CS 3750 Advanced Machine Learning

Word Models

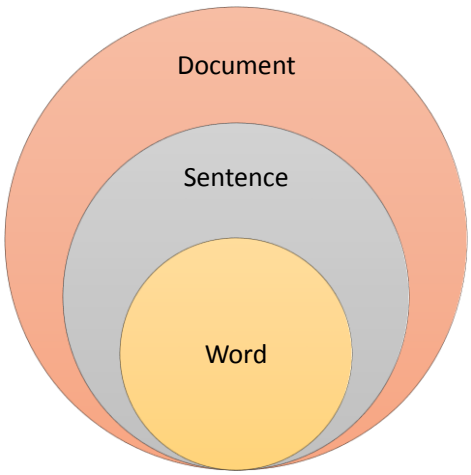
Presented by: Ahmed Magooda

10/04/2018

Text representation

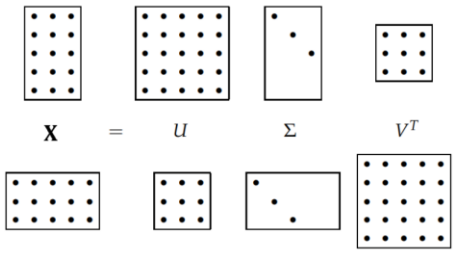
In the field of Natural language processing text can be presented using three levels of granularity.

1. Documents
2. Sentences
3. Words



Documents

Documents



We represented documents in different ways:

1. **Latent semantic indexing (LSI)**
    1. We constructed a co-occurrence matrix of words versus documents (**X**).
    2. Use SVD to decompose the Document-Word matrix into two lower dimension matrices (U, V) (Words and Documents)
    3. Each raw in the matrix represent a document, where similarity between documents can be calculate using cosine similarity

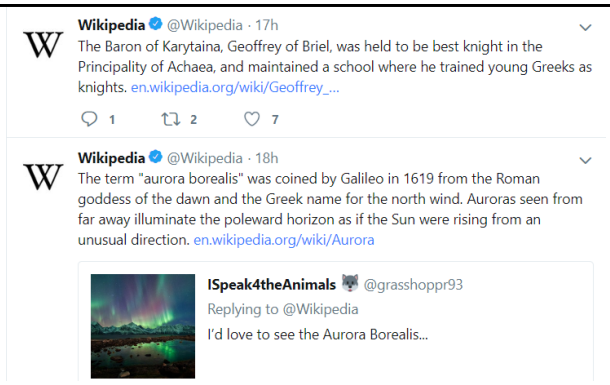
Documents Cont.

1. **LDA**
   1. LDA infers a list of latent topics from a set of documents.
   2. Each document is represented as a probability distribution over topics by calculating P(Z|D) where Z is topic and D is document.

	Z1	Z2	Z3	Z4	..	ZN
D1	0.1	0.25	0.005	0.001	..	0.1
D2	0.001	0.3	0.1	0.025	..	0.2

   3. Similarity between documents can be measured by how close the two distributions are.

Problem Definition

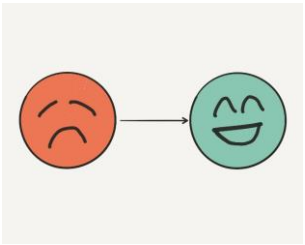


- Can we do everything on document level ?
- What about something like tweets, can it be treated as documents ?
- How can we measure similarity between sentences ?
- How can we handle tasks like automatic assessment of short answers, where an answer is just a couple of words.
- We can use a different level of granularity for our analysis.

Words

Words

- Words are the smallest meaningful unit of text.
- What we get from words:
  - **Meaning**
  - **Part of speech (POS):** Category of words which have similar grammatical properties.
  - **Sentiment:** Is it a positive or a negative word (good, bad, like, hate, etc..)



