Word Embedding

Yaqiang Yao

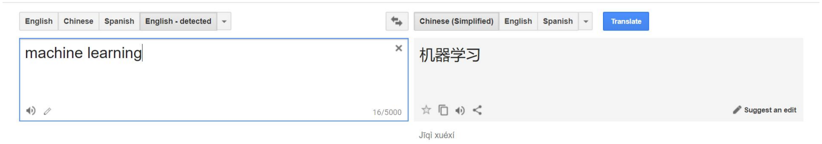
School of Computer Science and Technology
University of Science and Technology of China
Hefei China

April 25, 2018

Outline

Text Processing

1. You open Google and search for a news article on the ongoing Champions trophy and get hundreds of search results in return about it.
2. Nate silver analysed millions of tweets and correctly predicted the results of 49 out of 50 states in 2008 U.S Presidential Elections.
3. You type a sentence in google translate in English and get an Equivalent Chinese conversion.



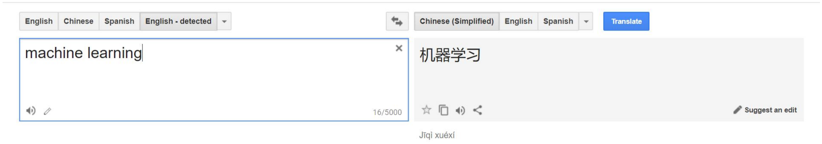
machine learning

机器学习

Jīqì xuéxí

Text Processing

1. You open Google and search for a news article on the ongoing Champions trophy and get hundreds of search results in return about it. (Clustering)
2. Nate silver analysed millions of tweets and correctly predicted the results of 49 out of 50 states in 2008 U.S Presidential Elections. (Classification)
3. You type a sentence in google translate in English and get an Equivalent Chinese conversion. (Machine Translation)



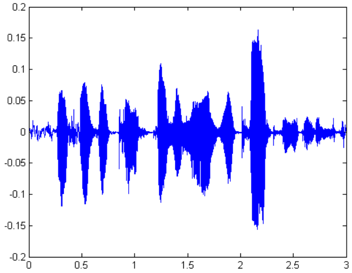
machine learning

机器学习

Jīqì xuéxí

Motivation

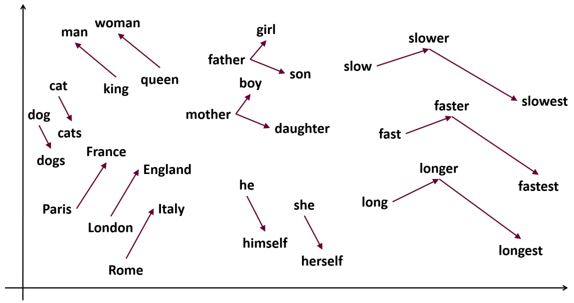
Image



Speech

- Humans can deal with text format quite intuitively
- How do you make a computer understand that “Apple” in “Apple is a tasty fruit” is a fruit that can be eaten and not a company?

Motivation



- The answer is creating a representation for words that capture their meanings, semantic relationships and the different types of contexts they are used in.
- All of these are implemented by using Word Embeddings or numerical representations of texts so that computers may handle them.

Basics

- Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text.
- Notations
  1. A sentence = "Word Embeddings are Word converted into numbers"
  2. A word may be "Embeddings" or "numbers" etc.
  3. A dictionary may be the list of all unique words in the sentence. So, a dictionary may look like ['Word','Embeddings','are','Converted','into','numbers']

Count Vector (Bag-of-Words)

- Consider a Corpus $\mathcal{C}$ of D documents $\{d_1, d_2, \cdots, d_D\}$ and N unique tokens extracted out of the corpus C.
- The N tokens will form our dictionary and the size of the Count Vector matrix $\mathbf{M}$ will be given by $N \times D$.
- Each row in the matrix $\mathbf{M}$ contains the frequency of tokens in document d_i.

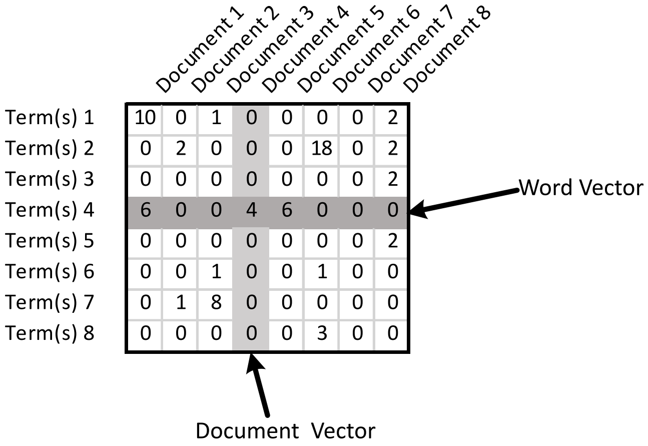
Count Vector (Bag-of-Words)

- Consider a Corpus $\mathcal{C}$ of D documents $\{d_1, d_2, \cdots, d_D\}$ and N unique tokens extracted out of the corpus C.
- The N tokens will form our dictionary and the size of the Count Vector matrix $\mathbf{M}$ will be given by $N \times D$.
- Each row in the matrix $\mathbf{M}$ contains the frequency of tokens in document d_i.
- Simple example:
  d_1: He is a lazy boy. She is also lazy
  d_2: Neeraj is a lazy person.

	He	She	lazy	boy	Neeraj	person
d_1	1	1	2	1	0	0
d_2	0	0	1	0	1	1

Count Vector (Bag-of-Words)

- A row can be understood as word vector for the corresponding word in dictionary.
- A column can be viewed as document vector for the corresponding document in corpus $\mathcal{C}$.



