Word Representation in Deep Learning

Zishan Ahmad
IIT Patna

Outline

1. Why word representation?
2. Non semantic word representations
   a. One-hot vector representation
3. Semantic word representation
   a. Distributional hypothesis
   b. Co-occurrence matrix based representation
   c. Language model
   d. FFNN language model
   e. Skip-gram model
   f. Continuous Bag of Words model (CBoW)
4. Cross-lingual word embeddings
   a. Why cross-lingual embeddings

Outline (continued...)

4. Crosslingual and Multi-lingual word embeddings
   a. Supervised Methods
      i. Parallel Corpus - Luong et al. 2015
      ii. Comparable Corpus - Vulić and Moens, 2015
      iii. Bilingual dictionary Induction
   b. Unsupervised Methods - Artext et.al., 2018

Why word representation?

Definition : Word (Oxford Dictionary)

A word is a single distinct meaningful element of speech or writing, used with others (or sometimes alone) to form a sentence

- Words are stitched together to form a sentence
- Proper representation of words is essential for text representation

Non-semantic word representation

The vast majority of rule-based and statistical NLP work regards words as atomic symbols

One-hot vector representation of words:

- Assign a unique id to each unique word in the corpus
- Convert these unique ids to one-hot vectors

Sentence: RMS Titanic was a British passenger liner.

Unique Ids: [1, 2, 3, 4, 5, 6, 7]

One-hot representation: [[1,0,0,0,0,0,0], [0,1,0,0,0,0,0], [0,0,1,0,0,0,0], [0,0,0,1,0,0,0], [0,0,0,0,1,0,0], [0,0,0,0,0,1,0], [0,0,0,0,0,0,1]]

Non-semantic word representation (continued...)

Python Code for categorical (one-hot) representation

```
from keras.utils import to_categorical
txt = "RMS Titanic was a British passenger liner that sank in the North Atlantic Ocean in 1912 after striking an iceberg during her maiden voyage from Southampton to New York City"
txt_list = txt.split()
word2id = {}
for i,j in enumerate(list(set(txt_list))):
  word2id[j] = i
txt_index = [word2id[i] for i in txt_list]
txt_one_hot = to_categorical(txt_index)
```

Non-semantic word representation (continued...)

Drawbacks of categorical representation:

- No semantics captured
- All the words are equally different from each other
  - The euclidean distance between any two words is 1.41 units
  - The cosine similarity between any two words is 0
- Curse of dimensionality (the length of the vector depends on the number of words in the corpus)
- The vectors formed are sparse

Semantic word representation

We can get a lot of value by representing a word by means of its neighbors:

“You shall know a word by the company it keeps” **(J. R. Firth 1957: 11)**

Built in Belfast, Ireland, in the United Kingdom the RMS **Titanic** was the second of the three Olympic-class ocean liners.

According to distributional hypothesis, all these words play a role in representing the meaning of the word **Titanic**

Semantic word representation (continued...)

Using co-occurrence matrix to make neighbours represent words.

- Window based co-occurrence matrix captures syntactic (POS) and semantic information
- The matrix is symmetric, i.e. an occurrence is counted irrespective of left or right context
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Semantic word representation (continued...)

Co-occurrence matrix example -

- Window size = 1

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

Semantic word representation (continued...)

Co-occurrence matrix example -

https://colab.research.google.com/drive/10XCsBjW88b9pYiLgWADxVaDLhZHhSeVV

Semantic word representation (continued…)

Code for co-occurence matrix creation:

```
import pandas as pd
import numpy as np
from collections import defaultdict
def co_occurrence(sentences, window_size):
    d = defaultdict(int)
    vocab = set()
    for text in sentences:
        text = text.lower().split()
        # iterate over sentences
        for i in range(len(text)):
            token = text[i]
            vocab.add(token)  # add to vocab
            next_token = text[i+1 : i+1+window_size]
```

Semantic word representation (continued…)

Code for co-occurence matrix creation:

```
for t in next_token:
                key = tuple( sorted([t, token]) )
                d[key] += 1
    # formulate the dictionary into dataframe
    vocab = sorted(vocab) # sort vocab
    df = pd.DataFrame(data=np.zeros((len(vocab), len(vocab)), dtype=np.int16),
                      index=vocab,
                      columns=vocab)
    for key, value in d.items():
        df.at[key[0], key[1]] = value
        df.at[key[1], key[0]] = value
    return df
docs = ["I like deep learning", "I enjoy NLP", "I enjoy flying"]
co_occurrence(docs, window_size=1)
```

Semantic word representation (continued...)

Problems with simple co-occurrence vectors:

- Increase in size with vocabulary
- Sparsity issue persists
- Very high dimensional: require a lot of storage

Semantic word representation (continued...)

Language Modeling:

Language Modeling (LM), is the development of probabilistic models that are able to predict the next word in the sequence given the words that precede it.

- A language model learns the probability of word occurrence based on examples of text
- Simpler models may look at a context of a short sequence of words, whereas larger models may work at the level of sentences or paragraphs
- Most commonly, language models operate at the level of words

Mathematically:

$P(x_1,x_2,x_3,...,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)...P(x_n|x_1,x_2,...,x_{n-1})$

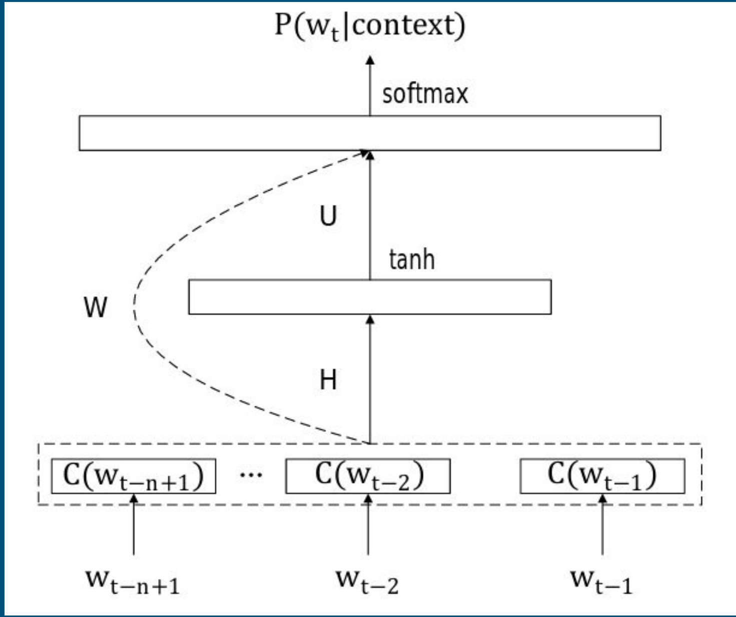
P(“its water is so transparent”) = P(“its”)P(“water”|”its”)P(“is”|”its”,”water”)... P(“transparent”|”its”, “water”, “is”, “so”)

P(“transparent”|”its water is so”) = count(transparent) / count(its water is so)

Semantic word representation (continued...)