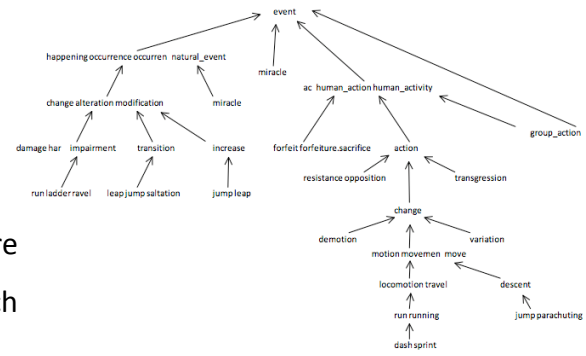
Word Models

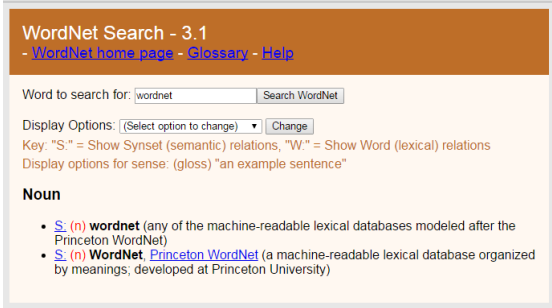
- Word models are a way to represent words of a language and the relations between them.
- Word models can help identifying if two words are semantically related.
  - Synonyms (King, ruler, emperor)
  - Antonyms (Good, Bad)
  - Hyponym (fry, stir-fry)
- Word models can also help in measuring word similarity.

Early work.

- WordNet [Princeton University]:
  - A large lexical database of words. Words are grouped into set of synonyms (synsets), each expressing a distinct concept.
  - Provides Word senses, POS, semantic relations between words.
  - Exists for many languages.
  - Needs manual effort to build.

Image credit: University of Princeton





Similarity in WordNet

To calculate similarity in word net, different approaches that depend on the tree structure were proposed:

- **Resnik:** $Sim_{Resnik}(W_1, W_2) = -\log P\big(LCS(W_1, W_2)\big)$, LCS is the least common parent (subsumer).
- **Lin:** $Sim_{Lin}(w_1, w_2) = \frac{2\ x\ \log P(LCS(w_1, w_2))}{(\log P(w_1) + \log P(w_2))}$
- **Leacock & Chodorow (Lch):** $Sim_{Lch}(w_1, w_2) = -\log(Length(w_1, w_2)/(2\ x\ D))$
- **Lesk:** Uses The number of common words in the WordNet gloss definition of Word1 and gloss definition of Word2.

WordNet conclusion

Pros:

- Very reliable as it is manually constructed.
- Provide different word properties along side gloss definition.
- Relation between words are explicit.

Cons:

- Low coverage because of manual effort.
- Hard to update with new words.
- Hard to represent words as features to use in different ML applications.

Word Embeddings

Word Embeddings

Motivation:

To overcome the limitations of dictionary based approaches like word net. Word embeddings can be used.

Definition:

Word embedding is a set of different techniques that can be used to map words into a numerical vector representation of some dimension k.

Each word is represented using a vector of either real or integer values. Each dimension can have an explicit or a latent meaning.

Example:

	d1	d2	d3	..	dk
W_i	**0.2**	**0.75**	**1.2**		**3.2**
W_j	**0.54**	**0.87**	**0.95**		**2.1**

Why word embeddings.

- Word embeddings can be produced with no manual effort, it can be done through statistical analysis over a collection of raw text.
- Can be trained in task based fashion, so a set of word embeddings tailored for your task can be generated. For example training a neural network for sentiment classification, the resultant embeddings can be very suitable to capture sentiment in words.
- Can capture semantic and syntactic features of words, embeddings trained based on small window of words can capture more syntactic features. Embeddings trained using large window of words can capture more semantic features.
- Easier to incorporate in a machine learning approaches, as words can be encoded in a vector of numerical values.

