Representing Words With Vectors

BILICI, M. Şafak
safakk.bilici.2112@gmail.com

Contents

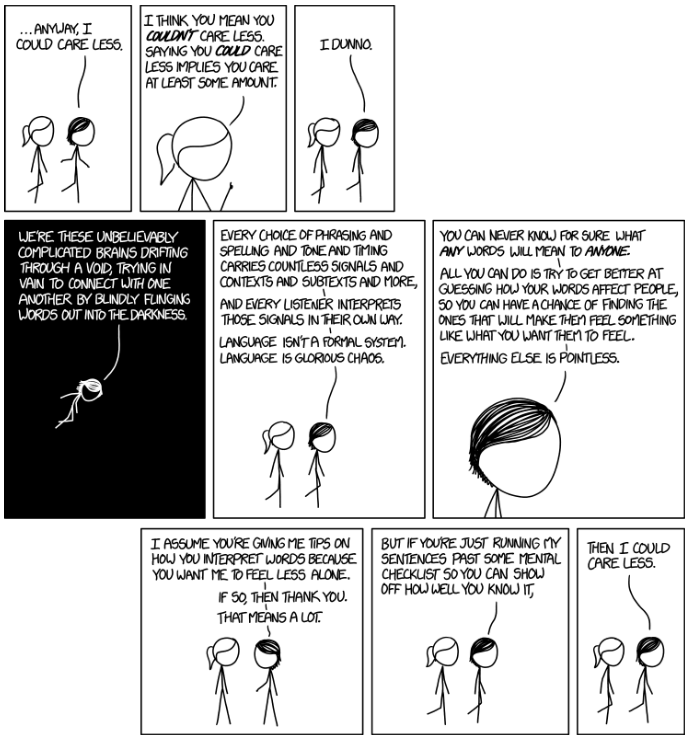

...ANYWAY, I COULD CARE LESS.
I THINK YOU MEAN YOU ***COULDN'T*** CARE LESS. SAYING YOU ***COULD*** CARE LESS IMPLIES YOU CARE AT LEAST SOME AMOUNT.
I DUNNO.
WE'RE THESE UNBELIEVABLY COMPLICATED BRAINS DRIFTING THROUGH A VOID, TRYING IN VAIN TO CONNECT WITH ONE ANOTHER BY BLINDLY FLINGING WORDS OUT INTO THE DARKNESS.
EVERY CHOICE OF PHRASING AND SPELLING AND TONE AND TIMING CARRIES COUNTLESS SIGNALS AND CONTEXTS AND SUBTEXTS AND MORE,
AND EVERY LISTENER INTERPRETS THOSE SIGNALS IN THEIR OWN WAY.
LANGUAGE ISN'T A FORMAL SYSTEM. LANGUAGE IS GLORIOUS CHAOS.
YOU CAN NEVER KNOW FOR SURE WHAT ***ANY*** WORDS WILL MEAN TO ***ANYONE***.
ALL YOU CAN DO IS TRY TO GET BETTER AT GUESSING HOW YOUR WORDS AFFECT PEOPLE, SO YOU CAN HAVE A CHANCE OF FINDING THE ONES THAT WILL MAKE THEM FEEL SOMETHING LIKE WHAT YOU WANT THEM TO FEEL.
EVERYTHING ELSE IS POINTLESS.
I ASSUME YOU'RE GIVING ME TIPS ON HOW YOU INTERPRET WORDS BECAUSE YOU WANT ME TO FEEL LESS ALONE.
IF SO, THEN THANK YOU.
THAT MEANS A LOT.
BUT IF YOU'RE JUST RUNNING MY SENTENCES PAST SOME MENTAL CHECKLIST SO YOU CAN SHOW OFF HOW WELL YOU KNOW IT,
THEN I COULD CARE LESS.

Figure 1: xkcd.com/1576

1 Word Representation With One-Hot Vectors

One-hot encoding is the most common, most basic way to turn a token into a vector. It consists in associating a unique integer index to every word, then turning this integer index i into a binary vector of size V, which is the size of our vocabulary, that would be all-zeros except for the i-th entry, which would be 1.

$$[X]_{\in\mathbb{R}^{|\times|}} = [W]_{\in\mathbb{R}^{|\times|}} \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_V \end{bmatrix}_{\in\mathbb{R}^{|\times|}} [C]_{\in\mathbb{R}^{|\times|}}$$

1.1 Little Example For One-Hot Representation

```
import numpy as np
samples = ['The cat sat on the mat.', 'The dog ate my homework.']
token_index = {}
for sample in samples:
    for word in sample.split():
        if word not in token_index:
            token_index[word] = len(token_index) + 1

sentence = "the cat ate the god"
seq = []
for word in sentence.split():
    one_hot = np.zeros((1,len(token_index)))
    if word not in token_index:
        seq.append(one_hot.squeeze())
    else:
        one_hot[0,token_index[word]-1] = \
        one_hot[0,token_index[word]-1] + 1
        seq.append(one_hot.squeeze())
print(seq)

#output: [array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]),
# array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.]),
# array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]),
# array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]),
# array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.])]
```

```
import numpy as np
with open('cats100.test.txt') as file_handler:
    data = file_handler.read()

token_idx = {}
for word in data.split():
    if word not in token_idx:
        token_idx[word] = len(token_idx) + 1

sentence = "Gectigimiz zamanlarda kotu olaylar yasandi"
one_hot_sentence = np.zeros((len(sentence.split()),
                             len(token_idx)),dtype=np.int32)
for word_idx,word in enumerate(sentence.split()):
    one_hot_sentence[word_idx,token_idx[word] - 1] = 1
```

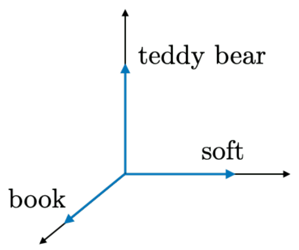


Figure 2: Stanford cs-230 cheatsheet

There are two major issues with this approach. First issue is the curse of dimensionality, which refers to all sorts of problems that arise with data in high dimensions. This requires exponentially large memory space. Most of the matrix is taken up by zeros, so useful data becomes sparse. Imagine we have a vocabulary of 50,000. (There are roughly a million words in English language.) Each word is represented with 49,999 zeros and a single one, and we need 50,000 squared = 2.5 billion units of memory space. Not computationally efficient.

Second issue is that every one hotted vector is orthogonal to each other. You cannot measure the similarity (like cosine similarity) on this vectors.

Those vectors can be visualized in 3 or 2-dimensional space as an example. In $\mathbb{R}^3$ (in other words, vocabulary size of 3, $\mid V \mid = 3$), the word vectors become our span set which is $\left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}$. The span set is linearly independent set. Therefore we cannot compute similarity metrics. The vectors $\vec{u_1}, \vec{u_2}, \vec{u_3}$ are orthogonal. Similarity metrics gives the result of zero. For example the cosine similarity:

$$\text{cos-sim}(\vec{u_1}, \vec{u_2}) = \frac{\langle \vec{u_1}, \vec{u_2} \rangle}{\|\vec{u_1}\|_2 \times \|\vec{u_2}\|_2} = \frac{\sum_{i=0}^{n} u_{1_i} \times u_{2_i}}{\sqrt{\sum_{i=0}^{n} u_{1_i}^2} \times \sqrt{\sum_{i=0}^{n} u_{2_i}^2}} = \frac{1 \times 0 + 0 \times 1 + 0 \times 0}{1 + 1} = 0$$

So, how to deal with those problems?

2 Lexical Semantics And Distributional Linguistics

Words that occur in **similar contexts** tend to have similar meanings. The link between similarity and words are distributed and similarity in what they mean is called **distributional hypothesis** or **distributional semantics** in the field of Computational Linguistics. So what can be counted when we say similar contexts? For example if you surf on the Wikipedia page of linguistics, the words in this page somehow related with each other in the context of linguistics. This was formulated firsty by Martin Joos (Description of Language Design, 1950), Zellig Harris (Distributional Structure, 1954), John Rupert Firth (Applications of General Linguistics, 1957).

