Word and Vectors

Instructor: Tom Ko

© 1992–2010 by Pearson Education, Inc. All Rights Reserved.

Objectives

- One hot encoding
- Word vectors
- Localist representation
- Distributed representation
- Word2Vec
- Skip-gram
- Negative sampling
- Embeddings

Meaning of a word

- How do you understand the meaning of a word?
- How does the meaning of a word represent in your mind?
- When you start to learning English, if you were told that
  - “cat” should be spelled as “cad”, or
  - “cat” => 狗, “dog” =>貓
  - Does it matters ?
- What’s more important is the meaning of the word.

Lexical Semantics

- Synonym
  - words with similar meaning
- Antonym
  - words with an opposite meaning
- Lemma
  - sing is the lemma for sing, sang, sung
- Connotation:
  - Positive (happy), negative (sad)
- Word similarity
  - Cat is not a synonym of dog, but cats and dogs are certainly similar words.

Lexical Semantics

- Principle of contrast
  - A difference in linguistic form is always associated with at least some difference in meaning.

- E.g. the word H2O is used in scientific contexts and would be inappropriate in a hiking guide, water would be more appropriate.
- Probably no two words are absolutely identical in meaning.
- The word synonym is therefore commonly used to describe a relationship of approximate

Representing words as discrete symbols

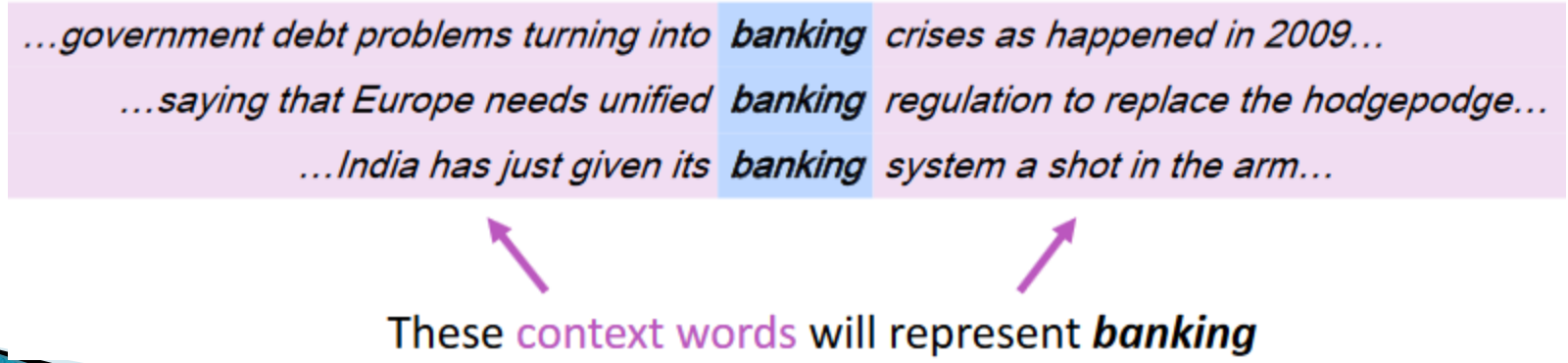
- In doing natural language processing, first we have to represent the words in the computers.
- In traditional NLP, words are represented as discrete symbols.
- We can assign an ID to each word
  - 001 for “dog”, 002 for “cat” …etc
  - This helps the computer to distinguish unique words.
- We cannot feed a word as a text string or its ID to a neural network.

One-hot encoding

- Words can be represented by one-hot vectors:
  - soccer = [0 0 0 0 0 0 0 1 0 0]
  - football= [0 0 0 1 0 0 0 0 0 0]
- One-hot means one 1, the rest 0s.
- Problems:
  - Any two vectors are orthogonal. No natural notion of similarity.
  - Vector dimension = number of words in vocabulary (e.g. half a million)
- A localist representation
- When we search for “American soccer”, we would like to match documents containing “American football”
  - We want a way to represent words which encapsulate the similarity.

Representing words by their context

- "You shall know a word by the company it keeps"(J. R. Firth 1957)
- Distributional semantics: A word's meaning is given by the words that frequently appear close-by
- 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



Understanding words by context

- Suppose you didn't know what the Cantonese word ongchoi meant, but you do see it in the following sentences or contexts:
  - (6.1) Ongchoi is delicious sauteed with garlic.
  - (6.2) Ongchoi is superb over rice.
  - (6.3) ...ongchoi leaves with salty sauces...
- And furthermore let's suppose that you had seen many of these context words occurring in contexts like:
  - (6.4) ...spinach sauteed with garlic over rice...
  - (6.5) ...chard stems and leaves are delicious...
  - (6.6) ...collard greens and other salty leafy greens
- The fact that ongchoi occurs with words like rice and garlic and delicious and salty, as do words like spinach, chard, and collard greens might suggest to the reader that ongchoi is a leafy green similar to these other leafy greens.

