Recursive Neural Networks for NLP

Spring 2020

Natural Language Processing

Understand natural language (e.g., English, Mandarin, Hindi) to perform useful tasks

Example tasks

- Sentiment analysis
- Language translation
  - Google Translate, Microsoft Translator, …
- Question answering
  - Cortana, Google Assistant, Siri, …

Major successes of deep learning

Rest of the Course

- Word embedding
  - Representing words succinctly while preserving “semantic distance”
- Neural language modeling (uses word embedding)
  - Prior approach (n-gram model)
  - RNN (basics)
  - LSTM (if there is time)
- Gender bias in word embedding and RNN

- + Missed lecture (real world adversarial examples)
- + Course review (or suggested topic)

Word Embedding

How to represent words?

First idea: integer, n^{th} word represented as integer n

Weaknesses

- Difficult for neural networks to extract useful information
  - Closer two words are in ordering, the more difficult for the network it is to tell them apart.
  - No relationship between adjacent/nearby words.
- Other weaknesses?

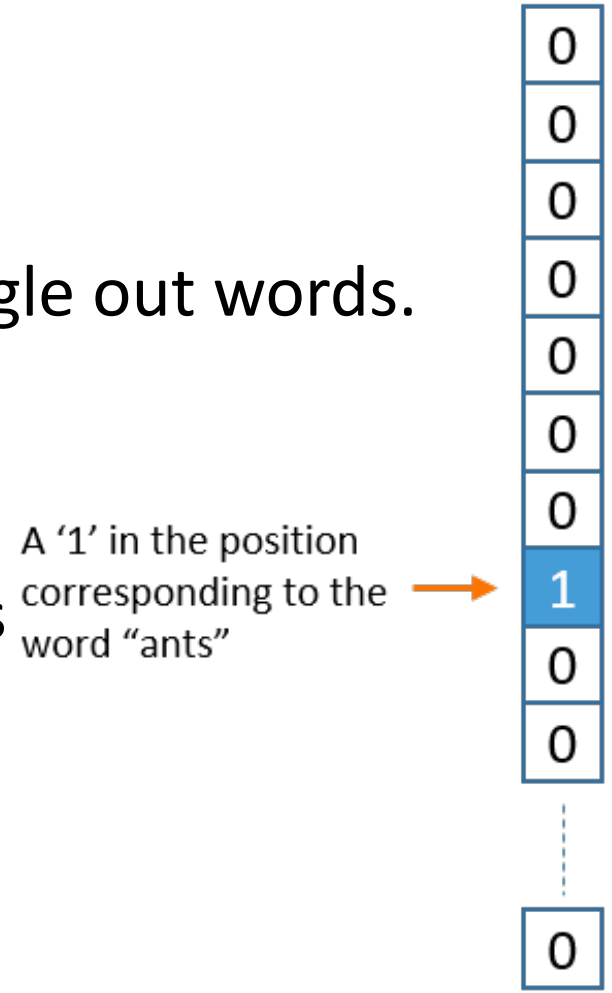
How to represent words?

Second idea: one-hot encoding

- Recall class/label encoding in MNIST, etc.
- Each word is separate dimension, easy to single out words.

Weaknesses

- Does not capture “similarity” between words (e.g., “motel” and “hotel”)
- Other weaknesses?



How to represent words?

- Insight: “You shall know a word by the company it keeps” - J. R. Wirth
- The context of a word is the set of words that appear nearby (within a fixed size window)
- Use the contexts of a word w to build up its representation

…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…*

How to represent words?

- Third idea: word embeddings (or word vectors)

- A dense vector for each word such that vectors of words that appear in similar contexts are similar

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

Popular word embedding: Word2vec

- Papers from Mikolov et al. (Google)
  - Efficient Estimation of Word Representations in Vector Space
  - Distributed Representations of Words and Phrases and their Compositionality

