).

Reminder: gradient descent

- At each step
  - Direction: We move in the reverse direction from the gradient of the loss function
  - Magnitude: we move the value of this gradient $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η
  - Higher learning rate means move *w* faster

$$w^{t+1} = w^{t} - h\frac{d}{dw}L(f(x;w),y)$$

The derivatives of the loss function

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t}$$

$$c_{neg}^{t+1} = c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t}$$

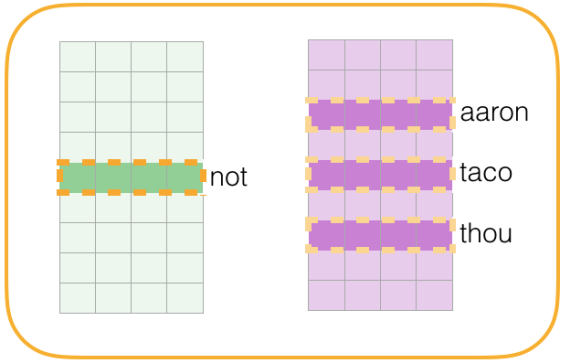
$$w^{t+1} = w^{t} - \eta\left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i}\right]$$

Skip-gram training example

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$
$$= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}$$



Update Model Parameters

https://jalammar.github.io/illustrated-word2vec/

Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together, representing word *i* as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

- Start with V random d-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - –Take a corpus and take pairs of words that co-occur as positive examples
  - –Take pairs of words that don't co-occur as negative examples
  - –Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - –Throw away the classifier code and keep the embeddings.

Word Embeddings

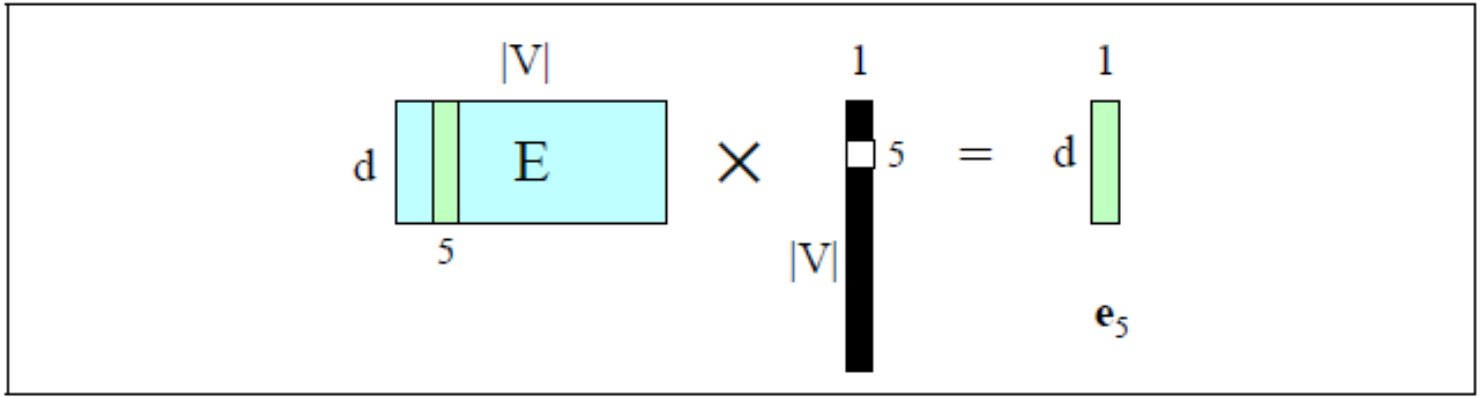


Figure 7.12 Selecting the embedding vector for word V_5 by multiplying the embedding matrix **E** with a one-hot vector with a 1 in index 5.

Example

Corpus = "Ned Stark is the most honourable man"

Current word-context pair = ("Ned", "Stark")

Current negative words = "pimples", "zebra", "idiot"

	one hot vector				
ned	1	0	0	0	0
stark	0	1	0	0	0
pimples	0	0	1	0	0
zebra	0	0	0	1	0
idiot	0	0	0	0	1

initial embedding matrix			
ned	-0.018	0.404	-0.317
stark	0.204	-0.007	-0.733
pimples	-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502

initial context matrix			
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

- https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling

Initial embeddings

Current word-context pair = *(" Ned ", " Stark ")*
Current negative words = *"pimples", "zebra", "idiot"*

#	Token	x
0	honorable	0
1	is	0
2	man	0
3	most	0
4	**ned**	**1**
5	**stark**	**0**
6	the	0
Neg.	pimples	0
	zebra	0
	idiot	0
	coins	0
	donkey	0
	machine	0

