Training Machines to Learn about Machine Learning: Word2vec for Technical Skills

Sam Goodgame
UC Berkeley School of Information
samuel.goodgame@ischool.berkeley.edu

Abstract

This paper trains a skip-gram Word2vec model in order to make accurate inferences about technical skills. The training data comes from 106,000 job descriptions scraped from the Internet, along with all posts from Stackoverflow with a score greater than 25. The resulting model has many practical uses, including the ability to generate a canonical list of technical skills and the ability to elucidate the relationships between them. Using the same analogy tasks that the original Word2vec paper proposed, the baseline accuracy of this model on a set of 1,954 general language model analogy tasks is 18.17%, and the final model achieved 28.08% accuracy on the same set. However, this paper proposes a novel evaluation method designed specifically for this data and task. The evaluation process involves three steps: (1) collecting sentences from Wikipedia from articles about technical skills; (2) "corrupting" the sentences by replacing the skill words with randomly selected incorrect skill words; and (3) evaluating how likely the model deems the set of correct sentences relative to the corrupted sentences. The best-performing model was a skip-gram Word2vec model with a 300 embedding dimensions and a window size of 15. Adjusting window size had the most significant effect on accuracy.

1 Introduction

The growing global knowledge economy is influencing the types of jobs that are available and desirable. Labor economics have always been important, but recent rapid changes to the global economy - and the associated uncertainty - has made employment and other economic issues top of mind for many citizens and the politicians that govern them.

The issue of re-skilling is particularly salient. Employment was one of the major themes of the 2016 U.S. presidential election. People all over the world seem to be interested in the same questions: What skills do I need in order to make a living? Which ones are the most impactful for my career--especially given what I already know?

One important step in answering those questions is to have an empirical understanding of the canonical set of skills that in-demand jobs require. Additionally, it is important to understand which skills are similar to one another. Armed with those insights, a variety of tools could be created to help workers re-skill. A language model is perfectly positioned to address these tasks. This project is an attempt to do so.

2 Background

This work leans heavily on Mikolov's canonical paper on Word2vec (2013). Mikolov et al. proposed that instead of treating words as atomic units and learning which words have the highest probabilities in a new setting, it is beneficial to use vectors that capture semantic and syntactic meaning. Neural networks trained using vector embeddings "significantly outperform N-gram models" (Mikolov, 2013).

A variety of model designs can use word2vec vector representations of words: probabilistic feedforward neural net language models, recurrent neural net language models, continuous bag-of-words models, and continuous skip-gram models, to name a few. Mikolov et al. assessed the quality of their results by examining algebraic relationships between sets of words, achieving 58.9% accuracy on the word pair task.

However, Mikolov's assessment technique was created for use with models trained on *general* text corpora. The algebraic relationships that the evaluation assesses compare entities like country capitals and currencies. As a result, the assessment technique cannot be used for models trained on more specific corpora that do not include the terms in the assessment.

Other papers have built on the work done by Mikolov. Skill2vec (Le, 2017) represents one attempt at pulling skills out of documents using Word2vec techniques. Le et al. created a dictionary using data from LinkedIn, then trained a Word2vec model over a corpus of job descriptions. The authors assessed their model using recruiters' domain knowledge. The recruiters determined that for any given skill, 76% of the top ten model outputs were valid and relevant skills.

This project intends to extend the work done by Le et al. The work in Skill2vec is interesting, but it could be improved in a few ways. First of all, Skill2vec's assessment criteria is too subjective. Additionally, their work could be improved by using additional sources and types of data.

3 Methods

This paper builds on work by Mikolov and Le et al. While this paper's primary task is similar to that of Le et al., it differs in a few important ways.

