Language modeling and word embeddings

John Arevalo

2016

http://www.mindlaboratory.org

Contents

Outline

Natural Language Understanding

Language Understanding?

Modeling?

Natural Language Understanding

Its all about how likely a sentence is...

- P (obama|president of U.S.)
- P (Good morning|Buenos días)
- P (about fifteen minutes from) $>$ P (about fifteen minuets from)
- P (I saw a bus) $\gg$ P (eyes awe a boss)

P (a man eating a sandwich)

Natural Language Understanding

- A sentence $(x_1, x_2, \ldots, x_T)$
  - Ex: $(\text{the}, \text{cat}, \text{is}, \text{eating}, \text{a}, \text{sandwich}, \text{on}, \text{a}, \text{couch})$
- How likely is this sentence?
- In other words, what is the probability of $(x_1, x_2, \ldots, x_T)$?
  - i.e: $P(x_1, x_2, \ldots, x_T) = ?$

Probability 101

- Joint probability $p(x,y)$
- Conditional probability $p(x|y)$
- Marginal probability $p(x)$ and $p(y)$

- They are related by $p(x,y) = p(x|y)p(y) = p(y|x)p(x)$

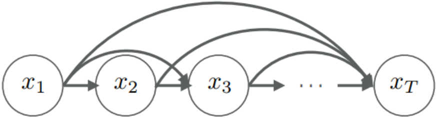
$$x \rightarrow y$$

Language Model as a product of conditionals

- Rewrite $p(x_1, x_2, \ldots, x_T)$ into

$$p(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$

- Graphically,



The goal

Maximize the (log-)probabilities of sentences in corpora:

$$\max \mathbb{E}_D \left[\log P \left(x_1, x_2, \ldots, x_T \right) \right]$$

Outline

n-gram language model

Use Markov assumption: Next word does not depend on all previous words, but only on last n words:

$$
\begin{aligned}
P(x_1, x_2, \ldots, x_T) &= \prod_{t=1}^{T} p(x_t | x_1, \ldots, x_{t-1}) \\
&\approx \prod_{t=1}^{T} p(x_t | x_{\boldsymbol{t-n}}, \ldots, x_{t-1})
\end{aligned}
$$

How to calculate such probabilities? Just counting:

$$
P(x_t | x_{t-1}) \frac{\text{count}(x_{t-1}, x_t)}{\text{count}(x_{t-1})}
$$

An example

$$P\left(x_t|x_{t-1}\right) \frac{\text{count}\left(x_{t-1}, x_t\right)}{\text{count}\left(x_{t-1}\right)}$$

<s> I am sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$P(\texttt{I}\,|\,\texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam}\,|\,\texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am}\,|\,\texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>}\,|\,\texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam}\,|\,\texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do}\,|\,\texttt{I}) = \tfrac{1}{3} = .33$$

Effects of n in the performance

- Ex) $p(\text{i, would, like, to}, \ldots, ., \langle/\text{s}\rangle)$
- Unigram Modelling
  $p(\text{i})p(\text{would})p(\text{like})\cdots p(\langle/\text{s}\rangle)$
- Bigram Modelling
  $p(\text{i})p(\text{would}|\text{i})p(\text{like}|\text{would})\cdots p(\langle/\text{s}\rangle\,|.)$
- Trigram Modelling
  $p(\text{i})p(\text{would}|\text{i})p(\text{like}|\text{i, would})\cdots$

$\vdots$

word	unigram	bigram	trigram	4-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	4.828	0.005	0.000	0.000
average	8.051	4.072	2.634	2.251
perplexity	265.136	16.817	6.206	4.758

Disadvantages

Data sparsity:# of all possible n-grams: $|V|^n$, where $|V|$ is the size of the vocabulary. Most of them never occur.