Some words have similar meanings, for example word cat and dog **similar**. Also, words can be **antonyms**, for example hot and cold. And words have **connotations** (TR: çağrışım), for example happy→positive connotation and sad→negative connotation. Can you feel the similarity of words, [study, exam, night, FF]?

Also each word can have multiple meanings. The word mouse can refeer to the rodent or the cursor control device. We call each of these aspects of the meaning of mouse a **word sense**. In other words words can be **polysemous** (have multiple senses), which can lead us to make word interpretations difficult!

- **Word Sense Disambiguation**: ”Mouse info” (person who types this into a web search engine, looking for a pet info or a tool?) (determining which sense of a word is being used in a particular context)

The word **similarity** is very useful in larger semantics tasks. Knowing how similar two words are can help in computing how similar the meaning of two phrases or sentences are, a very important component of natural language understanding tasks like **question answering**, **summarization** etc.

Word1	**Word2**	**Similarity (0-10)**
hline Vanish	Disappear	9.8
Behave	Obey	7.3
Belief	Impression	5.95
Muscle	Bone	3.65
Modest	Flexible	0.98
Hole	Agreement	0.3

We should look at **word relatedness** , the meaning of two words can be related in ways other than similarity. One such class of connections is called word **relatedness**, also traditionally called word **association** in pysch.

- The word *cup* and *coffee.*
- The word *inzva* and *deep learning.*

Also words can affective meanings. Osgood et al. 1957, proposed that words varied along three important dimensions of affective meaning: **valence, arousal, dominance**:

1. **valence**: the pleasantness of the stimulus.
2. **arousal**: the intensity of emotion provoked by the stimulus.
3. **dominance**: the degree of control exerted by the stimulus.

Examples:

- happy(1) $\uparrow$, satisfied(1) $\uparrow$; annoyed(1) $\downarrow$, unhappy(1)$\downarrow$
- excited(2) $\uparrow$, frenzied(2) $\uparrow$; relaxed(1) $\downarrow$, calm(2) $\downarrow$
- important(3)$\uparrow$, controlling(3)$\uparrow$; awed(3)$\downarrow$, influenced(3)$\downarrow$

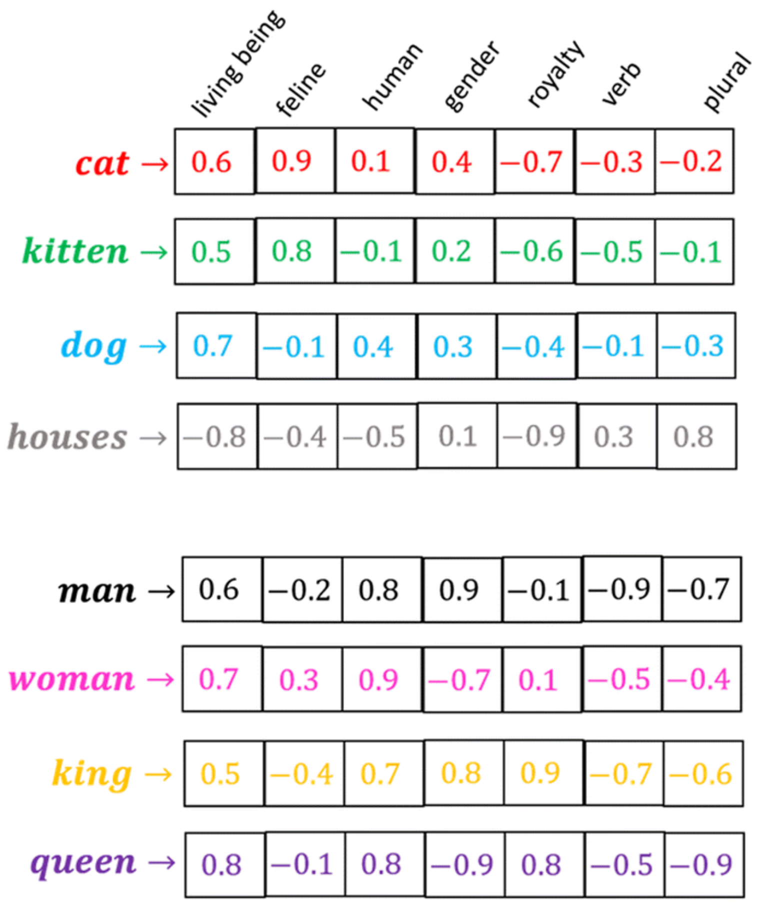
Question: does word embeddings has these dimesions?

3 Word Embeddings

How can we build a computational model that successfully deals with the different aspects of word meaning we saw above (word senses, word similarity, word relatedness, connotation etc.)?

Instead of representing words with one-hot vector, sparse; with word embeddings we represents words with dense vectors.

The idea of vector semantics is thus to represent a word as a point in some multi-dimensional semantic space. Vectors for representing words are generally called **embeddings**. We **LEARN** this embeddings form an arbitrary context.



	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8
man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Figure 3: source

Main two advantages of this word embeddings are:

- Now we represent words with dense vectors, which leads low memory requirements.

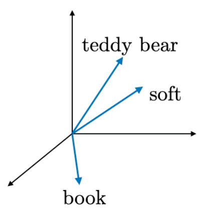
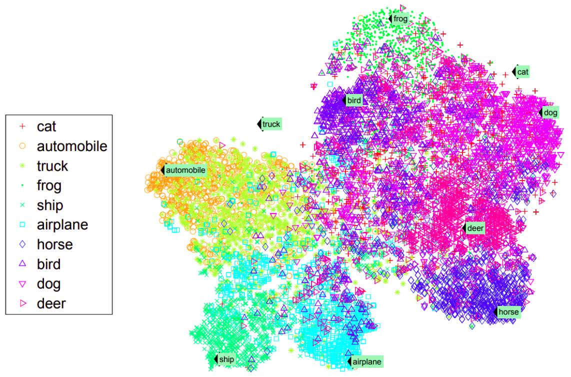


Figure 4: Stanford cs-230 cheatsheet

- We can now calculate similarity metrics on these vectors!

4 Visualizing Word Embeddings

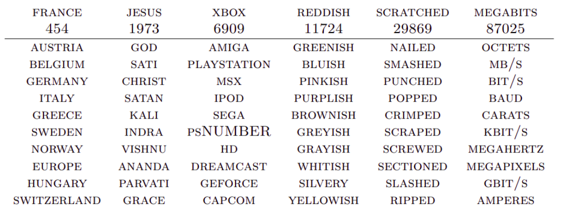
Word embeddings can be visualized with various dimensionality reduction/matrix factorization algorithms.



(a) Zero-Shot Learning Through Cross-Modal Transfer (Socher et al., 2013, NeurIPS)

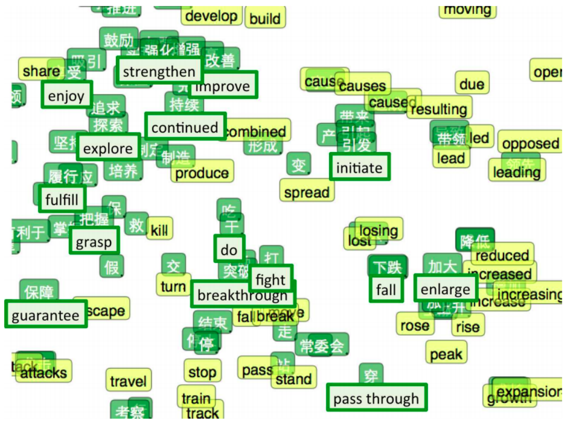
(b) Word representations: A simple and general method for semi-supervised learning (Turian et al., 2010)

Figure 5: t-SNE



FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

(a) Natural Language Processing (almost) from Scratch (Collobert et al., 2011)



(b) Bilingual Word Embeddings for Phrase-Based Machine Translation (Socher et al., 2013, EMNLP)

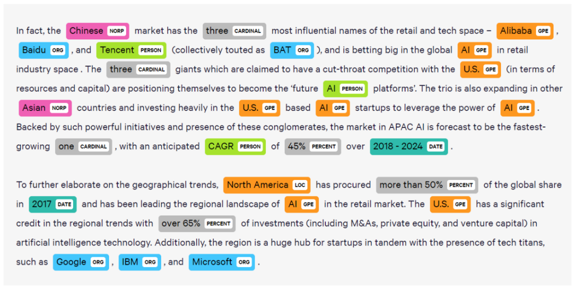
Figure 6: t-SNE

5 Using Word Embeddings

5.1 Named Entity Recognition (NER)

In Natural language processing, Named Entity Recognition (NER) is a process where a sentence or a chunk of text is parsed through to find entities that can be put under categories like names, organizations, locations, quantities, monetary values, percentages, etc. Traditional NER algorithms included only names, places, and organizations.

