Security and Fairness of Deep Learning

Recursive Neural Networks for NLP

Anupam Datta
CMU

Spring 2019

Announcement

- Class participation update
  - Pre-midterm: 2/5 points for presence in class instead of 1/5
  - Post midterm:
    - 3 points based on in-class quiz on Thur 4/18 covering adversarial attacks (Nicholas Carlini lecture 3/19) + word embeddings (4/9, 4/16).
    - 2/3 of these points is extra credit to recover points for pre-midterm

Goal

Understand Recurrent Neural Networks (RNNs) using Natural Language Processing (NLP) tasks as motivation

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

Outline for this module

- Word embedding
  - Representing words succinctly while preserving “semantic distance”
- Neural language modeling (uses word embedding)
  - RNN (basics), LSTM
- Neural machine translation (uses neural language modeling)
  - Sequence-to-sequence models, attention

- Gender bias in word embedding
- Explanations for and bias in RNNs

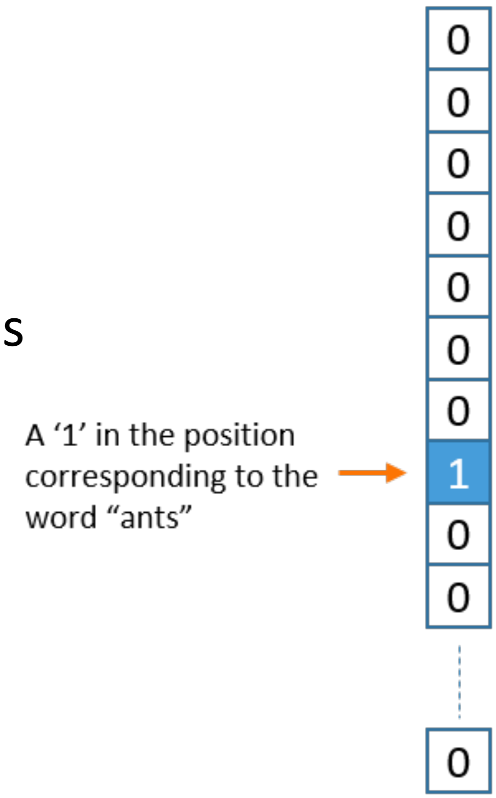
Word Embedding

How to represent words?

First idea: one-hot encoding

Weakness

- Does not capture "similarity" between words (e.g., "motel" and "hotel")



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?

- Second 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

Word2vec Skip-gram task

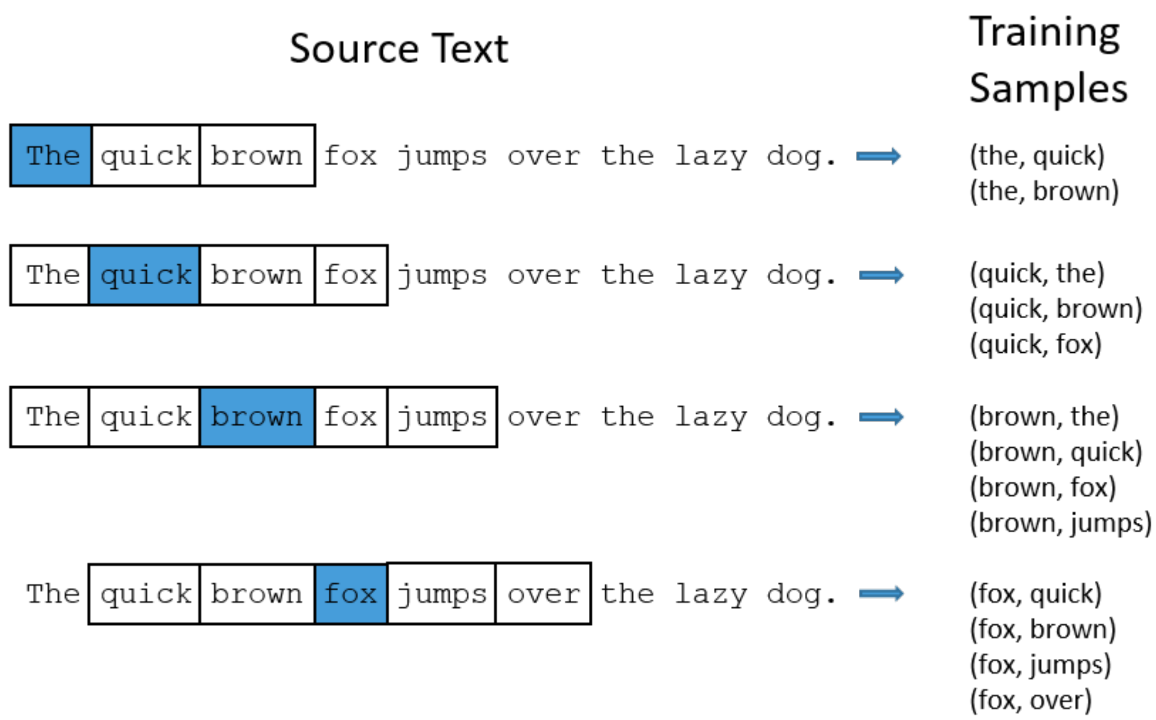
- Given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random.

