

Deep Learning with TensorFlow

http://cvml.ist.ac.at/courses/DLWT_W18

Lecture 4: Word Vectors

Word Vectors

Learning Representations of Words and Phrases

Divyansh Gupta

Deep Learning with TensorFlow

2018-12-10

How to meaningfully represent text?

- Most basic unit of text encoding is a character (ascii/unicode)
- A character in itself carries very little meaning
 ‘H’ , ‘G’ , ‘R’
- Words are the fundamental semantic and syntactic unit in language
 ‘Help’ , ‘garden’ , ‘running’
- In machine learning, we usually represent quantities as vectors (or tensors). Strings are difficult to operate on.
- How can we have a vector for each word?

One - Hot vectors

- Only one entry is 1, rest are all 0s.

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

←————— Vocabulary size (|V|) —————→

- Vector length = size of vocabulary (can be ~1,000,000 !)
- They are all orthogonal, no measure of similarity

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- Can we do better?

A close-up photograph of a person's hands writing on a document with a pen. The background is blurred, showing some bokeh lights. The text 'The Idea' is overlaid on the image in white.

The Idea

Represent words as dense vectors that capture semantic and syntactic similarity.

That is, similar words should have similar vectors.

Distributional Semantics

- Guiding dogma of distributional semantics:

“You shall know a word by the company it keeps”

(J. R. Firth 1957)

- Use context information to learn meaningful vectors

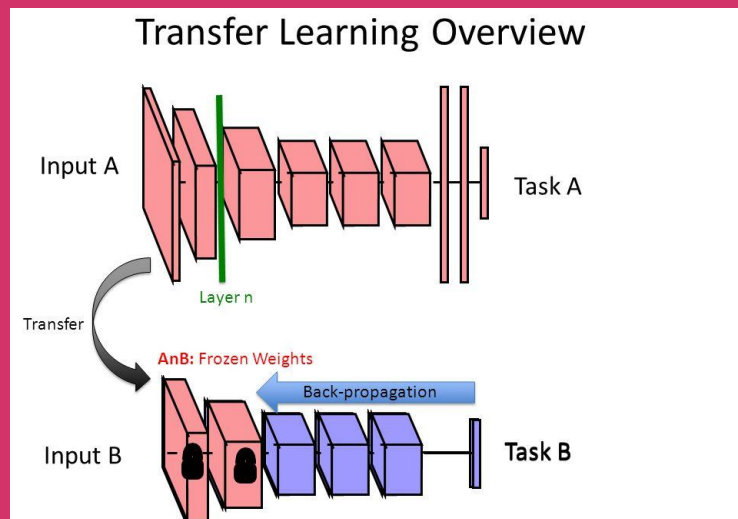
*...government debt problems turning into **banking** crises as happened in 2009...*

*...saying that Europe needs unified **banking** regulation to replace the hodgepodge...*

*...India has just given its **banking** system a shot in the arm...*

Transfer Learning

- We can't have a completely supervised way of training them, since we don't actually know what these vectors should look like.
- So we train for a “proxy” task and use the representations for the actual task
- ‘Actual’ NLP tasks:
 - Machine Translation
 - Summarization
 - Text Classification
 - Question Answering
 - ...



Methods

1. **Skip-gram** (Mikolov et al. 2013)

Predict context words from center word

2. **CBOW: Continuous Bag of Words**

(Mikolov et al. 2013)

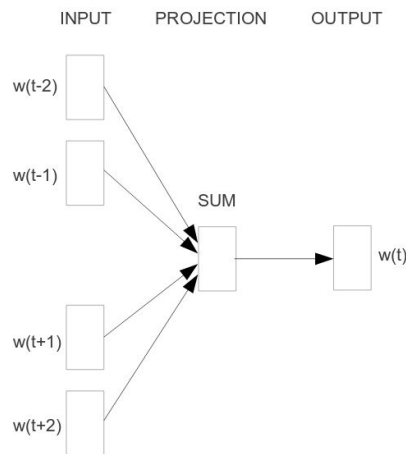
Predict center word from context words

3. **GloVe** (Socher et al. 2014)