	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

Word Vector
Document Vector

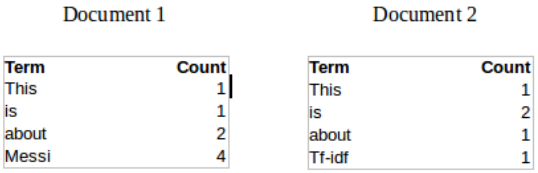
TF-IDF Vector

- TF-IDF works by penalizing these common words by assigning them lower weights.
  - TF: Term Frequency
  - IDF: Inverse Document Frequency
- Another method based on the frequency method that takes into account not just the occurrence of a word in a single document but in the entire corpus.
- Common words like 'is', 'the', 'a' etc. tend to appear quite frequently in comparison to the words which are important to a document.

TF-IDF Vector

- TF-IDF works by penalizing these common words by assigning them lower weights.
  - TF: Term Frequency
  - IDF: Inverse Document Frequency
- Another method based on the frequency method that takes into account not just the occurrence of a word in a single document but in the entire corpus.
- Common words like 'is', 'the', 'a' etc. tend to appear quite frequently in comparison to the words which are important to a document.
- Ideally, we want to down weight the common words occurring in almost all documents and give more importance to words that appear in a subset of documents.

Term Frequency (TF)



Document 1

Term	Count
This	1
is	1
about	2
Messi	4

Document 2

Term	Count
This	1
is	2
about	1
Tf-idf	1

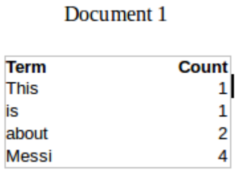
$$\text{TF} = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Number of terms in the document}}$$

- It denotes the contribution of the word to the document i.e words relevant to the document should be frequent. So,

$$\text{TF}(\text{This}, \text{Document1}) = \frac{1}{8}$$

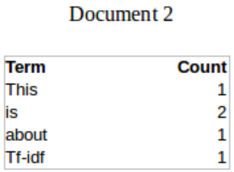
$$\text{TF}(\text{This}, \text{Document2}) = \frac{1}{5}$$

Inverse Document Frequency (IDF)



Document 1

Term	Count
This	1
is	1
about	2
Messi	4



Document 2

Term	Count
This	1
is	2
about	1
Tf-idf	1

$$\text{IDF} = \log\left(\frac{N \text{ (number of documents)}}{n \text{ (number of documents that a term } t \text{ has appeared in)}}\right)$$

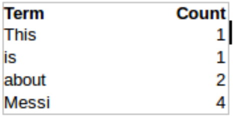
- Therefore,

$$\text{IDF(This)} = \log\left(\frac{2}{2}\right) = 0$$

$$\text{IDF(Messi)} = \log\left(\frac{2}{1}\right) = 0.301$$

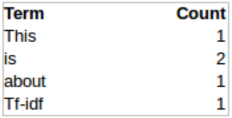
TF-IDF Vector

Document 1



Term	Count
This	1
is	1
about	2
Messi	4

Document 2



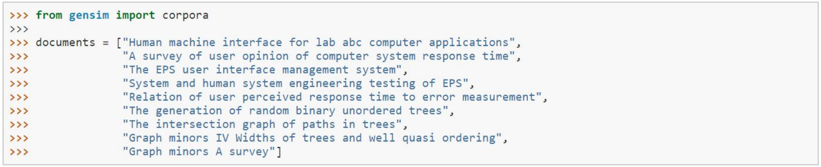
Term	Count
This	1
is	2
about	1
Tf-idf	1

$$\text{TF-IDF}(\text{This}, \text{Document1}) = \log\left(\frac{1}{8}\right) \times 0 = 0$$

$$\text{TF-IDF}(\text{This}, \text{Document2}) = \log\left(\frac{1}{5}\right) \times 0 = 0$$

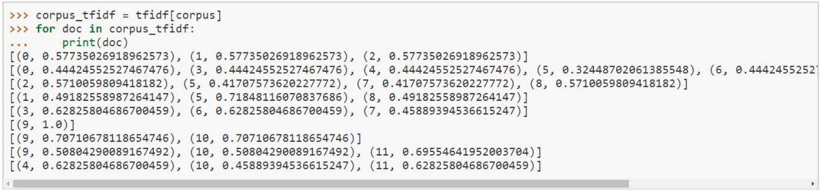
$$\text{TF-IDF}(\text{Messi}, \text{Document1}) = \log\left(\frac{4}{8}\right) \times 0.301 = 0.15$$

TF-IDF Example[1]



```
>>> from gensim import corpora
>>>
>>> documents = ["Human machine interface for lab abc computer applications",
>>>              "A survey of user opinion of computer system response time",
>>>              "The EPS user interface management system",
>>>              "System and human system engineering testing of EPS",
>>>              "Relation of user perceived response time to error measurement",
>>>              "The generation of random binary unordered trees",
>>>              "The intersection graph of paths in trees",
>>>              "Graph minors IV Widths of trees and well quasi ordering",
>>>              "Graph minors A survey"]
```

Documents



```
>>> corpus_tfidf = tfidf[corpus]
>>> for doc in corpus_tfidf:
...     print(doc)
[(0, 0.5773502691896257), (1, 0.5773502691896257), (2, 0.5773502691896257)]
[(0, 0.44424552527467476), (3, 0.44424552527467476), (4, 0.44424552527467476), (5, 0.3244870206138548), (6, 0.4442455252
[(2, 0.5710059809418182), (5, 0.41707573620227772), (7, 0.41707573620227772), (8, 0.5710059809418182)]
[(1, 0.49182558987264147), (5, 0.71848116070837686), (8, 0.49182558987264147)]
[(3, 0.62825804686700459), (6, 0.62825804686700459), (7, 0.45889394536615247)]
[(9, 1.0)]
[(9, 0.70710678118654746), (10, 0.70710678118654746)]
[(9, 0.50804290089167492), (10, 0.50804290089167492), (11, 0.69554641952003704)]
[(4, 0.62825804686700459), (10, 0.45889394536615247), (11, 0.62825804686700459)]
```

TF-IDF

[1] https://radimrehurek.com/gensim/

LSA (Latent Semantic Analysis)

- Any rectangular matrix can be decomposed into the product of three other matrices using the singular value decomposition

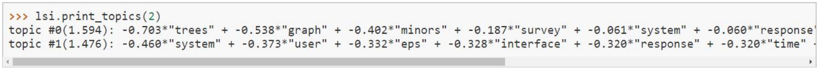
$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