Word Embeddings

Different types of word embeddings can be used.

1. Local representation:
   - Produces a representation that encodes information of only the word in concern.
   - Since vectors only encode information of single word, no relation between words can be inferred.
   - Example : **"One-hot vector representation"**
     - Each vector is of the size of the whole vocabulary, where all dimensions are set to 0 except for the dimension corresponding to the word in concern.
     - This representation is commonly used as an input to Neural network models.
     - Word similarity can't be estimated using such representation. Two vectors of Cat and Kitten have a cosine similarity of zero.

	1	2	3	..	i	..	j	…	v
W_i = cat	0	0	0	0	1	0	0	0	0
W_j = Kitten	0	0	0	0	0	0	1	0	0

Word Embeddings

- Distributed representations:
  - Since local representations only encode word related information, distributed representations produce vectors that encodes word information and information relating surrounding words.
  - Generated vectors encode semantic and/or syntactic information of words and their relation to other words in corpus/document.
  - Can be useful in calculating word to word similarity
    - For example LSA, in LSA lower dimension word matrix is also generated. The generated vectors encode relation to other words in form of within document cooccurrences. Similarity between words can measured using cosine similarity between word vectors.

Models of Word Embeddings

Models of word embeddings

- There are different types of models for word embeddings, it depends on the method used to train (produce) these embeddings.
- One way of classifying models:
  - Count based
  - Prediction based
  - Context Window based

Models of Word Embeddings

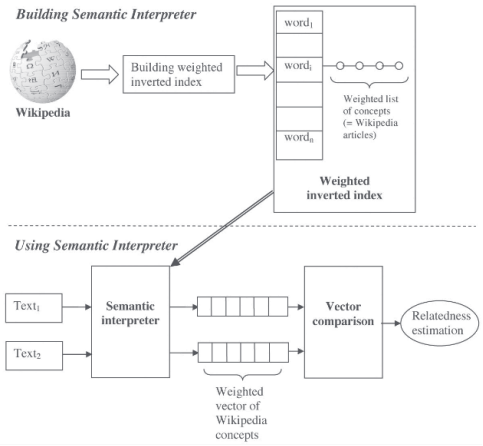
Count based

Models of Word Embeddings (Count based)

- Count based methods usually produce a matrix of cooccurrences between words and (words, documents, concepts, etc..)
- These matrices tend to be of a very large dimensions and sometimes suffer from sparsity. To overcome this issue, a form of dimensionality reduction can be used (SVD, PCA, LDA, …) to map vectors to a space of lower dimensions.
- Similarity between words can then be measured using distance measure like (Cosine, Euclidean, Manhattan, etc..)
- Examples:
  - Latent semantic analysis (LSA)
  - Explicit semantic analysis (ESA) (Gabrilovich, E., & Markovitch, S. (2007, January))
  - Extracting DIstributionally related words using CO-occurrences (DISCO)(Kolb, P. (2008))
  - An Efficient Framework for Constructing Generalized Locally-Induced Text Metrics

Explicit semantic analysis

- Similar words most likely appear with the same distribution of topics.
- ESA represents topics by Wikipedia concepts (Pages). ESA use Wikipedia concepts as dimensions to construct the space in which words will be projected
- For each dimension (concept), words in this concept article are counted.
- Inverted index is then constructed to convert each word into a vector of concepts.
- The vector constructed for each word represents the frequency of its occurrences within each (concept).



