Detecting semantically similar words using Stanford's GloVe

Suryaprasath Ramalingam

19315

Department of Electrical Engineering and Computer Science, IISER Bhopal

Abstract

The field of Natural Language Processing (NLP) has been under extensive research for decades. Under this field, Semantic Analysis is one of the most critical phases for any language processing task. Researchers have developed multiple learning algorithms for the same. However, many of these models have various shortcomings. This project will focus on Global Vectors for Word Representation (GloVe), one of the most efficient models that supposedly overcomes these shortcomings. The model was used to create text embeddings by finding hidden features between word co-occurrences. The word vectors for a set of words from the field of computer science were then extracted using predicted embeddings, and the top neighbors for each of these words were estimated for analysis. The results obtained were satisfactory, and it can be inferred that the model captured the hidden semantics within the text provided.

Introduction

This project aims at detecting semantically similar words within text using the GloVe model proposed by Stanford researchers. The text corpus involves words used extensively in the field of Computer Science. The project also aims at visually representing the obtained word vector space and finding the closest words to a specific set of computer-science related words.

Background Study

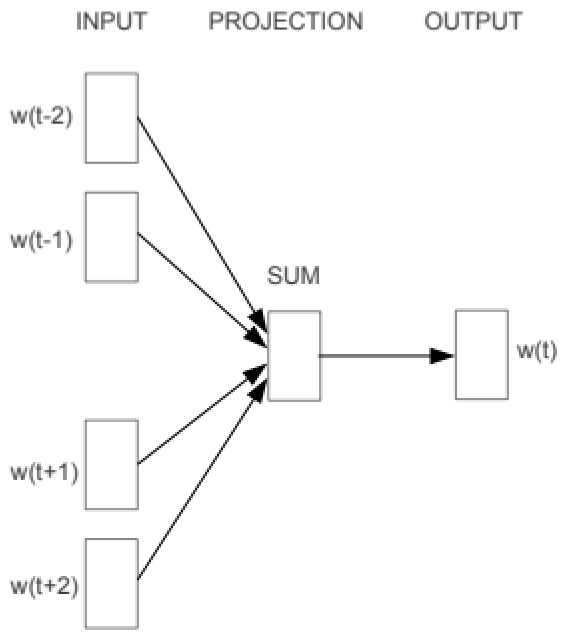
When it comes to finding hidden patterns within data, there are two approaches usually followed: using count-based methods and using predictive methods. One such method is Latent Semantic Analysis (LSA), where the cooccurrences of words can be statistically estimated. It is based on the distributional hypothesis, which states that correlated words often occur close to each other.

Using neural networks for language tasks is a predictive method for guessing words from learned distributions. With carefully chosen parameters and keeping the curse of dimensionality in mind, these networks can be constructed without compromising the accuracy. RNN (Recurrent Neural Network) is an example of modified NNs that have been explicitly proposed for many NLP tasks.

However, predictive models seem to outperform count-based models in multiple NLP tasks like the numeric estimation of semantic relatedness and analogy predictions. Google's Word2Vec, an unsupervised predictive model, is one such model which effectively captures semantics using Distributed Learning.

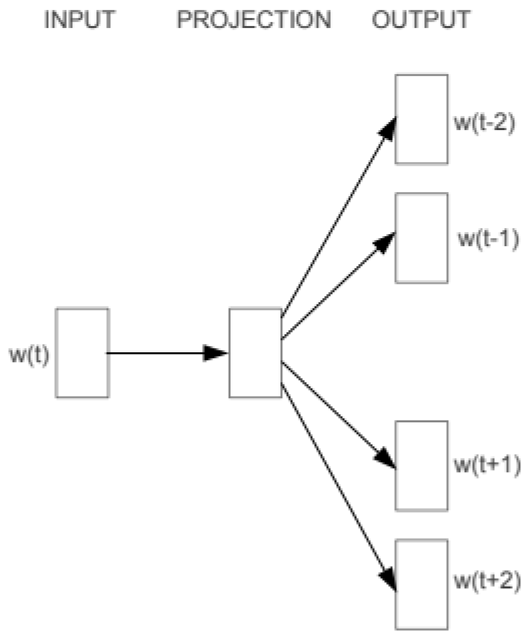
There are two model architectures for Word2Vec models, and either can be used depending on the task at hand: The Continuous Bag of Words (CBOW) model and the Skip-gram model. The CBOW model predicts the target word using source context words that often occur around the target word.