Vector semantics

- Defining the meaning of a word w as a vector, a list of numbers, a point in N dimensional space. (Osgood et al., 1957)
- The numbers are based in some way on counts of neighboring words.

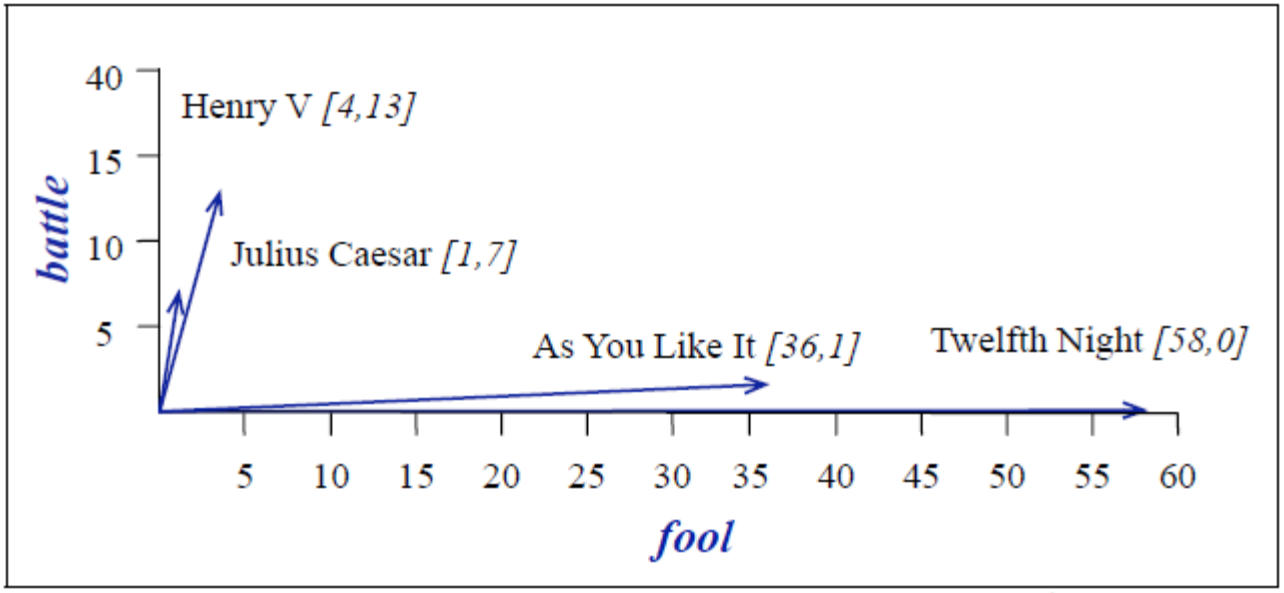
Properties of embeddings

- vector('king') – vector('man') + vector('woman') = vector(' ? ')
- vector('Paris') – vector('France') + vector('Italy') = vector(' ? ')

Document vectors

- The following table shows a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare.
- The word fool appeared 58 times in Twelfth Night

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3



A query system

- It is the task of finding the document d from the D retrieval documents in some collection that best matches a query q.
- A query is represented by a vector
- Need a way to compare two vectors to find how similar they are.
  - Cosine similarity

Cosine for measuring similarity

- The dot product

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

$$= |\vec{v}||\vec{w}|\cos\theta$$

- A problem: it favors long vectors. The raw dot product thus will be higher for frequent words. We need a similarity metric that tells us how similar two words are regardless of their frequency.
  - Normalized dot product

TF-IDF: Weighing terms in the vector

- Observations: Words like “the”, “it”, or “they”, which occur frequently with all sorts of words, aren’t informative.

- The tf-idf (stands for term frequency and inverse document frequency) algorithm captures two intuitions:
  - Term frequency: a word appearing 100 times in a document doesn’t make that word 100 times more likely to be relevant to the meaning of the document. So we down weight the raw frequency by log10.

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

  - Document frequency: to give a higher weight to words that occur only in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection.

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

TF-IDF: Weighing terms in the vector

- The tf-idf weighting of the value for word t in document d:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