- Will focus on Word2vec Skip-gram model

Word2vec approach

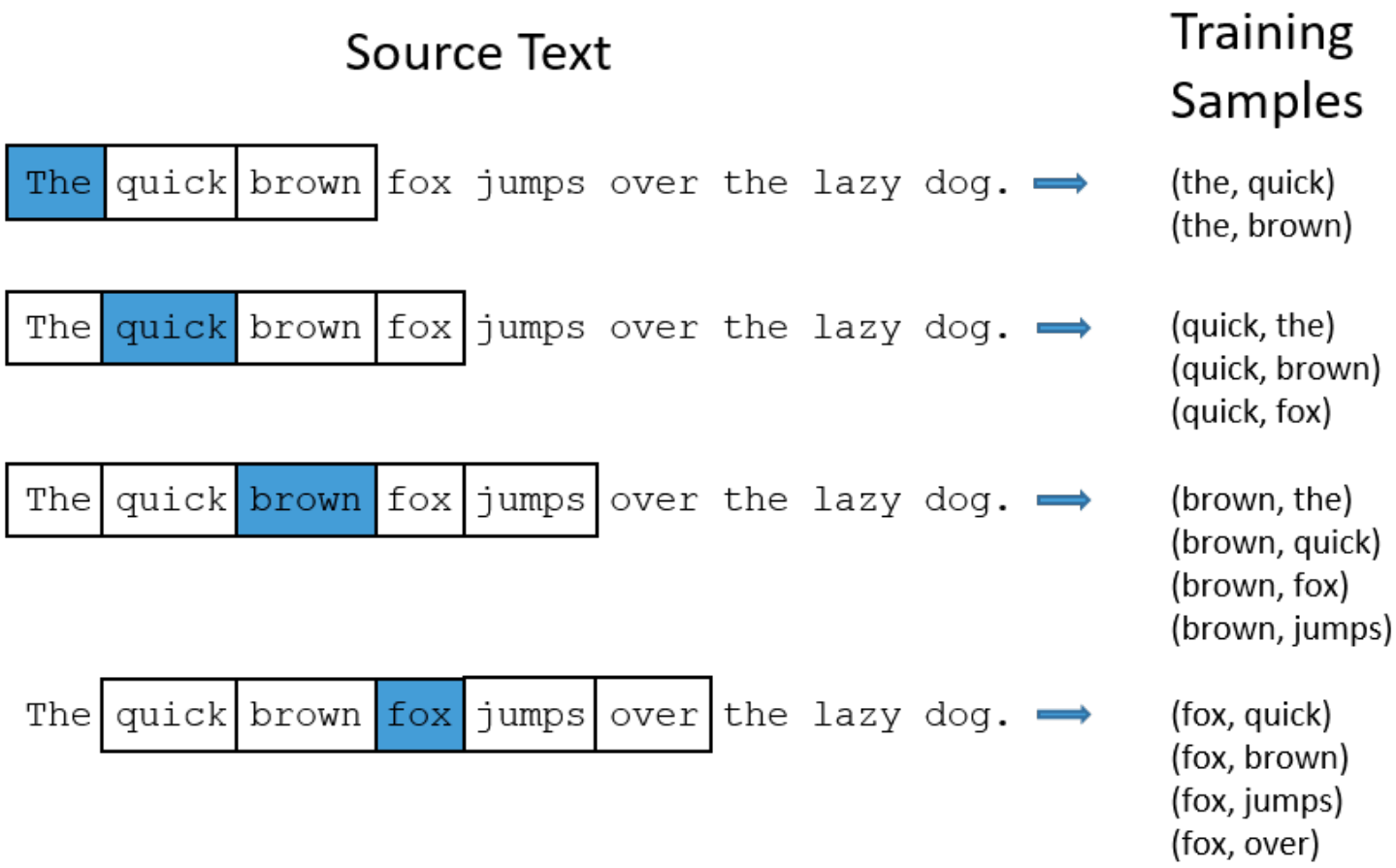
- Train neural network with single hidden layer to perform a specific task
- Weights of the hidden layer give us the word embeddings