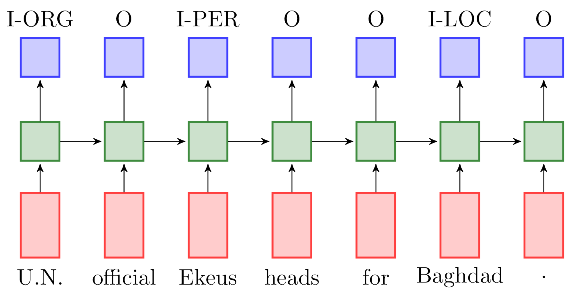
Since the embeddings can capture word senses etc., it is practical and beneficial to use word embeddings in NER task. Embeddings can capture entity informations while capturing word relations.



In fact, the Chinese NORP market has the three CARDINAL most influential names of the retail and tech space – Alibaba GPE , Baidu ORG , and Tencent PERSON (collectively touted as BAT ORG), and is betting big in the global AI GPE in retail industry space . The three CARDINAL giants which are claimed to have a cut-throat competition with the U.S. GPE (in terms of resources and capital) are positioning themselves to become the 'future AI PERSON platforms'. The trio is also expanding in other Asian NORP countries and investing heavily in the U.S. GPE based AI GPE startups to leverage the power of AI GPE . Backed by such powerful initiatives and presence of these conglomerates, the market in APAC AI is forecast to be the fastest-growing one CARDINAL , with an anticipated CAGR PERSON of 45% PERCENT over 2018 - 2024 DATE .

To further elaborate on the geographical trends, North America LOC has procured more than 50% PERCENT of the global share in 2017 DATE and has been leading the regional landscape of AI GPE in the retail market. The U.S. GPE has a significant credit in the regional trends with over 65% PERCENT of investments (including M&As, private equity, and venture capital) in artificial intelligence technology. Additionally, the region is a huge hub for startups in tandem with the presence of tech titans, such as Google ORG , IBM ORG , and Microsoft ORG .

(a) Source



(b) Source

Figure 7: NER

5.2 Transfer Learning

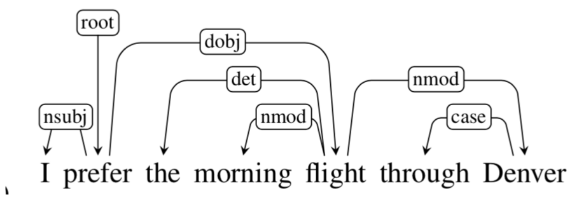
- To learn word embeddings, huge size of training data is always useful. For example GloVe is trained on 5 separate corpora:
  - 2010 Wikipedia dump with 1 billion tokens
  - 2014 Wikipedia dump with 1.6 billion tokens
  - Gigaword 5 which has 4.3 billion tokens
  - the combination Gigaword5 + Wikipedia2014, which has 6 billion tokens
  - 42 billion tokens of web data, from Common Crawl
- You can download pre-trained word embeddings online.
  - GloVe pre-trained vectors
    - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download)
    - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download)
    - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download)
  - Hellinger PCA vectors
  - word2vec pre-trained vectors
  - etc.
- Transfer embedding to new task with smaller training set.
  - Language Modelling
  - Predictive Typing, Spelling/Grammar Correction, Summarization, NMT etc.

5.3 Dependency Parsing

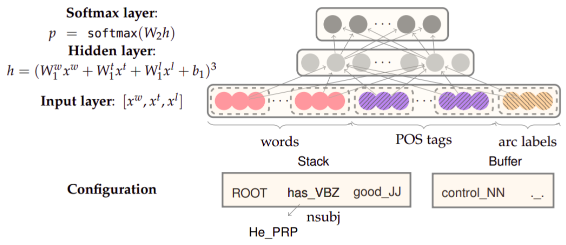
A dependency parser analyzes the grammatical structure of a sentence, establishing relationships between "head" words and words which modify those heads. The figure below shows a dependency parse of a short sentence.

Syntactic Parsing or Dependency Parsing is the task of recognizing a sentence and assigning a syntactic structure to it. The most widely used syntactic structure is the parse tree which can be generated using some parsing algorithms. These parse trees are useful in various applications like grammar checking or more importantly it plays a critical role in the semantic analysis stage.

Dependency parsing is the task of analyzing the syntactic dependency structure of a given input sentence S. The output of a dependency parser is a dependency tree where the words of the input sentence are connected by typed dependency relations. Formally, the dependency parsing problem asks to create a mapping from the input sentence with words $S = w_0 w_1 ... w_n$ (where w_0 is the ROOT) to its dependency tree graph G.



(a) A Fast and Accurate Dependency Parser using Neural Networks (Chen et al., 2014)

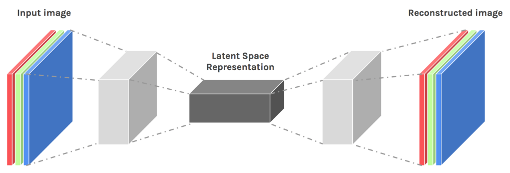


(b) CS224n Part IV

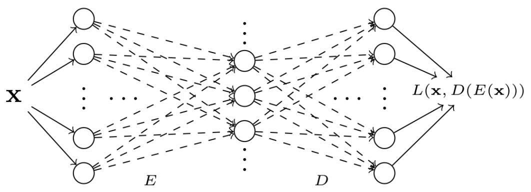
Figure 8: Dependency Parsing

5.4 Representation Learning

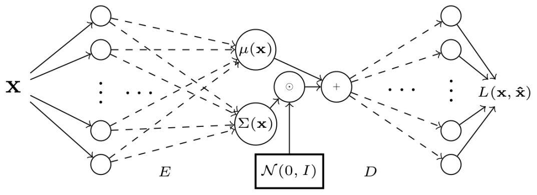
Various tasks can be modeled with representations. Since words can be represented by embeddings, images can be (or even signals/audio) represented by a latent space z. Autoencoders is a way to learn latent representation of data.



(a) Convolutional Autoencoder



(b) Autoencoder



(c) Variational Autoencoder

Figure 9: Autoencoders

6 Semantic & Syntactic Properties Of Embeddings

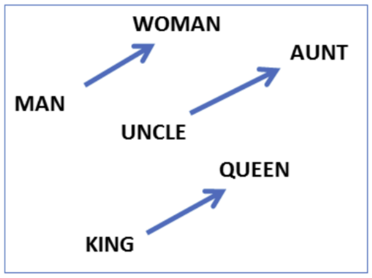
Word embeddings have very important property: analogies. Analogy is another semantic property of embeddings that can capture relational meanings. Simply in words, analogy is to find **X**:

- **A is to B as C is to X**

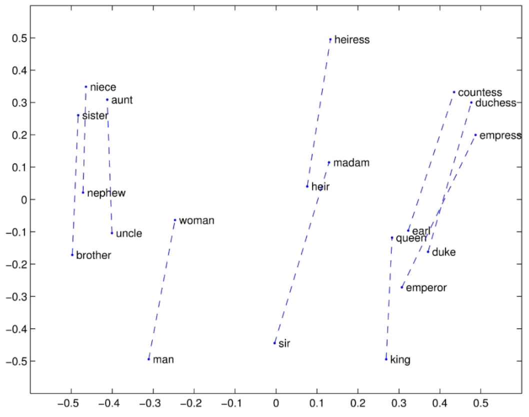
For example **"woman is to queen as man is to X**. In this example **X** should be the word **king**.