A specific window is chosen for extracting the surrounding words, whose numeric representations are computed through the embedding layer. The output from the embedding layer is then used to predict the target word from learned distributions.



CBOW

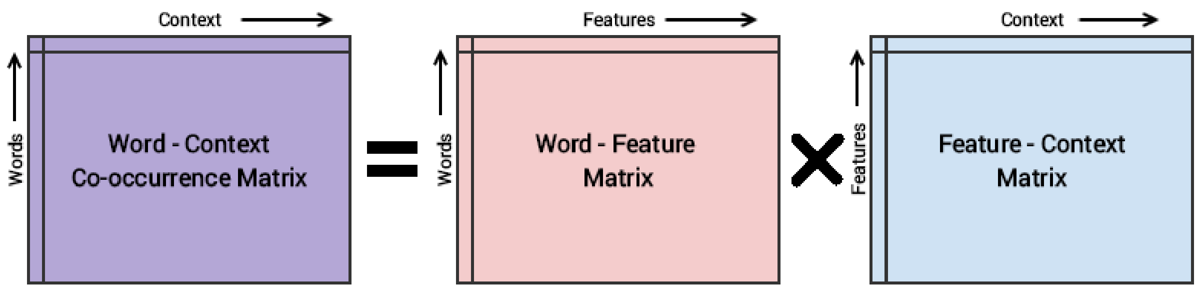
The Skip-gram model works almost opposite to the CBOW model. It predicts the surrounding words from the input word. In this way, multiple target words are predicted, rather than just one.



Skip-gram

Both the type of models, predictive and count-based, suffer some drawbacks. Predictive models do not capture semantics on the global level, while count-based models rely on statistics and do not perform well on analogy tasks and other tasks like the numeric estimation of semantic relatedness.

The GloVe model uses a different technique and is trained on the aggregated global word-to-word co-occurrences, thus capturing semantics and hidden substructures on the global level. For implementing this model, a co-occurrence matrix is constructed with preset window size. For each word, the context words of the same are represented by this matrix. This word co-occurrence matrix (word-context) can then be decomposed into two matrixes, one being the word-feature matrix and the other being the feature-context matrix. This decomposition can be performed either by SVD or can be taken as an optimization problem, and stochastic gradient descent can be used to approximate these matrices.



Methodology

Python was the primary language used in this project (excluding imported modules). Several modules were used and are listed below in detail. The techniques used in this project were carefully selected, with slight modifications to meet the objectives of the project.

Obtaining the text corpus

For this project, after multiple considerations, web articles from Wikipedia were extracted for the text corpus. The python modules *Wikipedia* and *BeautifulSoup* were used to extract the data.

Since Wikipedia articles follow a hierarchical structure, the articles under the category "Computer Science" at the first level were chosen. It included over 130 Wikipedia pages. Using *BeautifulSoup*, all these article names were extracted, and these names were passed as arguments to the Wikipedia module's search function. The text was then extracted using the in-built functions of the Wikipedia module.

Data cleaning

The module primarily used for this is the *Regex* python module. It uses pattern matching algorithms for substring search, where the pattern is to be passed as a string of a regular expression. Numeric data, single letters, stop words, and non-alphabetic characters were removed or replaced with a space bar. Full stops were kept until the end to differentiate between the next and previous documents, replaced by the newline character just before storing the text corpus as a text file. The stop words removed were obtained from the *NLTK* python library.

Training

The cleaned text data, with one or more spaces as a means of separation of words, was then passed as an argument to the bash run script of the *GloVe* module, appropriately modified for the given text. The module assured automatic training with the given corpus data. The word vector was then obtained after the training was completed.

Finding Closest Neighbors for the specific set of words

After the training phase, 6472 50-dimensional word vectors were generated and stored in a text file. Some of these word vectors may or may not represent a meaningful word due to the preprocessing errors and presence of alphabets used as variable names.

Two measures of "closeness" were chosen to find the closest neighbors: Euclidean Distance and Cosine Similarity. A specific set of words was manually selected for testing the learned representations. Six different words, which are the closest to the original word, were calculated and tabulated under results for each of these words. The words that is considered are: computer, process, cell, mouse, software, hardware, analog, data, storage, cpu, hub, network, learning, logic, trees, graphs, nodes, address, control and signal.

Visualizing

For visualizing the word vectors, *matplotlib* was used extensively. Since 50-dimensional vectors cannot be visualized, dimensionality reduction was applied to the word vectors. Two techniques were used for this phase: Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor (t-SNE). The word vectors were reduced to 2 dimensions and were plotted using *matplotlib*. The results after applying both the techniques are tabulated under results.