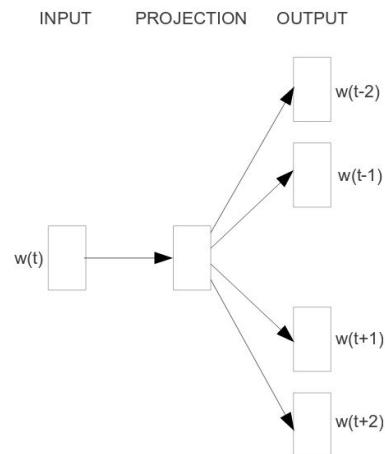
Incorporate co-occurrence counts into training

4. **FastText** (Bojanowski, et al. 2016)

Uses morphological elements of words



CBOW



Skip-gram

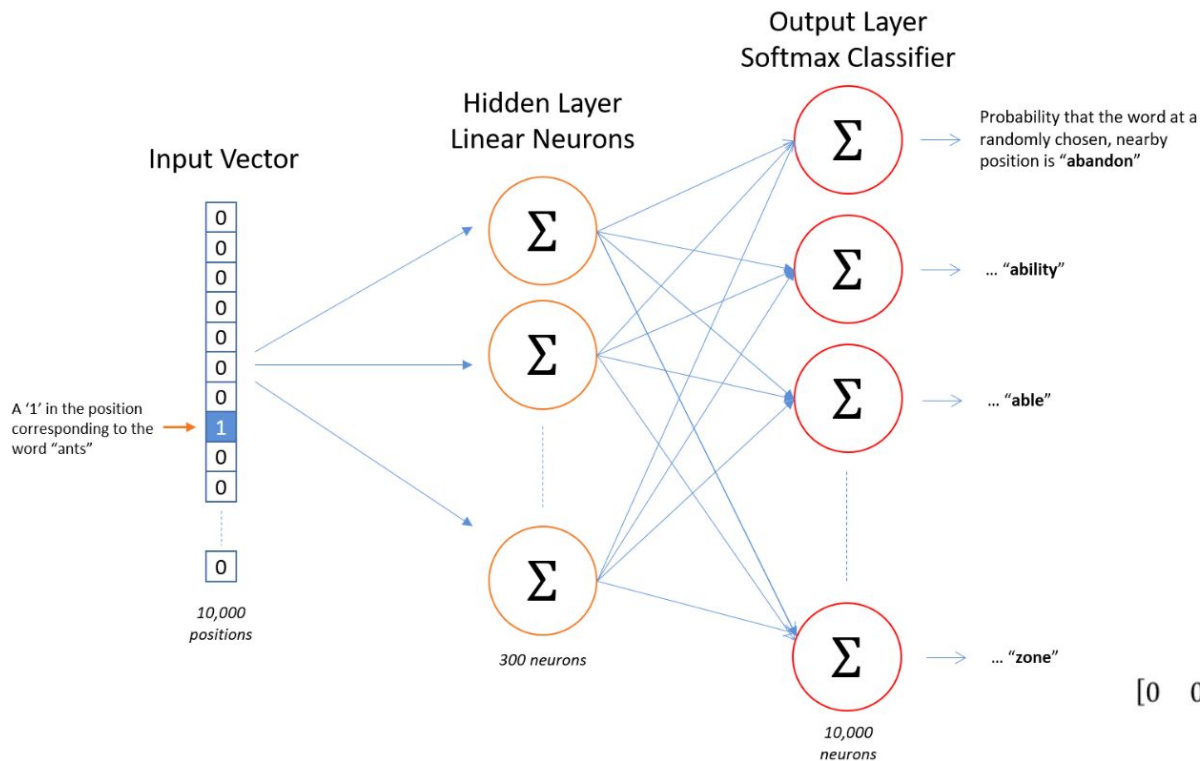
Remember!