Interestingly, such embeddings exhibit seemingly linear behaviour in analogies. This linear behaviour can be formulated as

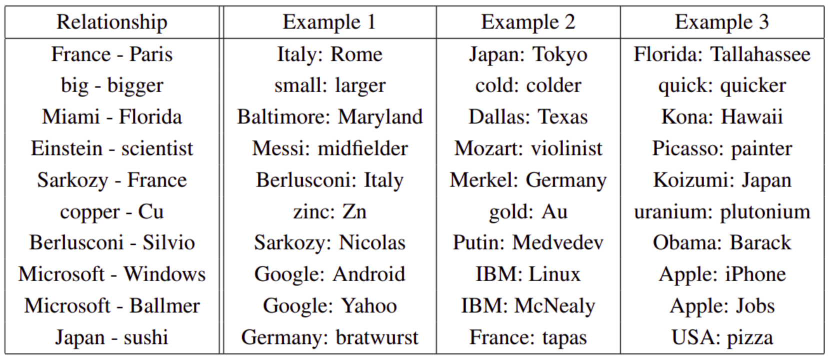
- w_a is to w'_a as w_b is to w'_b $\rightarrow\rightarrow\rightarrow$ $w'_a - w_a + w_b \approx w'_b$
- vec('*queen*') - vec('*woman*') + vec('*man*') $\approx$ vec('*king*')
- or
- vec('*queen*') - vec('*woman*') $\approx$ vec('*king*') - vec('textitman')
- Another example:
- vec('*Paris*') - vec('*France*') $\approx$ vec('*Italy*') - vec('*Rome*')



(a) Linguistic Regularities in Continuous Space Word Representations (Mikolov et al., 2013)



(b) Speech and Language Processing, Daniel Jurafsky, Third Edition

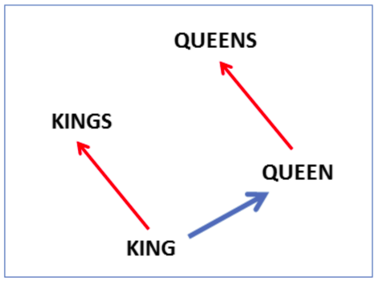


Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy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

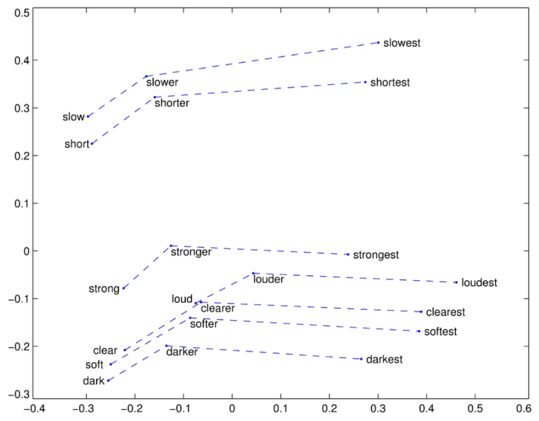
(c) Efficient Estimation of Word Representations in Vector Space (Mikolov etl al., 2013)

Figure 10: Semantic Properties Of Embeddings

Do vector embeddings capture syntactic relationships? Yes. Capture with linear behaviours? Yes.



(a) Stanford cs-230 cheatsheet



(b) Speech and Language Processing, Daniel Jurafsky, Third Edition

For more formal definition of analogy? Check Analogies Explained: Towards Understanding Word Embeddings (Allen et al., 2019)

7 Evaluating Word Vectors

asndjaklsşifkalg

- Extrinsic Evaluation
  - This is the evaluation on a real task.
  - Can be slow to compute performance.
  - Unclear if the subsystem is the problem, or our system.
  - If replacing subsystem improves performance, the change is likely good.
  - NER, Question Answering etc.
- Intrinsic Evaluation
  - Fast to compute
  - Helps to understand subsystem
  - Needs positive correlation with real task to determine usefulness

SimLex-999 dataset (Hill et al., 2015) gives values on a scale from 0 to 10 by asking humans to judge how similar one word is to another. Other datasets for evaluating word vectors:

- WordSim 353
- TOEFL Dataset
- SCWS Dataset
- Word-in-Context (WiC)
- Miller & Charles Dataset
- Rubenstein & Goodenough Dataset
- Stanford Rare Word (RW)

8 Learning Word Embeddings

8.1 Embedding Matrix

Embedding matrix can be represented as a single matrix $E \in \mathbb{R}^{d\times|V|}$ (or $E \in \mathbb{R}^{|V|\times d}$, it does not even matter), where d is embedding size and $\mid V \mid$ is size of the vocabulary V.

$$E = \begin{bmatrix} \text{hello} & \text{i} & \text{love} & \text{inzva} & \cdots & \text{sanctuary} \\ 0.1 & 0.137 & -0.03 & -0.44 & \cdots & 0.36 \\ -0.78 & -0.25 & 2.09 & 0.19 & \cdots & 0.32 \\ -3.1 & 1.54 & -2.52 & 0.2 & \cdots & -1.51 \\ 1.13 & -0.78 & -0.56 & 0.95 & \cdots & 0.2112 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -0.6 & 0.13 & 3.89 & -0.071 & -0.27 & 0.27 \end{bmatrix} \in \mathbb{R}^{d\times|V|}$$

But how to learn them?

8.2 A Neural Probabilistic Language Model (Bengio et al., 2003)

In neural language models, the prior context is represented by embeddings of the previous words. Representing the prior context as embeddings, rather than by exact words as used in n-gram language models, allows neural language models to generalize to unseen data much better than n-gram language models.

For example in our training set we see this sentence,

Today, after school, I am planning to go to cinema.

but we have never seen the word "concert" after the words "go to". In our test set we are trying to predict what comes after the prefix "Today, after school, I am planning to go to". An n-gram language model will predict "cinema" but not "concert". But a neural language model, which can make use of the fact that "cinema" and "concert" have similar embeddings, will be able to assign a reasonably high probability to "concert" as well as "cinema", merely because they have similar vectors.

In 2003 Bengio proposed a neural language model that can learns and uses embeddings to predict the next word in a sentence. Formally representation, for a sequence of words

$$x^{(1)}, x^{(2)}, ..., x^{(t)}$$

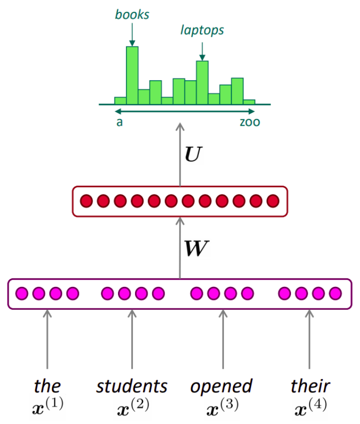
the probability distribution of the next word (output) is

$$p(x^{(t+1)} \mid x^{(1)}, x^{(2)}, ..., x^{(t)})$$

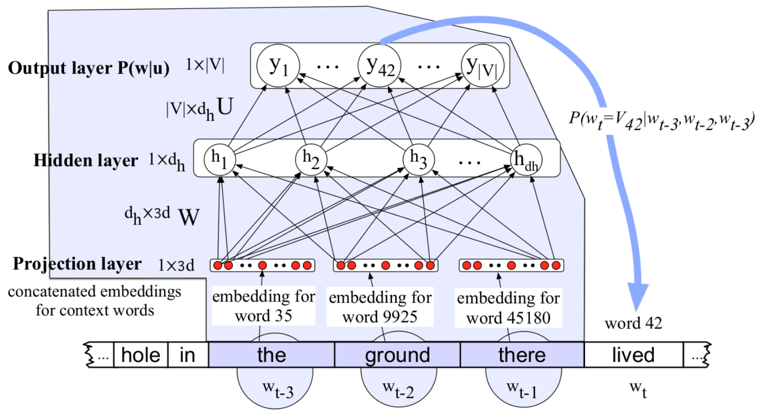
Bengio proposed a fixed-window neural Language Model which can be seen as the same approach of n-grams.