Neural Language Modeling:

Feed Forward Neural Network Language Model (FFNNLM):



Semantic word representation (continued...)

Neural Language Modeling:

- Previous *n-1* words are projected by shared projection matrix $C \in R^{|V| X m}$, where |*V*| is the size of the vocabulary and *m* is the size of the feature
- The input *x* of the FFNN is a concatenation of feature vectors of *n−1* words
- Model is followed by Softmax output layer to guarantee all the conditional probabilities of words positive and summing to one
- The final Softmax layer predicts the n^{th} word (next word given the previous context)

Semantic word representation (continued...)

Skip-gram Model:

This is one of the methods used for the creation of Word2Vec word embeddings

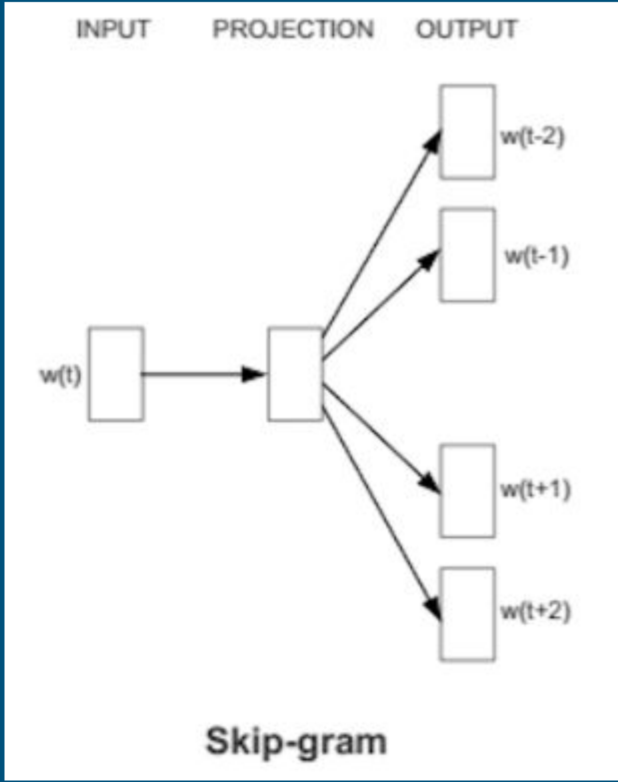
Main ideas behind this method

- Instead of capturing co-occurrence counts directly, predict surrounding words for every word
- Predict surrounding words in a window of length *m* for every word
- Objective function: Maximize the log probability of any context word given the current center word:

$$\text{minimize } J = -\log P(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} | w_c)$$

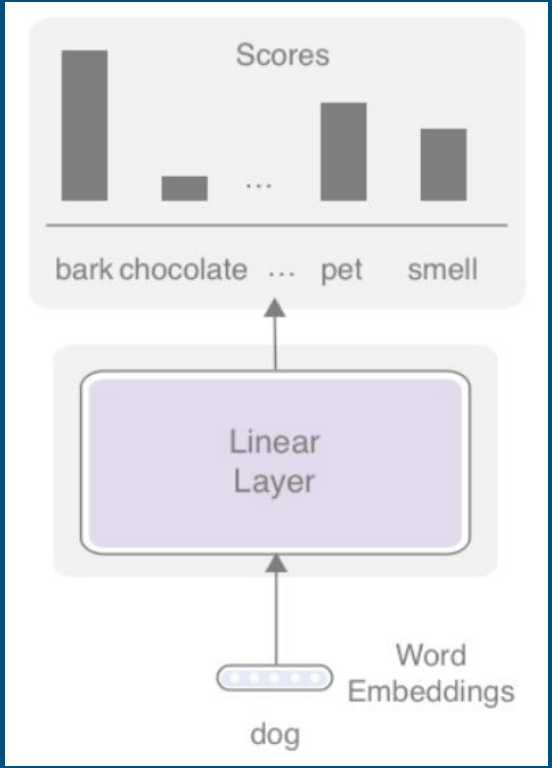
Semantic word representation (continued...)

Skip-gram Model:



Semantic word representation (continued...)

Skip-gram Model:



Semantic word representation

Continuous Bag of Words Model:

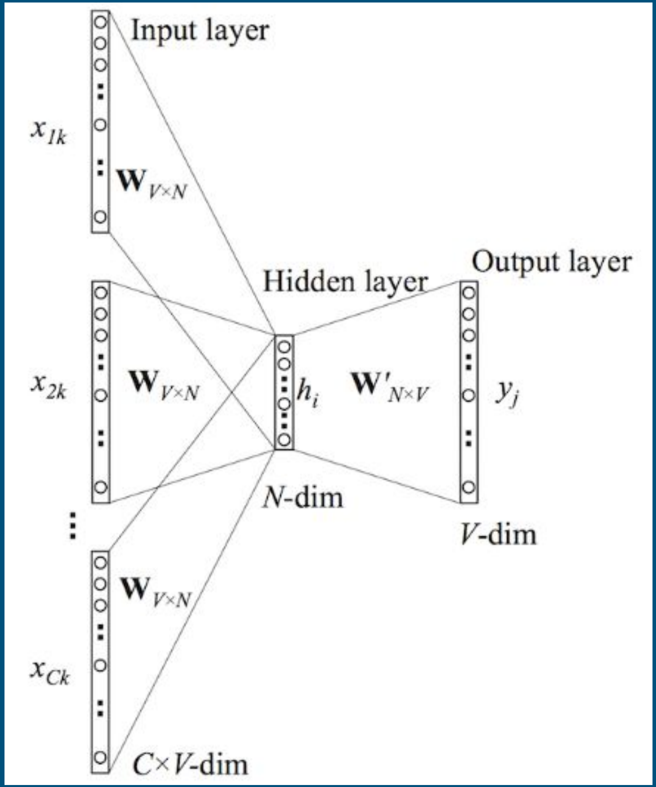
This is another method for creation of Word2Vec word embeddings

Main ideas behind this method

- Predict the current word based on other words in the context window *m*
- Objective function: Maximize the log probability of the current word given the context words

$$\text{minimize } J = -\log P(w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m})$$

Semantic word representation (continued...)