3.1 Data Gathering

The training data for this project comes from two sources: a collection of more than 106,000 job descriptions scraped from Indeed.com, and all of Stackoverflow's users' posts with scores greater than 25. For the sake of limiting the scope of the project, the job descriptions were restricted to the following keywords:

Accounting	Human Resources	Sales Engineer
Analytics	Legal Operations	Social Media Marketing
App Developer	Logistics	Software Architect
Business Intelligence	Marketing	Software Consultant
Client services	Mobile Engineer	Software Developer
Compliance Operations	People Operations	Software Engineer
Customer success	Product Designer	Software Manager
Data Analyst	Product Manager	Software Test Engineer
Data Engineer	Professional Services	Strategy
Data Scientist	Program Manager	Supply Chain Management
Database Administrator	Project Manager	Technical Program Manager
DevOps	QA Engineer	Technical Sales
Financial Analysis	Quality Assurance Engineer	UX Designer
Frontend Engineer	Release Engineer	UX Engineer
Hardware Operations	Risk	UX Researcher

3.2 Data Preprocessing

After gathering the data, I processed it to prepare it for ingestion into the model. First, I conducted some basic preprocessing: segmentation into sentences, removing punctuation and whitespace, and lemmatization. I used spaCy's lemmatization library for lemmatization.

Second, I created a list of skill bigrams and trigrams: terms that I wanted represented within the model as single entities. I then found all occurrences of those terms in the corpus, and replaced whitespace separating words with underscores. This process turned tokens like `machine learning` into underscored terms like `machine_learning`. It is worth noting that my initial approach involved using phrase modeling to automatically detect bigrams and trigrams. I abandoned this approach after realizing that phrase modeling was concatenating distinct skills that happened to be colocated in the data, such as `java_python`.

It is worth noting that my preprocessing steps differ from those of Mikolov. First of all, he did not use lemmatization, whereas I did. Because I am using my language model for a very specific purpose -- extracting semantic meaning from a certain class of words -- I want to reduce the number of permutations that exist in the corpus that represent the same entity. Lemmatization is a clean way to achieve my goal; it removes inflectional endings such as *-ing*, and it converts all plurals to singular form. Lemmatizing does have a downside, in the sense that certain skill words are improperly lemmatized ("AWS" becomes "aw", because the lemmatizer thinks that it's a plural object). Secondly, Mikolov did not process bigrams, whereas I did. Certain skill terms only have their semantic meaning when processed as bigrams; "machine" and "learning" have much different meanings separately than when combined into "machine_learning." I needed to preserve as much semantic meaning as possible in those terms, which is why I manually preserved them in the corpus using underscores.

My process also differs from that of Le et al, who generated their list of skills from LinkedIn. This paper takes a different approach; I extract the skills from the data itself. The primary method I used to create a list of hard skills was to cluster the embeddings from the baseline model, using a k-means algorithm with k of 100. I was then able to find "skill clusters", from which I created a list of hard skills.

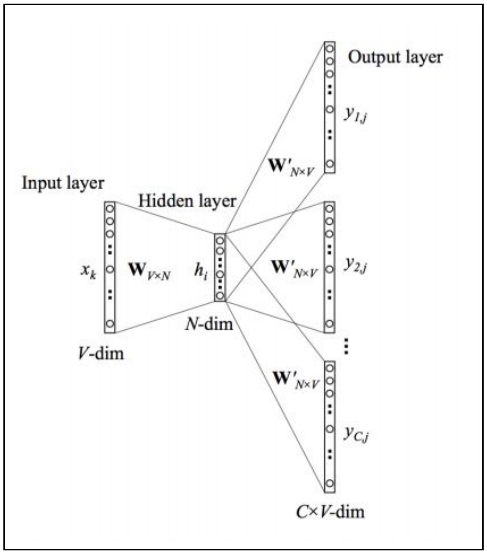
3.3 Model Specification

After preprocessing, the model ingested data and learned word vectors.

I used Gensim's implementation of Word2vec for modeling (Rehurek, 2010). The Gensim implementation was created based on Mikolov's work; the majority of the code was ported from Google's implementation, written in the C programming language for performance. I use Word2vec instead of similar algorithms like GloVe for a few reasons. First, Gensim's implementation of Word2vec is easy to parallelize, so I was able to train models that took more than a week of CPU time in a few hours. Second, Word2vec is designed to allow the model to understand context for each word, so that similar words have similar representations in embedding space.