$\times$

W_{input}

-0.014	0.135	0.109
-0.665	0.669	0.309
0.702	0.621	-0.981
0.214	-0.813	-0.561
-0.018	0.404	-0.317
0.204	-0.007	-0.733
-0.652	-0.097	0.499
-0.706	0.745	0.002
-0.492	-0.709	0.133
-0.628	-0.073	0.502
0.456	0.767	-0.629
0.348	0.544	-0.740
-0.627	0.487	0.466

$=$

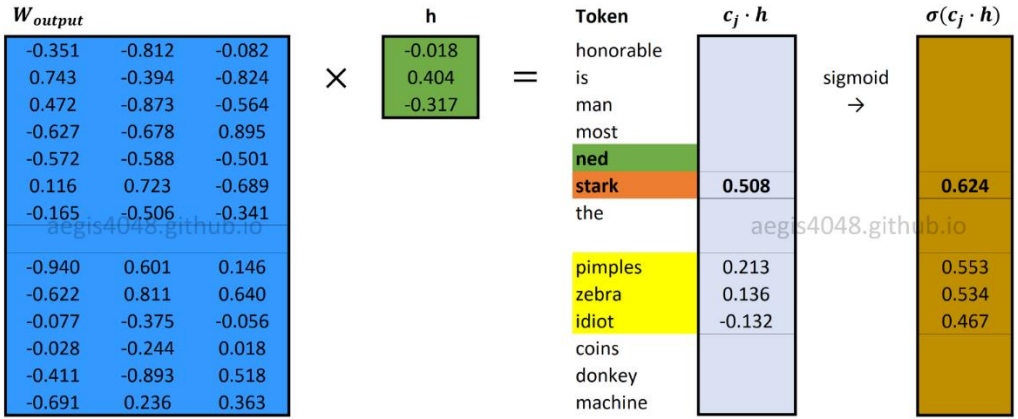
h

-0.018
0.404
-0.317

	input embeddings initial		
ned	-0.018	0.404	-0.317
stark	0.204	-0.007	-0.733
pimples	-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502

	context embeddings initial		
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

Forward Propagation: Sigmoid output layer



$$P(+|w,c) = \sigma(\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

old context embeddings				old i/p word embedding	dot product	sigmoid()
stark	0.116	0.723	-0.689	-0.018	0.508417	0.624435309
pimples	-0.94	0.601	0.146	0.404	0.213442	0.446841162
zebra	-0.622	0.811	0.64	-0.317	0.13596	0.466062263
idiot	-0.077	-0.375	-0.056		-0.132362	0.533042273

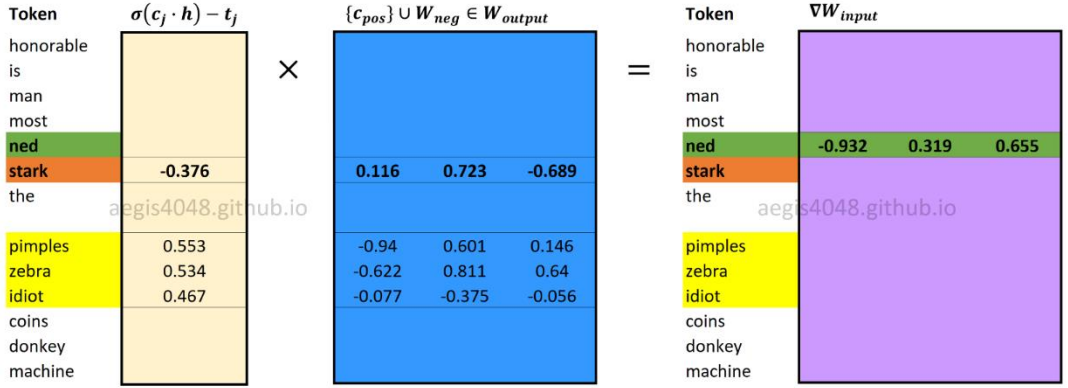
Backward Propagation: Prediction Error

our current positive word is Stark, t_j=1 for Stark and t_j=0 for other negative words (pimples, zebra, idiot).

Token	$\sigma(c_j \cdot h)$		t_j		$\sigma(c_j \cdot h) - t_j$
honorable					
is					
man					
most					
ned					
stark	0.624		1		-0.376
the					
pimples	0.553		0		0.553
zebra	0.534		0		0.534
idiot	0.467		0		0.467
coins					
donkey					
machine					

Sigmoid()	t	sig-t (pred error)
0.624435309	1	-0.375564691
0.446841162	0	0.446841162
0.466062263	0	0.466062263
0.533042273	0	0.533042273