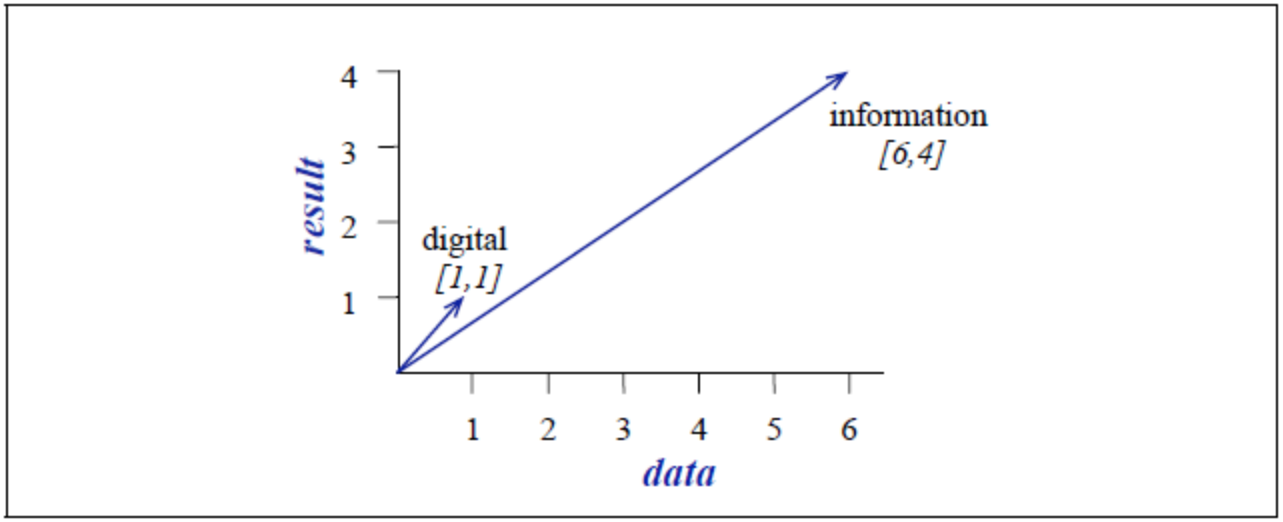
TF-IDF

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Word vectors

sugar, a sliced lemon, a tablespoonful of **apricot** jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	...	computer	data	pinch	result	sugar
apricot	0	...	0	0	1	0	1
pineapple	0	...	0	0	1	0	1
digital	0	...	2	1	0	1	0
information	0	...	1	6	0	4	0



Word vectors

- Word vectors are also called word embeddings or word representations.
- They are a distributed representation.
- The word vectors computed from co-occurrence matrix are usually long and sparse.
  - Long: the length of the vector, which is the size of the vocabulary, is usually between 10k and 50k
  - Sparse: most of the dimensions are zeros

$$\text{Football} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \text{ or } \begin{pmatrix} 0 \\ 79 \\ 0 \\ 0 \\ 0 \\ 0 \\ 34 \\ 0 \end{pmatrix}$$