Training Set:

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Test Set:

- ... denied the offer
- ... denied the loan

$$P\left(\text{offer}|\text{denied the}\right) = 0$$

Disadvantages

False independence assumption: Because in an n-gram language model we assume that each word is only conditioned on the previous n-1 words

False conditional independence assumption

"The dogs chasing the cat bark". The tri-gram probability $P\,(\text{bark}|\text{the cat})$ is very low (not observed in the corpus by the model, because the cat never barks and the plural verb "bark" has appeared after singular noun "cat"), but the whole sentence totally makes sentence.

Outline

Neural language model

~~Non-parametric estimator~~ $\longrightarrow$ parametric estimator

$$\begin{aligned} P\left(x_t | x_{t-n}, \ldots, x_{t-1}\right) &= \xcancel{\frac{\text{count}\left(x_{t-n}, \ldots, x_t\right)}{\text{count}\left(x_{t-1}, \ldots, x_{t-1}\right)}} \\ &= f_\Theta\left(x_{t-n}, \ldots, x_{t-1}\right) \end{aligned}$$

Neural language model

~~Non-parametric estimator~~ $\longrightarrow$ parametric estimator

$$\begin{aligned} P\left(x_t | x_{t-n}, \ldots, x_{t-1}\right) &= \xcancel{\frac{\text{count}\left(x_{t-n}, \ldots, x_t\right)}{\text{count}\left(x_{t-1}, \ldots, x_{t-1}\right)}} \\ &= f_\Theta\left(x_{t-n}, \ldots, x_{t-1}\right) \end{aligned}$$

Somehow, we need numerical representation for words... i.e. **Word vectors**

Word representation

One simple approach is the one-hot or 1-of-K encoding.

$$
\begin{aligned}
\begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0 & 0\end{bmatrix} &\rightarrow \text{I}\\
\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & 0\end{bmatrix} &\rightarrow \text{liked}\\
\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix} &\rightarrow \text{the}\\
\begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix} &\rightarrow \text{hotel}
\end{aligned}
$$

Word representation

One simple approach is the one-hot or 1-of-K encoding.

$$\begin{matrix} [0 & 0 & 1 & 0 & 0 & 0 & 0] & \rightarrow & \text{I} \\ [0 & 1 & 0 & 0 & 0 & 0 & 0] & \rightarrow & \text{liked} \\ [0 & 0 & 0 & 0 & 0 & 0 & 1] & \rightarrow & \text{the} \\ [0 & 0 & 0 & 1 & 0 & 0 & 0] & \rightarrow & \text{hotel} \end{matrix}$$

Drawbacks

- Highly dimensional ($|V|$)
- Representations are orthogonal, so there is no natural notion of similarity in a set of one-hot vectors.

$$\begin{matrix} [1 & 0 & 0 & 0 & 0 & 0 & 0] & \rightarrow & \text{motel} \\ [0 & 0 & 0 & 1 & 0 & 0 & 0] & \rightarrow & \text{hotel} \end{matrix}$$

Word representation

How to learn a continuous representation for words?:

$$\begin{array}{ccc}
\begin{bmatrix}0.3 & 0.2 & 0.8 & 0.1\end{bmatrix} & \rightarrow & \text{I} \\
\begin{bmatrix}0.4 & 1.2 & 0.1 & 0.9\end{bmatrix} & \rightarrow & \text{liked} \\
\begin{bmatrix}1.3 & -2.1 & 0 & 1.2\end{bmatrix} & \rightarrow & \text{the} \\
\begin{bmatrix}0.5 & 1.4 & 0.3 & -0.4\end{bmatrix} & \rightarrow & \text{hotel} \\
\begin{bmatrix}0.3 & 1.0 & 0.6 & -0.1\end{bmatrix} & \rightarrow & \text{motel}
\end{array}$$