Derivative of loss w.r.t input word embeddings



$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Sigmoid()	t	sig-t (pred error)
0.624435309	1	-0.375564691
0.446841162	0	0.446841162
0.466062263	0	0.466062263
0.533042273	0	0.533042273

	context embeddings initial		
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

sig-t(pred error)	C*(sig-t)			derivative of loss w.r.t input word embeddings		
-0.37556469	-0.04356550	-0.271533272	0.258764072	-0.794531	0.1751039	0.5924323
0.44684116	-0.42003069	0.268551538	0.06523881			
-0.466062263	0.289890727	0.377976495	0.298279848			
-0.533042273	0.041044255	-0.199890852	-0.029850367			

Input word embedding update

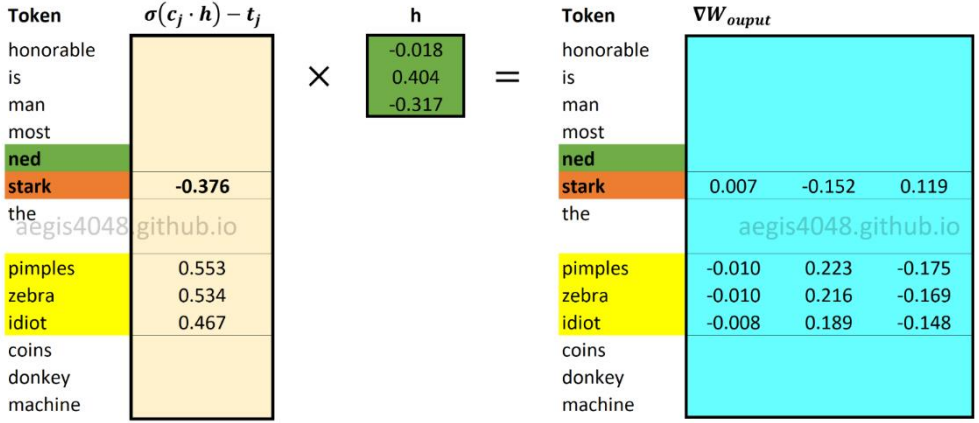
$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \left[[\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}^{t}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k} [\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w}^{t})]\mathbf{c}_{neg_i} \right]$$

Token	$W_{input}^{(old)}$			−	η	×	∇W_{input}			=	$W_{input}^{(new)}$		
honorable	-0.014	0.135	0.109		0.05						-0.014	0.135	0.109
is	-0.665	0.669	0.309								-0.665	0.669	0.309
man	0.702	0.621	-0.981								0.702	0.621	-0.981
most	0.214	-0.813	-0.561								0.214	-0.813	-0.561
ned	-0.018	0.404	-0.317				-0.932	0.319	0.655		0.029	0.388	-0.350
stark	0.204	-0.007	-0.733								0.204	-0.007	-0.733
the	-0.652	-0.097	0.499								-0.652	-0.097	0.499
pimples	-0.706	0.745	0.002								-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133								-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502								-0.628	-0.073	0.502
coins	0.456	0.767	-0.629								0.456	0.767	-0.629
donkey	0.348	0.544	-0.740								0.348	0.544	-0.740
machine	-0.627	0.487	0.466								-0.627	0.487	0.466

derivative of loss w.r.t input word embeddings		
-0.93154	0.318454	0.65541

word	old input word embedding			LR	LR* derivative w.r.t input			new i/p word embedding		
ned	-0.018	0.404	-0.317	0.05	-0.0397265	0.0087551	0.0296216	**0.0217265**	**0.3952448**	**-0.346**

Derivative of loss w.r.t context word embeddings



$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

sigmoid-t (pred error)	i/p word embedding	derivative of loss w.r.t context word embeddings		
-0.375564691	-0.018	0.006760164	-0.151728135	0.119054007
0.446841162	0.404	-0.008043141	0.18052383	-0.141648648
0.466062263	-0.317	-0.008389121	0.188289154	-0.147741737
0.533042273		-0.009594761	0.215349078	-0.168974401

Context word embedding update

Token	$W^{(old)}_{output}$				∇W_{ouput}			$W^{(new)}_{output}$		
honorable	-0.351	-0.812	-0.082	−				-0.351	-0.812	-0.082
is	0.743	-0.394	-0.824					0.743	-0.394	-0.824
man	0.472	-0.873	-0.564					0.472	-0.873	-0.564
most	-0.627	-0.678	0.895					-0.627	-0.678	0.895
ned	-0.572	-0.588	-0.501					-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689		0.007	-0.152	0.119	**0.116**	**0.731**	**-0.695**
the	-0.165	-0.506	-0.341					-0.165	-0.506	-0.341
pimples	-0.94	0.601	0.146		-0.010	0.223	-0.175	**-0.940**	**0.590**	**0.155**
zebra	-0.622	0.811	0.64		-0.010	0.216	-0.169	**-0.622**	**0.800**	**0.648**
idiot	-0.077	-0.375	-0.056		-0.008	0.189	-0.148	**-0.077**	**-0.384**	**-0.049**
coins	-0.028	-0.244	0.018					-0.028	-0.244	0.018
donkey	-0.411	-0.893	0.518					-0.411	-0.893	0.518
machine	-0.691	0.236	0.363					-0.691	0.236	0.363