Dense vs. sparse vectors

- It turns out that dense vectors work better in every NLP task than sparse vectors.
- Possible reasons are
  - Classifiers can have much less parameters
  - Avoid overfitting
  - dense vectors may do a better job of capturing similarity than sparse vectors.
- We would like to look for a way to learn short and dense word embeddings automatically

Localist representation

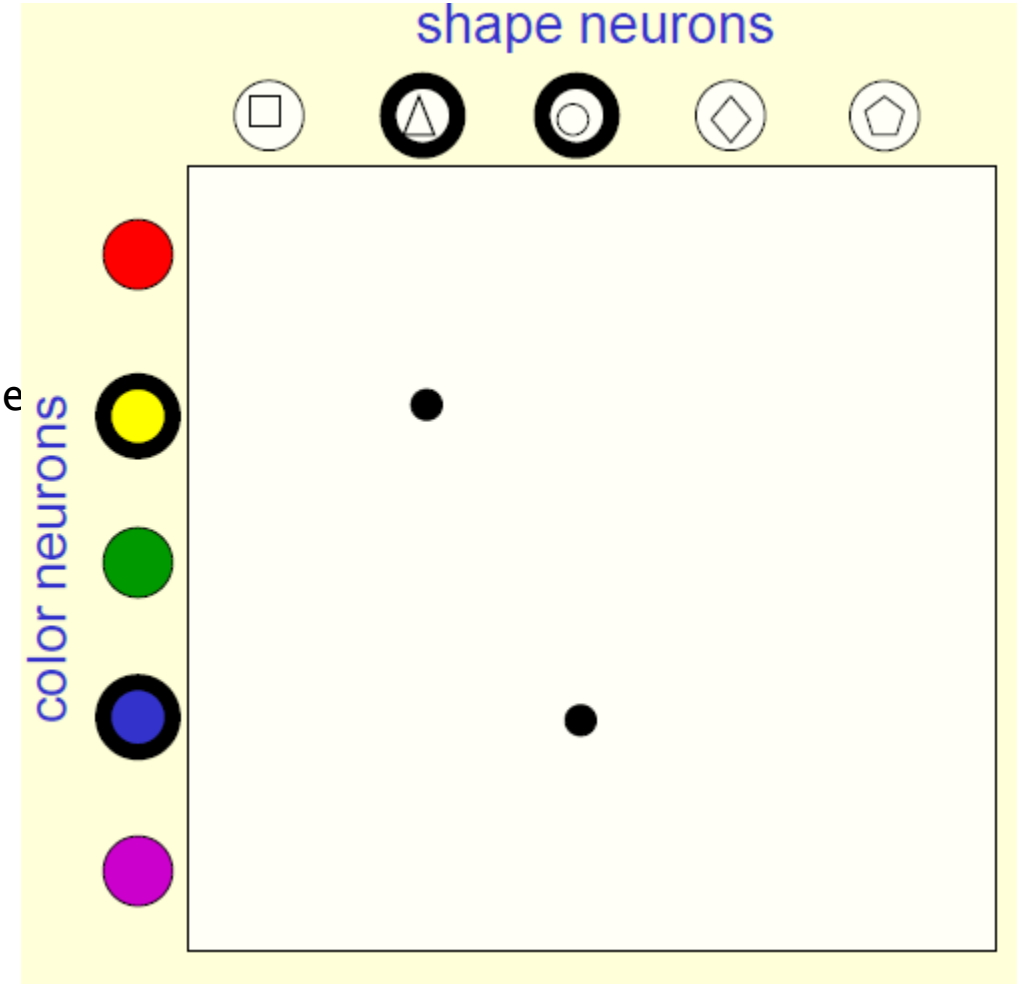
- The simplest way to represent things with neural networks is to dedicate one neuron to each thing.
- One neuron for each shape: square, triangle, circle
  - If we want to represent a circle and a triangle, we activate both neurons
- Easy to understand
- Very inefficient whenever the data has componential structure.