- We are not actually interested in the “fake” tasks that we are optimizing for
- What we really care about are the **intermediate representations** that are learnt in the process
- Also known as self-supervised learning as we are using implicit labels derived from the input data itself

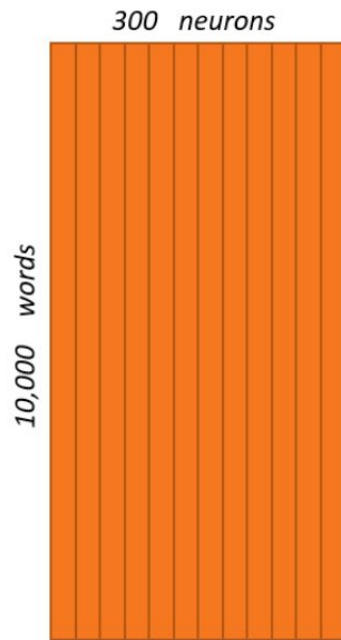
Skip-gram Training Data

Source Text	Training Samples
The quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Skip-gram Architecture

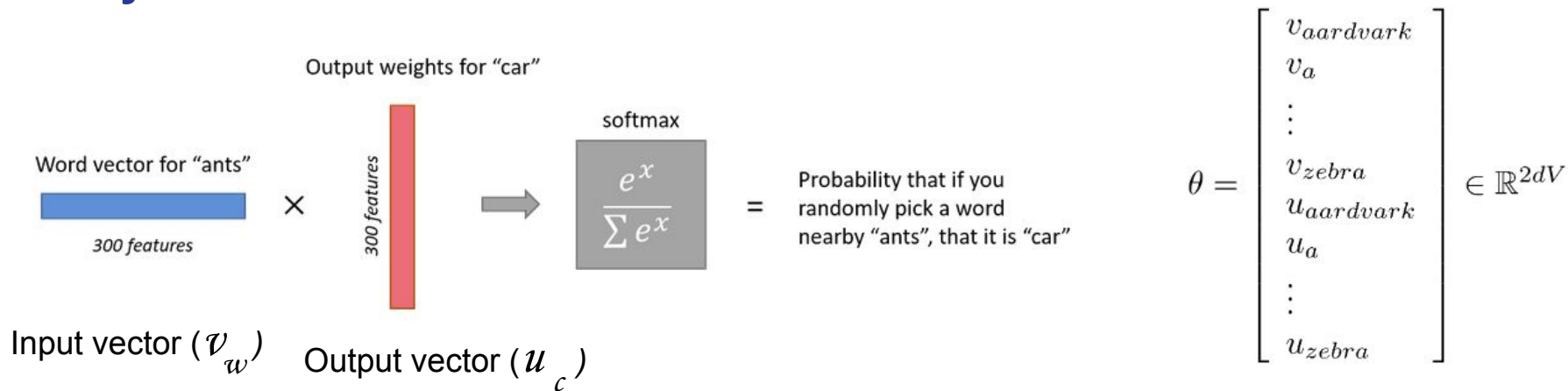


Hidden Layer Weight Matrix



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

Objective Function



$$J(\theta) = \frac{1}{|\text{Text}|} \sum_{w \in \text{Text}} \sum_{c \in C(w)} \log P(c|w; \theta)$$

$$P(c|w; \theta) = \frac{\exp(u_c^T v_w)}{\sum_{c' \in V} \exp(u_{c'}^T v_w)}$$

Upgrades! (Mikolov et al. 2013b)

1. Treating common word pairs or phrases as single “words” in their model. Example: New York
2. Subsampling frequent words to decrease the number of training examples. Probability of getting discarded:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.

Negative Sampling

- Softmax is too costly to compute
- So we convert the problem to binary classification of whether a given (word, context_word) pair belongs to the dataset, D

$$J(\theta) = \sum_{(w,c) \in D} P(D = 1|w, c; \theta) = \sum_{(w,c) \in D} \log \frac{1}{1 + \exp(-u_c \cdot v_w)}$$

- Has a trivial solution if $P(D = 1|w, c; \theta) = 1$ for all (w, c)
- So we randomly sample some negative examples that need not be in the text

$$J(\theta) = \sum_{(w,c) \in D} P(D = 1|w, c; \theta) + \sum_{(w,c') \in D'} P(D = 0|w, c'; \theta)$$

Negative Sampling (contd.)

$$\begin{aligned} J(\theta) &= \sum_{(w,c) \in D} P(D = 1 | w, c; \theta) + \sum_{(w,c') \in D'} P(D = 0 | w, c'; \theta) \\ &= \sum_{(w,c) \in D} \log \frac{1}{1 + \exp(-u_c \cdot v_w)} + \sum_{(w,c') \in D'} \log \frac{1}{1 + \exp(u_{c'} \cdot v_w)} \\ &= \sum_{(w,c) \in D} \log \sigma(u_c \cdot v_w) + \sum_{(w,c') \in D'} \log \sigma(-u_{c'} \cdot v_w) \end{aligned}$$