η = 0.05 ×

$$c^{t+1}_{pos} = c^{t}_{pos} - \eta[\sigma(c^{t}_{pos} \cdot w^{t}) - 1]w^{t}$$
$$c^{t+1}_{neg} = c^{t}_{neg} - \eta[\sigma(c^{t}_{neg} \cdot w^{t})]w^{t}$$

LR= 0.05

word	old context word embedding			LR* derivative w.r.t context			new context word embedd		
stark	0.116	0.723	-0.689	0.000338	-0.00758	0.00595	0.1156619	0.730586	-0.69495
pimples	-0.94	0.601	0.146	-0.0004	0.009026	-0.00708	-0.939597843	0.591973	0.153082
zebra	-0.622	0.811	0.64	-0.00042	0.009414	-0.00738	-0.621580544	0.801585	0.647387
idiot	-0.077	-0.375	-0.056	-0.00047	0.0107	-0.00845	-0.076520262	-0.3857	-0.04755

The kinds of neighbors depend on window size

- **Small windows** (C= +/- 2) : nearest words are syntactically similar words in same taxonomy
  - *Hogwarts* nearest neighbors are other fictional schools
    - *Sunnydale, Evernight, Blandings*
- **Large windows** (C= +/- 5) : nearest words are related words in same semantic field
  - *Hogwarts* nearest neighbors are Harry Potter world:
    - *Dumbledore, half-blood, Malfoy*

Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data

William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

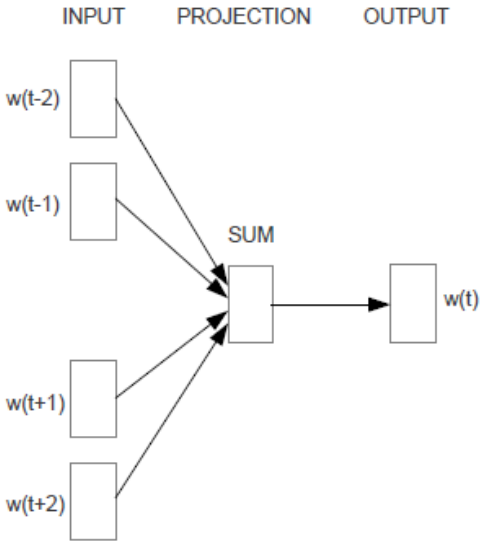
Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical)* are biased toward men, a bias slowly decreasing 1960-1990
  - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

CBOW vs Skip-gram



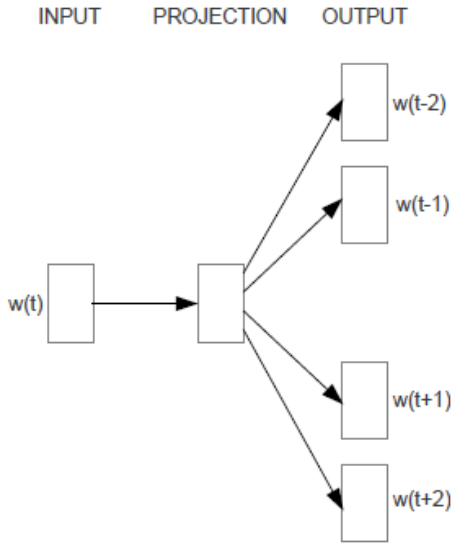
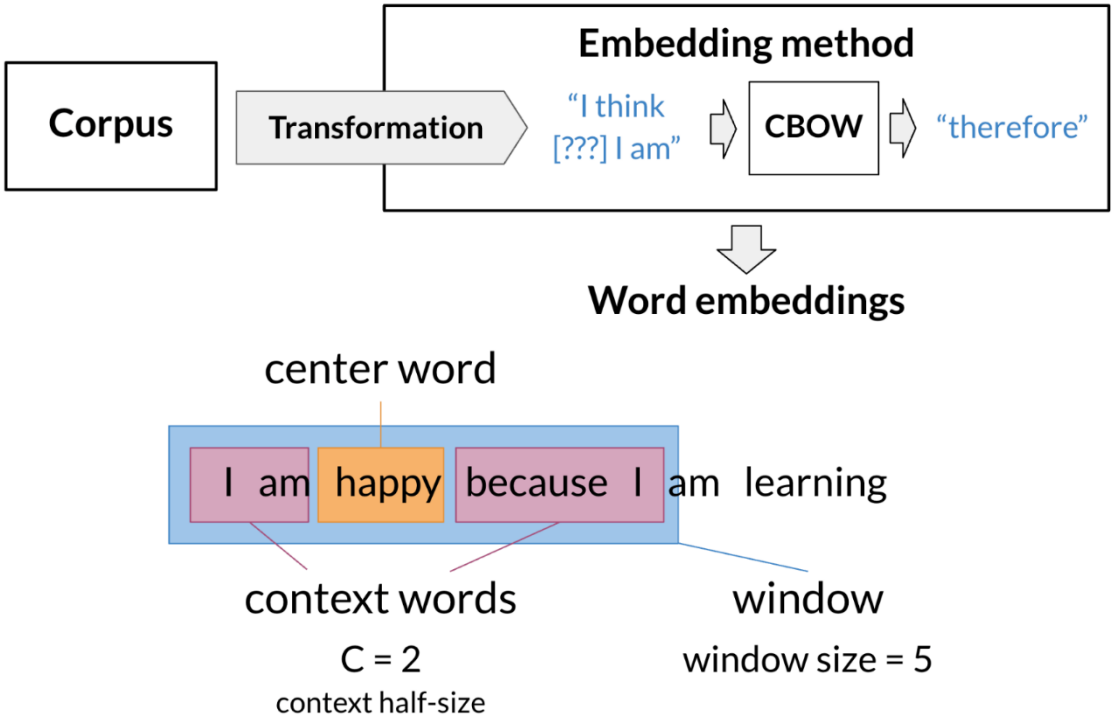