Semantic word representation (continued…)

Code for word embedding creation:

```
from gensim.models import Word2Vec

sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
       ['this', 'is', 'the', 'second', 'sentence'],
       ['yet', 'another', 'sentence'],
       ['one', 'more', 'sentence'],
       ['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1, size=300, sg=0) #sg ({0, 1}, optional) – Training algorithm: 1
for skip-gram; otherwise CBOW.

print(model)
# summarize vocabulary
words = list(model.wv.vocab)
print(words)
# access vector for one word
print(model['sentence'])
```

Semantic word representation (continued...)

Code for word embedding creation:

```
model['this'].size

# save model
model.save('model.bin')
# load model
new_model = Word2Vec.load('model.bin')
print(new_model)
```

Semantic word representation (continued...)

Word2Vec demo:

```
from gensim.test.utils import common_texts, get_tmpfile
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import numpy as np

def cos(x1, x2):
  return np.dot(x1, x2)/(np.linalg.norm(x1)*np.linalg.norm(x2))

!wget -P /root/input/ -c
"https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz"
EMBEDDING_FILE = '/root/input/GoogleNews-vectors-negative300.bin.gz' # from above
word2vec = KeyedVectors.load_word2vec_format(EMBEDDING_FILE, binary=True)
print(word2vec["cat"].shape)
print(cos(word2vec['cat'],word2vec['purr']))

print(word2vec.similar_by_vector(word2vec["cat"], topn=10, restrict_vocab=None))
```

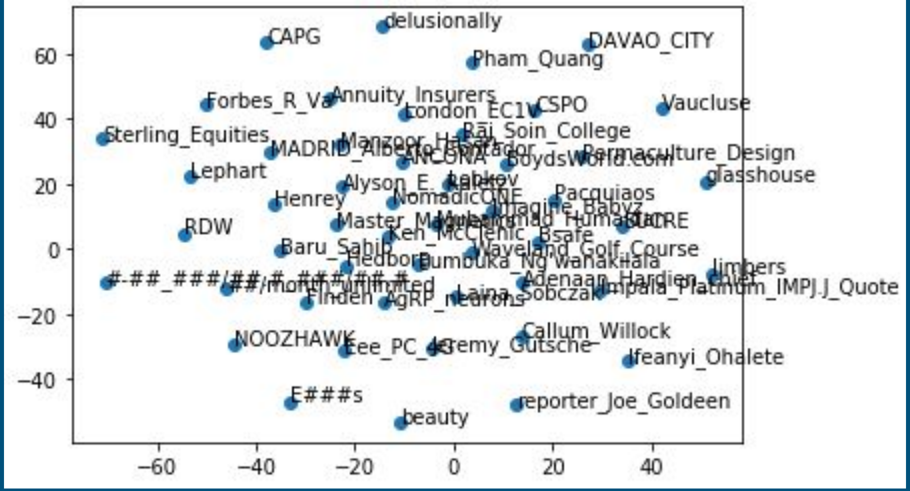
Semantic word representation (continued...)

Word2Vec demo:

Plotting word vectors:

```
import random
vocab = random.sample(list(word2vec.vocab), 50)
X = np.array([word2vec[v] for v in vocab])
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=0)
np.set_printoptions(suppress=True)
Y = tsne.fit_transform(X)

plt.scatter(Y[:, 0], Y[:, 1])
for label, x, y in zip(vocab, Y[:, 0], Y[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(0, 0), textcoords='offset points')
plt.show()
```



Cross-lingual word embeddings

Why do we need Cross-lingual Embeddings?

- Bridge the language divergence

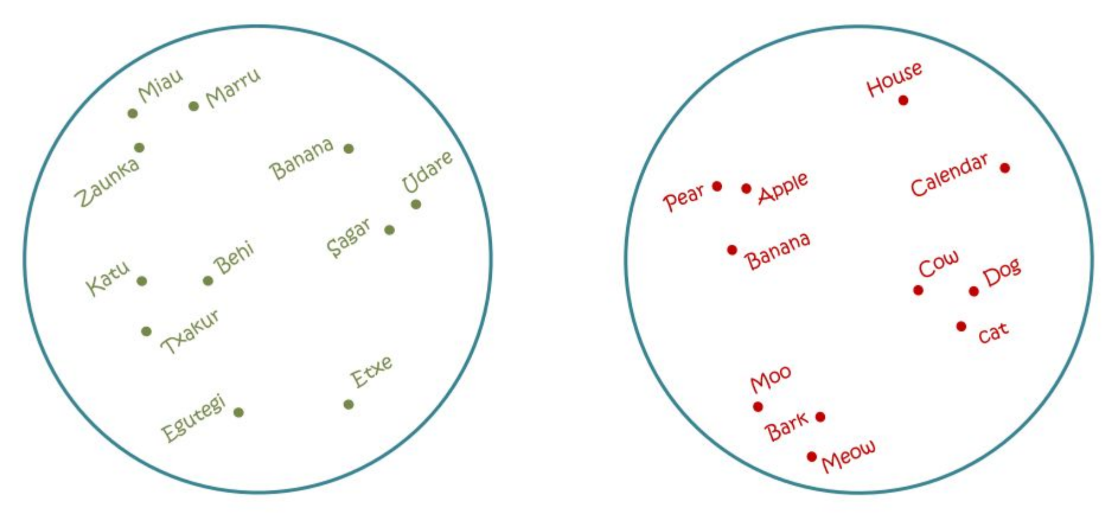
Applications

- Leverage the resource-richness of one language (e.g., English) in solving a problem in resource-constrained languages (e.g., Hindi, Marathi etc.)
- Useful for unsupervised machine translation

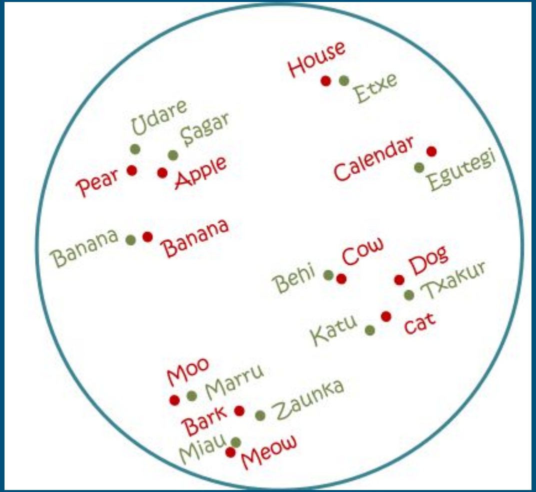
Cross-lingual word embeddings (continued...)