- The network is going to tell us the probability for every word in our vocabulary of being the “nearby word” that we chose.

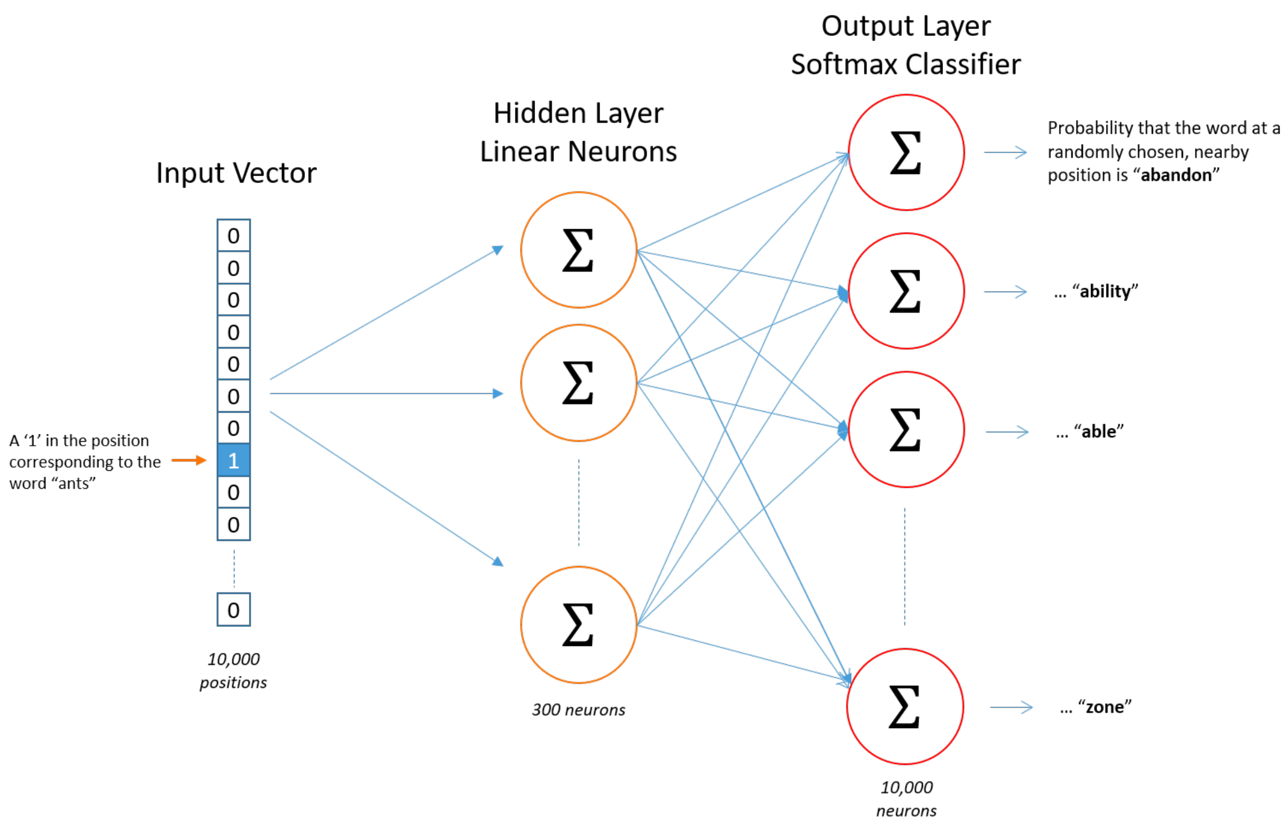
- “Nearby” words: A typical window size might be 2, meaning 2 words behind and 2 words ahead (4 in total).