- LSA uses a truncated SVD, keeping only the k largest singular values and their associated vectors, so

$$\mathbf{M} = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top$$

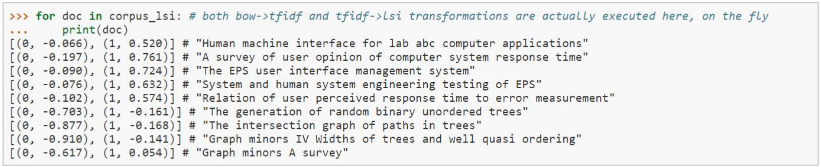
  where $\mathbf{U}_k \in \mathbb{R}^{N\times k}$, $\mathbf{\Sigma}_k \in \mathbb{R}^{k\times k}$, and $\mathbf{V}_k \in \mathbb{R}^{M\times k}$.
- The rows in $\mathbf{U}_k$ are the term vectors, and the rows in $\mathbf{V}_k$ are the document vectors.

LSA Example[2]



```
>>> lsi.print_topics(2)
topic #0(1.594): -0.703*"trees" + -0.538*"graph" + -0.402*"minors" + -0.187*"survey" + -0.061*"system" + -0.060*"response'
topic #1(1.476): -0.460*"system" + -0.373*"user" + -0.332*"eps" + -0.328*"interface" + -0.320*"response" + -0.320*"time" -
```

Topic-Term



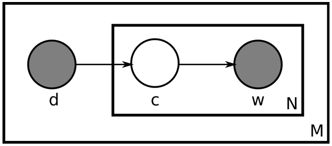
```
>>> for doc in corpus_lsi: # both bow->tfidf and tfidf->lsi transformations are actually executed here, on the fly
...     print(doc)
[(0, -0.066), (1, 0.520)] # "Human machine interface for lab abc computer applications"
[(0, -0.197), (1, 0.761)] # "A survey of user opinion of computer system response time"
[(0, -0.090), (1, 0.724)] # "The EPS user interface management system"
[(0, -0.076), (1, 0.632)] # "System and human system engineering testing of EPS"
[(0, -0.102), (1, 0.574)] # "Relation of user perceived response time to error measurement"
[(0, -0.703), (1, -0.161)] # "The generation of random binary unordered trees"
[(0, -0.877), (1, -0.168)] # "The intersection graph of paths in trees"
[(0, -0.910), (1, -0.141)] # "Graph minors IV Widths of trees and well quasi ordering"
[(0, -0.617), (1, 0.054)] # "Graph minors A survey"
```

Doc-Topic

[2]https://radimrehurek.com/gensim/

Probabilistic LSA

- Introduce a latent variable: **words' topic** c.



- The probability of each co-occurrence as a mixture of conditionally independent multinomial distributions

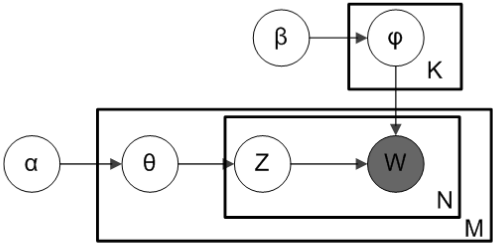
$$p(w,d) = \sum_c p(c)p(d|c)p(w|c) = p(d) \sum_c p(c|d)p(w|c)$$

  where c is the words' topic.

- The parameters are learned using the EM algorithm.

Latent Dirichlet Allocation

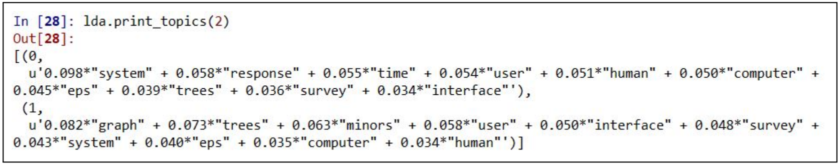
- Introduce Dirichlet prior to topic distribution.



  - α: Dirichlet prior on the per-document topic distributions
  - β: Dirichlet prior on the per-topic word distribution
  - θ_m: topic distribution for document m
  - φ_k: word distribution for topic k
  - z_{mn}: topic for the n-th word in document m
  - w_{mn}: is the specific word

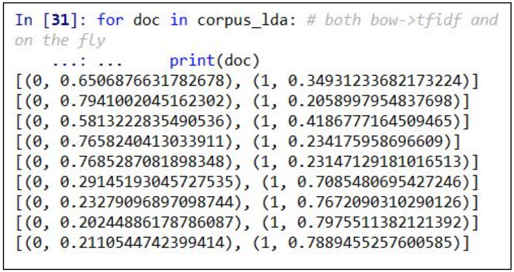
- The parameters are learned using the variational Bayes algorithm.

LDA Example



```
In [28]: lda.print_topics(2)
Out[28]:
[(0,
  u'0.098*"system" + 0.058*"response" + 0.055*"time" + 0.054*"user" + 0.051*"human" + 0.050*"computer" +
0.045*"eps" + 0.039*"trees" + 0.036*"survey" + 0.034*"interface"'),
 (1,
  u'0.082*"graph" + 0.073*"trees" + 0.063*"minors" + 0.058*"user" + 0.050*"interface" + 0.048*"survey" +
0.043*"system" + 0.040*"eps" + 0.035*"computer" + 0.034*"human"')]
```

Topic-Term



```
In [31]: for doc in corpus_lda: # both bow->tfidf and
on the fly
    ...: ...      print(doc)
[(0, 0.6506876631782678), (1, 0.34931233682173224)]
[(0, 0.7941002045162302), (1, 0.2058997954837698)]
[(0, 0.5813222835490536), (1, 0.4186777164509465)]
[(0, 0.7658240413033911), (1, 0.234175958696609)]
[(0, 0.7685287081898348), (1, 0.23147129181016513)]
[(0, 0.29145193045727535), (1, 0.7085480695427246)]
[(0, 0.23279096897098744), (1, 0.7672090310290126)]
[(0, 0.20244886178786087), (1, 0.7975511382121392)]
[(0, 0.2110544742399414), (1, 0.7889455257600585)]
```

Doc-Topic

Distributional Hypothesis

- Words that are used and occur in the same contexts tend to purport similar meanings.
  — (Harris, 1954)
- A word is characterized by the company it keeps.
  — (Firth, 1957)

He is reading a **magazine**	I was reading a **newspaper**
This **magazine** published my story	The **newspaper** published an article
She buys a **magazine** every month	He buys this **newspaper** every day

Co-Occurrence Vector

- Idea: Similar words tend to occur together and will have similar context.
- For example:
  - Apple is a fruit.
  - Mango is a fruit.
- Context window is specified by a number and the direction.