Word representation

How to learn a continuous representation for words?:

$$\begin{array}{rcl} \begin{bmatrix}0.3 & 0.2 & 0.8 & 0.1\end{bmatrix} & \rightarrow & \text{I} \\ \begin{bmatrix}0.4 & 1.2 & 0.1 & 0.9\end{bmatrix} & \rightarrow & \text{liked} \\ \begin{bmatrix}1.3 & -2.1 & 0 & 1.2\end{bmatrix} & \rightarrow & \text{the} \\ \begin{bmatrix}0.5 & 1.4 & 0.3 & -0.4\end{bmatrix} & \rightarrow & \text{hotel} \\ \begin{bmatrix}0.3 & 1.0 & 0.6 & -0.1\end{bmatrix} & \rightarrow & \text{motel} \end{array}$$

With the context of each word

Distributional hypothesis

You 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)

One of the most successful ideas of modern NLP

government debt problems turning into **banking** crises as has happened in

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

↖ These words will represent *banking* ↗

- Distributed: Represent a word as a point in a vector space (e.g. as a vector).
- Distributional: The meaning of a word is given by the context where it appears.

Distributed representations with NN

Distributed representations of words can be obtained from various neural network based language models:

- Feedforward neural net language model
- Recurrent neural net language model

Neural network language model

Topics: Neural Language Modelling

$$p(x_t|x_{t-n},\ldots,x_{t-1}) = f_\Theta\left(x_{t-n},\ldots,x_{t-1}\right)$$

- Building a neural language model (Bengio et al., 2000)

(1) *1*-of-*K* encoding of each word $x_{t'}$

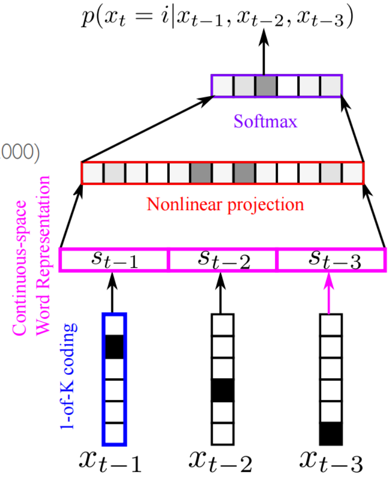
(2) Continuous space word representation

$$s_{t'} = W^\top x_{t'}, \text{ where } W \in \mathbb{R}^{|V|\times d}$$

(3) Nonlinear hidden layer

$$h = \tanh(U^\top [s_{t-1}; s_{t-2}; \cdots; s_{t-n}] + b)$$

$$\text{, where } U \in \mathbb{R}^{nd\times d'} \text{ and } b \in \mathbb{R}^{d'}$$



Neural network language model

Topics: Neural Language Modelling

$$p(x_t|x_{t-n},\ldots,x_{t-1}) = f_\Theta\left(x_{t-n},\ldots,x_{t-1}\right)$$

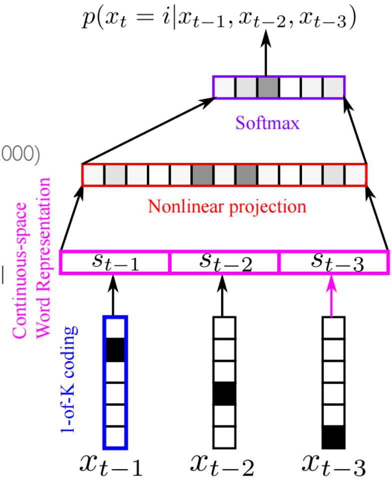
- Building a neural language model (Bengio et al., 2000)

(1)Unnormalized probabilities

$$y = Vh + c, \text{ where } V \in \mathbb{R}^{|V|\times d'} \text{ and } c \in \mathbb{R}^{|V|}$$

(2)Softmax normalization

$$p(x_t = i|x_{t-n},\ldots,x_{t-1}) = \frac{\exp(y_i)}{\sum_{j=1}^{|V|}\exp(y_j)}$$



Neural network language model complexity

For all the following models, the training complexity is proportional to:

$$O = E \times T \times Q$$

where E is number of the training epochs, T is the number of the words in the training set and Q is defined further for each model architecture.
The computational complexity, defined as the number of parameters that need to be accessed to fully train the model, In a NNLM is given by

$$Q = n \times d + n \times d \times d' + d' \times |V|$$

with n the size of the context, d the dimensionality of the word space, d' the number of units in the hidden layer and $|V|$ the size of the vocabulary.