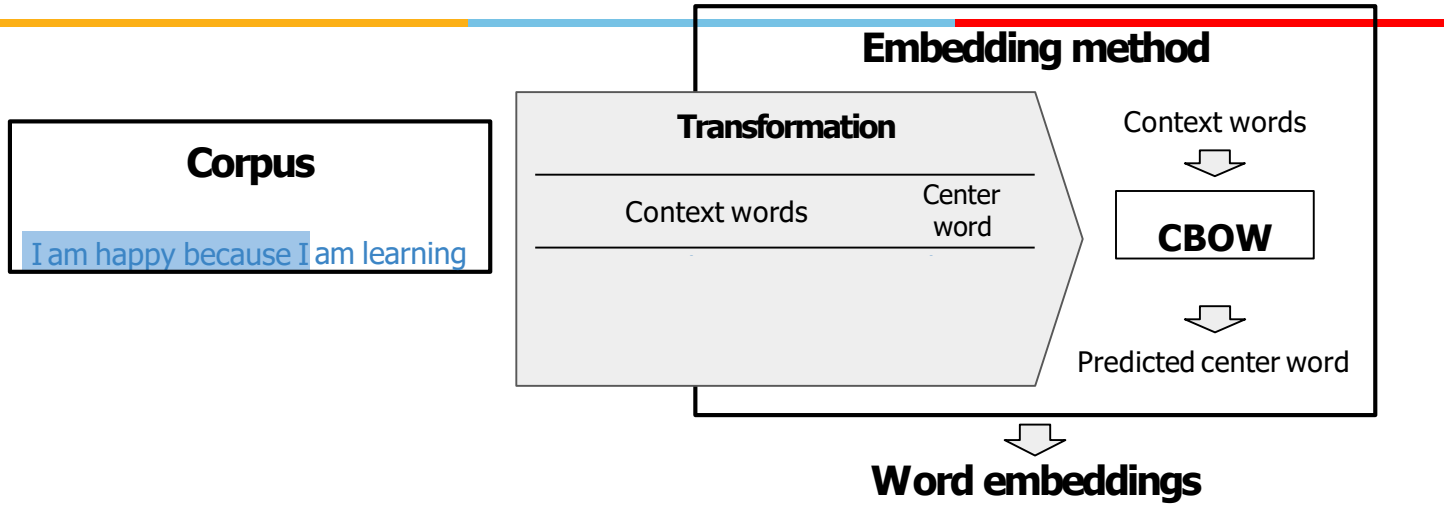
Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Original word2vec paper: https://arxiv.org/pdf/1301.3781.pdf