Componential structure

- For example, each object has many properties like shape, color, size.
- One neuron for each shape, color and size
- How to represent a big blue circle?
  - Activate the neurons for big, blue and circle
- How to represent a yellow triangle and a blue circle at the same time?

Componential structure

- Confusion
- Create another neuron specially for a big yellow triangle
  - Can't represent unseen combination
  - Total number of neurons will be huge: #shapes * #size * #color



Distributed representation

- Each neuron must represent something, so this must be a local representation.
- "Distributed representation" means a many-to-many relationship between two types of representation (such as concepts and neurons).
  - Each concept is represented by many neurons
  - Each neuron participates in the representation of many concepts

Measuring vector similarity

- The cosine and the dot product
- Dot product favors long vectors
  - $\boldsymbol{w} \cdot \boldsymbol{x} = |\boldsymbol{w}|\ |\boldsymbol{x}|\ \cos\theta$

- The cosine of the angle between the two vectors, which is the normalized dot product, is the most common similarity metric.

Word2Vec

- We want to represent words with short and dense vector which encapsulate similarity among words.
- Can they be learned automatically ?
- Word2vec (Mikolovet al. 2013) is a framework for learning word vectors

The basic idea

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position 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

Objective function

- to minimize the objective function:

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

- How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

Prediction function

- We will use two vectors per word w:
  - v_w when w is a center word
  - u_w when w is a context word
- For a center word c and a context word o:

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

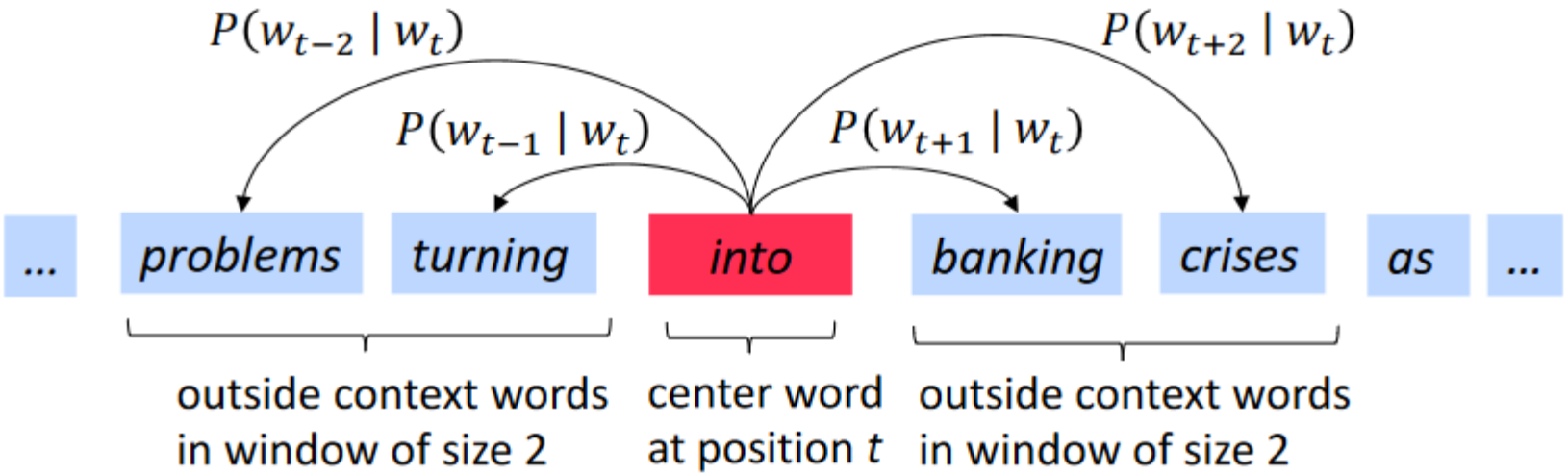
② Exponentiation makes anything positive