Extracting DIstributionally related words using CO-occurrences (DISCO)

- DISCO representation is based on scanning the corpus by a window of variable width (preferred ±3).
- Co-occurrence matrix is constructed using the unique words as rows. Columns are the unique combination of (word, position) pair in the scanning window.
- The Matrix is then filled by the frequency of co-occurrence between row words and column words on that position.
- The absolute values in the matrix are then converted using the formula.

$$log \frac{(f(w,r,w') - 0.95) f(*,r,*)}{f(w,r,*) f(*,r,w')}$$

- Where:
  - w and w' stand for words. r is a window position
  - f is the frequency of occurrence, and "0.95" is an adjustment factor

DISCO

- Unlike basic word-word cooccurrence matrix, DISCO accounts for position of words when counting cooccurrences.
- For example using a window of size ±1, we construct a matrix with size Vx2V where V is the vocabulary size. A sentence like "$W_1W_2W_3W_4W_5$" can be represented as follows.

	W1, -1	W1, +1	W2, -1	W2, +1	W3, -1	W3, +1	…	..	Wv, +1
W1	0	0	0	**1**	0	0	0	0	0
W2	**1**	0	0	0	0	**1**	0	0	0
W3	0	0	**1**	0	0	0	0	0	0
W4	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0
Wv	0	0	0	0	0	0	0	0	0

Framework for Constructing Generalized Locally-Induced Text Metrics

- Construct a graph based on cooccurrences of words in the same sentence. Where each node is a word and edges represent within sentence cooccurrence.
- Edges are weighted with the number of documents, in which the two words appeared in the same sentence.
- With the assumption that there is an underlying function *r(.)* that can map a vector to a different space. Given two words that are related according to some task. Applying such function over the two vectors should produce new vectors that are close in the new space.
- So to get an estimation for the mapping function, graph Laplacian is applied over the weighted adjacency matrix. Then spectral decomposition (eigen decomposition) is applied to produce the new vector space.

Models of Word Embeddings

Prediction based

Models of Word Embeddings (Prediction based)

- Prediction based models can produce word embeddings while performing some prediction task. The weights optimized during training can then be used as embeddings.
- For example training a language model. In a language model training, a sequence of words is passed as input and the model tries to predict the next word.
- Neural networks are commonly used for language modeling. After training the set of weights are used as word embeddings.
- Two NN based LMs are:
  - Feed forward NN LM
  - Recurrent NN LM

Feed forward NN for LM (Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003))

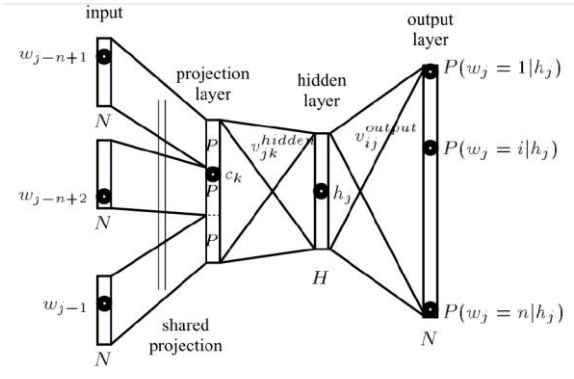


Image credit: Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010)

The objective of the training is to find θ to minimize

$$L(\theta) = -\frac{1}{T}\left(\sum_j Log\ P(w_j|w_{j-1}, \dots, w_{j-n+1})\right) + R(\theta)$$

Where R(θ) is regularization term.

- $P(w_j|w_{j-1}, \dots, w_{j-n+1}) = \frac{e^{y_{wj}}}{\sum_{i!=j} e^{y_{wi}}}$
- $Y = b + W\ tanh(d + Hx)$
- $x = (C(w_{j-1}), C(w_{j-2}), \dots, C(w_{j-n+1}))$

b is the output layer bias
d is hidden layer bias
W is hidden-output weights
H is the hidden layer weights.