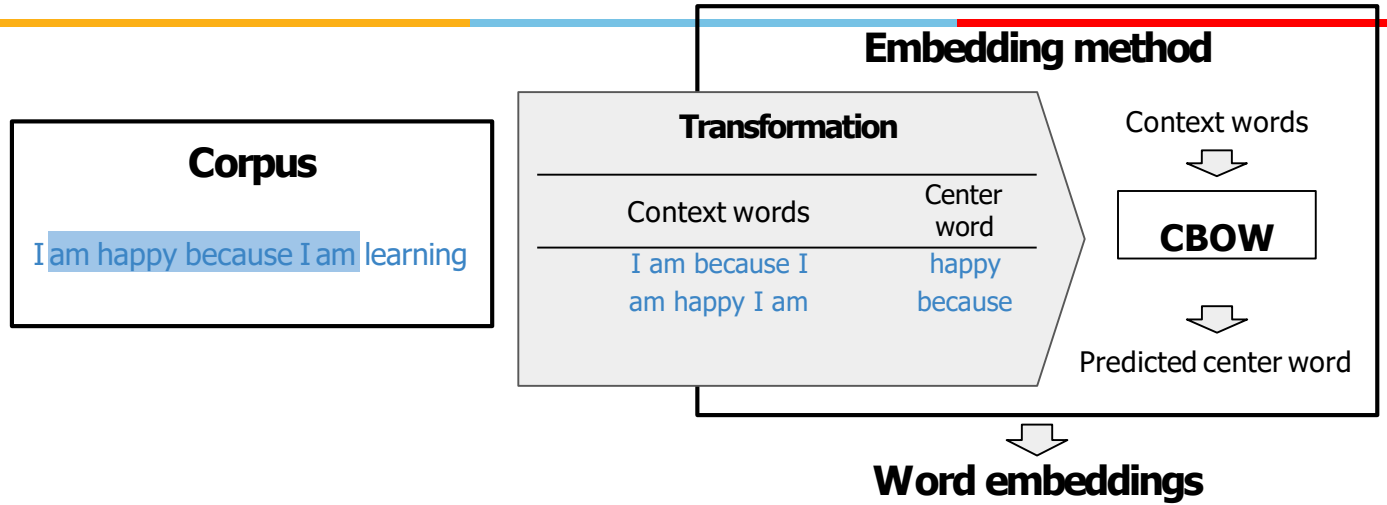
CBOW


Corpus
Transformation
Embedding method
"I think [???] I am"
CBOW
"therefore"
Word embeddings
center word
I am happy because I am learning
context words
C = 2
context half-size
window
window size = 5