Table 1: Top six closest neighbors for each word (Euclidean Distance as distance measure)

Word	**Top 6 closest neighboring words***					
computer	input	classes	computational	object-oriented	images	simultaneous
process	mathematical	finding	positive	Labs	connection	relays
cell	permits	restriction	latency	Nmos	convolution	constants
mouse	editing	landin	increment	Dimensionality	pram	stating
software	memory	increase	bug	Statically	complexity	Capabilities
hardware	firewall	service	unrooted	Scope	complete	communications
analog	signals	grows	polish	Broader	ashby	adjacency
data	hash	limits	model	Computational	determining	encoded
storage	large-scale	devices	mass	Responsibility	determining	primitives
cpu	partial	bound	counting	Tax	incorporated	instructions
hub	inexpensive	singularity	field-programmable	Administrative	colors	representatives
network	uses	one-dimensional	suitable	Parent	circle	asked

learning	cells	regula	solved	Reused	suction	disadvantage
logic	feature	formalism	yield	Triangle	misleading	desired
trees	rewriting	trend	ipc	Ben	tree	rewrite
graphs	graph	acids	ability	Node	hierarchy	divisions
nodes	newell	kutta	article	Produces	ability	hierarchy
address	decision	head	events	Transactions	framework	tutorials
control	exact	closely	reductions	Voltage	charge	real-world
signal	quantity	politics	employ	Sector	attack	shape

Table 2: Top six closest neighbors for each word (Cosine Similarity as distance measure)

Word	**Top 6 closest neighboring words***					
computer	problems	system	operations	Type	theory	computational
process	engineering	software	functions	Languages	model	data
cell	memory	instance	open	Algorithmic	levels	keys
mouse	provability	android	card	Reals	guidance	neg
software	memory	computer	functions	Logic	true	create
hardware	computer	execution	study	Basic	problem	software
analog	signal	digital	processing	Bioinformatics	information	data
data	model	hardware	theory	Program	functions	system
storage	data	testing	instruction	Attacks	geometry	frac
cpu	instruction	infinite	practical	Code	memory	hierarchy
hub	private	characterized	robert	Strategies	addressing	anonymous

network	uses	algebra	shared	Living	cpu	standard
learning	cells	solved	models	Machine	computability	implementation
logic	programming	object	theory	Child	formal	solutions
trees	tree	regulation	characters	Classes	cpu	statistics
graphs	graph	architectures	detection	Previous	input	node
nodes	node	tree	network	Small	basis	real
address	decision	head	array	Memory	ciphers	machines
control	systems	algebra	problem	Constructed	system	function
signal	clock	processing	shape	Graph	probability	proposed

**Note that some words were deliberately excluded from the list. These include common symbols and stop words that may have appeared after preprocessing. These also include words that are very common but not present in the NLTK stop words list.*