"~~as the proctor started the clock~~ $\underline{\text{the students opened their}}$.........."

We have a moving window at time t with an embedding vector representing each of the window size previous words. For window size 3, words $w_{t-1}, w_{t-2}, w_{t-3}$. These 3 vectors are concatenated together to produce input x. The task is to predict w_t.



(c) CS224n lecture 5



(d) Speech and Language Processing, Daniel Jurafsky, Third Edition

Figure 11: Neural Probabilistic Language Model

For this task, we are going to represent each of the N prevuios words as a one-hot-vector of length $\mid V \mid$. The forward equation for neural language model

- Input $x_i \in R^{1\times|V|}$
- Learning word embeddings: $e = concat(x_1E^T, x_2E^T, x_3E^T)$
  - $E \in \mathbb{R}^{d\times|V|}$
  - $e \in \mathbb{R}^{1\times 3d}$
- $h = \sigma(eW^T + b_1)$
  - $W \in \mathbb{R}^{d_h\times 3d}$
  - $h \in \mathbb{R}^{1\times d_h}$
- $z = hU^T + b_2$
  - $U \in \mathbb{R}^{|V|\times d_h}$
  - $z \in \mathbb{R}^{1\times|V|}$
- $\hat{y} = softmax(z) \in \mathbb{R}^{1\times|V|}$

Then the model is trained, at each word w_t, the negative log likelihood loss is:

$$L = -\log p(w_t \mid w_{(t-1)}, w_{(t-2)}, ..., w_{(t-n+1)}) = softmax(z)$$

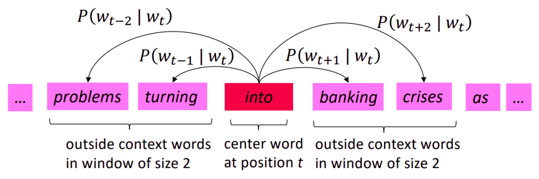
$$\theta_{t+1} = \theta_t - \eta\frac{\partial - \log p(w_t \mid w_{(t-1)}, w_{(t-2)}, ..., w_{(t-n+1)})}{\partial\theta}$$

9 word2vec

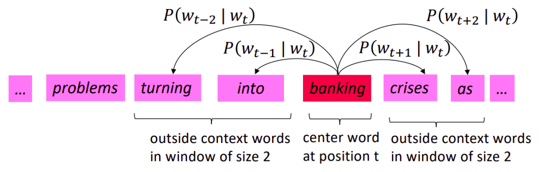
word2vec is proposed in the paper called Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al., 2013) . It allows you to learn the high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. Word2vec algorithm uses Skip-gram Efficient Estimation of Word Representations in Vector Space (Mikolov et al., 2013) model to learn efficient vector representations. Those learned word vectors has interesting property, words with semantic and syntactic affinities give the necessary result in mathematical similarity operations.

Suppose that you have a sliding window of a fixed size moving along a sentence: the word in the middle is the "target" and those on its left and right within the sliding window are the context words.

The skip-gram model is trained to predict the probabilities of a word being a context word for the given target.



(a) Learning Word Embedding



(b) nLearning Word Embedding

Figure 12: Target - Context pairs.

For example consider this sentence,

"A change in Quantity also entails a change in Quality"

Our target and context pairs for window size of 5:

Sliding window (size = 5)	**Target word**	**Context**
[A change in]	a	change, in
[A change in Quantity]	change	a, in, quantitiy
[A change in Quantity also]	in	a, change, quantitiy,also
...	...	...
[entails a change in Quality]	change	entails, a, in, Quality
[a change in Quality]	in	a, change, Quality
[change in Quality]	quality	change, in

Each context-target pair is treated as a new observation in the data.

For each position $t = 1, .., T$ predict context words within a window of fixed size m, given center word w_j. In Skip-gram connections we have an objective to maximize, likelihood (or minimize log-likelihood):

$$\max_{\theta} \prod_{\text{center}} \prod_{\text{context}} p(\text{context}|\text{center}; \theta)$$

$$= \max_{\theta} \prod_{t=1}^{T} \prod_{-c \leq j \leq c, j \neq c} p(w_{t+j} | w_t; \theta)$$

$$= \min_{\theta} -\frac{1}{T} \prod_{t=1}^{T} \prod_{-c \leq j \leq c, j \neq c} p(w_{t+j} | w_t; \theta)$$

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

So, How can we calculate those probabilities? Softmax gives the normalized probabilities.

9.1 Parameterization Of Skip-Gram Model

Let's say w_t is our target word and w_c is current context word. The softmax is defined as

$$p(w_c \mid w_t) = \frac{\exp(v_{w_c}^T v_{w_t})}{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})}$$

maximizing this log-likelihood function under v_{w_t} gives you the most likely value of the v_{w_t} given the data.

$$\frac{\partial}{\partial v_{w_t}} \cdot \log \frac{\exp(v_{w_c}^T v_{w_t})}{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})}$$

$$= \frac{\partial}{\partial v_{w_t}} \cdot \log \underbrace{\exp(v_{w_c}^T v_{w_t})}_{\text{numerator}} - \frac{\partial}{\partial v_{w_t}} \cdot \log \underbrace{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})}_{\text{denominator}}$$

$$\frac{\partial}{\partial v_{w_t}} \cdot v_{w_c}^T v_{w_t} = v_{w_c} \quad \text{(numerator)}$$

Now, it is time to derive denominator.

$$\frac{\partial}{\partial v_{w_t}} \cdot \log \sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t}) = \frac{1}{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})} \cdot \frac{\partial}{\partial v_{w_t}} \sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})$$

$$= \frac{1}{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})} \cdot \sum_{i=0}^{|V|} \frac{\partial}{\partial v_{w_t}} \cdot \exp(v_{w_i}^T v_{w_t})$$

$$= \frac{1}{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})} \cdot \sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t}) \frac{\partial}{\partial v_{w_t}} v_{w_i}^T v_{w_t}$$

$$= \frac{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t}) \cdot v_{w_i}}{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})} \quad \text{(denominator)}$$

To sum up,

$$\frac{\partial}{\partial w_t} \log p(w_c \mid w_t) = v_{w_c} - \frac{\sum_{j=0}^{|V|} \exp(v_{w_j}^T v_{w_t}) \cdot v_{w_j}}{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})}$$

$$= v_{w_c} - \sum_{j=0}^{|V|} \frac{\exp(v_{w_j}^T v_{w_t})}{\sum_{i=0}^{|V|} \exp(v_{w_i}^T v_{w_t})} \cdot v_{w_j}$$

$$\underbrace{= v_{w_c} - \sum_{j=0}^{|V|} p(w_j \mid w_t) \cdot v_{w_j}}_{\nabla_{w_t} \log p(w_c|w_t)}$$

This is the observed representation subtract $\mathbb{E}[w_j \mid w_t]$.

9.2 Negative Sampling (Noise Contrastive Estimation (NCE))

The Noise Contrastive Estimation (NCE) metric intends to differentiate the target word from noise samples using a logistic regression classifier (Noise-contrastive estimation: A new estimation principle for unnormalized statistical models, Gutmann et al., 2010).