Quick	Brown	Fox	Jump	Over	The	Lazy	Dog
Quick	Brown	Fox	Jump	Over	The	Lazy	Dog

- For a given corpus, the co-occurrence of a pair of words say w_1 and w_2 is the number of times they have appeared together in a context window.

Co-Occurrence Matrix

- Corpus = He is not lazy. He is intelligent. He is smart.
- The co-occurrence matrix with context window 2 is

	He	is	not	lazy	intelligent	smart
He	0	4	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

Variations of Co-occurrence Matrix

- Let's say there are V unique words in the corpus. The columns of the Co-occurrence matrix form the context words.
- A co-occurrence matrix of size $V \times V$ is very large and difficult to handle.
- Instead, this Co-occurrence matrix is decomposed using techniques like PCA, SVD etc. into factors and combination of these factors forms the word vector representation.

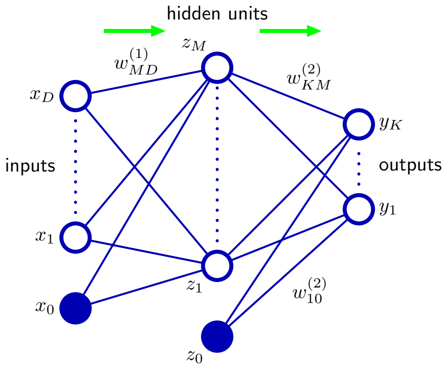
$$\overset{\hat{X}}{\underset{m \times n}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}}} \approx \overset{U}{\underset{m \times r}{\begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix}}} \overset{S}{\underset{r \times r}{\begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix}}} \overset{V^{\intercal}}{\underset{r \times n}{\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}}}$$

- Dot product of $\mathbf{U}$ and $\mathbf{S}$ gives the word vector representation and $\mathbf{V}$ gives the word context representation.

Pros and Cons of Co-occurrence Matrix

- Advantages
  1. Preserve the semantic relationship between words.
  2. Use factorization at its core and produce more accurate word vector representations than existing methods.
  3. Be computed once and can be used anytime once computed.
- Disadvantages: require huge memory to store the co-occurrence matrix.

Feed-forward Network Functions

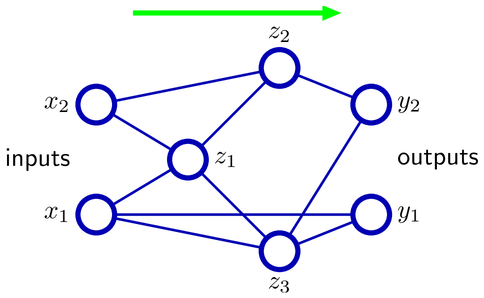


- M linear combinations of the input variables $a_j = \sum_{i=1}^{D} \omega^{(1)}_{j,i} x_i + \omega^{(1)}_{j,0}$.
- A Differentiable, nonlinear activation function $h(\cdot)$ to give $z_j = h(a_j)$.
- Output unit activations $a_k = \sum_{j=1}^{M} \omega^{(2)}_{k,j} z_j + \omega^{(2)}_{k,0}$.
- Network output $y_k = f(a_k)$.

Overall network function

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = f(\sum_{j=0}^{M} \omega^{(2)}_{kj} h(\sum_{i=0}^{D} \omega^{(1)}_{ji} x_i))$$

Forward Propagation



- Each unit computes a weighted sum of its inputs of the form

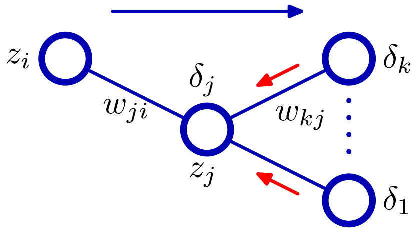
$$a_j = \sum_i \omega_{ji} z_i$$

- The sum is transformed by $h(\cdot)$ to give the activation z_j of units j in the form

$$z_j = h(a_j)$$

This process is called **forward propagation** because it can be regareded as a forward flow of information through the network.

Error Back Propagation



Error function

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{t}_n\|^2$$

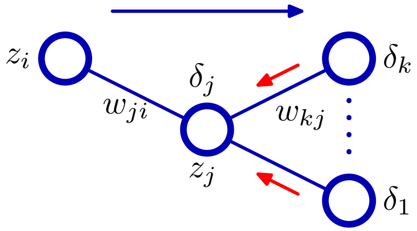
- E_n depends on the weight ω_{ji} only via the summed input a_j to unit j

$$\frac{\partial E_n}{\partial \omega_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial \omega_{ji}}$$

- Introduce a useful notation $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$, where the δ's are often referred to as errors
- Write $\frac{\partial a_j}{\partial \omega_{ji}} = z_i$, then

$$\frac{\partial E_n}{\partial \omega_{ji}} = \delta_j z_i.$$

Error Back Propagation Cont.