Figure 1: Word Vector Representation after t-SNE dimensionality reduction

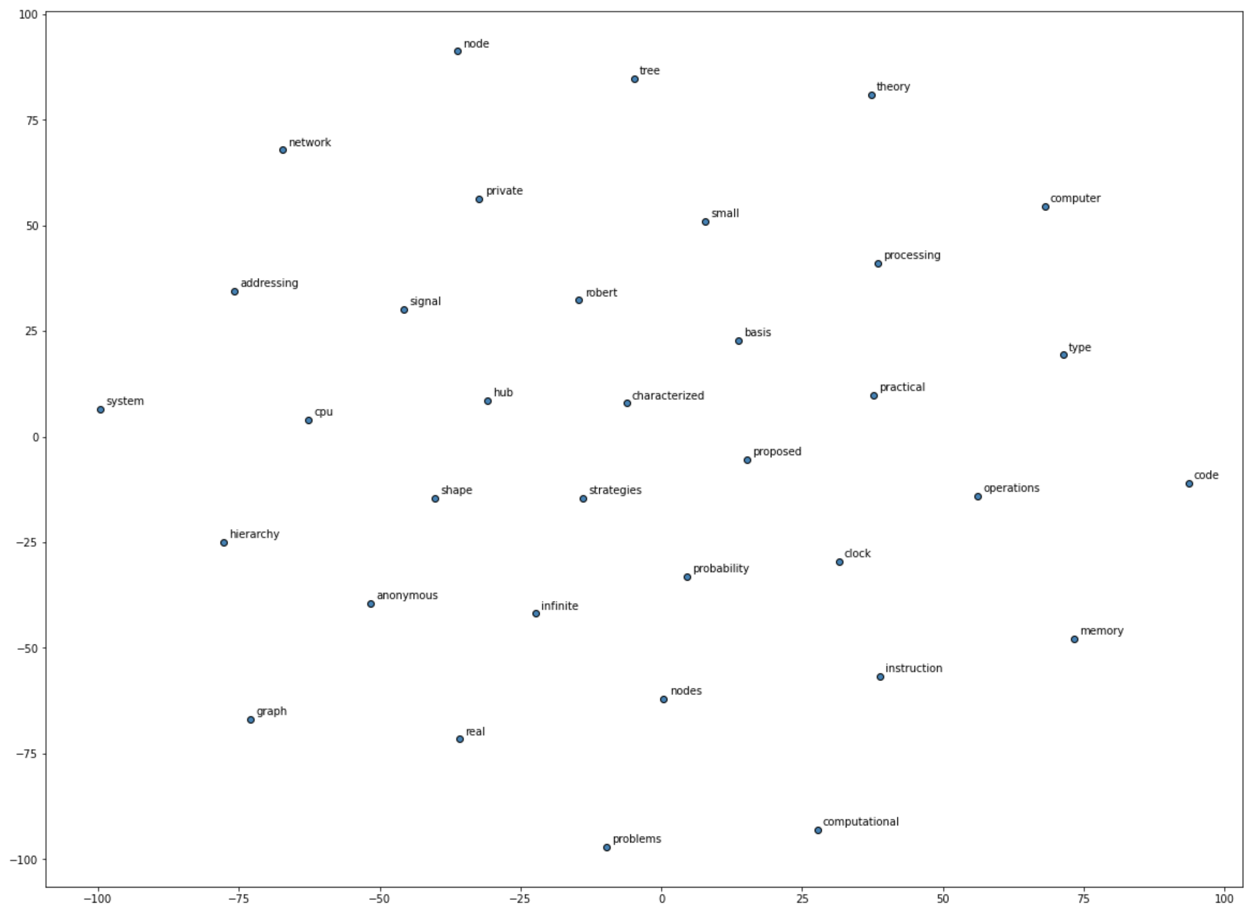
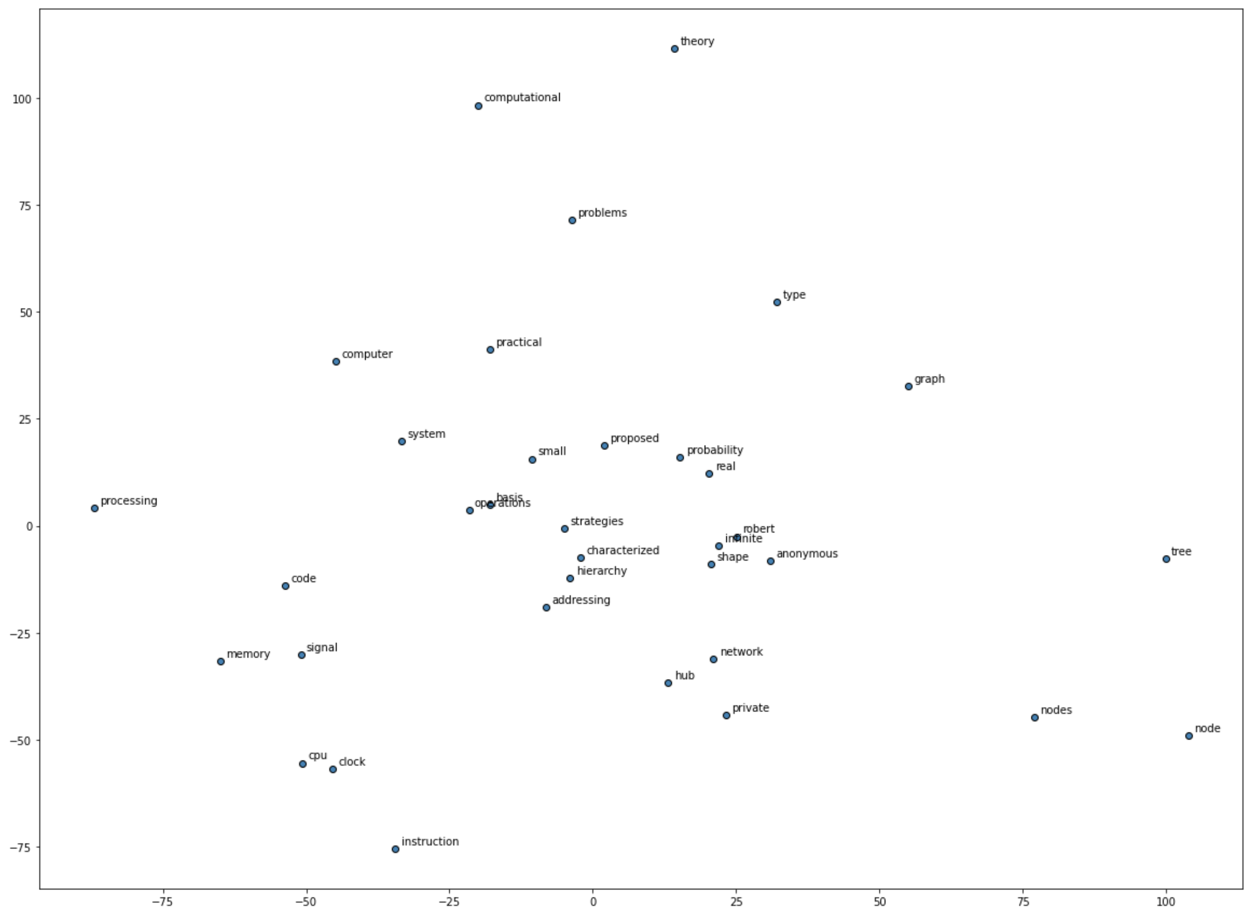