In softmax computation, look at the denominator. The summation over $\mid V \mid$ is computationally expensive. The training or evaluation takes asymptotically $O(\mid V \mid)$. In a very large corpora, the most frequent words can easily occur hundreds or millions of times ("in", "and", "the", "a" etc.). Such words provides less information value than the rare words. For example, while the skip-gram model benefits from observing co-occurences of "inzva" and "deep learning", it benefits much less from observing the frequent co-occurences of "inzva" and "the".In a very large corpora, the most frequent words can easily occur hundreds or millions of times ("in", "and", "the", "a" etc.). Such words provides less information value than the rare words. For example, while the skip-gram model benefits from observing co-occurences of "inzva" and "deep learning", it benefits much less from observing the frequent co-occurences of "inzva" and "the".

For every training step, instead of looping over the entire vocabulary, we can just sample several negative examples! We "sample" from a noise distribution $P_n(w)$ whose probabilities match the ordering of the frequency of the vocabulary.

Consider a pair (w_t, w_c) of word and context. Did this pair come from the training data? Let's denote by $p(D = 1 \mid w_t, w_c)$ the probability that (w_t, w_c) came from the corpus data. Correspondingly $p(D = 0 \mid w_t, w_c)$ will be the probability that (w_t, w_c) didn't come from the corpus data. First, let's model $p(D = 1 \mid w_t, w_c)$ with sigmoid:

$$p(D = 1 \mid w_t, w_c) = \sigma(v_{w_c}^T v_{w_t}) = \frac{1}{1 + \exp(-v_{w_c}^T v_{w_t})}$$

Now, we build a new objective function that tries to maximize the probability of a word and context being in the corpus data if it indeed is, and maximize the probability of a word and context not being in the corpus data if it indeed is not. Maximum likelihood says:

$$\max \prod_{(w_t,w_c)\in D} p(D = 1 \mid w_t, w_c) \times \prod_{(w_t,w_c)\in D'} p(D = 0 \mid w_t, w_c)$$

$$= \max \prod_{(w_t,w_c)\in D} p(D = 1 \mid w_t, w_c) \times \prod_{(w_t,w_c)\in D'} 1 - p(D = 1 \mid w_t, w_c)$$

$$= \max \sum_{(w_t,w_c)\in D} \log p(D = 1 \mid w_t, w_c) + \sum_{(w_t,w_c)\in D'} \log(1 - p(D = 1 \mid w_t, w_c))$$

$$= \max \sum_{(w_t,w_c)\in D} \log \frac{1}{1+\exp(-v_{w_c}^T v_{w_t})} + \sum_{(w_t,w_c)\in D'} \log \left(1 - \frac{1}{1+\exp(-v_{w_c}^T v_{w_t})}\right)$$

Note that $\frac{\exp(-x)}{(1+\exp(-x))} \times \frac{\exp(x)}{\exp(x)} = \frac{1}{(1+\exp(x))}$

$$= \max \sum_{(w_t,w_c)\in D} \log \frac{1}{1+\exp(-v_{w_c}^T v_{w_t})} + \sum_{(w_t,w_c)\in D'} \log \frac{1}{1+\exp(v_{w_c}^T v_{w_t})}$$

Maximizing the likelihood is the same as minimizing the negative log likelihood:

$$L = -\sum_{(w_t,w_c)\in D} \log \frac{1}{1+\exp(-v_{w_c}^T v_{w_t})} - \sum_{(w_t,w_c)\in D'} \log \frac{1}{1+\exp(v_{w_c}^T v_{w_t})}$$

The Negative Sampling (NEG) proposed in the original word2vec paper. NEG approximates the binary classifier's output with sigmoid functions as follows:

$$p(d=1|v_{w_c}, v_{w_t}) = \sigma(v_{w_c}^T v_{w_t})$$

$$p(d=0|v_{w_c}, v_{w_t}) = 1 - \sigma(v_{w_c}^T v_{w_t}) = \sigma(-v_{w_c}^T v_{w_t})$$

So the objective is

$$L = -[\log \sigma(v_{w_c}^T v_{w_t}) + \sum_{\substack{i=1 \\ \tilde{w}_i \sim Q}}^{K} \log \sigma(v_{\tilde{w}_i}^T v_{w_t})]$$

In the above formulation, $v_{\tilde{w}_i} \mid i = 1...K$ are sampled from $P_n(w)$. How to define $P_n(w)$? In the word2vec paper $P_n(w)$ defined as

$$P_n(w_i) = 1 - \sqrt{\frac{t}{freq(w_i)}} \quad t \approx 10^{-5}$$

This distribution assigns lower probability for lower frequency words, higher probability for higher frequency words.

Hence, this distribution is sampled form a unigram distribution $U(w)$ raised to the $\frac{3}{4}$rd power. The unigram distribuiton is defined as

$$P_n(w) = \left(\frac{U(w)}{Z}\right)^{\alpha}$$

Or just by Andrew NG's definition:

$$P_n(w_i) = \frac{freq(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{M} freq(w_j)^{\frac{3}{4}}}$$

Raising the unigram distribution $U(w)$ to the power of α has an effect of smoothing out the distribution. It attempts to combat the imbalance between common words and rare words by decreasing the probability of drawing common words, and increasing the probability drawing rare words.

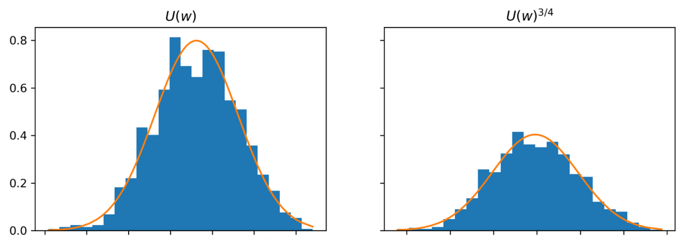


Figure 13

```
import numpy as np
unig_dist  = {'inzva': 0.023, 'deep': 0.12,
'learning': 0.34, 'the': 0.517}
print(sum(unig_dist.values()))
#output: 1

alpha      = 3 / 4
noise_dist = {key: val ** alpha for key, val in unig_dist.items()}
Z = sum(noise_dist.values())
noise_dist_normalized = {key: val / Z for key,
                         val in noise_dist.items()}
print(noise_dist_normalized)

#output: {'inzva': 0.044813853132981724,
#  'deep': 0.15470428538870049,
#  'learning': 0.33785130228003507,
#  'the': 0.4626305591982827}

print(sum(noise_dist_normalized.values()))
#output1.0

K = 10
print(np.random.choice(list(noise_dist_normalized.keys()),
     size=K, p=list(noise_dist_normalized.values())))
#output: array(['the', 'the', 'deep',
#'learning', 'the', #'learning', 'the',
#'inzva', 'the', 'learning'], dtype='<U8')
```

9.3 Hierarchical Softmax

Hierarchical softmax is proposed to make the sum calculation faster with the help of a binary tree structure. The model uses a binary tree to represent all words in the vocabulary. The $| V |$ words must be leaf units of the tree.

Hieararchical softmax is a technique for **aproximating** naive softmax. Hieararchical softmax uses binary tree, all nodes have 2 children, and is organized like Huffmann Tree. Rare words are down at deeper levels and frequent words are at shallower levels.