Problems with monolingual word embeddings

- Embedding of a word in one language (say, Spanish) and embedding of the same word (translated) in other language (say, English) *do not possess any association* between them.
- Therefore, they cannot represent each other in the vector space (i.e., they *cannot correlate*).



Monolingual embeddings (Spanish and English)



Cross-lingual embedding (Spanish and English)

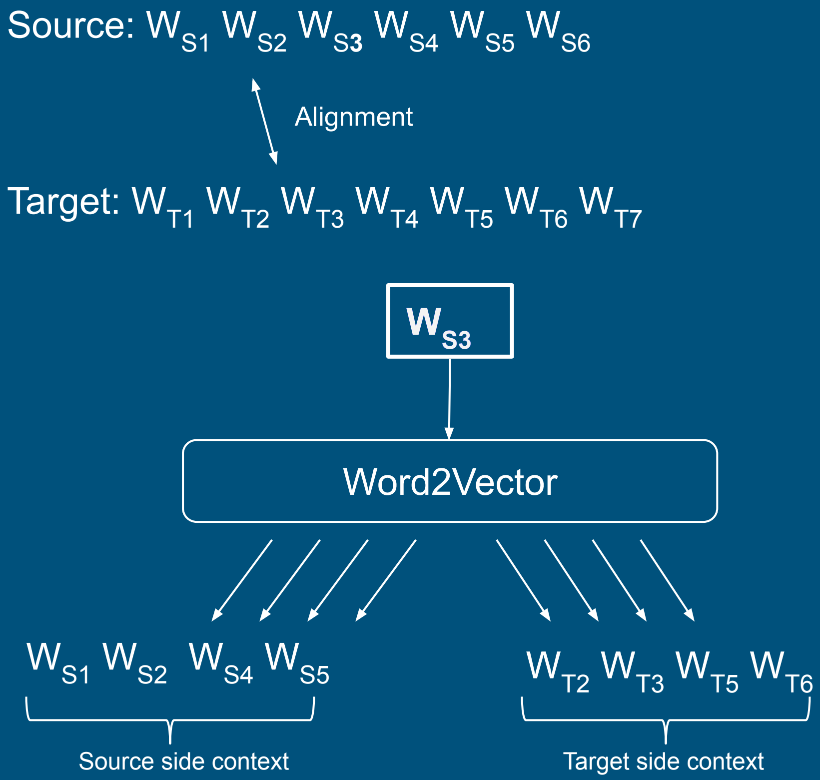
Cross-lingual word embeddings (continued...)

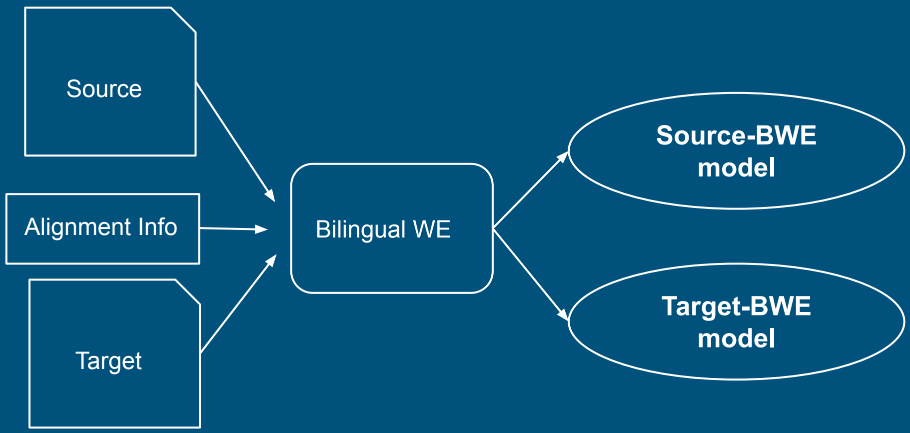
Luong et al. 2015, Bilingual Word Representations with Monolingual Quality in Mind. In *NAACL Workshop on Vector Space Modeling for NLP*.

Bi-lingual word embeddings aims to *bridge the language divergence* in the vector space.

- Idea is pretty simple
  - Utilize existing word2vec skip-gram model (Mikolov., 2013a)
  - For each word, define its context to include words from both the source and target languages
- Requires a *parallel corpus* and alignment information among parallel sentences

Cross-lingual word embeddings (continued...)



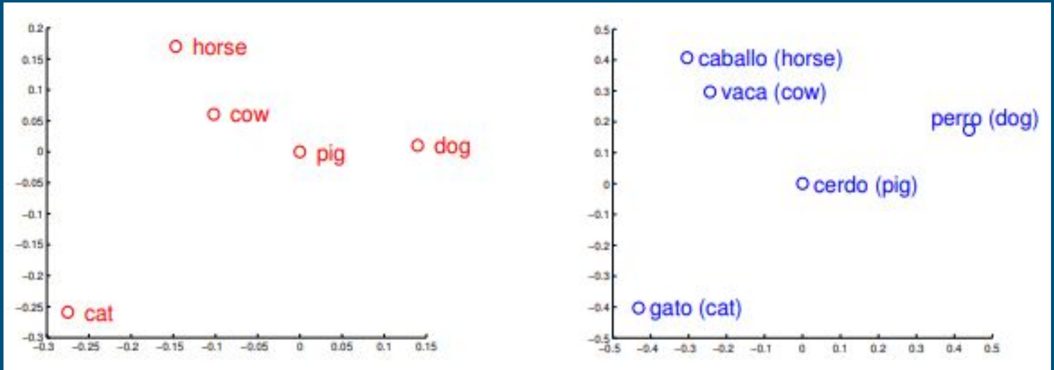


Bi-lingual word embeddings

Cross-lingual word embeddings (continued...)

Tomas Mikolov, Quoc V. Le, and Ilya Sutskever, 2013. Exploiting Similarities among Languages for Machine Translation. In arXiv:1309.4168v1.