*Figure 2: Word Vector Representation after PCA dimensionality reduction***



**Note that PCA returns dimensions that are within one and zero. However, for clear visualization, the dimensions have been scaled by the same amount.

Discussion

The GloVe model used in this project worked as expected. Despite having a relatively high number of data points, very rare words were also able to be located close to the respective surrounding words. Moreover, very commonly used words across multiple documents (like 'data') had their closest words from different articles. Thus, it is evident that it captured the global semantic structure.

For the distance measure, cosine similarity was chosen as the base measure. The closest neighbors from the tables above clearly indicate that Euclidean distance is not a good measure as many words do not have any relations and yet occur close. Thus, cosine similarity was used to pick the closest neighbors.

For each of the words chosen to analyze, the closest six words were chosen to be plotted. The results after using PCA and t-SNE for dimensionality reduction is clearly different. This can be attributed to the complexity in choosing the right parameters for t-SNE as clearly, the points haven't clustered due to lack of convergence. Clearly, choosing the right parameters is also important for clustering.

Lemmatization hasn't been considered in this project. This is evident from the fact that the closest word to 'nodes' is 'node'. If the text corpus had been lemmatized, then the model could have removed duplicate entries and provided better results.

The GloVe module used a high cooccurrence window size of 25 words per target word. This helped in capturing larger hidden structures in longer sentences. This can also reveal relations between consecutive sentences.

The training for this corpus was much faster due to optimized codes written in low-level languages (C and C++). Since the model was pre-built, only the input text corpus with space separations was required to feed into the model.

Although we have trained over a large dataset, this is usually not enough. This is because one sentence cannot effectively reveal all the hidden connections between words. The original GloVe model trained with over 840 billion word-tokens, and therefore showed promising results in the original paper. Many more sentences could have helped the model show even better results.

Wikipedia articles, that were the source material for the text corpus, are topic-specific. Therefore, many of the important words often occur only on a small subset of the total number of articles. Hence, it was difficult to capture global semantics efficiently.

Conclusion

Despite the limited data, the GloVe model proved to be effective. Choosing the appropriate parameters and techniques is important in every phase as described in the project, as the outcome is heavily dependent on the same. Hence, we were able to detect semantics between the words with minor errors only. Given a better, and bigger text corpus, the model is guaranteed to perform better as the quality of the model improves with the quality and size of the data.

References

Baroni, Marco, Georgiana Dinu and German Kruszewski. "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors." 2014.

Mikolov, Tomas, et al. "Efficient Estimation of Word Representations in Vector Space." 2013.

Pennington, Jeffrey, Richard Socher and Christopher D. Manning. "GloVe: Global Vectors for Word Representation." 2014.

Sarkar, Dipanjan. "A hands-on intuitive approach to Deep Learning Methods for Text Data — Word2Vec, GloVe and FastText." 14 May 2018. *Towards Data Science.* 7 October 2021.