- Seems familiar?
  - Recall autoencoders

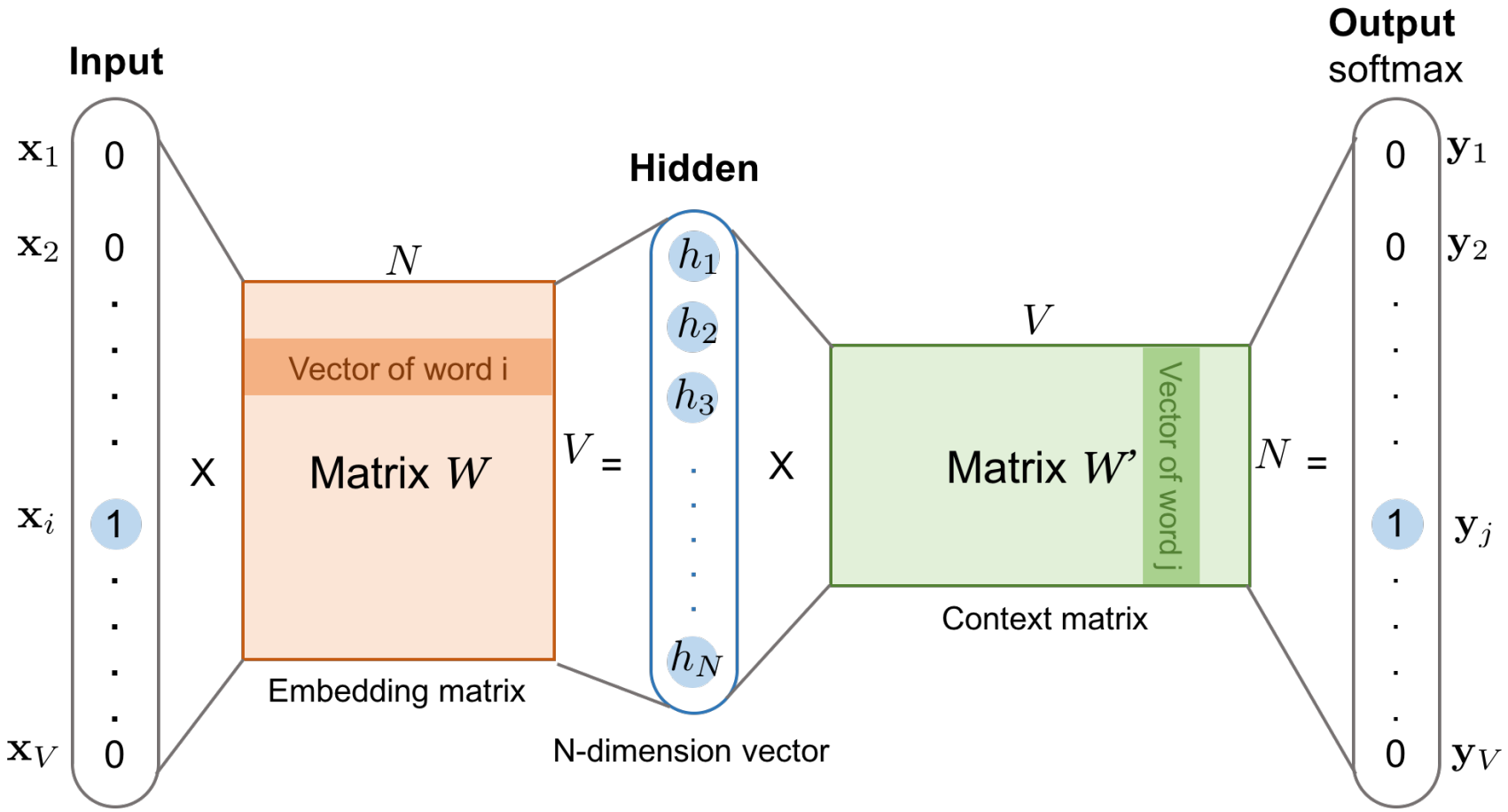
Word2vec Skip-gram task

- Given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random.
- The network is going to tell us the probability for every word in our vocabulary of being the “nearby word” that we chose.
- “Nearby” words: A typical window size might be 2, meaning 2 words behind and 2 words ahead (4 in total).
- Example: If input word “Soviet”, the output probabilities are going to be much higher for words like “Union” and “Russia” than for unrelated words like “watermelon” and “kangaroo”.