- Requires
  - *Two monolingual embeddings*
  - *Bi-lingual dictionary*

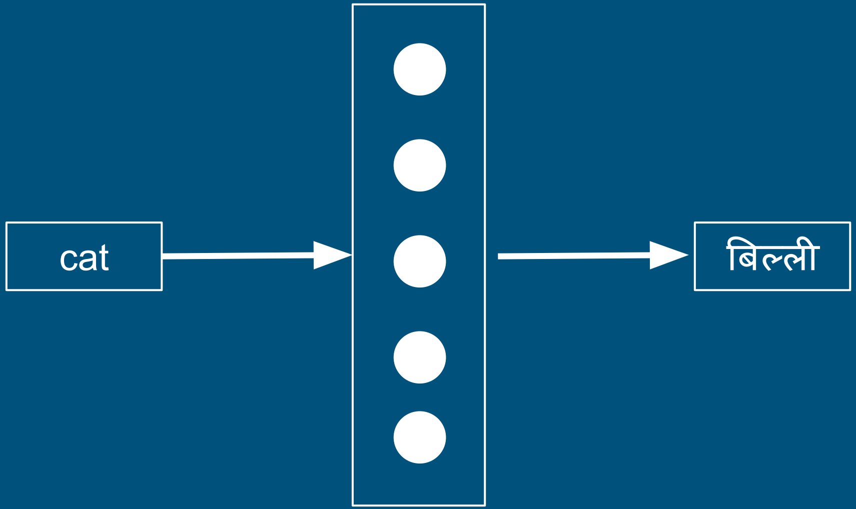


- Approach
  - Suppose we are given a set of word pairs and their associated vector representations $\{x_i, z_i\}$.
  - Goal is to find a transformation matrix W

$$\min_W \sum_{i=1}^{n} \|Wx_i - z_i\|^2$$

  - For any given new word and its vector representation x, we can compute $z = Wx$.

Cross-lingual word embeddings (continued...)



Linear layer (W) for transforming *English* words to *Hindi*

Cross-lingual word embeddings (continued...)

Normalized word embedding and orthogonal transform for bilingual word translation (Xing et al. 2015):

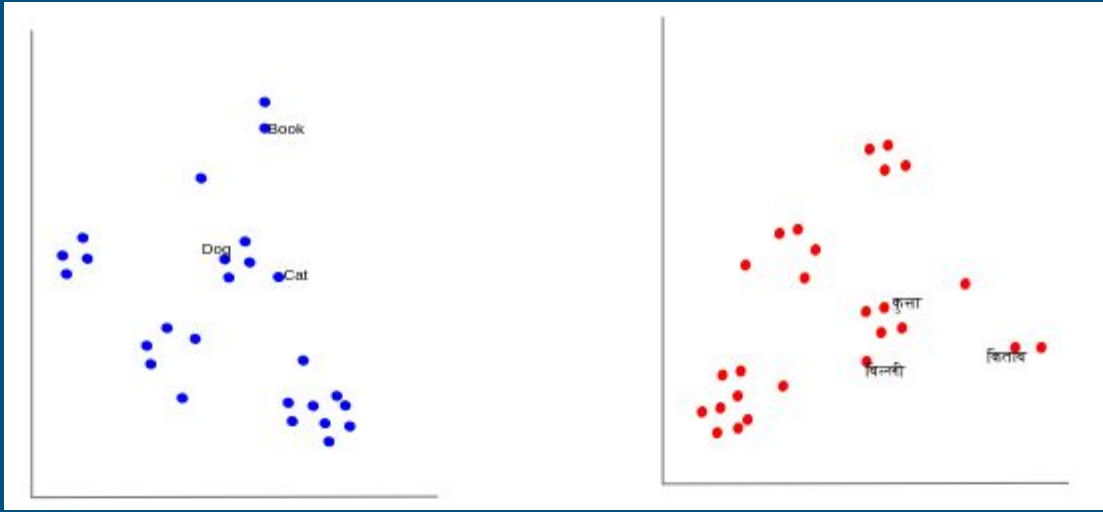
- In, *Exploiting Similarities among Languages for Machine Translation (Mikolov et at. 2013)*
  - Given a set of n word pairs and their vector representations $\{x_i, y_i\}$, where x_i is a d_1 dimensional vector and y_i is a d_2 dimensional vector
  - Goal is to find W (dimension: d_2 ❌ d_1) such that Wx_i approximates y_i min$_W$ $||WX\text{-}Y||$
  - These results can be improved by enforcing an orthogonality constraint on W

$$WW^T = I$$

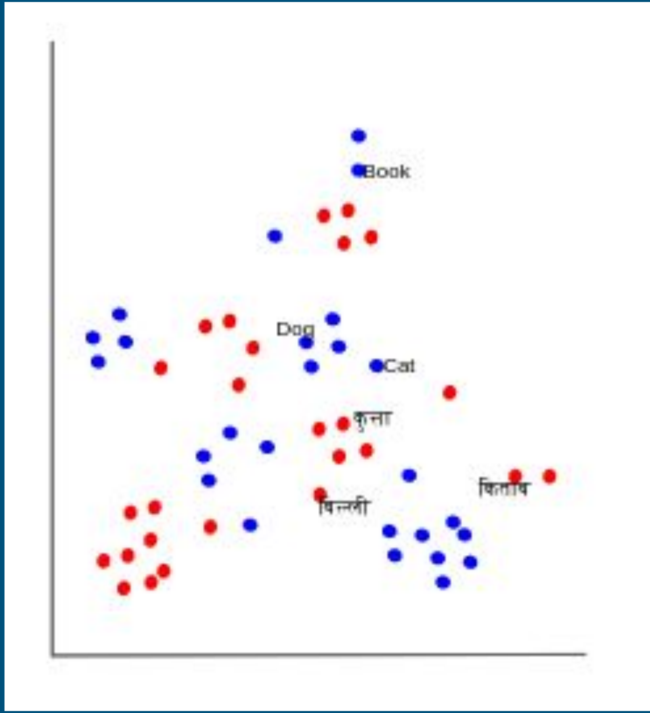
Cross-lingual word embeddings (continued...)