Backpropagation formula

$$\delta_j = h^{'}(a_j) \sum_k \omega_{kj} \delta_k$$

- Evaluate the δ's for hidden units

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

- The sum runs over all units k to which unit j sends connections, and

$$a_k = \sum_j \omega_{kj} h(a_j)$$

$$\Rightarrow \quad \frac{\partial a_k}{\partial a_j} = \omega_{kj} h^{'}(a_j)$$

Language Modeling Fundamentals: N-grams

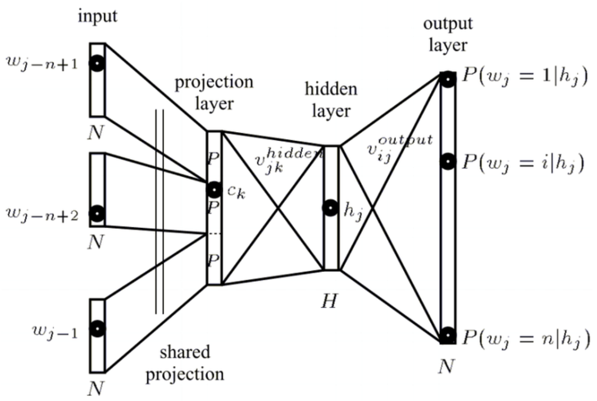
- Language models generally try to compute the probability of a word w_t given its $n-1$ previous words, i.e. $p(w_t|w_{t-1}, \cdots, w_{t-n+1})$.
- Applying the chain rule together with the Markov assumption

$$p(w_1, \cdots, w_T) = \prod_i p(w_i|w_{i-1}, \cdots, w_{i-n+1})$$

- In n-gram based language models, a word's probability is based on the frequencies of its constituent n-grams:

$$p(w_t|w_{t-1}, \cdots, w_{t-n+1}) = \frac{\text{cout}(w_{t-n+1}, \cdots, w_{t-1}, w_t)}{\text{cout}(w_{t-n+1}, \cdots, w_{t-1})}$$

Neural Language Model



- In neural networks, the same objective is achieved using softmax layer

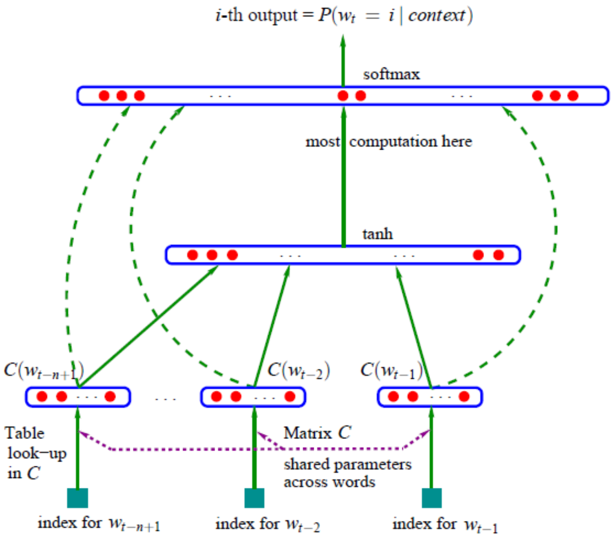
$$p(w_t|w_{t-1}, \cdots, w_{t-n+1}) = \frac{\exp(\mathbf{h}^\top \mathbf{v}'_{w_t})}{\sum_{w_i \in V} \exp(\mathbf{h}^\top \mathbf{v}'_{w_i})}$$

Neural Language Model

- $\mathbf{h}^\top \mathbf{v}'_{w_t}$ computes the (unnormalized) log-probability of word w_t.
- $\mathbf{h}$ is the output vector of the penultimate network layer
- $\mathbf{v}'_w$ is the output embedding of word w, i.e. its representation in the weight matrix of the softmax layer.
- The model tries to maximize the probability of predicting the correct word at every timestep t

$$\begin{aligned} J_\theta &= \frac{1}{T} \log p(w_1, \cdots, w_T) \\ &= \frac{1}{T} \sum_t \log p(w_t | w_{t-1}, \cdots, w_{t-n+1}) \end{aligned}$$

Classic Neural Language Model



Classic Neural Language Model: Building Blocks

1. **Embedding Layer**: a layer that generates word embeddings by multiplying an index vector with a word embedding matrix.
2. **Intermediate Layer(s)**: one or more layers that produce an intermediate representation of the input, e.g. a fully-connected layer that applies a non-linearity to the concatenation of word embeddings of n previous words.
3. **Softmax Layer**: the final layer that produces a probability distribution over words in V.

The final softmax layer (more precisely: the normalization term) as the network's main bottleneck, as the cost of computing the softmax is proportional to the number of words in V.

CBOW (Continuous bag of words)

- CBOW tends to predict the probability of a word given a context.
- Suppose, we have a corpus $\mathcal{C} =$"**The quick brown fox jumps over the lazy dog**", and define a context window of 1.

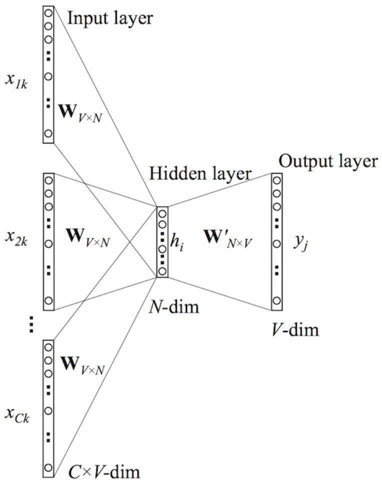
No.	Training Samples (Input, Output)
1	(quick, the)
2	([the brown], quick)
3	([quick fox], brown)
4	([brown jumps], fox)
...	...

- Objective function

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t | w_{t-n}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+n})$$

CBOW Model Cont.

- There is a no activation function between any layers.
- The weight between the hidden layer and the output layer is taken as the word vector representation of the word.
- An average is taken over all the corresponding rows of the matrix.



CBOW Model Cont.