Recurrent NN for LM (Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010))

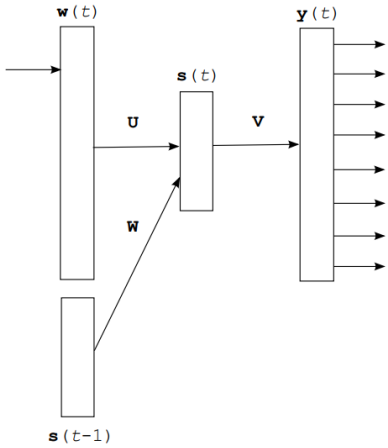


Image credit: Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010)

The objective of the training is the same as the feedforward NN.

$$s(t) = f\big(Uw(t) + Ws(t-1)\big)$$

$$y(t) = g(Vs(t))$$

Where ***f*** is the sigmoid function and **g** is the Softmax function

Models of Word Embeddings

Context window based

Models of Word Embeddings (Context window based)

- In these models we iterate over a corpus of raw text using a window of fixed size. The model uses information from within the window to learn word embeddings.
- Smaller windows allow learning more syntactic features, while larger windows are good for learning more semantic features.
- Examples:
  - Skip-gram (Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013).)
  - CBOW (Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013).)

Skip-gram

- Skip gram model aims to predict context words within a window given the center word. So for each word t = 1…T in the corpus, the goal is to predict surrounding words in a window of radius m.
- Objective function is maximizing the probability of correct context words given the center word.

Skip gram Cont.

- Input is one-hot vector.
- Each word is represented with two different vectors (context vector, output vector). Each vector is N dimensions.
- The model parameters are the word vectors.
- We need to learn 2 *x* V *x N* parameters

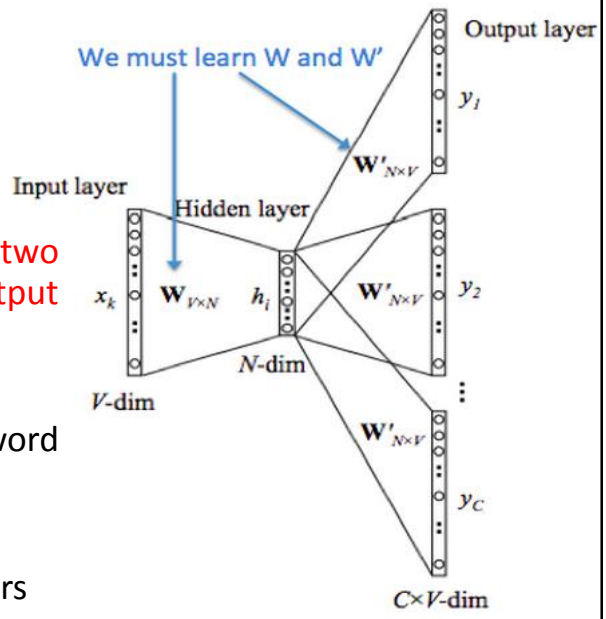


Image credit: Francois Chaubard, Michael Fang, Guillaume Genthial, Rohit Mundra, Richard Socher

Skip-gram Cont.

- Objective function is maximizing the probability of correct context word given the center word.
- $L(\theta) = \prod_{t=1}^{T} \prod_{\substack{j=-m \\ J!=0}}^{j=m} P(w_{t+j}|w_t;\theta)$
- Which is the same as minimizing
- $L'(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=-m}^{j=m} \log P\left(w_{t+j}\middle|w_t\right)$
- Θ is the set of variable that will be optimized (e.g. W, W').

Skip-gram Cont.

- $P(w_{t+j}|w_t) = P(O|C) = \frac{\exp(u_o^T.v_c)}{\sum_{w=1}^{V}\exp(u_w^T.v_c)}$

- o is the **output** word index, c is the **center** word index.
- v_c is the center word vector.
- u_o is the output word vector.

- Now we need to calculate the gradient to update the parameters.

Skip-gram Cont.