Why is Orthogonality important

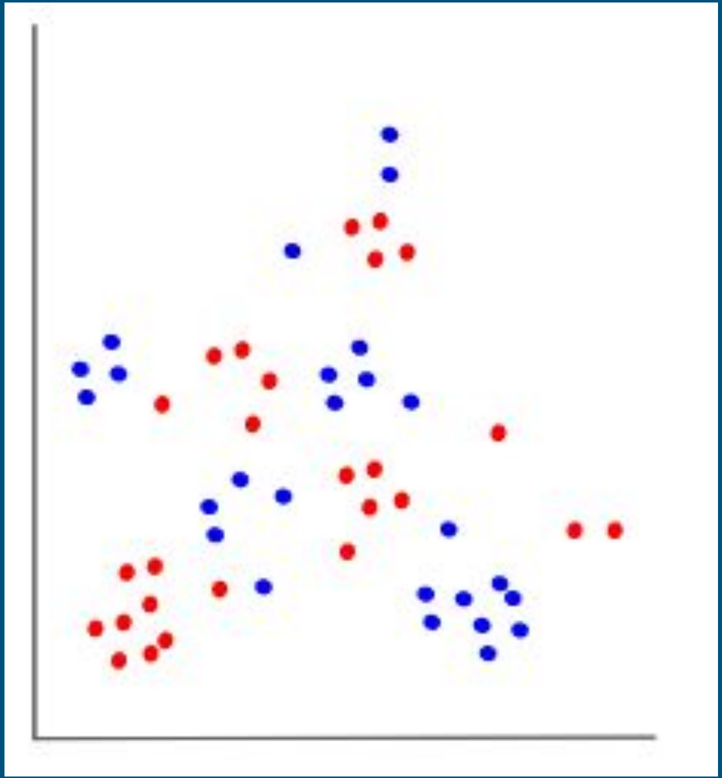
- It restricts transformation to only rotation
- Orthogonal transformation is length and angle preserving.
- Therefore it is an isometry of the Euclidean space (such as a rotation).



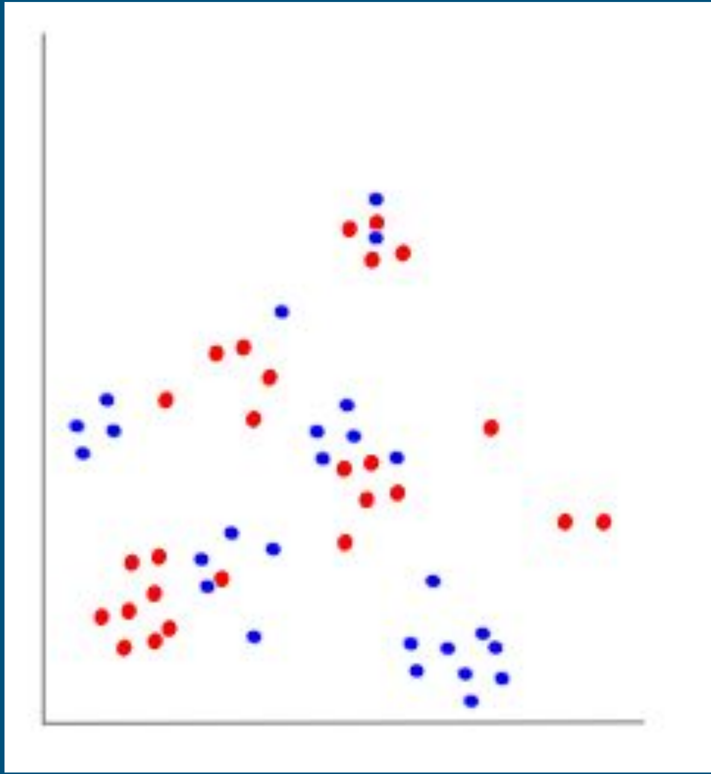
Cross-lingual word embeddings (continued...)



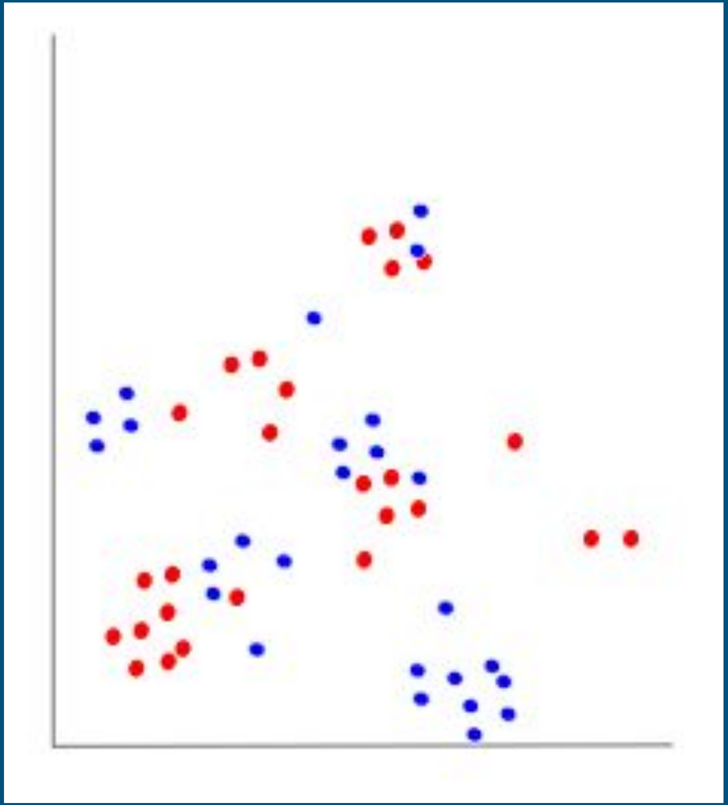
Cross-lingual word embeddings (continued...)



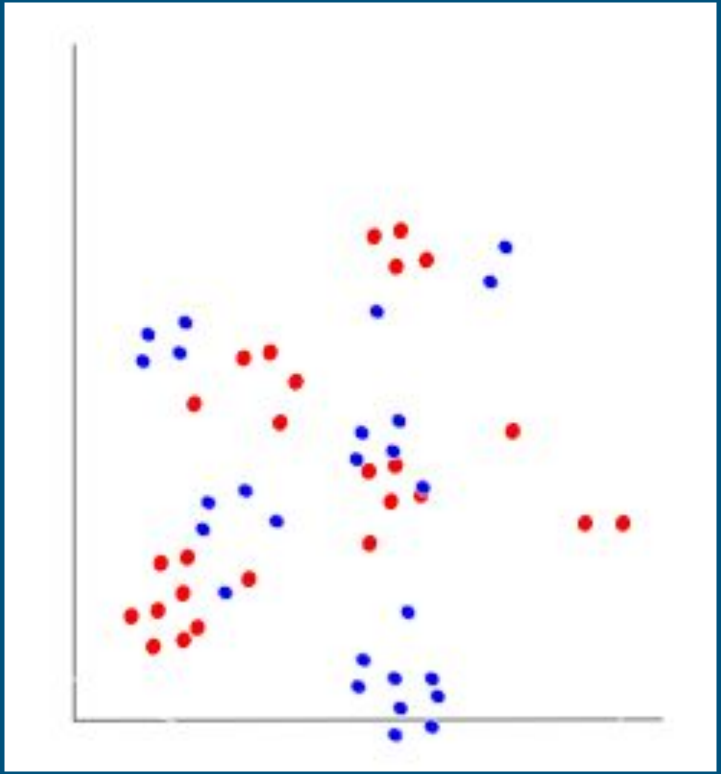
Cross-lingual word embeddings (continued...)