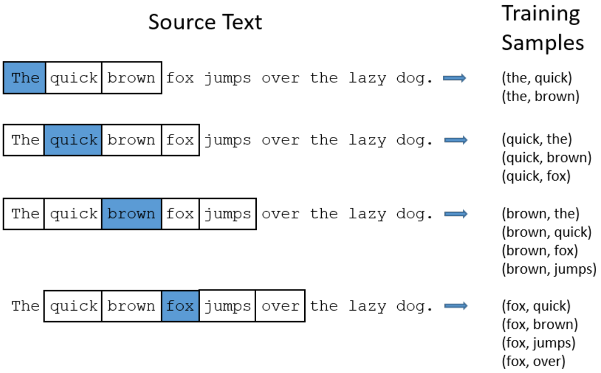
- The objective function in MLP is a MSE(mean square error)
- In CBOW, it is negative log likelihood of a word given a set of context i.e $-\log(p(w_O/w_I))$, where $p(w_O/w_I)$ is given as

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

  - w_O: output word
  - w_I: context words
  - $\mathbf{v}_w$: "input" vector representations of w
  - $\mathbf{v}'_w$: "output" vector representations of w
  - W: the number of words in the vocabulary

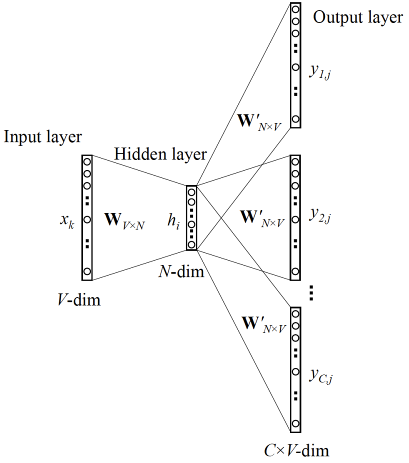
Skip-Gram Model

- Skip-gram follows the same topology as of CBOW. It just flips CBOWs architecture on its head.
- The aim of skip-gram is to predict the context given a word.
- $\mathcal{C}$="**The quick brown fox jumps over the lazy dog**".

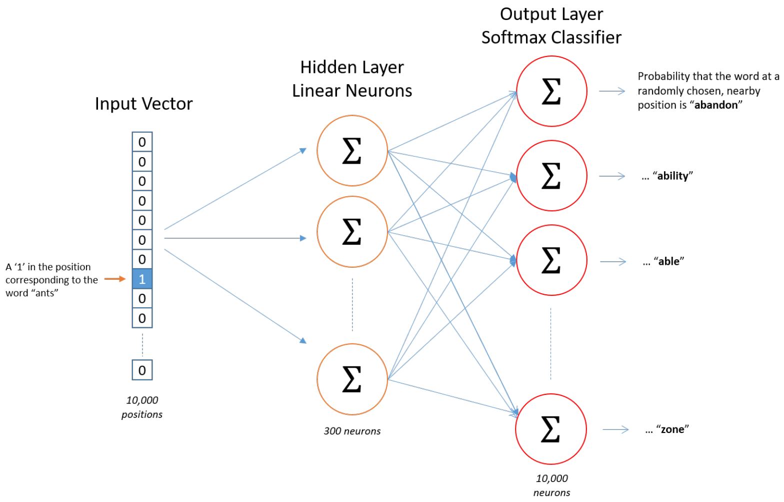


Skip-Gram Model

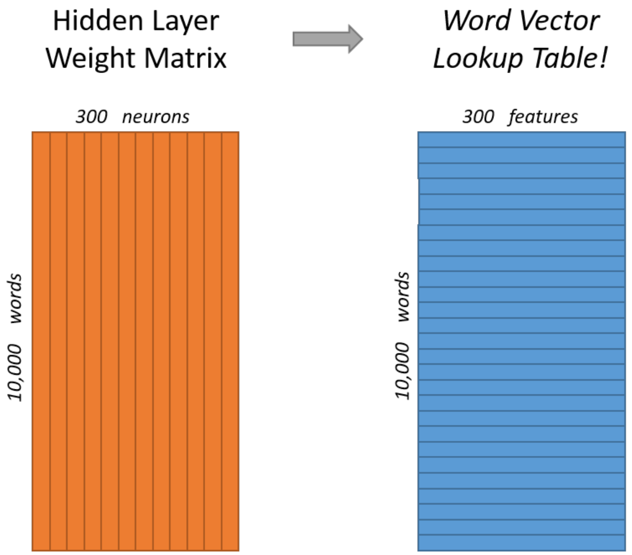
- There will be C one hot encoded target variables and C corresponding outputs.
- C separate errors are calculated with respect to the C target variables.
- These C error vectors obtained are added element-wise to obtain a final error vector which is propagated back to update the weights.



Model Details



The Hidden Layer



The Hidden Layer Cont.

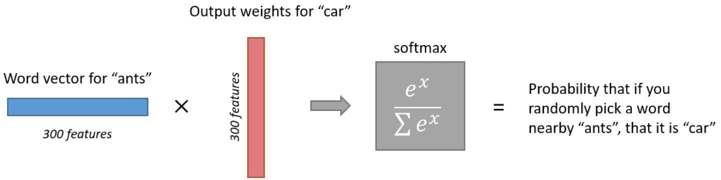
- If you multiply a $1 \times 10,000$ one-hot vector by a $10,000 \times 300$ matrix, it will effectively just select the matrix row corresponding to the '1'.

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

- This means that the hidden layer of this model is really just operating as a lookup table.
- The output of the hidden layer is just the "word vector" for the input word.

The Output Layer

- The 1×300 word vector for "**ants**" then gets fed to the output layer.
- An illustration of calculating the output of the output neuron for the word "**car**".



- Note that neural network does not know anything about the offset of the output word relative to the input word.

Tips for Learning Word Embedding

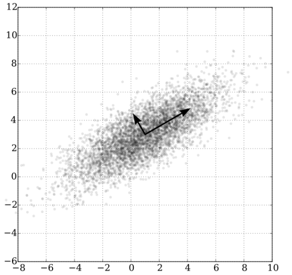
- Recall that the neural network had two weight matrices—a hidden layer and output layer.
- Both of these layers would have a weight matrix with $300 \times 10,000 = 3$ million weights each!

Tips for Learning Word Embedding

- Recall that the neural network had two weight matrices—a hidden layer and output layer.
- Both of these layers would have a weight matrix with $300 \times 10,000 = 3$ million weights each!
- Three innovations
  1. Treating common word pairs or phrases as single “words” in their model.
  2. Subsampling frequent words to decrease the number of training examples.
  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.