- D' is constructed by randomly sampling c' from the following distribution $P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^n (f(w_j)^{3/4})}$
- There are k (5-15) negative samples for each $(w, c) \in D$

Skip-gram Summary

- The model needs to predict similar outputs for words that occur in similar contexts
 - This can be done by making their vectors similar
 - For example, **intelligent** and **smart** would appear in similar contexts
 - Similarly for contexts like “**There are 11 players in a team**”, and “**Basketball is played by five members**”...
- CBOW is just the opposite: predict center word from context words

A slightly different approach: co-occurrence

- Construct co-occurrence matrix
- Reduce with SVD (patented for IR in 1988)
- Example corpus:
 - I like deep. learning
 - I like NLP.
 - I enjoy flying.

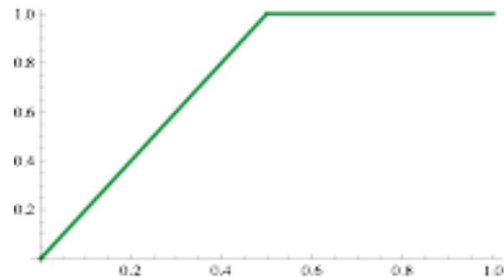
counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

GloVe (2014) - Combining both methods

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^W f(P_{ij}) (u_i^T v_j - \log P_{ij})^2$$

- P_{ij} is the co-occurrence count of w_i and w_j
- $f(P_{ij})$ is used to give less weight to infrequently occurring pairs
- Faster to train as it does not make multiple passes through the corpus

$f \sim$




Evaluation

Intrinsic

- Analyse the word vectors themselves to see what they represent
- For instance, Analogy task, similarity task, clustering

Extrinsic

- Use the generated representation on actual NLP tasks
 - See which embedding does best on Part-of-speech tagging, machine translation etc.
- 

Analogy Task

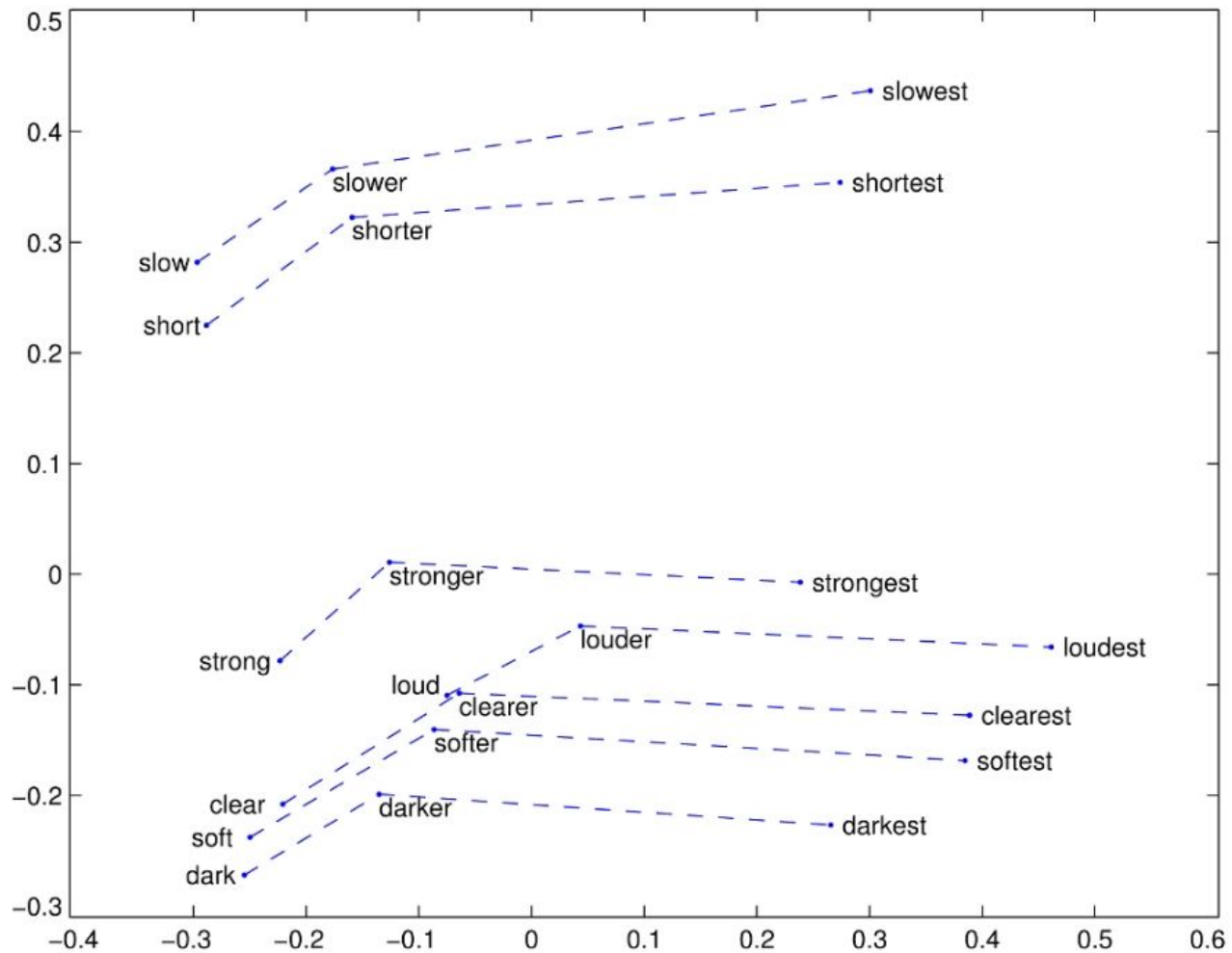
a:b :: c:?