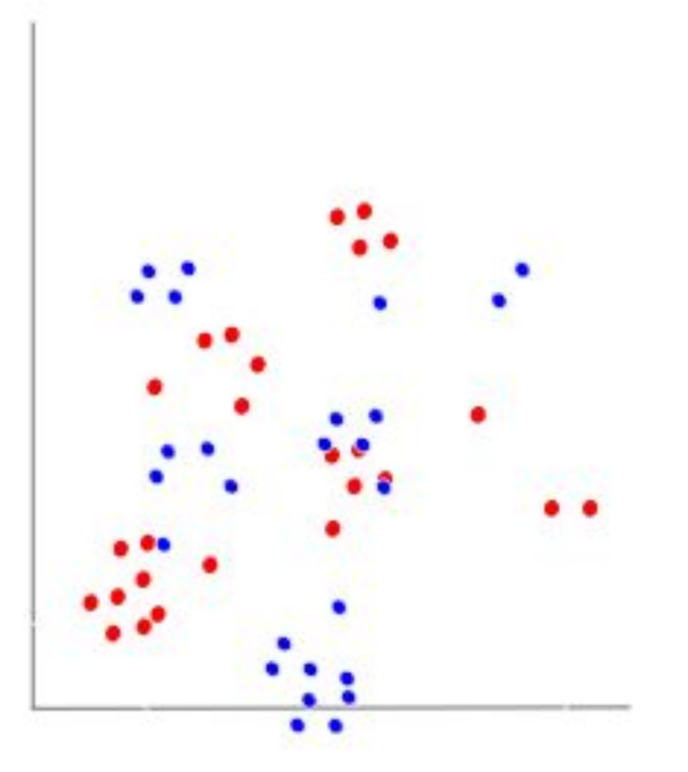
Cross-lingual word embeddings (continued...)



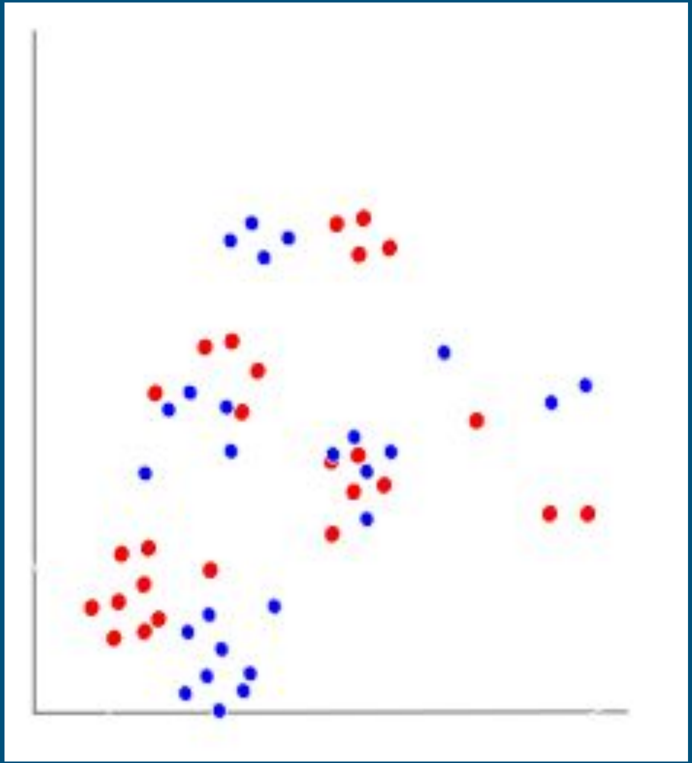
Cross-lingual word embeddings (continued...)



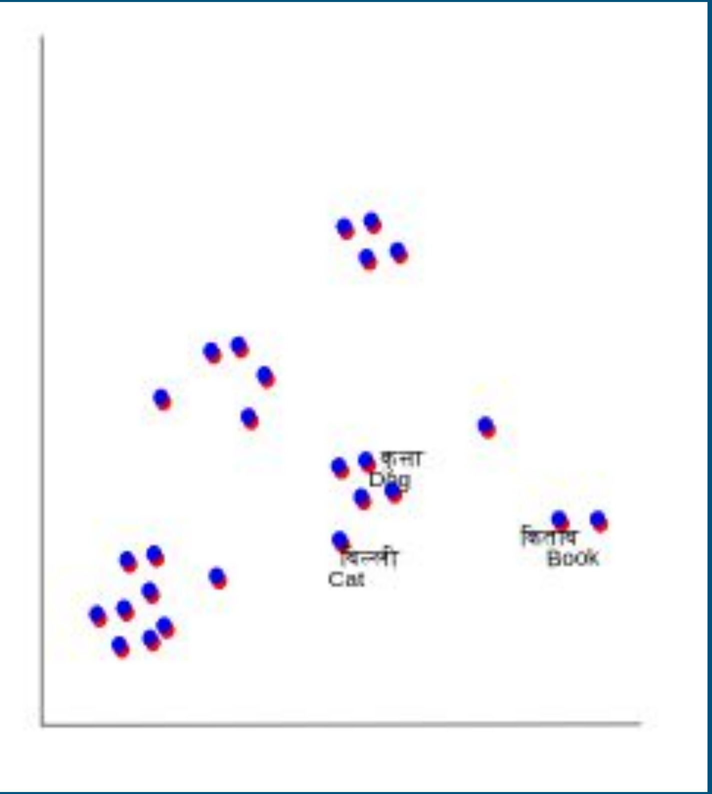
Cross-lingual word embeddings (continued...)



Cross-lingual word embeddings (continued...)



Cross-lingual word embeddings (continued...)



Cross-lingual word embeddings (continued...)

Word translation without parallel data (Conneau et al. 2018)

Proposed complete unsupervised approach to cross-lingual mapping:
Basic steps:

- Learn W from domain adversarial training
- Use W to induce initial bilingual dictionary $X, Y = \{x_i, y_i\}^{n}_{i=1}$ using CSLS (Cross-domain Similarity Local Scaling) metric
- Iteratively update, applying
  - $W = UV^T$ where $U\Sigma V^T = \mathrm{SVD}(YX^T)$
    - Also done using the following formula for weight updates:

$$W \leftarrow (1+\beta)W - \beta(WW^T)W$$

  - And finding new $X, Y = \{x_i, z_i\}^{n}_{i=1}$ using CSLS metric
  - Continue till there are no new addition to the dictionary

Cross-lingual word embeddings (continued...)