Canonical Correlation Analysis (CCA)

- CCA is the analog to Principal Component Analysis (PCA) for pairs of matrices.
- PCA computes the directions of maximum covariance between elements in a single matrix.



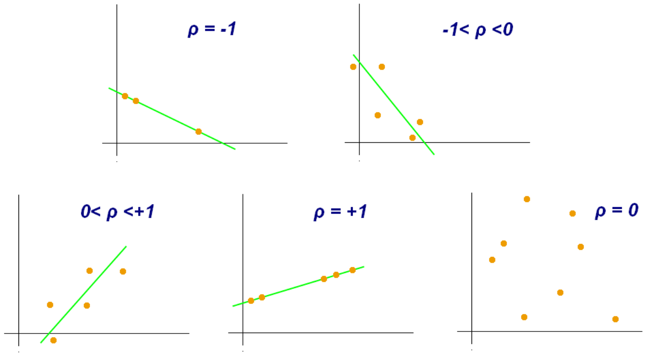
- N observations $\{\mathbf{x}_n\}_{n=1}^N$ with mean $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^N \mathbf{x}_n$.
- Project the observations onto a 1-dimensional space with unit vector $\mathbf{u}_1$.
- The variance of the projected data is $\frac{1}{N}\sum_{n=1}^N \{\mathbf{u}_1^\top \mathbf{x}_n - \mathbf{u}_1^\top \bar{\mathbf{x}}\}^2 = \mathbf{u}_1^\top \mathbf{S}\mathbf{u}_1$.

- Maximize the projected variance $\mathbf{u}_1^\top \mathbf{S}\mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^\top \mathbf{u}_1)$.

CCA

- Pearson correlation coefficient

$$\rho_{x,y} = \frac{\text{cov}(x,y)}{\sigma_x \sigma_y} = \frac{\mathbb{E}\left[(x-\bar{x})(y-\bar{y})\right]}{\sigma_x \sigma_y}$$



- CCA computes the directions of maximal correlation between a pair of matrices.

CCA Cont.

- Given n samples from two sets of multivariate data $\mathcal{D}_z = \{\mathbf{z}_1, \cdots, \mathbf{z}_n\} \in \mathbb{R}^{m_1}$ and $\mathcal{D}_w = \{\mathbf{w}_1, \cdots, \mathbf{w}_n\} \in \mathbb{R}^{m_2}$.
- CCA tries to find a pair of linear transformation $\boldsymbol{\phi}_z \in \mathbb{R}^{m_1 \times k}$ and $\boldsymbol{\phi}_w \in \mathbb{R}^{m_2 \times k}$ such that the correlation between the projection of $\mathbf{z}$ onto $\boldsymbol{\phi}_z$ and $\mathbf{w}$ onto $\boldsymbol{\phi}_w$ is maximized

$$\max_{\boldsymbol{\phi}_z, \boldsymbol{\phi}_w} \frac{\boldsymbol{\phi}_z^\top \mathbf{C}_{zw} \boldsymbol{\phi}_w}{\sqrt{\boldsymbol{\phi}_z^\top \mathbf{C}_{zz} \boldsymbol{\phi}_z} \sqrt{\boldsymbol{\phi}_w^\top \mathbf{C}_{ww} \boldsymbol{\phi}_w}}$$

  where $\mathbf{C}_{zw} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{z}_i - \bar{\mathbf{z}})(\mathbf{w}_i - \bar{\mathbf{w}})^\top$ is the sample covariance matrices.
- Define the operation

$$(\boldsymbol{\phi}_z^{proj}, \boldsymbol{\phi}_w^{proj}) \equiv \text{CCA}(\mathbf{Z}, \mathbf{W})$$

One Step CCA for Word Embedding

- The left and right context of a given word provides two natural views and one could use CCA to estimate the hidden state of word.
- Define $\mathbf{L}, \mathbf{R} \in \mathbb{R}^{n \times vh}$ as the matrices specifying the left and right contexts of the tokens, and $\mathbf{W} \in \mathbb{R}^{n \times v}$ as the matrix of the tokens themselves.
- One step CCA estimate the eigenword dictionary by

$$(\boldsymbol{\phi}_w, \boldsymbol{\phi}_c) = \text{CCA}(\mathbf{W}, \mathbf{C})$$

where the matrix $\boldsymbol{\phi}_w \in \mathbb{R}^{v \times k}$ contains the eigenword dictionary that characterizes each of the v words in the vocabulary using a k dimensional vector.

Two Step CCA for Word Embedding

- First take the CCA between the left and right contexts and use the result of that CCA to estimate the state $\mathbf{S}$ (an empirical estimate of the true hidden state) of all the tokens in the corpus from their contexts.

1. Take the CCA between the left and right contexts

$$(\phi_l, \phi_r) = \text{CCA}(\mathbf{L}, \mathbf{R})$$

   then,

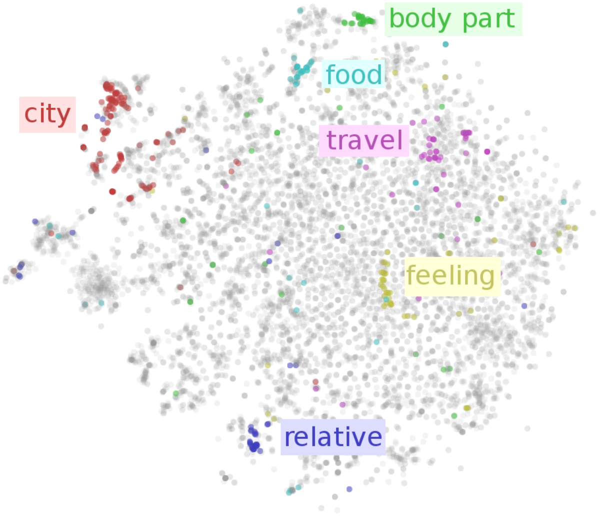
$$\mathbf{S} = [\mathbf{L}\phi_l \ \mathbf{R}\phi_r]$$

2. Take the CCA between the state and token

$$(\phi_s, \phi_w) = \text{CCA}(\mathbf{S}, \mathbf{W})$$

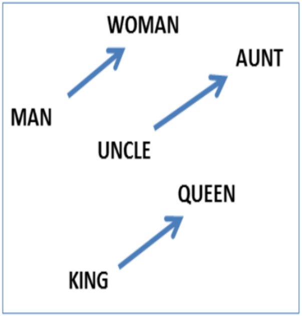
- Obtain the eigenword dictionary ϕ_w.