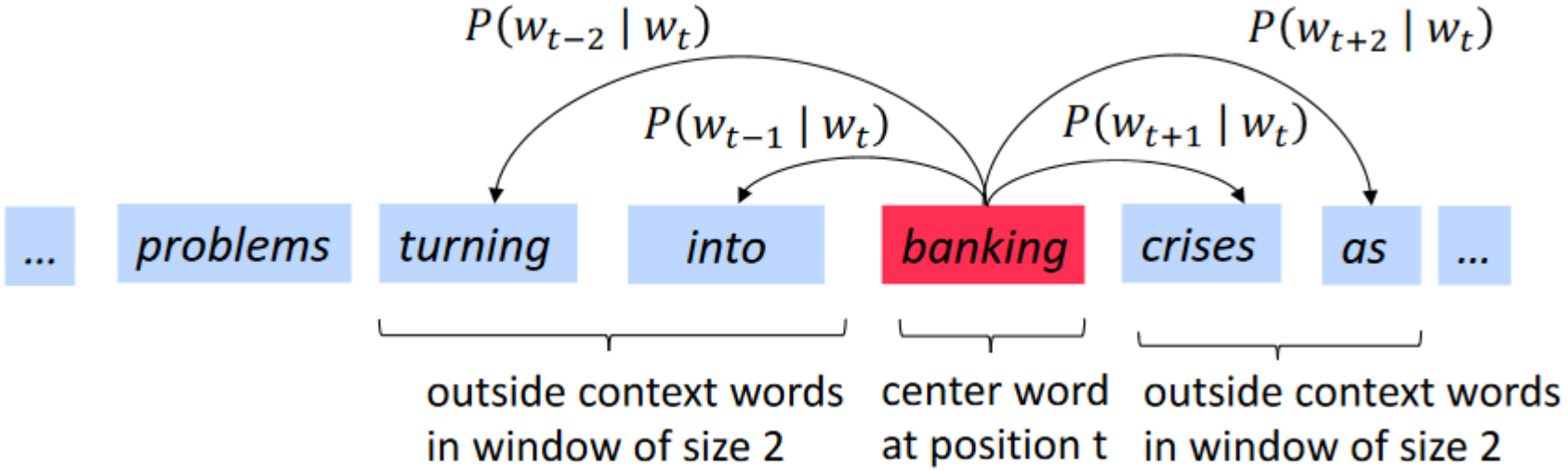
① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

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

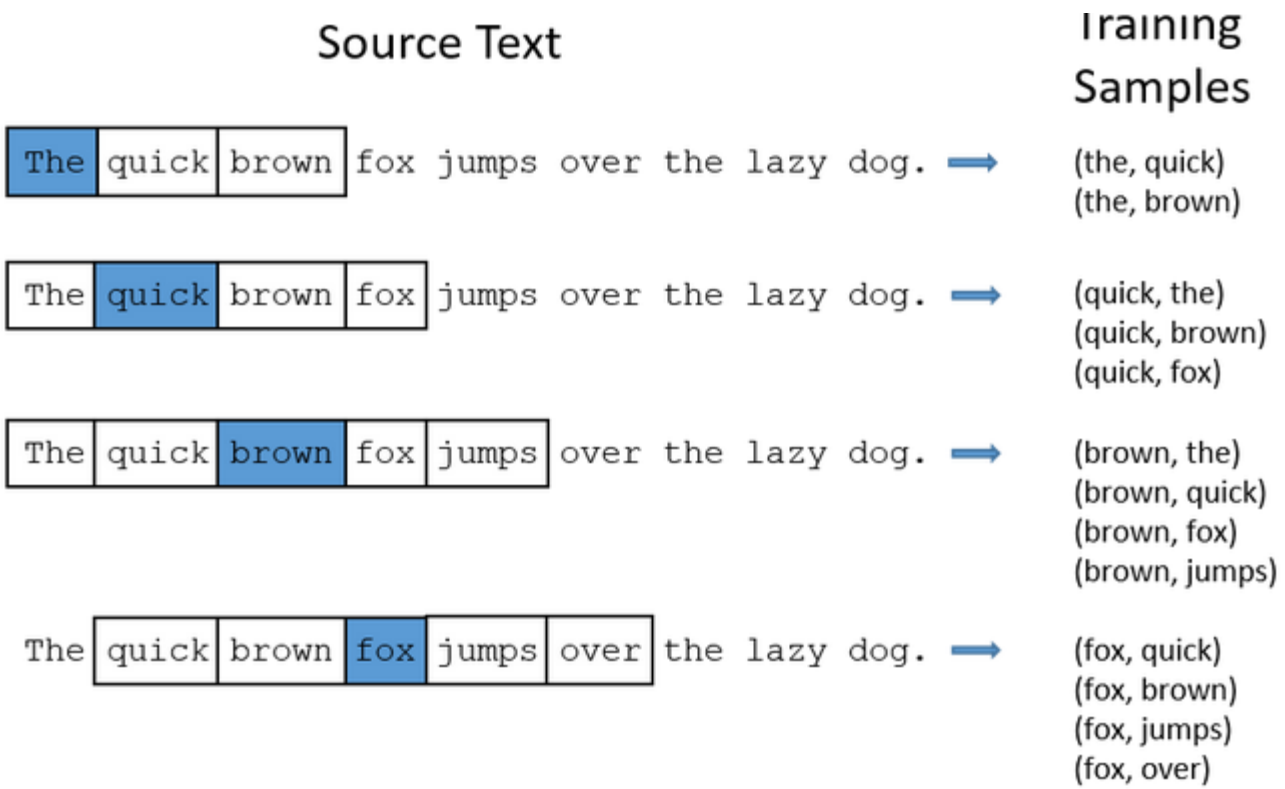
③ Normalize over entire vocabulary to give probability distribution