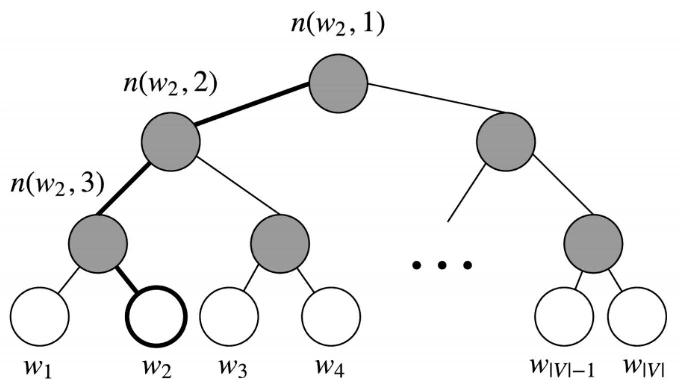


Figure 14: Hierarchical Softmax

Let's introduce some notation. Let $L(w)$ be the number of nodes in the path from the root to the leaf w. For instance, above figure, $L(w_2)$ is 3. Let's write $n(w,i)$ as the i-th node on this path with associated vector $v_{n(w,i)}$. Now for each inner node n, we arbitrarily choose one of its children and call it $ch(n)$ (e.g. always the left node). Then, we can compute the probability as

$$P(w \mid w_i) = \prod_{j=0}^{L(w)-1} \sigma([n(w,j+1) == ch(n(w,j))] \cdot v_{n(w,j)}^T v_{w_i})$$

. First, we are computing a product of terms based on the shape of the path from the root $n(w,i)$ to leaf (w). If we assume $ch(n)$ is always the left node of n, then term $[n(w,j+1) == ch(n(w,j))]$ returns 1 when the path goes left, and -1 if right. Furthermore, the term $[n(w,j+1) == ch(n(w,j))]$ provides normalization. At a node n, if we sum the probabilities for going to the left and right node, you can check that for any value of $v_{n(w,j)}^T v_{w_i}$

$$\sigma(v_{n(w,j)}^T v_{w_i}) + \sigma(-v_{n(w,j)}^T v_{w_i}) = 1$$

Finally, we compare the similarity of our input vector v_{w_i} to each inner node vector $v_{n(w,j)}^T$ using a dot product.

Taking w_2 in Figure we must take two left edges and then a right edge to reach w_2 from the root, so

$$P(w_2 \mid w_i) == p(n(w_2,1), left) \cdot p(n(w_2,2), left) \cdot p(n(w_2,3), right)$$

$$= \sigma(v_{n(w,1)}^T v_{w_i}) \cdot \sigma(v_{n(w,2)}^T v_{w_i}) \cdot \sigma(-v_{n(w,3)}^T v_{w_i})$$

To train the model, our goal is still to minimize the negative log likelihood $-\log P(w \mid w_i)$.

So, instead $O(\mid V \mid)$, our complexity is the depth of the tree, $O(\log \mid V \mid)$, in worst-case.

10 GloVe: Global Vectors for Word Representation

GloVe (GloVe: Global Vectors for Word Representation, Pennington et al., 2014) combines the advantages of global matrix factorization (such as Latent Semantic Analysis) and local context window methods. Methods like skip-gram (word2vec) may do better on the analogy task, but they poorly utilize the statistics of the corpus since they train on separate local context windows insted of on global co-occurence counts. GloVe defines matrix $\mathbf{X} = [X_{ij}]$ which represents word-word co-occurence counts. Entries X_{ij} tabluate the number of times word j occurs in the context of word i. If we choose our context-target pairs as in word2vec algorithm, then $\mathbf{X}$ would be symmetric. If we choose our context-target pairs in one direction (left or right), then $X_{ij} \neq X_{ji}$. The objective function of GloVe comes from a relation between symmetry and homomorphism between $(\mathbb{R}, +)$ and $(\mathbb{R}_{>0}, \times)$. Since this is not scope of the course, we will evaluate the objective function intuitively. The objective function of GloVe is defined as

$$J = \sum_{i,j=0}^{|V|} f(X_{ij})(w_i^T \tilde{w_j} + b_i + \tilde{b_j} - \log X_{ij})^2$$

i and j are playing the role of target and context. The term $w_i^T \tilde{w_j} - \log X_{ij}$ says that "how related are those two words?" as measured by how often they occur with each other. But what if words i and j never accour together? Then $X_{ij} = 0$, and $\log 0$ is not defined. So we need to define a weigthing term $f(X_{ij})$:

1. $f(0) = 0$. We are going to use a convention that "$0 \log 0 = 0$".
2. $f(x)$ should be non-decreasing so that rare word co-occurences are not overweighted.
3. $f(x)$ should be relatively small for large values of x, so that frequent co-occurences are not overweighted.

Yes there are a lot of functions that have characteristic like f. GloVe choose f as:

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^{\alpha}, & \text{if } x < x_{\max} \\ 1, & \text{otherwise} \end{cases}$$

The most beautiful part of this algoirthm is that w_i and $\tilde{w_j}$ are symmetric. This the role of derivation of objective is same for w_i and $\tilde{w_j}$ when $i = j$. $\rightarrow$ The model generates two set of vectors, W and $\tilde{W}$. When X is symmetrix, W and $\tilde{W}$ are equivalent and differ only as a result of their random initializations. At final level, we are able to choose

$$\text{word}_k^{\text{final}} = \frac{w_k + \tilde{w_k}}{2}$$

11 Final Notes On Word Vectors

GloVe outperforms word2vec, but word2vec is more popular than Glove.

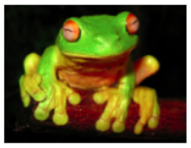
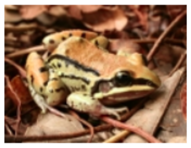
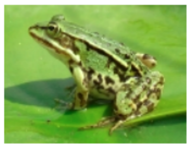
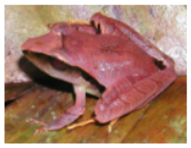
nearest neighbors of *frog*	**Litoria**	**Leptodactylidae**	**Rana**	**Eleutherodactylus**
Pictures				

Figure 15: Source: Official GloVe implementation repository

11.1 Roles Of Symmetry, Dimension & Window Size

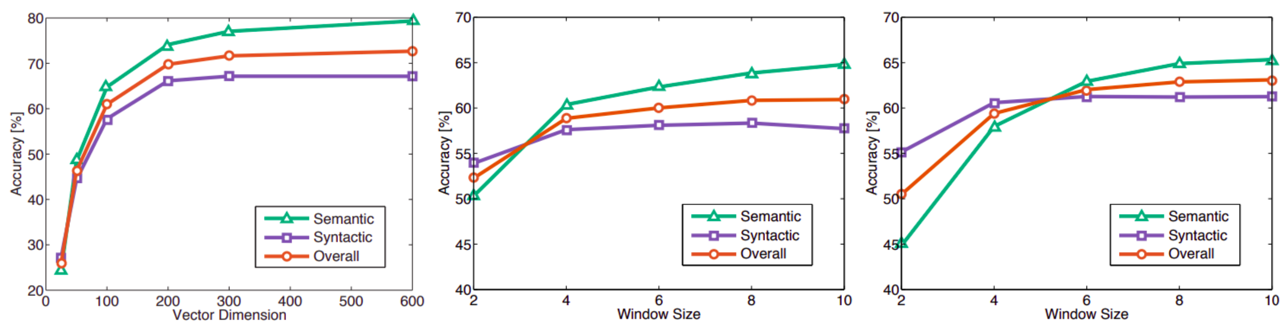


Figure 16: GloVe: Global Vectors for Word Representation, Pennington et al., 2014)

They observed diminishing returns for vectors larger than about 200 dimensions. Performance is better on the syntactic subtask for small and asymmetric context windows, which aligns with the intuition that syntactic information is mostly drawn from the immediate context and can depend strongly on word order. Semantic information, on the other hand, is more frequently non-local, and more of it is captured with larger window sizes.