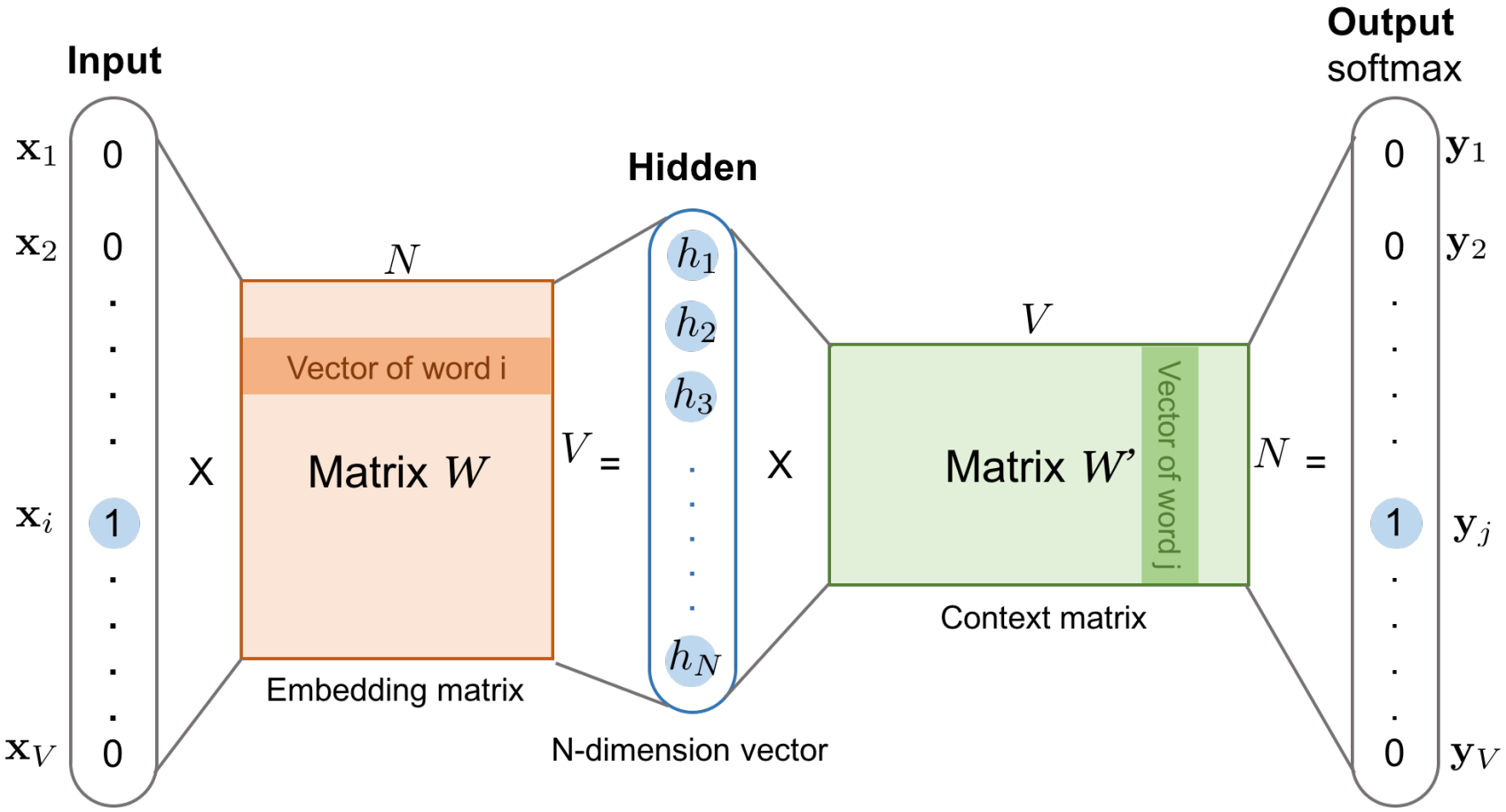
Training samples



Model


Input
Output
softmax
Hidden
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_i$
$\mathbf{x}_V$
0
1
X
N
Vector of word i
Matrix W
Embedding matrix
V =
h_1
h_2
h_3
h_N
N-dimension vector
X
V
Matrix W'
Vector of word j
Context matrix
N =
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_j$
$\mathbf{y}_V$