Computational cost

The training complexity of the feedforward NNLM is high:

- Propagation from projection layer to the hidden layer
- Softmax in the output layer

Using this model just for obtaining the word vectors is very inefficient.

Outline

Goal

"The main goal of this paper is to introduce techniques that can be used for **learning high-quality word vectors from huge data sets** with billions of words, and with millions of words in the vocabulary"

Efficient learning

The full softmax can be replaced by:

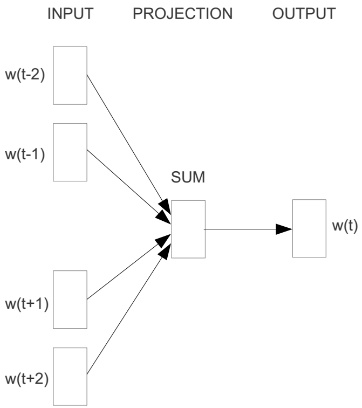
- Hierarchical softmax
- Negative sampling

We can further remove the hidden layer: for large models, this can provide additional speedup $1000x$

- Continuous bag-of-words model
- Continuous skip-gram model

CBOW

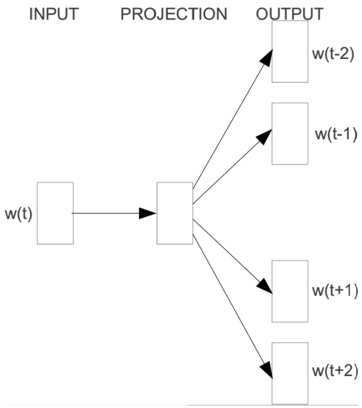
Predicts the current word given the context



Complexity

$$Q = n \times d + d \log_2 (|V|)$$

Skip-gram



$$Q = C \times (d + d \times \log_2 (|V|))$$

For each training word we will select randomly a number R in range $< 1; C >$and then use R words from history and R words from the future.

Skip-gram formulation

The training objective of the Skip-gram model is to find word representations that are useful for predicting the surrounding words in a sentence or a document. More formally, given a sequence of training words $w_1, w_2, w_3, \ldots, w_T$, the objective of the Skip-gram model is to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

where c is the size of the training context (which can be a function of the center word w_t). Larger c results in more training examples and thus can lead to a higher accuracy, at the expense of the training time. The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^\top v_{w_I}\right)}$$

where v_w and v'_w are the "input" and "output" vector representations of w, and W is the number of words in the vocabulary. This formulation is impractical because the cost of computing $\nabla \log p(w_O|w_I)$ is proportional to W, which is often large (10^5–10^7 terms).

Efficient learning - Summary

- Efficient multi-threaded implementation of the new models greatly reduces the training complexity.
- The training speed is in order of 100K - 5M words per second.
- Quality of word representations improves significantly with more training data.

Outline

Criterion

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Results

Table 2: *Accuracy on subset of the Semantic-Syntactic Word Relationship test set, using word vectors from the CBOW architecture with limited vocabulary. Only questions containing words from the most frequent 30k words are used.*

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Table 3: *Comparison of architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [20]*

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

Results

Table 4: *Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.*

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Results

Table 5: *Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Compute time results

Table 6: *Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Outline

Additive compositionality

$$vector('smallest') - vector('small') = vector('biggest') - vector('big')$$

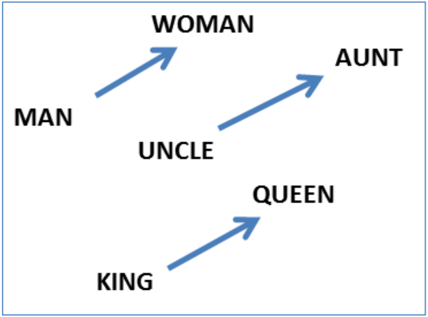
$$vector('smallest') = vector('biggest') - vector('big') + vector('small')$$