- First we calculate how to change the center vector.
- $\frac{\partial}{\partial v_c} log \frac{\exp(u_o^T.v_c)}{\sum_{w=1}^{V}\exp(u_w^T.v_c)}$

$= \frac{\partial}{\partial v_c}\log\exp(u_o^T.v_c) - \frac{\partial}{\partial v_c}\log\sum_{w=1}^{V}\exp(u_w^T.v_c)$

Using chain rule

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

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

$= u_o - \frac{1}{\sum_{w=1}^{V}\exp(u_w^T.v_c)}.\sum_{x=1}^{V}\exp(u_x^T.v_c)\,u_x = u_o - \sum_{x=1}^{V}\frac{\exp(u_x^T.v_c)\,u_x}{\sum_{w=1}^{V}\exp(u_w^T.v_c)} = u_o - \sum_{x=1}^{V}P(x|c)\,u_x$

We calculate the change in each output vector the same way.

CBOW

- In this model given a window of 2m words we predict the word in the center.
- Objective function is maximizing the probability of correct center word given the context words.
- The context vectors are averaged before using in prediction.

$$v_c = \frac{w_{t-m} + w_{t-m-1} + \ldots + w_{t+m}}{2m}$$

CBOW Cont.

- Each word is represented with two different vectors (context vector, output vector). Each vector is N dimensions.
- The model parameters are the word vectors.
- We need to learn 2 *x* V *x N* parameters

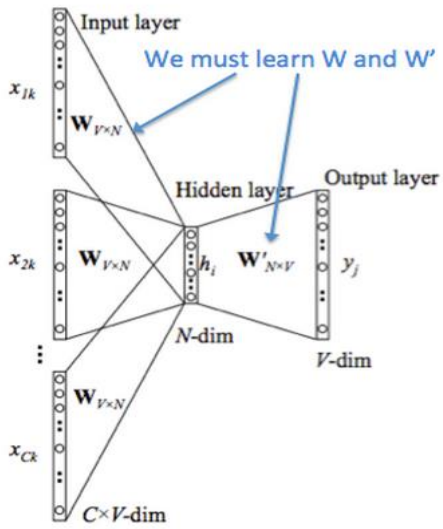


Image credit: Francois Chaubard, Michael Fang, Guillaume Genthial, Rohit Mundra, Richard Socher

CBOW Cont.

- Objective function is maximizing the probability of correct output word given the context words.

- $L(\theta) = \prod_{t=1}^{T} P(w_t|w_{t-m}, w_{t-m-1}, \dots, w_{t+m}; \theta) = \prod_{t=1}^{T} P(w_t|v_c; \theta)$
- Which is the same as minimizing
- $L'(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \log P(w_t|v_c)$

- Θ is the set of variable that will be optimized (e.g. W, W').

Hierarchical Softmax

- What is wrong with regular softmax ?
  - For each sample, we need to normalize over the whole vocabulary. "order of millions"

- Hierarchical softmax:
  - Leaves are words
  - There is a unique path from root to each word.
  - No output representation for words
  - Every intermediate node has a vector representation.

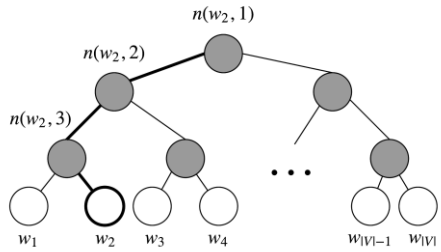
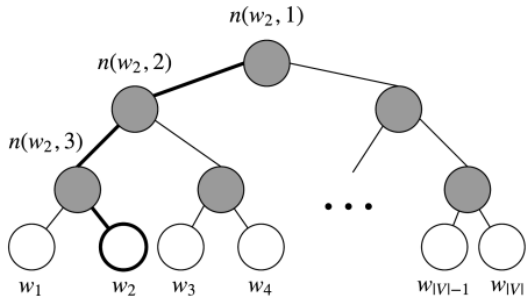


Image credit: Francois Chaubard, Michael Fang, Guillaume Genthial, Rohit Mundra, Richard Socher