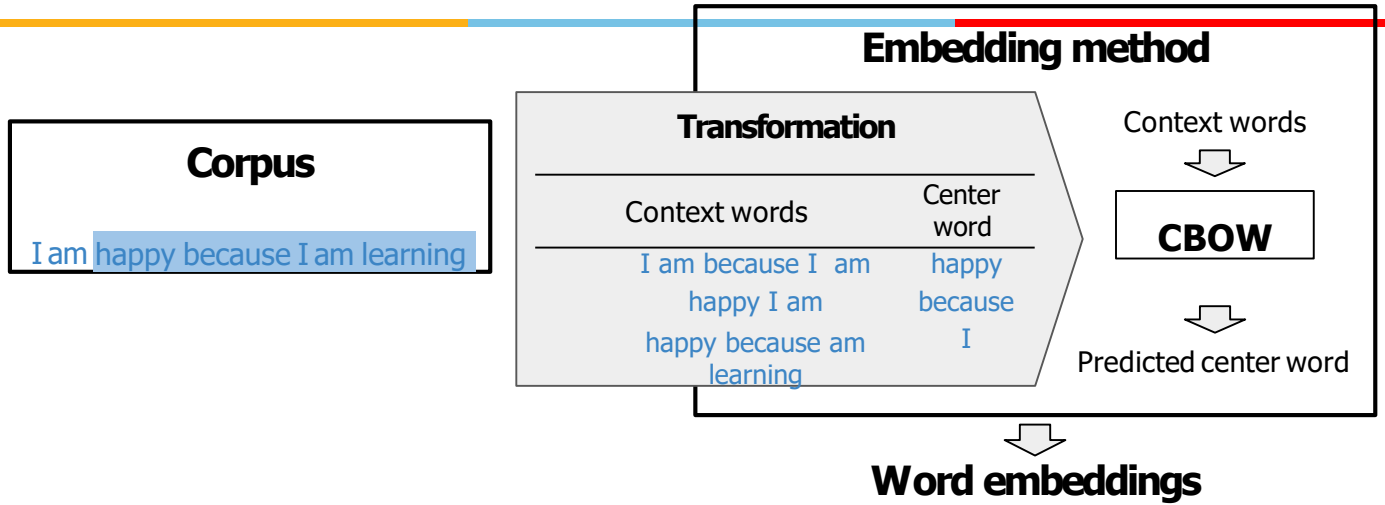
From corpus to training



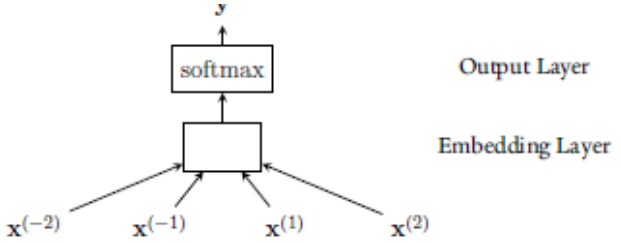
From corpus to training

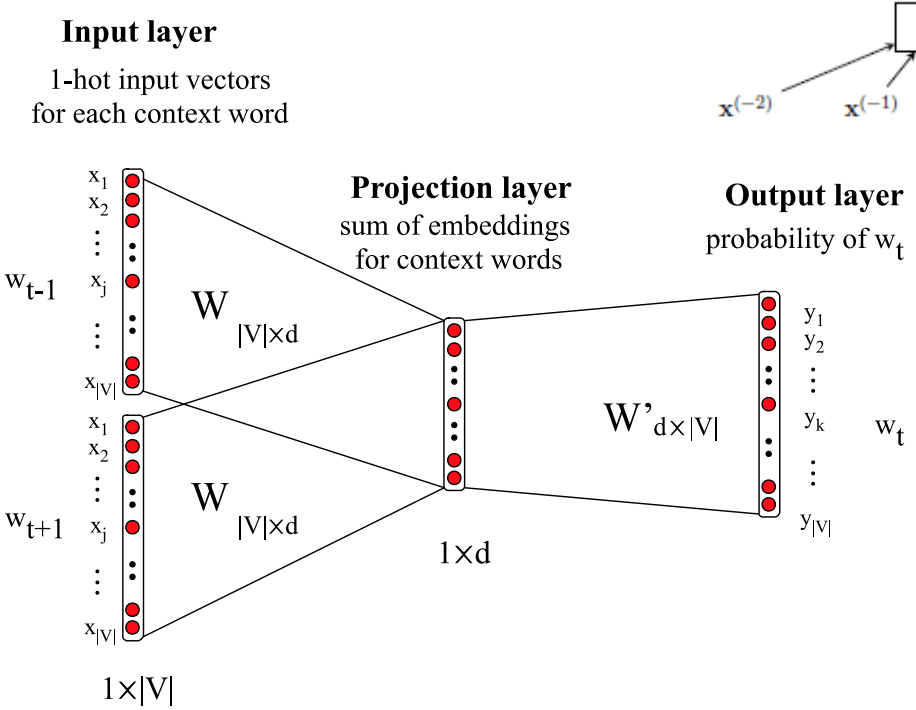


From corpus to training



CBOW (Continuous Bag of Words)





Skip gram versus CBOW

Skip-gram

– works well with a small amount of the training data,

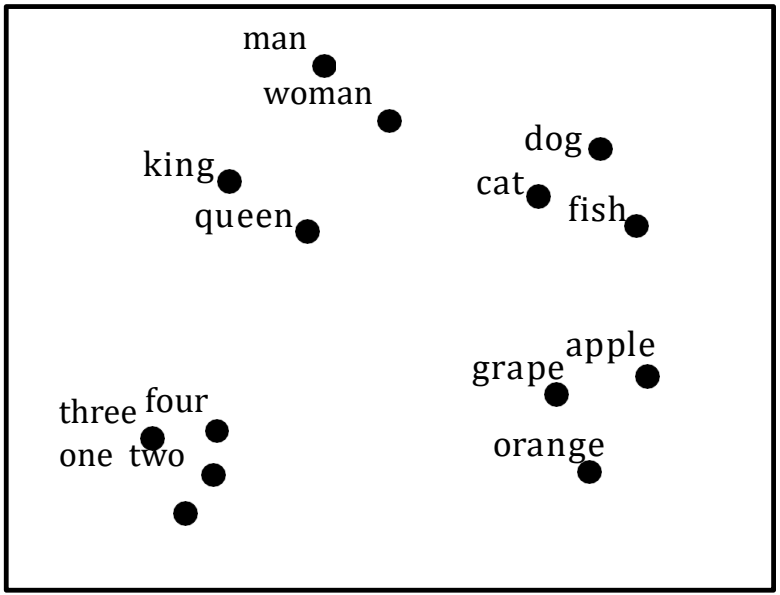
– represents well even rare words or phrases.

CBOW

– several times faster to train than the skip-gram,

– slightly better accuracy for the frequent words.

Visualizing word embeddings



[van der Maaten and Hinton., 2008. Visualizing data using t-SNE]