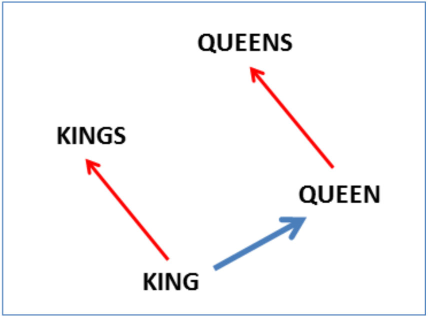
Expression	*Nearest tokens*
Czech + currency	koruna, Czech crown, Polish zloty, CTK
Vietnam + capital	Hanoi, Ho Chi Minh City, Viet Nam, Vietnamese
German + airlines	airline Lufthansa, carrier Lufthansa, flag carrier Lufthansa
Russian + river	Moscow, Volga River, upriver, Russia
French + actress	Juliette Binoche, Vanessa Paradis, Charlotte Gainsbourg

Linguistic regularities

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

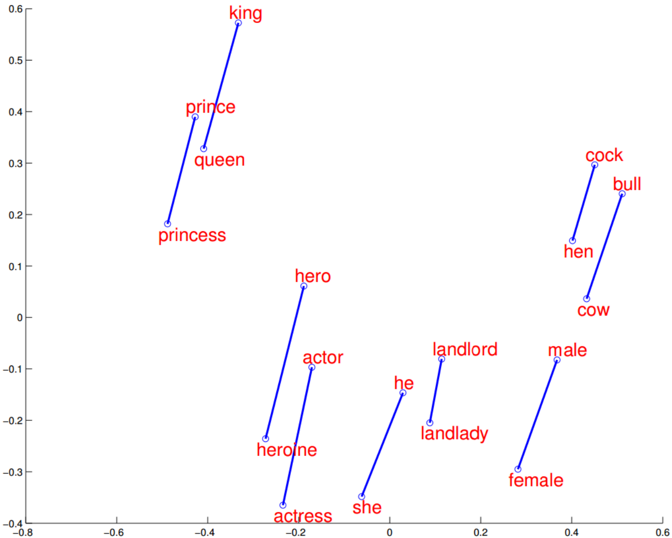
Linguistic regularities



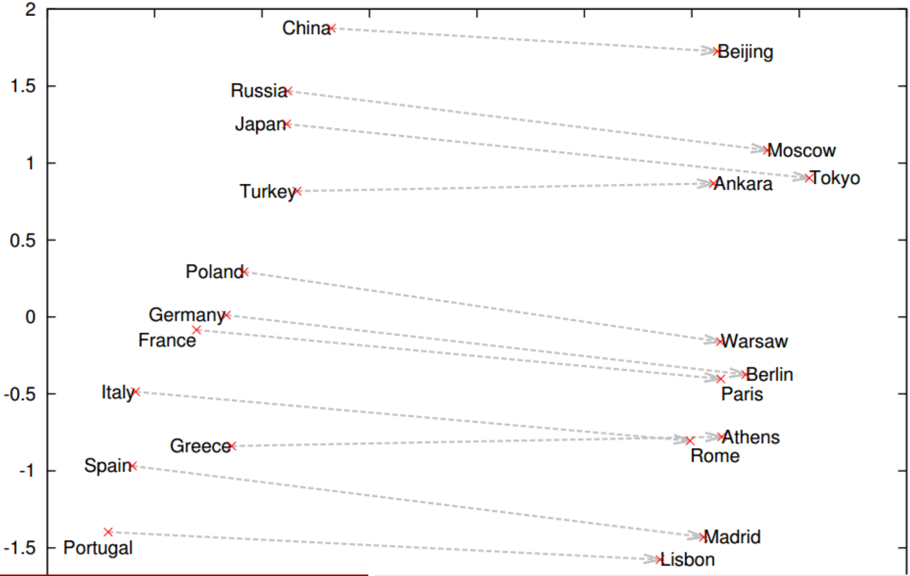


The word vector space implicitly encodes many regularities among words.