Embedding matrix for hidden layer



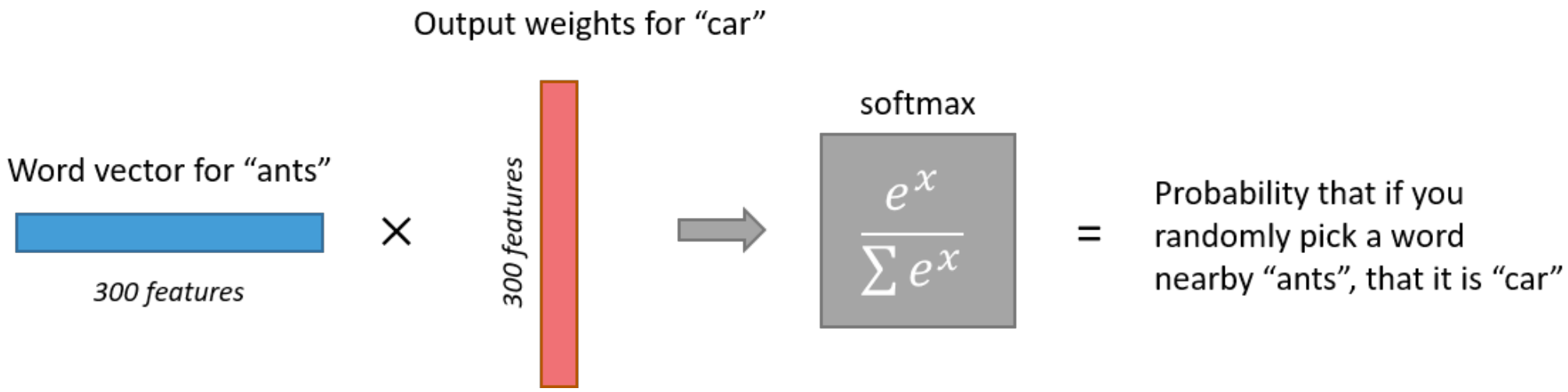
- Embedding matrix is 10,000 x 300

Embedding matrix: lookup table for word vectors

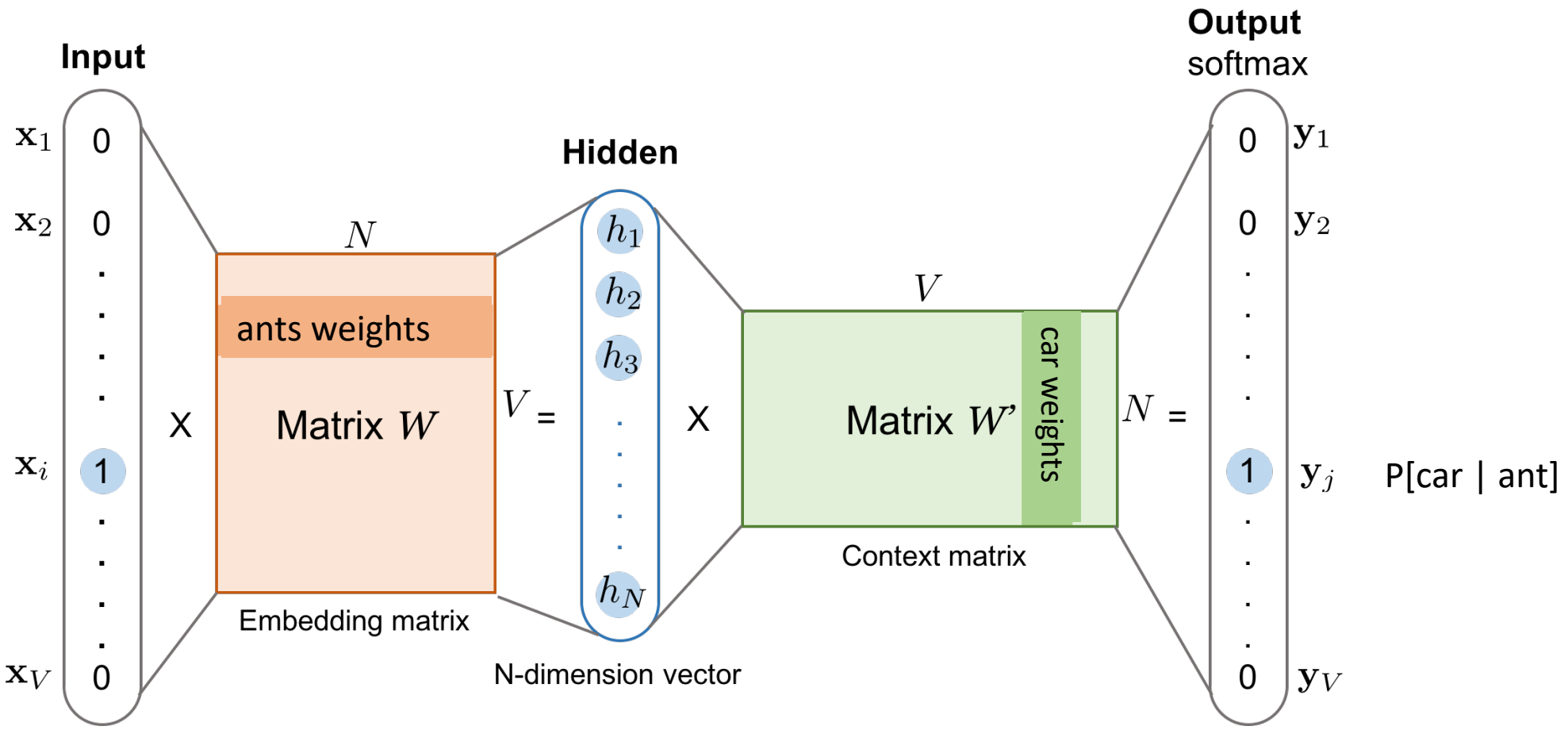
$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