The model uses the skip-gram architecture, as proposed by Mikolov:

Skip-Gram Architecture from Mikolov, 2013



$$p(w_{c,j} = w_{O,c} | w_I) = y_{c,j} = \frac{exp(u_{c,j})}{\sum_{j'=1}^{V} exp(u_{j'})}$$

The model generates vectors for each token. Each vector consists of a token followed by the x-dimensional embeddings that characterize the that token:

	0	1	2
rubygem	-0.007805	-0.056915	-0.025744
charlotte	0.027095	-0.042693	0.014534
ct	-0.043666	0.029391	-0.036102
governmental	-0.027602	0.003340	0.066653
magento	0.016457	0.027155	-0.048232

The specification of the baseline model is below:

Training Corpus	Job descriptions + Stackoverflow
Embedding dimensions	100
Training Algorithm	Skip-gram Hierarchical softmax
Word Window Size	5
Epochs over corpus	15
Minimum required corpus word frequency	20

3.4 Model Evaluation

The word algebra technique for model evaluation employed by Mikolov is reasonable for general corpora, such as news articles or government proceedings. However, due to the specific nature of this project's data, I needed to generate a different method of evaluating the model.

My evaluation technique capitalizes on the predictive nature of the Word2vec algorithm. If the model truly understands skills, then it should assign a relatively high probably to a sentence that includes a skill within its correct context. Correspondingly, it should assign a relatively low probability to a sentence that misuses a skill word. This logic is core to the evaluation scheme.

Accordingly, the first step in creating the evaluation scheme was to identify sentences that correctly use skills. To generate those sentences, I first constructed a list of all of the relevant skills I wanted to search; there were 915 in total. Next, I used the `wikipedia` package on PyPI to search for each skill. I manually adjusted search terms that did not yield Wikipedia results. For example, I needed to change the term "`chef`" to "`chef (software)`" to disambiguate the software package from other senses of the word. I then captured the summary associated with each of the 915 skills, split the summary into sentences, and removed any sentences that did not contain any of my 915 skills.

The resulting set included 2,114 unique sentences, each containing at least one of the 915 skills. I ran each sentence through the same lemmatizer that I used for the training corpus. Then, I created a copy of the sentences, and I replaced each occurence of a skill with

a randomly selected *different* skill, thus creating a "corrupted" set of sentences. Here is an example of a sentence before and after the corruption process:

Clean:

```
apache groovy be an object_orient programming_language for the java platform
```

Corrupted:

```
akka standard_operating_procedure be an zoho spreadsheet for the aurelia platform
```

After constructing the two sets of sentences, they could be used to evaluate models. The specific metric I used was the average negative log likelihood difference (ANLLD) between the clean sentence set and the corrupted sentence set. Good models should have a higher ANLLD, because they should consistently generate high negative log likelihoods for the corrupted sentences and low negative log likelihoods for clean sentences. Therefore, a high *difference* between the two sets is the objective.

4 Results and Discussion

After training the word embeddings, I conducted exploratory data analysis (EDA) on the results.

The first part of the EDA was relatively straightforward, and consisted of checking that the model's embeddings made sense. I used a few techniques for this analysis.

4.1 Qualitative Evaluation

Finding Related Terms. I found the closest related terms, measured using cosine similarity, for a variety of skill words. Here is an example:

```
get_related_terms(u'neural_network')
```

neural_networks	0.86
decision_tree	0.842
support_vector	0.828
neural_net	0.827
svm	0.825
convolutional	0.805
bayesian	0.805
svms	0.802
rnn	0.788
reinforcement_learning	0.772
baye	0.772
markov	0.765
deep_learning	0.748
xgboost	0.748
lstm	0.745

The terms that the model regards as the most similar to "neural network" make sense; it surfaces terms related to support vector machines, along with key terms related to Bayesian inference.