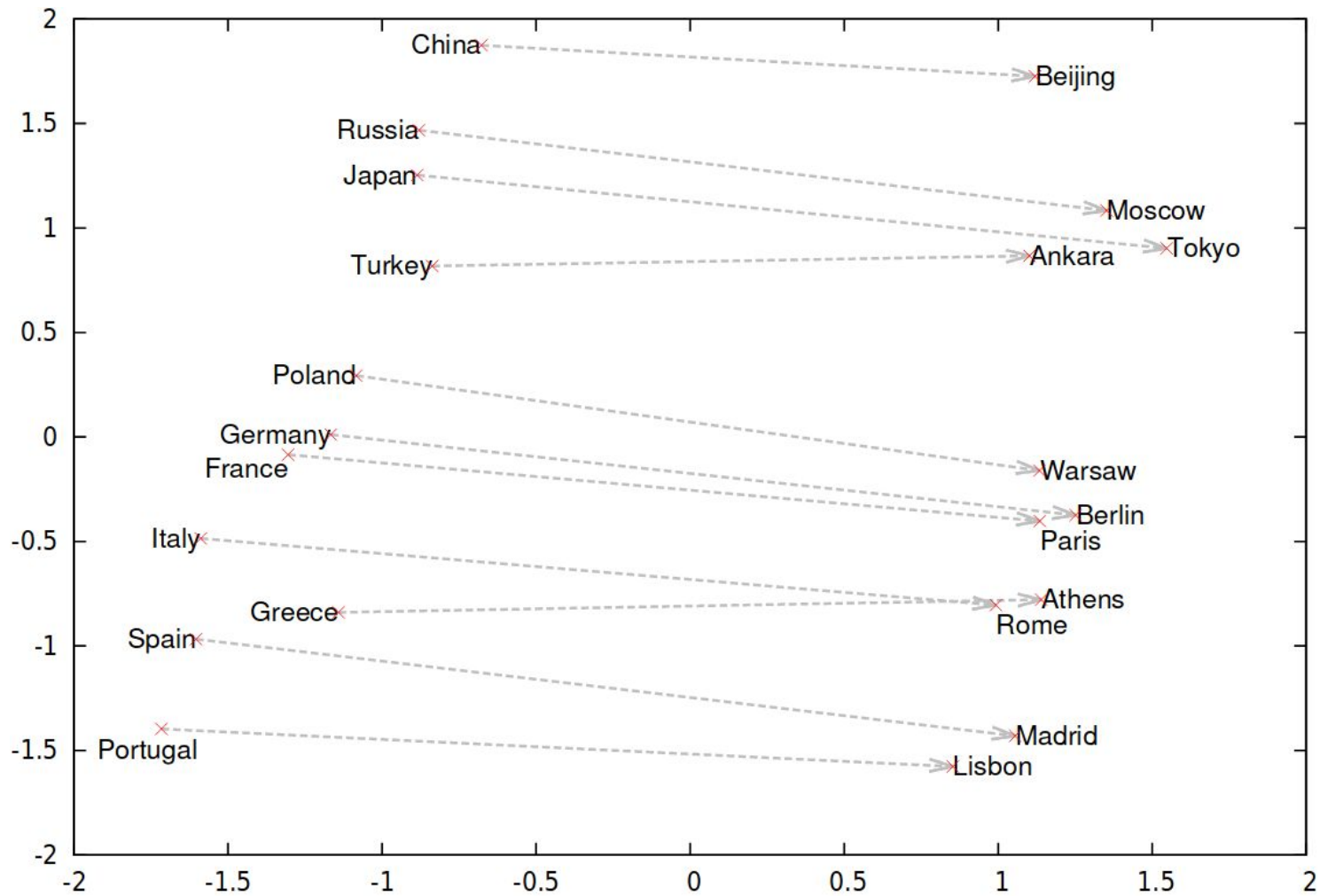
man:woman :: king:?