- 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

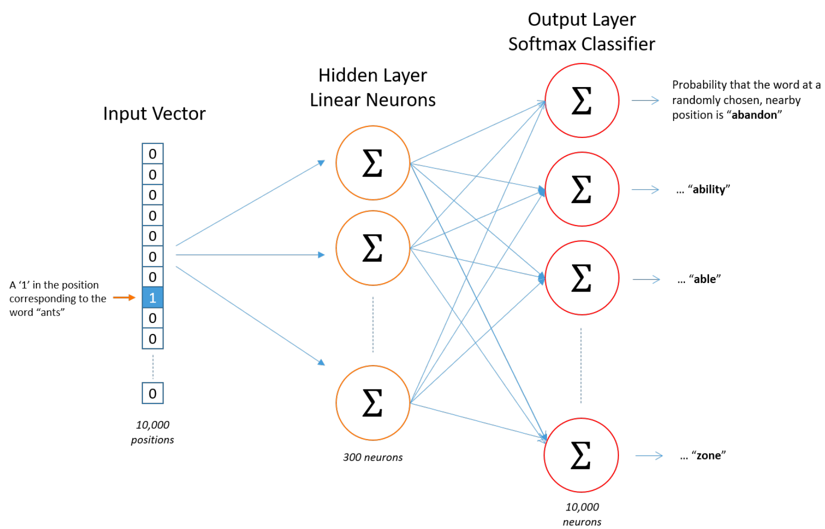


Model



Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
0
0
0
0
0
0
0
1
0
0
0
A '1' in the position corresponding to the word "ants"
10,000 positions
300 neurons
10,000 neurons
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"