Skill Analogies. I decided against using skill analogies for my formal evaluation technique, because it is too difficult in most cases to establish an objectively "correct" set of analogies. However, using analogies is helpful in determining the relationships between embeddings. Here are three examples:

```
word_algebra(add=[u'nosql', u'rdbms'],
             subtract=[u'mongodb'])
```

```
relational_databas
```

```
word_algebra(add=[u'python3', u'ruby'],
             subtract=[u'pip3'])
```

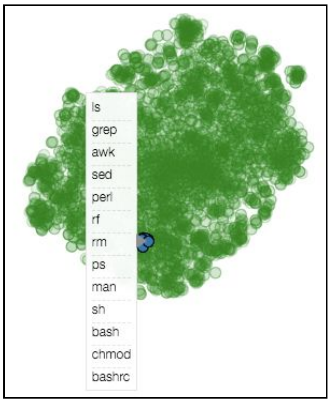
```
gem
```

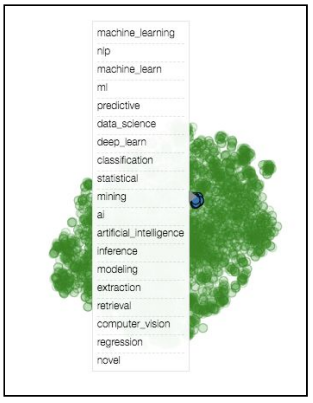
```
word_algebra(add=[u'python', u'javascript'],
             subtract=[u'pip'])
```

```
js
```

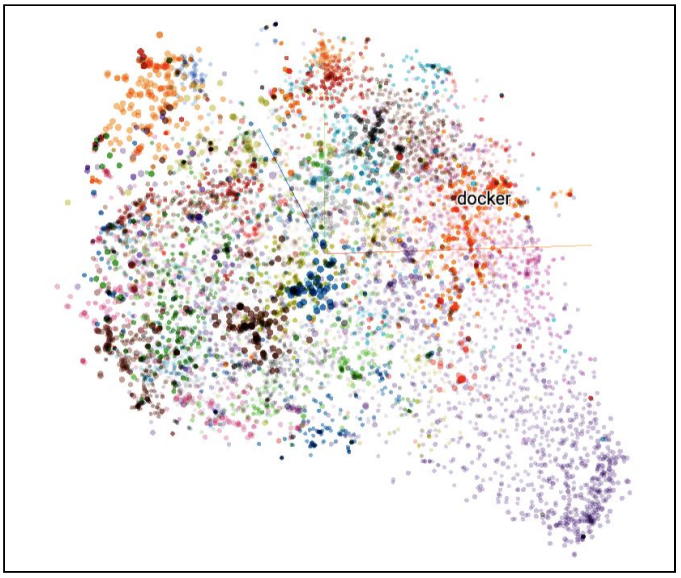
These analogies operate by finding distances between points, so saying "*a* is to *b* as *c* is to *d*" is equivalent to "*d* = *a* + *b* - *c*". Above, the first analogy is plausible, and the second is about as accurate as I could reasonably expect. But the third misses the mark; I would have expected *npm* as the result, which is Javascript's package manager. This type of mistake is characteristic of the model's performance on most "skill analogy" tasks.

Visualization. Visualizing the word embeddings is instructive. If the models correct understand skills, then skill terms with semantically similar meanings should be close together. For example, a gaggle of Unix commands are more or less on top of each other, and the model correctly lumps machine learning terminology together:

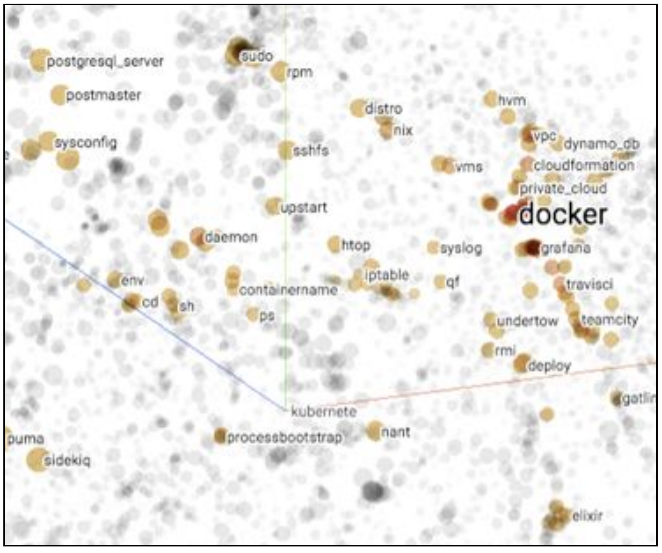




I also clustered the skills using k-means. I was able to pipe the cluster labels into Google's Tensorboard product to visualize the embeddings in three dimensions, colored by cluster:



Isolating a particular term helps gauge the accuracy of the embeddings:



In this case, the term "Docker" (referring to the containerization technology) is closely related to a set of terms that is, in fact, semantically related to the technology in question: other DevOps tools. They include virtualization services such as VMS (or "virtual memory system"), cloud analytics products like Grafana, databases, and Linux process terms like "sudo" and "daemon".

4.2 Error Analysis

The most intuitive way to conduct error analysis would be to look at the most common types of errors, hypothesize why the model was incorrect, and tune the model in a way that might reduce the error. Accordingly, I constructed a counter that kept track of the number of times each unique error occurred. In this context, an "error" is defined as the model preferring a corrupted sentence over the correct one; the counter kept track of the skills in each set, constituting a sort of confusion matrix. My hypothesis was that some sort of useful pattern would emerge regarding the skills that the model consistently fumbled. Here is an example of the output, sorted by descending frequency:

```
(['blockchain'], ['ffmpeg'])    1
(['openal'], ['prism']) 1
(['openvpn'], ['hpc'])  1
(['jscript'], ['scp'])  1
(['semaphore'], ['oscommerce']) 1
(['umbraco'], ['wordpress'])    1
(['openwrt'], ['saucelab']) 1
(['bpo'], ['selinux'])  1
(['ocr', 'artificial_intelligence'],
   ['stack_overflow', 'lua'])  1
(['ingre'], ['nhibernate']) 1
(['silverlight'], ['macro'])    1
```

As the example indicates, this method was fruitless, because even for the baseline model specification, *every single error was unique.*

Given that the confusion matrix approach of examining confused pairs of skills was unhelpful, a logic next step was examining the most often confused *singular* skills. That analysis did not yield much of a pattern; the most commonly referenced skills are the ones that were also the most confused. Here are the top confused skills for the baseline model:

Skill	Confused Frequency
c	16
programming_language	13
sql	11
stress_load	9
gnu	9
linux	8
java	7
silverlight	7
cobol	7
operating_system	7

Given that the skills contained in each error were unique, my next step was to examine the sentences for which the model made the most egregious errors. The machinery for this analysis was already in place, as I calculated sentence-level probabilities as a part of the evaluation scheme.

Here is the baseline model's most egregious error, in which the model preferred the corrupted sentence by 230 log likelihood "points." It is worth examining this error, as it is representative of the others:

Error 0 Negative Log Likelihood Difference: -230.66
Clean:

```
kubernete commonly stylize as k8s be an open source system for automate deployment scaling and management of containerized application that be originally design by google and now maintain by the cloud native computing foundation
```

Corrupted:

```
dyndns commonly stylize as k8s be an open source system for automate deployment scaling and management of ramdajs application that be originally design by google and now maintain by the cloud native computing foundation
```

Kubernetes is a containerization service; DynDNS is a method of updating a name server in the Domain Name System, and RamdaJS is a functional programming library for JavaScript. I hypothesize that this error was primarily caused by the fact that Kubernetes and DynDNS both pertain to DevOps/infrastructure workflows, which makes the sentence more plausible than if DynDNS had been replaced with a totally irrelevant term.