11.2 Roles Of Corpus

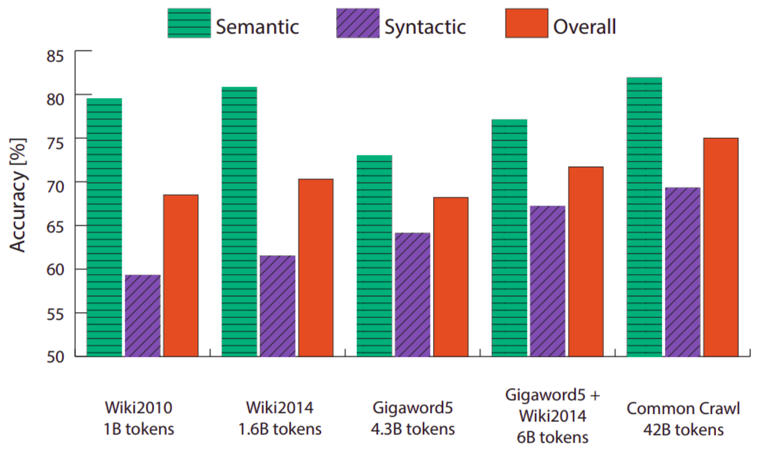


Figure 17: GloVe: Global Vectors for Word Representation, Pennington et al., 2014)

On the syntactic subtask, there is a monotonic increase in performance as the corpus size increases. This is to be expected since larger corpora typically produce better statistics. Interestingly, the same trend is not true for the semantic subtask, where the models trained on the smaller Wikipedia corpora do better than those trained on the larger Gigaword corpus. Due to, Wikipedia has fairly more comprehensive articles than news.

11.3 Are Word Vectors Only Learned With Neural Networks?

No:

- Term-Document Matrix in information retrieval.
- Word-Word Co-occurence Matrix.

- TF-IDF:

$$\text{tf}_{t,d} = \text{count}(t, d)$$

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

$$N : \text{total number of documents}, \ \text{df}_t : \text{total number of documents includes word} \ \ t$$

- Positive Pointwise Mutual Information (PPMI):

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}, \quad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}, \quad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$\text{PPMI}_{ij} = \max\left(\log_2 \frac{p_{ij}}{p_{i*}p_{*j}}\right)$$

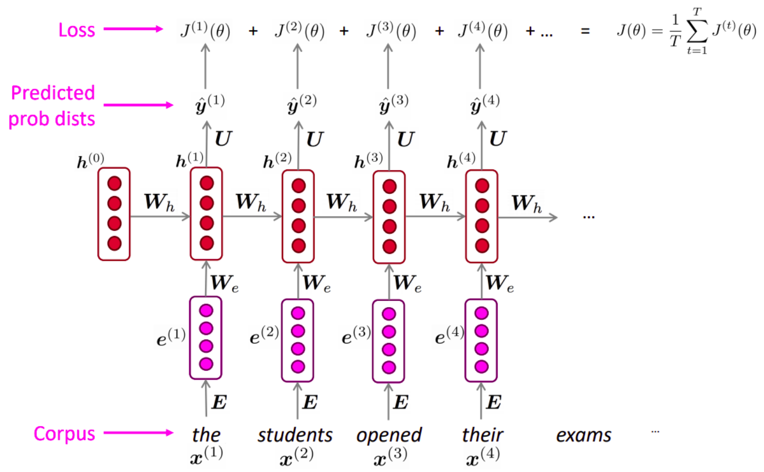
- Singular Value Decomposition (SVD):

$$X = U\Sigma V^T$$

- Hellinger-PCA
- etc.

12 Word Vectors In Various Tasks

Word vectors can be used in sentiment analysis, dependency parsing, language modeling etc. But also, we'd like to learn the embeddings simultaneously with training the network.



Loss
$J^{(1)}(\theta)$ + $J^{(2)}(\theta)$ + $J^{(3)}(\theta)$ + $J^{(4)}(\theta)$ + ... = $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$
Predicted prob dists
$\hat{y}^{(1)}$ $\hat{y}^{(2)}$ $\hat{y}^{(3)}$ $\hat{y}^{(4)}$
U
$h^{(0)}$ $h^{(1)}$ $h^{(2)}$ $h^{(3)}$ $h^{(4)}$
W_h
...
W_e
$e^{(1)}$ $e^{(2)}$ $e^{(3)}$ $e^{(4)}$
E
Corpus
the students opened their exams ...
$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$

Figure 18: Natural Language Processing with Deep Learning CS224N/Ling284

13 Bias In Word Vectors

Embeddings also reproduce the implicit biases and stereotypes that were latent in text. Word embeddings can reflect gender, ethnicity, age, sexual orientation etc. Bolukbasi et al., 2016 found that 'man' - 'computer programmer' + 'woman' in word2vec embeddings trained on news text is 'homemaker', alsı 'father' is to 'doctor' as 'mother' is to 'nurse'. Historical embeddings are also being used in to measure biases in the past.

14 References

1 Deep Learning, NLP, and Representations. Christopher Olah. July 7, 2014. URL: https:// colah.github.io /posts/2014-07-NLP-RNNs-Representations/

2 CS 230 - Deep Learning. Afshine Amidi and Shervine Amidi. URL: https:// stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

3 Named Entity Recognition (NER) with keras and tensorflow. Nasir Safdari. Dec 13, 2018. URL: https:// towardsdatascience.com/ named-entity-recognition-ner-meeting-industrys - requirement - by-applying-state-of-the-art-deep-698d2b3b4ede

4 Named-entity recognition. URL: https://deepai.org/ machine-learning-glossary-and-terms/ named - entity-recognition

5 CS224n: Natural Language Processing with Deep Learning. Christopher Manning, Richard Socher, Lisa Wang, Juhi Naik, and Shayne Longpre. URL: https://web.stanford.edu/class/ cs224n/

6 CS224n: Natural Language Processing with Deep Learning - Word Vectors Slides. Christopher Manning, Richard Socher, Lisa Wang, Juhi Naik, and Shayne Longpre. URL: https:// web.stanford.edu /class/cs224n/slides/cs224n-2021-lecture01-wordvecs1.pdf

7 CS224n: Natural Language Processing with Deep Learning - Word Vectors 2 Notes. Christopher Manning, Richard Socher, Lisa Wang, Juhi Naik, and Shayne Longpre. URL: https:// web.stanford.edu /class/cs224n/readings/cs224n-2019-notes01-wordvecs1.pdf

8 CS224n: Natural Language Processing with Deep Learning - Word Vectors 2 Slides. Christopher Manning, Richard Socher, Lisa Wang, Juhi Naik, and Shayne Longpre. URL: https:// web.stanford.edu /class/cs224n/slides/cs224n-2021-lecture02-wordvecs2.pdf

9 CS224n: Natural Language Processing with Deep Learning - Word Vectors 2 Notes. Christopher Manning, Richard Socher, Lisa Wang, Juhi Naik, and Shayne Longpre. URL: https:// web.stanford.edu /class/cs224n/readings/cs224n-2019-notes02-wordvecs2.pdf

10 CS224n: Natural Language Processing with Deep Learning - Dependency Parsing Notes. Christopher Manning, Richard Socher, Lisa Wang, Juhi Naik, and Shayne Longpre. URL: https:// web.stanford.edu /class/cs224n/readings/cs224n-2019-notes04-dependencyparsing.pdf

11 CS224n: Natural Language Processing with Deep Learning - RNNLM Slides. Christopher Manning, Richard Socher, Lisa Wang, Juhi Naik, and Shayne Longpre. URL: https:// web.stanford.edu /class/cs224n/slides/cs224n-2021-lecture05-rnnlm.pdf

12 Stanford Parser. URL: https:// nlp.stanford.edu /software/nndep.html

13 Learning Word Embedding. Oct 15, 2017. Lilian Weng. URL: https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html#context-based-skip-gram-model

14 Speech and Language Processing. Dan Jurafsky and James H. Martin. URL: https:// web.stanford.edu / jurafsky/slp3/

15 Natural Language Processing (almost) from Scratch (Collobert et al., 2011)

16 Efficient Estimation of Word Representations in Vector Space (Mikolov etl al., 2013)

17 Analogies Explained: Towards Understanding Word Embeddings (Allen et al., 2019)

18 Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al., 2013)

19 GloVe: Global Vectors for Word Representation, Pennington et al., 2014