- Each one-hot encoding selects a row of the matrix (its word vector)

Output layer slice



Output layer slice



Note

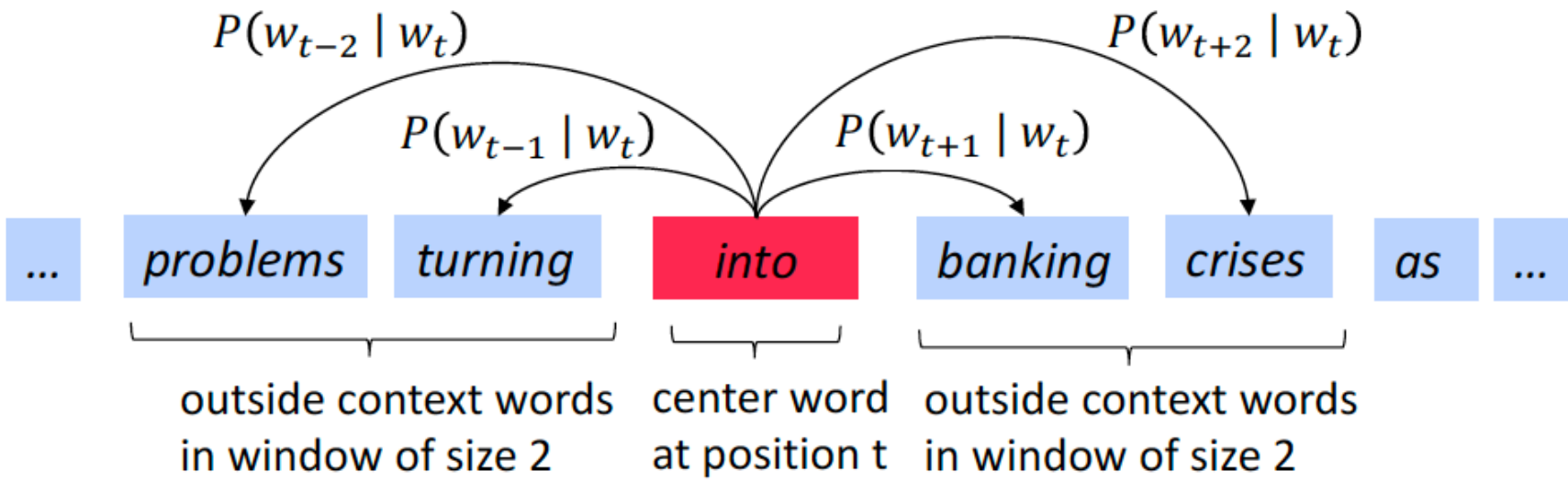
If two words have similar contexts, then the network is motivated to learn similar word vectors for these two words

Examples

- “smart”, “intelligent”
- “ant”, “ants”

Word2Vec overview

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



Word2Vec: toward objective function

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

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

Example m=1: Problems turning into banking crises as …

P(turning | problems)	(center is problems)
* P(problems | turning) * P(into | turning)	(center is turning)
* P(turning | into) * P(banking | into)	(center is into)
* P(into | banking) * P(crises | banking)	(center is banking)
* P(banking |crises) * P(as | crises)	(center is crises)
* …	

Word2Vec: objective function

- Objective function is negative log likelihood

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

- Objective now looks like the familiar batchable per-instance loss

-log P(turning | problems)	(center is problems)
-log P(problems | turning)	(center is turning)
-log P(into | turning)	(center is turning)
-log P(turning | into)	(center is into)
-log P(banking | into)	(center is into)
-log P(into | banking)	(center is banking)
-log P(crises | banking)	(center is banking)
-log P(banking |crises)	(center is crises)

Word2Vec: objective function

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*:

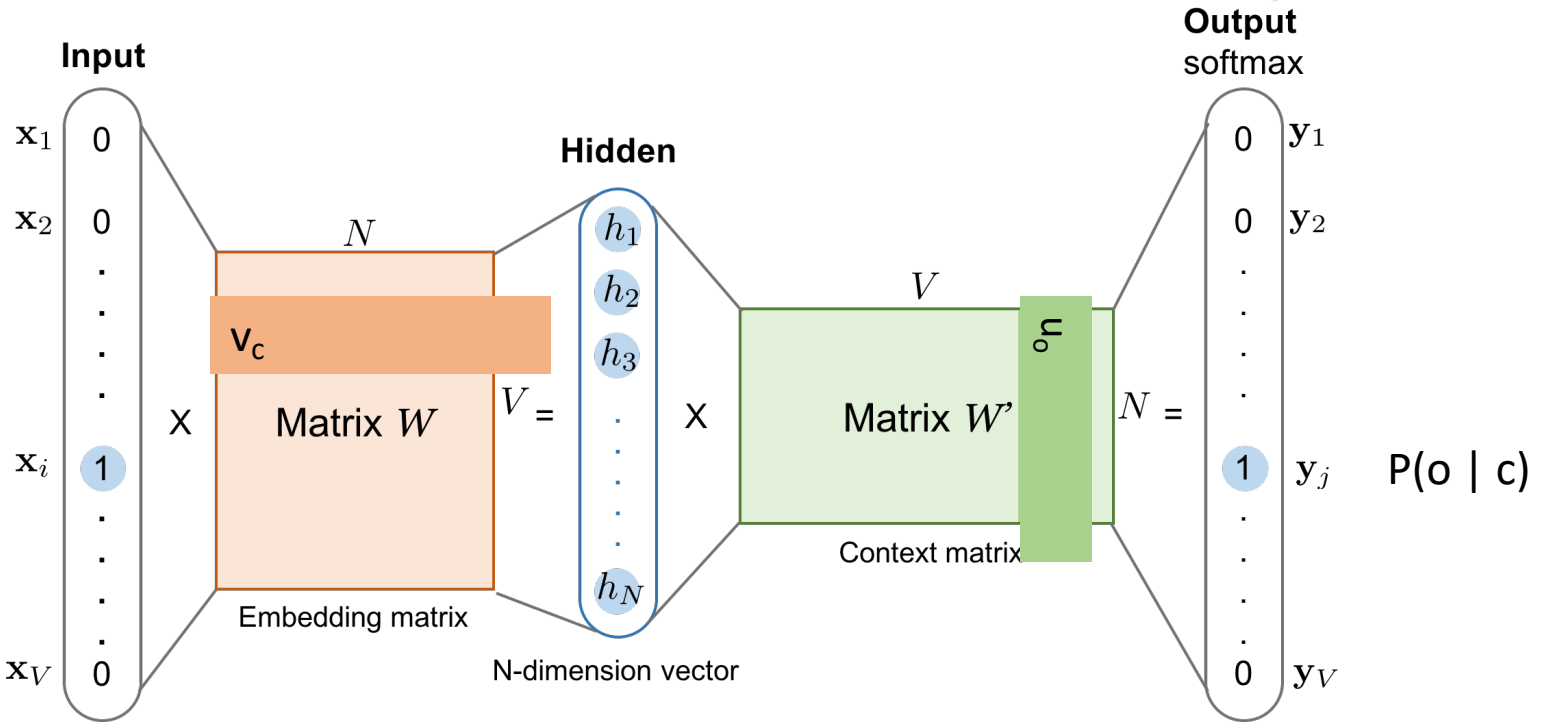
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Word2Vec: prediction function

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

Dot product compares similarity of *o* and *c*.
Larger dot product = larger probability

After taking exponent,
normalize over entire vocabulary



Word2Vec: train model using SGD

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

Remember: every word has two vectors

We then optimize these parameters

Scalability is a challenge

- With 300 features and a vocab of 10,000 words, that's 3M weights in the hidden layer and output layer each!

- Two techniques in Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality
  - Subsampling frequent words
  - Negative sampling

Subsampling frequent words

- There are two “problems” with common words like “the”:
  1. 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.
  2. We will have many more samples of (“the”, …) than we need to learn a good vector for “the”.

Subsampling frequent words

w_i is the word
$z(w_i)$ is the fraction of the total words in the corpus that are that word

Probability of keeping word w_i

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

- $P(wi)=1$ (100% chance of being kept) when $z(wi)<=0.0026$
- $P(wi)=0.5$ (50% chance of being kept) when $z(wi)=0.00746$

Negative sampling

- Scalability challenge
  - For each training sample, update all weights in output layer
  - 3M weights in our running example!