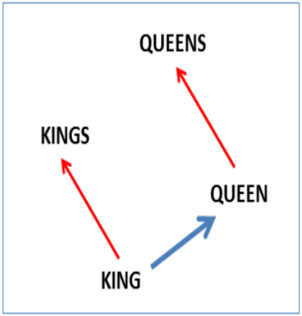
Word Similarity



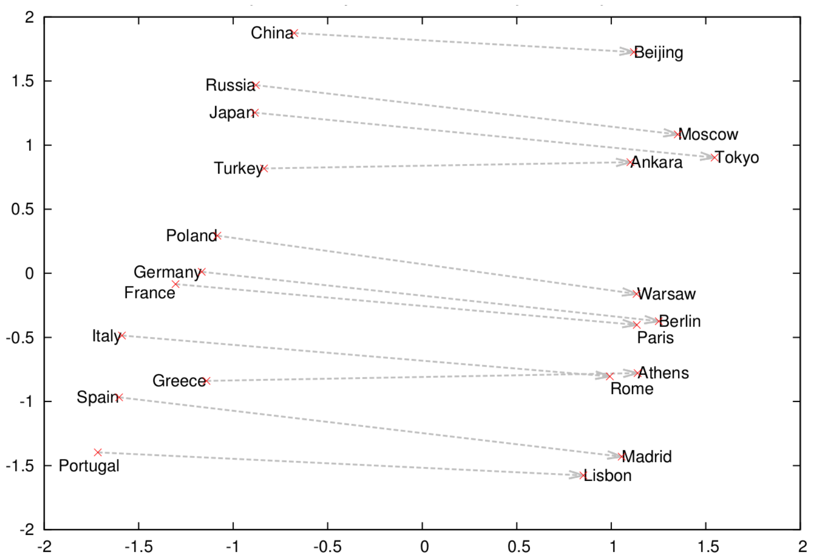
Relationship between Words

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

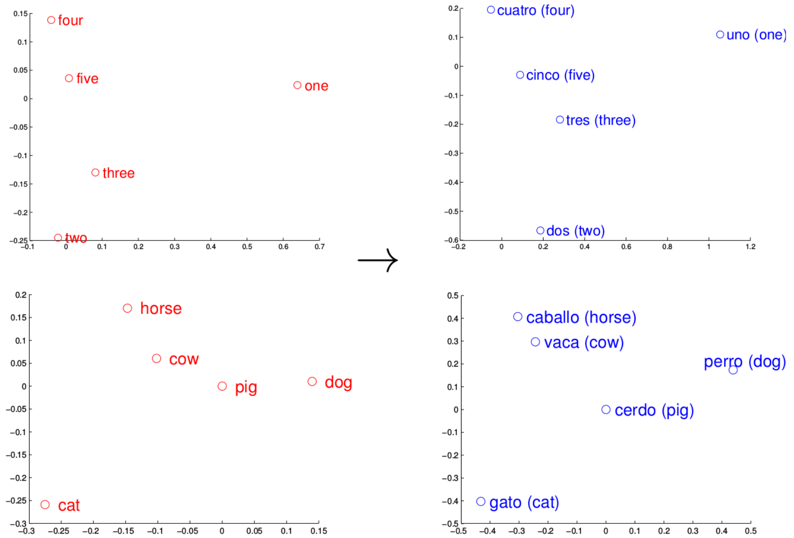




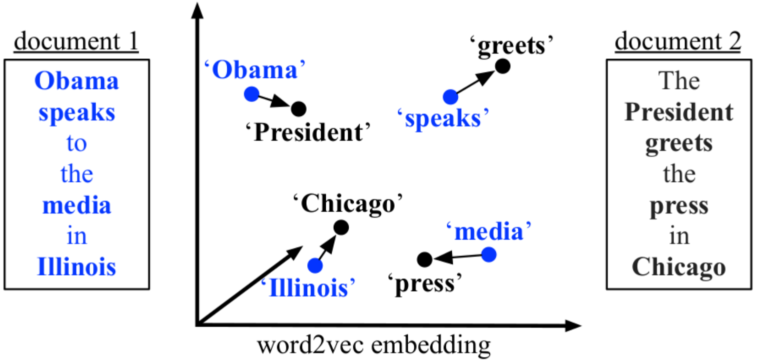
Relationship between Words



Machine Translation



Document Similarity



Reference

- Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. The Journal of Machine Learning Research, 3, 11371155.
- Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of the International Conference on Learning Representations (ICLR 2013), 112.
- Dhillon, Paramveer S., Dean P. Foster, and Lyle H. Ungar. Eigenwords: spectral word embeddings. Journal of Machine Learning Research 16 (2015): 3035-3078.
- `http://ruder.io/word-embeddings-1/index.html#fn:9`
- `http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/`
- `http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/`

Assignment

Q: Given overall network function

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = f(\sum_{j=0}^{M} w_{kj}^{(2)} h(\sum_{i=0}^{D} w_{ji}^{(1)} x_i))$$

and cross-entropy error function for the multiclass classification problem

$$E(\boldsymbol{w}) = -\ln\left(\prod_{k=1}^{K} y_k^{t_k}\right) = -\sum_{k=1}^{K} t_k \ln y_k$$

where $t_k = \{0,1\}^K$ and $\sum_{k=1}^{K} t_k = 1$, $f(z_k) = \frac{\exp(z_k)}{\sum_{l=1}^{K}\exp(z_l)}$ is the softmax function, $h(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$. Compute the derivatives of error function w.r.t the first-layer and second-layer weights with Back Propagation

$$\frac{\partial E}{\partial w_{ji}^{(1)}} \text{ and } \frac{\partial E}{\partial w_{kj}^{(2)}}$$