The basic idea



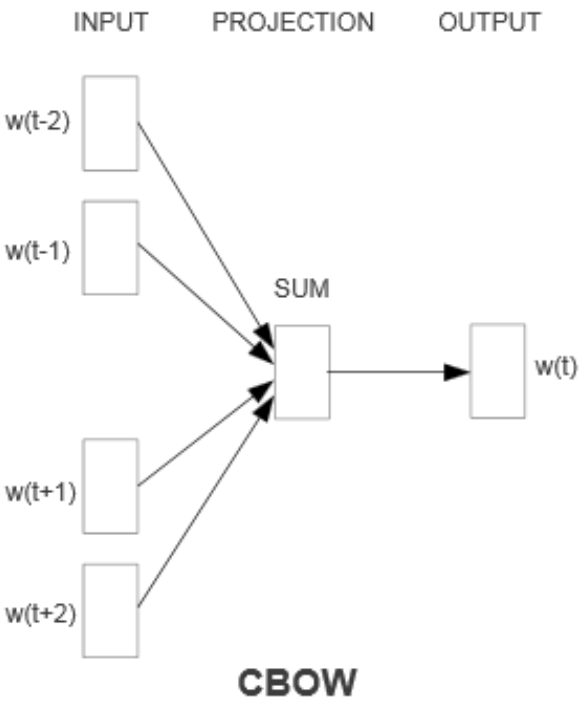


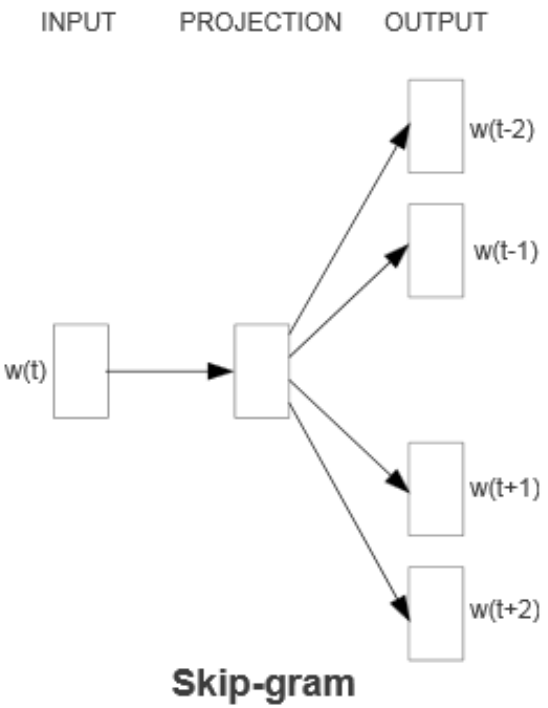
Training of Word2Vec



Word2Vec models

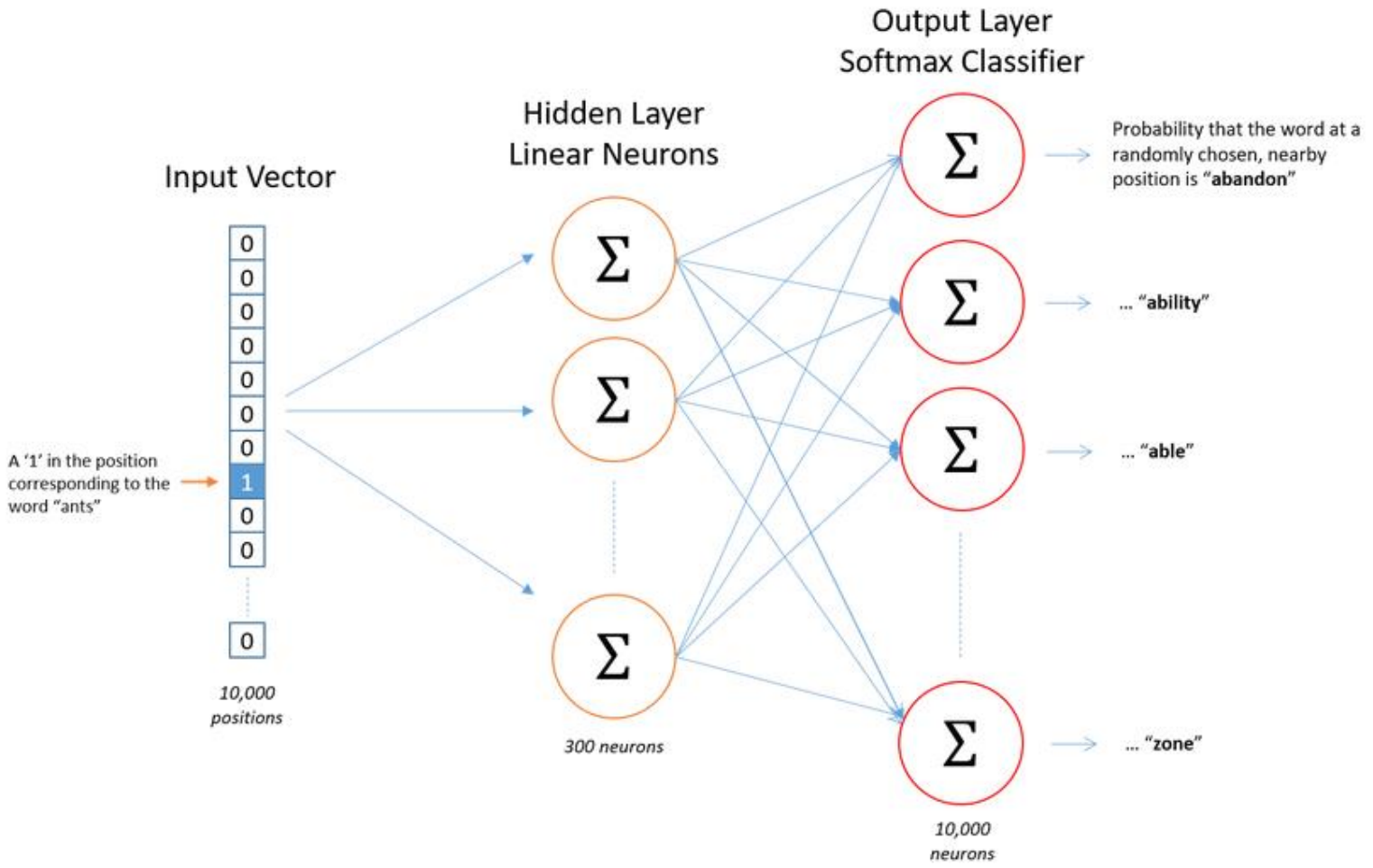
- The word2vec model can be thought of as a simplified neural network model with one hidden layer and no non-linear activation.
- Skip-grams (SG)
  - Predict context ("outside") words (position independent) given center word
- Continuous Bag of Words (CBOW)
  - Predict center word from (bag of) context words