Weight matrix for hidden layer



- Weight matrix is

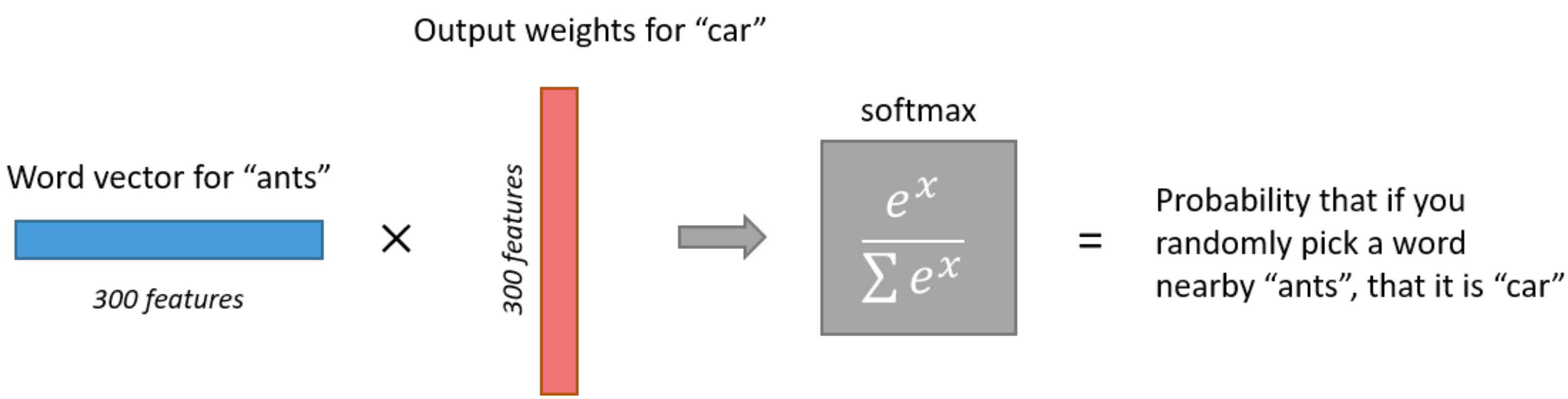
  10,000 x 300

Weight matrix: lookup table for word vectors

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

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

Output layer



Note

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

Examples

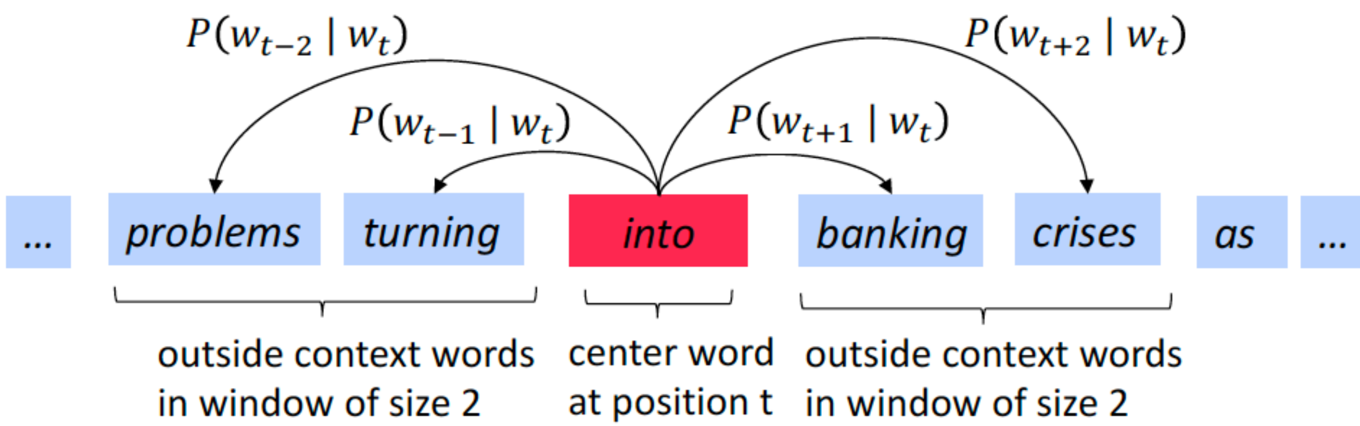
- “smart”, “intelligent”
- “ant”, “ants”

In more detail

- We have a large corpus of text
- 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

Word2Vec overview

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



Word2Vec: toward objective function

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

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

Word2Vec: objective function

- Objective function is negative log likelihood

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

- Minimizing objective function equivalent to maximizing predictive accuracy

Word2Vec: objective function

<u>Question:</u> How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

<u>Answer:</u> 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*:

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

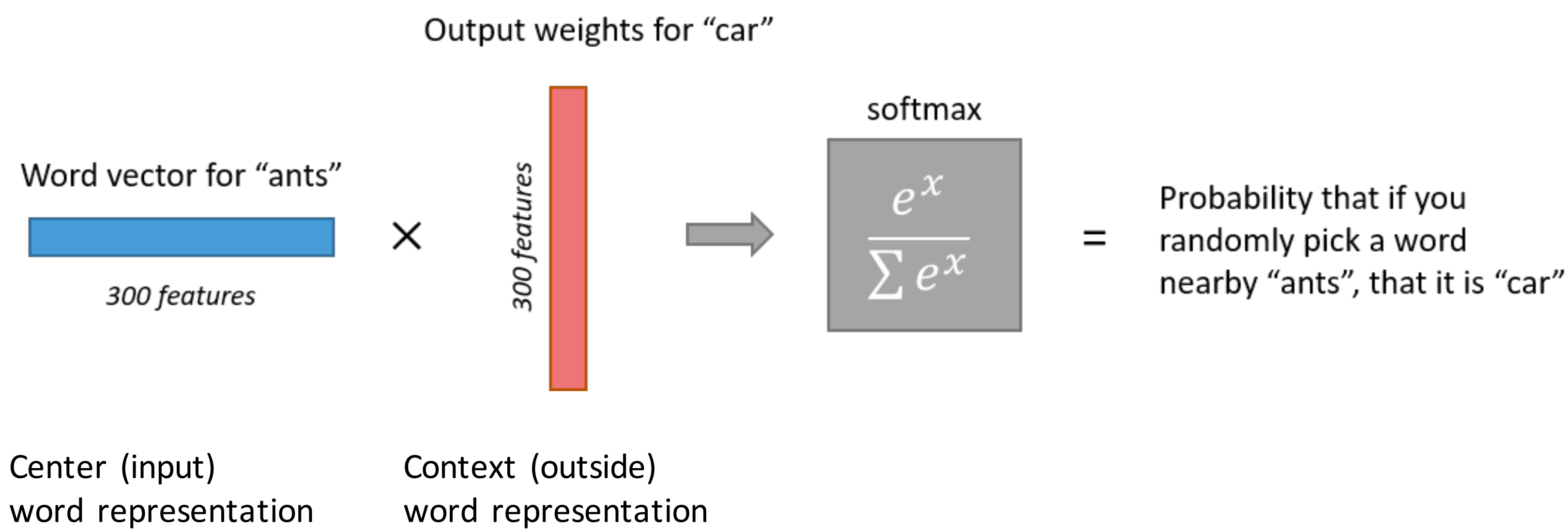
Word2Vec: prediction function

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

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

After taking exponent,
normalize over entire vocabulary

Output layer



Word2Vec: train model using SGD

Recall: θ represents **all** model parameters, in one long vector

In our case with *d*-dimensional vectors and *V*-many words:

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

Remember: every word has two vectors

We then optimize these parameters

Gradient with respect to center word

$$\frac{\partial}{\partial v_c}\left(J(\theta)\right) = u_o - \sum_{x=1}^{V} p(x|c)\, u_x$$

observed (training sample)

expectation (given current word embeddings)

Gradient with respect to center word

$$\frac{\partial}{\partial v_c}\left(J(\theta)\right) = u_o - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

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

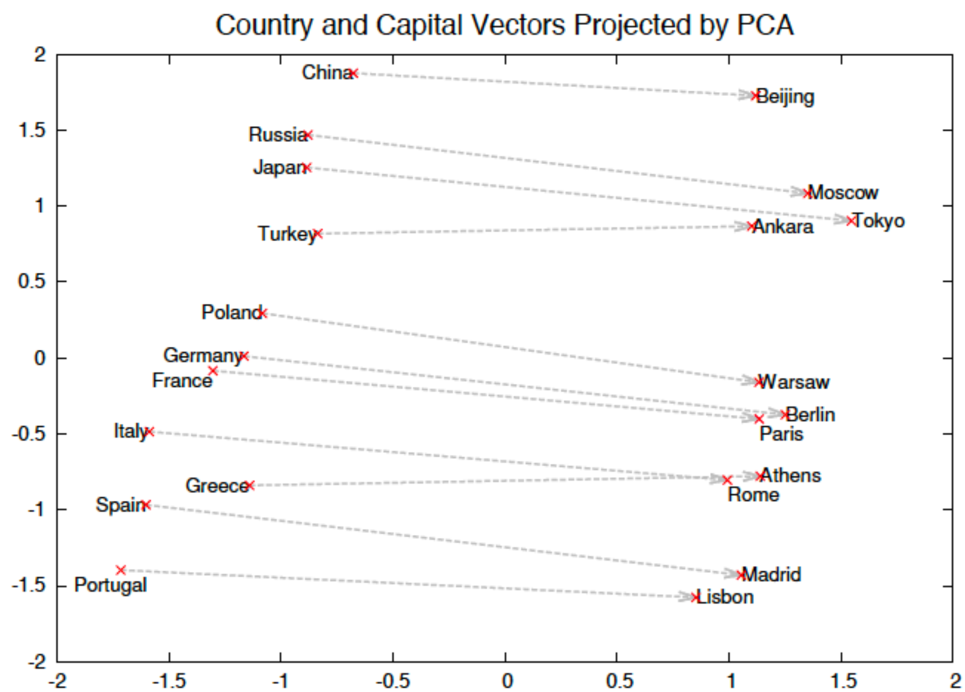


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised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

Outline for this module

- Word embedding
  - Representing words succinctly while preserving “semantic distance”
- Neural language modeling (uses word embedding)
  - RNN (basics), LSTM
- Neural machine translation (uses neural language modeling)
  - Sequence-to-sequence models, attention

- Gender bias in word embedding
- Explanations for and bias in RNNs

Acknowledgments

- Word2Vec tutorial by Chris McCormick
- Stanford cs224n lecture notes on word vectors