Cross-domain Similarity Local Scaling (CSLS):

The following is the formula for CSLS:

$$CSLS(Wx_s, y_t) = 2cos(Wx_s, y_t) - r_T(Wx_s) - r_S(y_t)$$

- Here W_x is the transformation of source embedding (x) into the target space.

$$r_T(Wx_s) = \frac{1}{K} \sum_{y \in N_T(Wx_s)} cos(Wx_s, y_t)$$

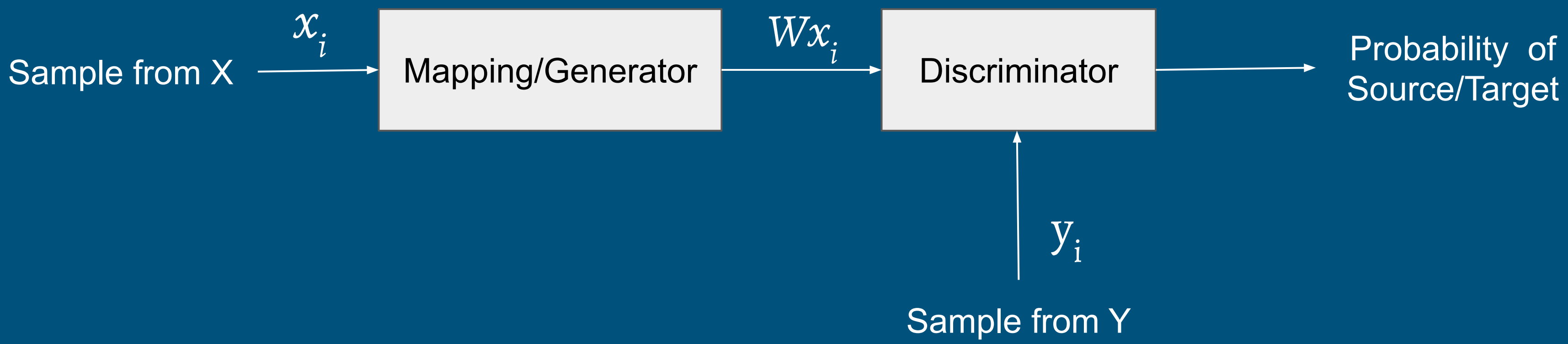
- Here $N^T(Wx_s)$ is used to denote the neighborhood, associated with a mapped source word embedding Wx_s

This process increases the similarity associated with isolated word vectors, but decreases the similarity of vectors lying in dense areas

Cross-lingual word embeddings (continued...)

Adversarial Training:

- Let $X = \{x_1, x_2, x_3 \ldots, x_n\}$ and $Y = \{y_1, y_2, y_3, \ldots, y_m\}$ be two sets of n and m word embeddings coming from a source and a target language respectively.
- A model is trained to discriminate between elements randomly sampled from $WX = \{Wx_1, Wx_2, \ldots, Wx_n\}$ and Y



Cross-lingual word embeddings (continued...)

Codes:

```python
import io
import numpy as np
def load_vec(emb_path, nmax=50000):
    vectors = []
    word2id = {}
    with io.open(emb_path, 'r', encoding='utf-8', newline='\n', errors='ignore') as f:
        next(f)
        for i, line in enumerate(f):
            word, vect = line.rstrip().split(' ', 1)
            vect = np.fromstring(vect, sep=' ')
            assert word not in word2id, 'word found twice'
            vectors.append(vect)
            word2id[word] = len(word2id)
            if len(word2id) == nmax:
                break
    id2word = {v: k for k, v in word2id.items()}
```

Cross-lingual word embeddings (continued...)

Codes:

```
     embeddings = np.vstack(vectors)
     return embeddings, id2word, word2id
def get_nn(word, src_emb, src_id2word, tgt_emb, tgt_id2word, K=5):
    print("Nearest neighbors of \"%s\":" % word)
    word2id = {v: k for k, v in src_id2word.items()}
    word_emb = src_emb[word2id[word]]
    scores = (tgt_emb / np.linalg.norm(tgt_emb, 2, 1)[:, None]).dot(word_emb / 
np.linalg.norm(word_emb))
    k_best = scores.argsort()[-K:][::-1]
    for i, idx in enumerate(k_best):
        print('%.4f - %s' % (scores[idx], tgt_id2word[idx]))
src_path = '/content/en.cross.vec'
#tgt_path = '/content/hi.cross.vec'
tgt_path = '/content/hi.mono.vec'
nmax = 50000  # maximum number of word embeddings to load

src_embeddings, src_id2word, src_word2id = load_vec(src_path, nmax)
tgt_embeddings, tgt_id2word, tgt_word2id = load_vec(tgt_path, nmax)
```

References:

- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. **Efficient Estimation of Word Representations in Vector Space**. In *arXiv preprint arXiv:1301.3781*
- Tomas Mikolov, Quoc V. Le, and Ilya Sutskever, 2013b. **Exploiting Similarities among Languages for Machine Translation**. In *arXiv preprint arXiv:1309.4168v1*
- Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. **Bilingual Word Representations with Monolingual Quality in Mind**. In *NAACL Workshop on Vector Space Modeling for NLP*. Denver, United States, pages 151–159.
- Ivan Vulić and Marie-Francine Moens. 2015. **Bilingual Word Embeddings from Non-Parallel Document-Aligned Data Applied to Bilingual Lexicon Induction**. In *53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, Beijing, China, 719–725.
- Manaal Faruqui and Chris Dyer. 2014. **Improving Vector Space Word Representations Using Multilingual Correlation**. In *14th Conference of the European Chapter of the Association for Computational Linguistics.* Association for Computational Linguistics, Gothenburg, Sweden, 462–471.
- *Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2016.* ***Learning principled bilingual mappings of word embeddings while preserving monolingual invariance****. In 2016 Conference on Empirical Methods in Natural Language Processing, pages 2289-2294*
- *Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2017.* ***Learning bilingual word embeddings with (almost) no bilingual data****. In 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 451-462.*
- Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. 2015. **Normalized word embedding and orthogonal transform for bilingual word translation**. In *Proceedings of NAACL*.
- Peter H Schonemann. 1966. **A generalized solution of the orthogonal procrustes problem**. *Psychometrika*, 31(1):1–10.
- Conneau, Alexis, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2017. **Word translation without parallel data."** In *arXiv preprint arXiv:1710.04087*.