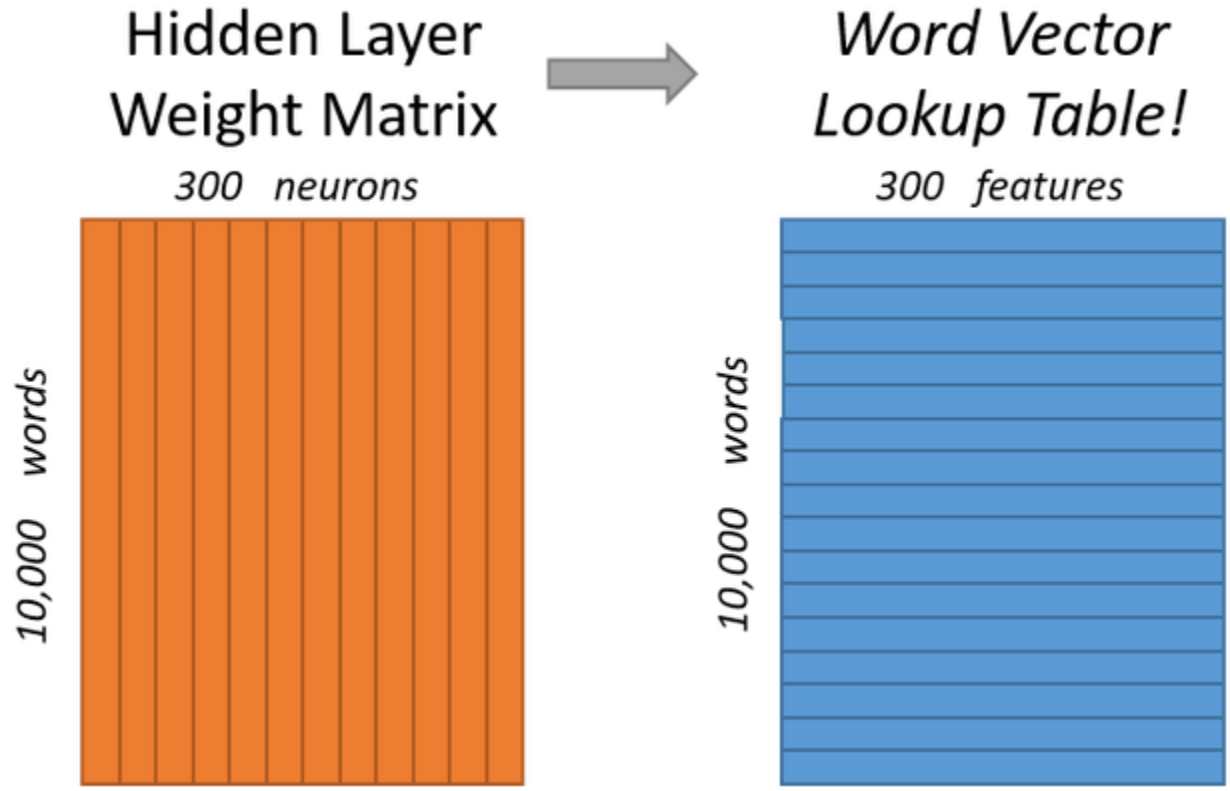


Skip-gram model

- Unlike the traditional N-gram, the words in skip-gram need not to be consecutive. That's why it is called skip-gram.



Layer weights as word vectors



$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Intuition of Word2Vec

- If two different words have very similar “contexts” (that is, what words are likely to appear around them), then our model needs to output very similar results for these two words.
- And one way for the network to output similar context predictions for these two words is if *the word vectors are similar*.
- So, if two words have similar contexts, then our network is motivated to learn similar word vectors for these two words

The use of Word2Vec

- We're going to train a simple neural network with a single hidden layer to perform a certain task, but then we're not actually going to use that neural network for the task we trained it on! Instead, the goal is actually just to learn the weights of the hidden layer–we'll see that these weights are actually the "word vectors" that we're trying to learn.
- Two vectors for each word, one represents center word, one represents nearby word
  - You can take the average of the two.

Two representations for each word

- Two vectors for each word, one representation when it is a center word, one representation when it is a context word
  - You can take the average of the two. (A common way)
  - Summing or concatenate them
- Consider the case where both the word dog and the context dog share the same vector v. Words hardly appear in the contexts of themselves, and so the model should assign a low probability to p(dog|dog), which entails assigning a low value to $v \cdot v$ which is impossible.

Practical issues of Word2Vec

- In the example I gave, we had word vectors with 300 components, and a vocabulary of 10,000 words. Recall that the neural network had two weight matrices–a hidden layer and output layer. Both of these layers would have a weight matrix with 300 x 10,000 = 3 million weights each!
- Running gradient descent on a neural network that large is going to be slow. And to make matters worse, you need a huge amount of training data in order to tune that many weights and avoid over-fitting. millions of weights times billions of training samples means that training this model is going to be a beast.
- Solutions:
  - Subsampling frequent words
  - Negative Sampling

Subsampling Frequent Words

- There are two “problems” with common words like “the”:
  - When looking at word pairs, (“fox”, “the”) doesn’t tell us much about the meaning of “fox”. “the” appears in the context of pretty much every word.
  - We will have many more samples of (“the”, …) than we need to learn a good vector for “the”.
- For each word we encounter in our training text, there is a chance that we will effectively delete it from the text. The probability that we cut the word is related to the word’s frequency.

Negative Sampling

- Skip-gram neural network has a tremendous number of weights, all of which would be updated slightly by every one of our billions of training samples!
- Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them.
- With negative sampling, we are instead going to randomly select just a small number of “negative” words (let’s say 5) to update the weights for. (In this context, a “negative” word is one for which we want the network to output a 0 for). We will also still update the weights for our “positive” word (which is the word “quick” in our current example).

Negative Sampling

- Recall that the output layer of our model has a weight matrix that's 300 x 10,000. So we will just be updating the weights for our positive word ("quick"), plus the weights for 5 other words that we want to output 0. That's a total of 6 output neurons, and 1,800 weight values total. That's only 0.06% of the 3M weights in the output layer!
- In the hidden layer, only the weights for the input word are updated (this is true whether you're using Negative Sampling or not).

Reading list

- Speech and Language Processing, version 3
  - Chapter 6
- http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/