- Negative sampling
  - For each training sample, update only a small number of weights in output layer
  - Weights for the correct output word (300 weights) + 5 randomly selected “negative words” for whom the output should be 0 (5x 300 weights)

Negative sampling

- Negative samples are chosen according to their empirical frequency

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4} \right)}$$

Negative sampling: objective function

$$J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$$

- Maximize probability that real words appear around center word; and
- Minimize probability that random words appear around center word

Word embeddings capture relationships

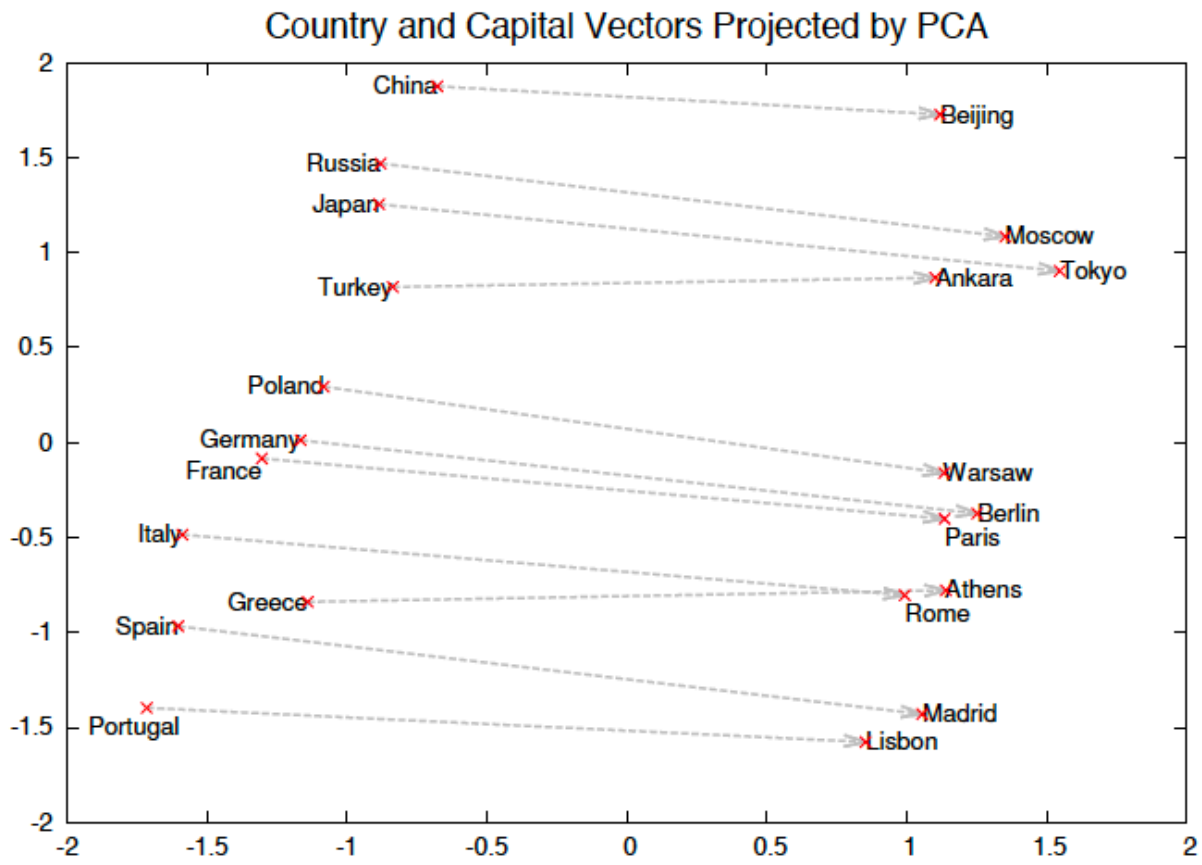


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Additive compositionality

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

Rest of the Course

- Word embedding
  - Representing words succinctly while preserving “semantic distance”
- Neural language modeling (uses word embedding)
  - Prior approach (n-gram model)
  - RNN (basics)
  - LSTM (if there is time)
- Gender bias in word embedding and RNN

- + Missed lecture (real world adversarial examples)
- + Course review (or suggested topic)

Acknowledgments

- Word2Vec tutorial by Chris McCormick
- Learning Word Embedding by Lilian Weng
- Stanford cs224n lecture notes on word vectors
- Spring 2019 course