Hierarchical Softmax

- How to calculate probability of a word given a vector P(w|w_i)
  - The probability of random walk from root to the desired word. (Log(V) steps).
  - $P(w|w_i) = \prod_{j=1}^{L(w)-1} \sigma\left(\left[n(w,j+1) =? \; LeftChild\left(n(w,j)\right)\right].v_{n(w,j)}^{T} v_{wi}\right)$
  - Where:
    - L(w) it the length from root to word w
    - n(w,j) is the node on distance j-1 from the root on the path of word w
    - [x] = 1 if x is true, and -1 if false. The condition here means is the next node on the path to w going to be left child or not.
    - σ is the sigmoid function
    - $v_{n(w,j)}^{t}$ is the vector corresponding to the intermediate node.

Example:



- If we take w_2, The path should be (Left-> left -> right)

$$P(w_2|w_i) = P(n(w_2|1), Left).\, P(n(w_2|2), Left).\, P(n(w_2|3), Rigth)$$
$$= \sigma\left(v_{n(w_2,1)}^{T} v_{wi}\right).\, \sigma\left(v_{n(w_2,2)}^{T} v_{wi}\right).\, \sigma\left(-v_{n(w_2,3)}^{T} v_{wi}\right)$$

- The objective is still to minimize the negative log likelihood $-\log P(w_2|w_i)$. However instead of updating output vectors of words, we update the vectors of nodes on the path to the word.

Negative sampling

- Instead of summing over the whole vocabulary in every training step. We sample some negative samples to use.
- Sampling can be done from a distribution $P_n(w)$, with probabilities correspond to the vocabulary unigram probabilities.

Negative sampling Cont.

- The new objective to optimize: given a word w, and context c
  - We want to maximize the probability of the word and context came from the same training data, if that is the case. P(D=1|w,c)
  - Maximize the probability of the word and context not from the same training data, if that is the case. P(D=0|w,c)
  - Model P(D=1|w,c) using sigmoid function. P(D=1|w,c) = $\sigma(v_c^T v_w)$ = $\frac{1}{1+ e^{-v_c^T v_w}}$

Negative sampling Cont.

- The new objective function would be Maximizing:

$\prod_{w,c \in D} P(D = 1|w, c) \prod_{w,c \in D'} P(D = 0|w, c)$

$= \prod_{w,c \in D} P(D = 1|w, c) \prod_{w,c \in D'} (1 - P(D = 1|w, c))$

$= \sum_{w,c \in D} \log(\frac{1}{1+ e^{-u_c^T v_w}}) + \sum_{w,c \in D'} \log(1 - \frac{1}{1+ e^{-u_c^T v_w}})$, as $\sigma(x) + \sigma(-x) = 1$

$= \sum_{w,c \in D} \log(\frac{1}{1+ e^{-u_c^T v_w}}) + \sum_{w,c \in D'} \log(\frac{1}{1+ e^{u_c^T v_w}})$

So we are minimizing:

$$-\sum_{w,c \in D} \log\left(\frac{1}{1+ e^{-u_c^T v_w}}\right) - \sum_{w,c \in D'} \log(\frac{1}{1+ e^{u_c^T v_w}})$$

$$-\sum_{w,c \in D} \log \sigma(u_c^T v_w) - \sum_{w,c \in D'} \log \sigma(-u_c^T v_w)$$