Analogical relations

The classic parallelogram model of analogical reasoning

To solve: *"apple is to tree as grape is to _____"*

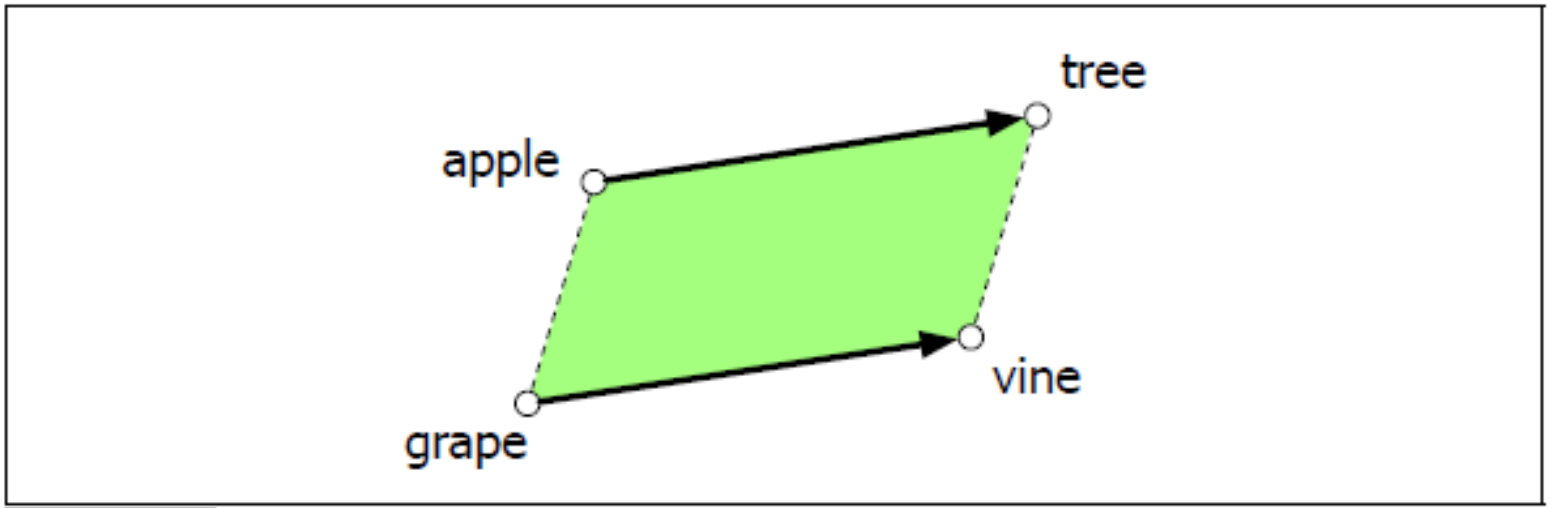


Figure 6.15 The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of $\overrightarrow{vine}$ can be found by subtracting $\overrightarrow{apple}$ from $\overrightarrow{tree}$ and adding $\overrightarrow{grape}$.

Analogical relations via parallelogram

- The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

  king – man + woman is close to queen

  Paris – France + Italy is close to Rome

- For a problem a:a*::b:b*, the parallelogram method is:

$$\hat{b}^* = \underset{x}{\text{argmax}}\ \text{distance}(x, a^* - a + b)$$

GloVe

- GloVe stands for global vectors for word representation.
- Unsupervised learning algorithm developed by Stanford for generating word embeddings
- GloVe captures both global statistics and local statistics of a corpus, in order to come up with word vectors
- Aggregates global word-word co-occurrence matrix from a corpus
- Idea is learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves.
- The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors.

Matrix of word-word co-occurrence counts

- 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

- let the matrix of word-word co-occurrence counts be denoted by X,
- whose entries X_{ij} tabulate the number of times word j occurs in the context of word i
- let $X_i = \sum_k X_{ik}$ be the number of times any word appears in the context of word i
- let $P_{ij} = P(i|j) = X_{ij}/X_i$ be the probability that word j appear in the context of word i

Intuition behind GloVe

ratio of conditional probabilities represents the word meanings

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

Very small or large:
solid is related to ice but not steam, or
gas is related to steam but not ice

close to 1:
water is highly related to ice and steam, or
fashion is not related to ice or steam.

- Given a prob word, the ratio can be small, large or equal to 1 depends on their correlations.
- For example, if the ratio is large, the probe word is related to w_i but not w_j.
- This ratio gives us hints on the relations between three different words

References

- Ch-6 : Speech and Language Processing Daniel Jurafsky and James H. Martin
- https://jalammar.github.io/illustrated-word2vec/
- http://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture02-wordvecs2.pdf
- https://arxiv.org/pdf/1301.3781.pdf
- https://nlp.stanford.edu/pubs/glove.pdf
- https://jonathan-hui.medium.com/nlp-word-embedding-glove-5e7f523999f6
- www.deeplearning.ai
- https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling

References

Word embedding

- https://www.youtube.com/watch?v=ERibwqs9p38
- https://www.youtube.com/watch?v=EsfNYiLVtHI&t=10s
- https://www.youtube.com/watch?v=lrPxo-92GC0
- https://www.coursera.org/lecture/nlp-sequence-models/
- https://www.youtube.com/watch?v=UqRCEmrv1gQ
- https://www.youtube.com/watch?v=g7wEfamF0Eg
- https://www.youtube.com/watch?v=Sho-b4-9ODE