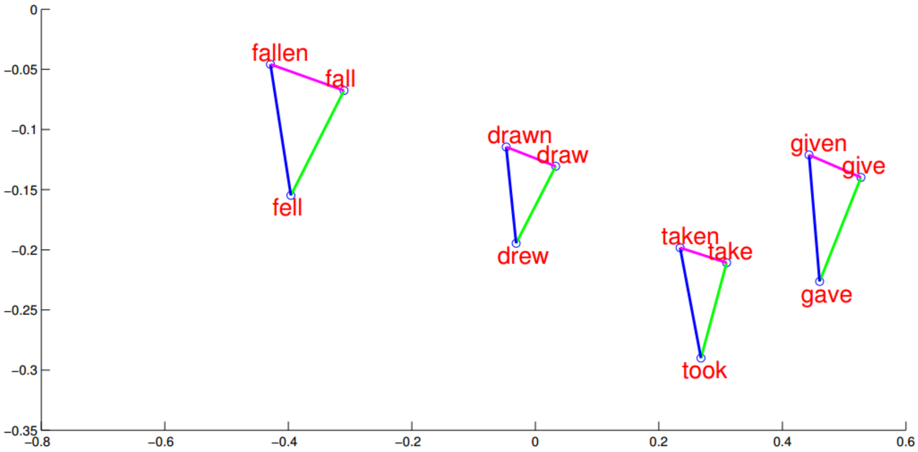
Visualization in word space



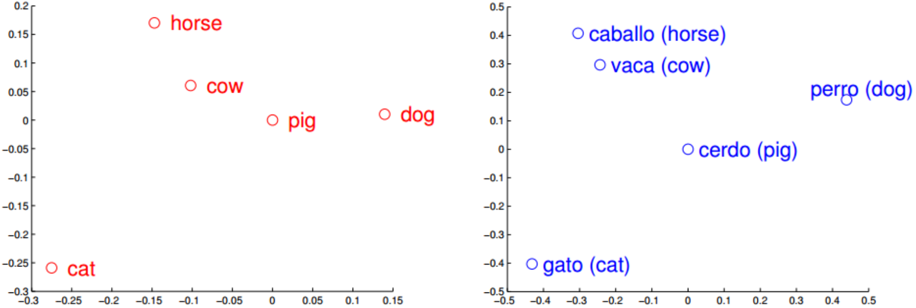
Visualization in word space



Visualization in word space



Machine translation



- For translation from one vector space to another, we need to learn a linear projection.
- Small starting dictionary can be used to train the linear projection.
- Then, we can translate any word that was seen in the monolingual data.

Limitations

- How to represent a phrases or documents?
- Ignores the order of the elements.

In summary

Representation of text is very important for performance of many real-world applications. The most common techniques are:

- N-grams: Bag-of-words (Based on 1-of-N coding)
- Continuous representations
  - Feed-forward Neural language models
  - word2vec
  - RNN Models

Heavily based on...

- Live demo: https://ronxin.github.io/wevi/
- Language modeling: https://web.stanford.edu/class/cs124/lec/languagemodeling.p
- C. Manning - Human Language & vector words: http://videolectures.net/deeplearning2015_manning_language_
- K. Cho - Deep Natural Language Understanding http://videolectures.net/deeplearning2016_cho_language_u

Further readings

- Original papers: https://arxiv.org/abs/1301.3781,https://arxiv.org/abs/1310.4546
- word2vec Explained: https://arxiv.org/abs/1402.3722