Testing that hypothesis is straightforward: I switched "dyndns" out for a totally unrelated skill word, as determined by distance in three-dimensional TSNE-reduced space via the Tensorboard. Replacing "dyndns" with "neo4j" (the Python graphing library) nearly halved the error, cutting it to -121. The terms "numpy", "photoshop", "powerpoint," and other skills unrelated to "dyndns" had the same effect: they drastically lowered the error. Meanwhile, terms semantically similar to Kubernetes (or, the lemmatized version, "kubernete"), worsened the error or kept it roughly consistent. Replacing "dyndns" with "docker", another DevOps tool, resulted in an error of -245. Swapping in the term "google_compute_engine" increased the error to -329.

This pattern continued through other examples; for the sake of brevity, I will avoid listing them here.

It seems as though the model is making errors where the corruption process swaps in terms that are similar to the correct term, or at least plausible given the rest of the sentence's context. In other words, the model has learned enough about the language to know that certain contexts are better than others, but for similar terms, it has a hard time deciding which is the most accurate.

One seemingly logical solution could be to adjust the evaluation process to ensure that similar terms are not used in the corruption process, but changing the evaluation metric midway through an experiment is dishonest.

Because the problem seems to be that the model just needs to learn more about each term, the best solution is to adjust the hyperparameters to brute-force coerce the model to increase the amount of semantic meaning that the it can extract from the data. In concrete terms, this means increasing the window size and increasing the number of embedding dimensions.

4.3 Results After Hyperparameter Tuning

The results are below. They are slightly easier to interpret in the visualization enclosed in the Supplemenary Materials section.

Embedding Dimensions	Window Size	Algorithm	Score
300	**15**	**Skip-Gram**	**398**
600	12	Skip-Gram	340
300	10	Skip-Gram	286
200	8	Skip-Gram	234
600	5	Skip-Gram	162
400	5	Skip-Gram	160
200	5	Skip-Gram	155
100	5	Skip-Gram	149
100	5	CBOW	17
50	5	CBOW	15

Increasing window size. The baseline model has a window size of 5. I trained models with window sizes of 5, 8, 10, 12, and 15. Increasing window size seemed to have the largest effect on results.

Increasing embedding dimensions. Mikolov used embedding dimensions of 100, 200, 300, and 600. Those are reasonable dimensions, so I used the same ones, plus one 400-dimension model and a 50-dimension model.

Different algorithms. I abandoned CBOW early on, as its results were consistently lower than those of the skip-gram architecture. However, I did include the results from two CBOW models above. Given the types of errors that I have encountered, it makes sense that a continuous bag-of-words would perform worse. In the context of skills-based text, it is easier to predict context from terms (skip-gram's approach) than it is to predict a term from its context (CBOW's approach).

The best-performing model had 300 embedding dimensions and a window size of 15. Increasing window size had the largest effect on accuracy. This phenomenon makes intuitive sense; models with larger window sizes are able to retain more information about a skill's context, which causes the model to prefer sentences that put skills in their proper contexts.

5 Conclusion

This project created a model to understand skills within their contexts. While it could be useful in its current form, future work could advance this research in a few ways:

Increase the amount of training data. I used 106,000 jobs and all of Stackoverflow's high-quality posts. Increasing the number of job descriptions by a factor of 10 or more would likely improve the model's understanding of the terms.

Use a deeper neural network architecture. The results from different model specifications indicate that the more detail that the model was able to learn about the text, the better it performed on the evaluation task.

Use additional computational power. For budgetary purposes, I limited the most computational intensive training to eight days of vCPU training time. A model with a larger window would likely perform better, but it would be more expensive to train.

References

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Word2vec. 2013. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*. https://arxiv.org/abs/1301.3781.

Van-Duyet Le, Vo Minh Quan, and Dang Quang An. 2017. Skill2Vec: Machine Learning Approaches for Determining the Relevant Skill from Job Description. *ArXive e-prints*. https://arxiv.org/abs/1707.09751v1.

Radim Rehurek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. *LREC*. https://radimrehurek.com/gensim/lrec2010_final.pdf.

A Supplementary Materials