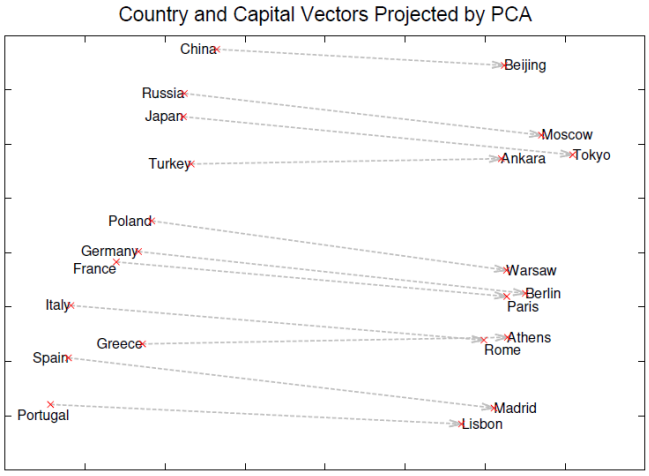
Evaluation of Word Embeddings

Evaluating (CBOW, Skipgram) Word embeddings

- Intrinsic evaluation:
  - How good are the features of the embeddings.
- Extrinsic evaluation:
  - How useful the embeddings are in other downstream tasks.

Intrinsic Evaluation

- High quality word embeddings should be able to produce a form of clusters based on word similarity. So similar words should appear close to each others.
- Visualizing CBOW embeddings in lower dimensions, we can see that countries are aligned with their corresponding capitals in the other side on an almost similar distance.



Intrinsic Evaluation

- Analogy task:
  - The task was proposed in (Mikolov etal. (2013))
  - For a specific word relation, given a,b,y find x so that "*a is to b, as x is to y*".
    - For example given the relation between "brother-sister", what is the word that has the same relation to grandson ? "granddaughter".

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

Extrinsic Evaluation

Word embeddings can be evaluated extrinsically by measuring the outcome of using word embeddings on different downstream tasks like:

- Sentiment analysis.
- Part of speech tagging.
- Parsing.
- Question answering
- Automatic short answer grading.
- etc..

Extrinsic Evaluation: Automatic Short answer grading.

In this task, the aim is to automatically score a student answer in the light of a model answer.

Question : why ecosystem is characterized by multiple components ?

Model Answer : Because the system consists of: non-living factors including physical factors and chemical agents. The factors include living organisms and food-producing organisms consuming food and objects analyst

Student Answer : There are non-living factors such as physical and chemical factors, and there are factors that vary from live objects produced and consumed and analyzed (5)

Student Answer: Multiple components of the ecosystem where there are objects ground, sea creatures and inanimate objects interact with each other. (0)

	RMSE	Correlation
IAA	0.659	0.586
Graph Alignment SVM	0.998	0.518
Manually crafted features	NA	0.50
Word embeddings (Skipgram, CBOW) + Cosine	0.95	0.53
Word embeddings (Skipgram, CBOW) + SVR	**0.79**	**0.7**

Conclusion

- Document representation is helpful but not always applicable.
- Word embeddings can be used to measure similarity between words.
- Word embeddings can be trained by different means.
- Skipgram, CBOW models are faster to train than regular language model based ones.
- Evaluating Skipgram, CBOW models intrinsically and extrinsically proved the high quality of these two representations.

Software packages.

- **Gensim** in Python for
  - LSA
  - Skipgram
  - CBOW
- DISCO API in Java.
  https://www.linguatools.de/disco/disco_en.html#download
- Pytorch, Tensorflow in python for:
  - Feed forward NNs
  - RNNs

References

- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality.
- Mikolov, T., Yih, W. T., & Zweig, G. (2013, June). Linguistic Regularities in Continuous Space Word Representations.
- Kolb, P. (2008). Disco: A multilingual database of distributionally similar words.
- Princeton University "About WordNet." WordNet. Princeton University. 2010. http://wordnet.princeton.edu
- Gabrilovich, E., & Markovitch, S. (2007, January). Computing semantic relatedness using wikipedia-based explicit semantic analysis.
- Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis.
- Rong, X. (2014). word2vec parameter learning explained.
- Stanford CS224n: Natural Language Processing with Deep Learning notes.
- CMU CS11-747 Neural Networks for NLP notes
- Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010)
- Mikolov, T., Kopecky, J., Burget, L., & Glembek, O. (2009, April). Neural network based language models for highly inflective languages.
- Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model.

Questions ?

Thanks