$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza





Other cool applications

Context

Input

Context

TITLE	ARTIST	ALBUM	🕒
✓ Voyeur	Phantoms, Ni...	Broken Halo	4:40
✓ Somebodies Something	Tyne	Somebodies ...	3:44
✓ No Words - Kasbo Remix	Erik Hassle, C...	No Words (Re...	5:08
✓ I Will Wait	Aaron Krause	I Will Wait	3:54
✓ Lost	Ficci	Lost	4:07
✓ Falling Short (DarkO Remix)	Låpsley, DarkO	Falling Short (...)	4:12
✓ Self Defined	Maya Payne	The Lucky On...	3:54
✓ Escape	Tongues.	Kitsuné Hot S...	3:07

Implementation

- Pre-trained vectors available for all these methods!

- Word2Vec (Skip-gram)

<https://code.google.com/archive/p/word2vec/>

- GloVe

<https://nlp.stanford.edu/projects/glove/>

- FastText (also for 156 other languages!)

<https://fasttext.cc/docs/en/english-vectors.html>

TensorFlow Code

```
embeddings = tf.Variable(tf.random_uniform([vocabulary_size, embedding_size],
-1.0, 1.0))
nce_weights = tf.Variable(tf.truncated_normal([vocabulary_size,
embedding_size], stddev=1.0 / math.sqrt(embedding_size)))
nce_biases = tf.Variable(tf.zeros([vocabulary_size]))

# Placeholders for inputs
train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])

# Embedding helper function:
embed = tf.nn.embedding_lookup(embeddings, train_inputs)
```

TensorFlow Code (Continued)

```
# Compute the NCE loss, using a sample of the negative labels each time.
```

```
loss = tf.reduce_mean(  
    tf.nn.nce_loss(weights=nce_weights,  
                  biases=nce_biases,  
                  labels=train_labels,  
                  inputs=embed,  
                  num_sampled=num_sampled,  
                  num_classes=vocabulary_size))
```

```
# We use the SGD optimizer.
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1.0).minimize(loss)
```

```
with tf.Session as session:
```

```
    for inputs, labels in generate_batch(...):  
        feed_dict = {train_inputs: inputs, train_labels: labels}  
        _, cur_loss = session.run([optimizer, loss], feed_dict=feed_dict)
```

References

1. Stanford NLP CS224n - <http://web.stanford.edu/class/cs224n/syllabus.html>
2. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. In Proceedings of Workshop at ICLR, 2013.
3. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. *Distributed Representations of Words and Phrases and their Compositionality*. In Proceedings of NIPS, 2013.
4. McCormick, C. (2016, April 19). Word2Vec Tutorial - The Skip-Gram Model. Retrieved from <http://www.mccormickml.com>

TensorBoard Demo

Adapted from:

<http://www.cse.chalmers.se/~richajo/dit865/files/Word%20embeddings%20in%20Gensim.html>

and

<https://github.com/sudharsan13296/visualise-word2vec>



Questions?