Report on Contextual Features for Metaphor Detection on Running Text

Edwin Zhou
edwinmzhou@gmail.com

Abstract

Metaphor is a fundamental tool in communication, with uses ranging from invoking emotion to conveying new ideas. Metaphor detection is thus crucial to many applications in natural language processing. While it used to be that metaphor detection methods focused on immediate linguistic features, we now understand that contextual information is crucial for detecting metaphors, especially on running text. However, there has not been enough work on how best and by what scope to integrate contextual information into systems. This project explores the effectiveness of both global and local contextual features as well as which features to properly represent them. I build upon a baseline one-hot encoding unigram context with features discussed in Jang 2015 and train on a simple logistic regression model. Experiments on the VUAMC Corpus show a noticeable improvement over the baseline on some feature combinations, confirming the validity of those features for incorporating context. Some features do not perform so well, and invites further investigation.

1 Introduction

Metaphor is a fundamental component of human language which is used frequently in our everyday lives. As a form of figurative language, metaphors can be used to evoke emotion, simplify complex ideas, bring in poetic or humorous elements (Glucksberg, 2001), and connect different concepts (Strang et al., 1982), thus allowing us to express ourselves more effectively. However, although humans can effortlessly interpret metaphors using a number of subconscious linguistic devices and background knowledge,how machines might have trouble with them, and there are many challenges facing the problem of metaphor processing (Tsvetkov et al., 2014).

The ability to process metaphors will open many doors in the field of NLP. For example, it can help machines correctly translate sentences containing metaphors from one language to another. Consequently, metaphor research has gained much traction and interest in the recent decade, but there still exists many areas of improvement.

1.1 Metaphor Detection

The first step of any metaphor processing system is metaphor detection (Mu et al., 2019). My directed studies project will focus on applying existing metaphor detection methods on the VU Amsterdam Metaphor corpus (Steen, 2010). The VU Amsterdam Metaphor corpus is a publicly available metaphor corpus which was used for a shared task on detecting metaphors in a 2018 workshop on figurative language processing (Leong et al., 2018). This corpus was manually annotated based on the MIPVU metaphor annotation scheme. In addition, this corpus has annotations for different metaphor types. Since there is no clear consensus even among researchers regarding how we differentiate a metaphor from a non-metaphor, there exists many different types of metaphor annotations.

Although my proposal implied the exploration of a variety of systems on the VU Amsterdam Metaphor Corpus, after reading a number of papers, I found that the corpus has been fairly well-explored. Instead, I choose to focus on contextual the features described in (Jang et al., 2015), as well as a few baseline models for comparison.

1.2 Contextual Information for Metaphor Detection

The following two examples illustrate how sometimes context is necessary to detect metaphoricity:

> *As the deadline to the directed studies project approached, he felt an over-*

whelming anxiety. ***It began to eat away at him from the inside.***

After a few months, the alien piranha hatched in his stomach. ***It began to eat away at him from the inside.***

In the first example, "it ate him up inside" is used metaphoricially, while in the second, it is used literally. In these examples, it is difficult to identify (Martin, 1996; Shutova, 2010; Shutova et al., 2013; Shutova and Teufel, 2010; Huang, 2013) or contrast in lexical concreteness and abstractness (Turney et al., 2011; Tsvetkov et al., 2013), as the sentence makes sense when interpreted literally. It is obvious that global context beyond a single sentence is needed to process and detect metaphorical expressions like these which have external contextual information.

Intuitively, contextual information is most useful when combined to selectional restriction violations, specifically in scenarios where verb arguments are unclear without context, for example when pronouns are used. This is an area for further exploration.

While we now understand that contextual information is crucial for detecting metaphors, especially in running text (Mu et al., 2019), there remains work to be done on how to optimally incorporate contextual information into metaphor detection systems.

While Jang's global and local contextual features have been previously implemented on a dataset built from an online breast cancer support forum, there is a significant difference compared to the VU Amsterdam Metaphor Corpus, which is comprised of four different genres of running text. Also, while every word in the VU Amsterdam dataset is labeled as a metaphor or a literal word, for Jang et al.'s dataset, the words that are inspected and classified are restricted to a set of seven candidate words. While this constrained dataset is good for inspecting the features defined in Jang et al., (2015), it does not showcase overall performance of the use of contextual features in a metaphor detection system on unrestricted text, which is what I explore in my project.

I also found that discourse-level features have been very recently investigated by Mu et al. (2019), and determined by qualitative analysis to be often necessary for effective metaphor processing. However, there are three main differences between their paper to this project. Firstly, of the two common types of models for detecting metaphors in running text (Gao et al., 2018): one which determines the metaphoricity of each word (sequence labeling), and one which only classifies verbs, Mu et al. chose to focus on the latter, while I attempt to label all words in running text as literal or metaphorical. Overall, metaphor detection appears to be easier for verbs than other parts-of-speech (2018). Also, Mu et al. only uses paragraphs for its broader context feature, whereas I employ both global and sentence-level local features. Lastly, Mu et al. represents features using word embeddings, whereas I employ simpler one-hot encodings and numerical features.

There is a knowledge gap in what optimal features for representing global and textual features on running text should be. With the VU Amsterdam Metaphor Corpus, which contains a wide range of running texts from newspaper articles to conversations, I fully explore the effectiveness of global and local contextual features proposed by Jang et al. by applying them to sentence, paragraph, and full text-level contexts.

The rest of the project review will be organized as follows. Section 2 discusses the VU Amsterdam Metaphor Corpus used for my experiments. Section 3 gives an overview of the method inspected, and walks through the process of extracting relevant data from the corpus XML file, the divison of datasets according to Klebanov (2014), as well as the engineering of global and local contextual features following Jang et al. (2015). In Section 4, I present my results over the baseline models, and discuss the effectiveness of my contextual features on running text for metaphor detection, and in Section 7, I give my concluding remarks on the project and reflect on potential future work.

2 Data

We use the VU Amsterdam Corpus (Steen, 2010) for our investigations. The corpus consists of 117 running text fragments sampled from the British National Corpus (BNC)'s Baby Corpus across four genres: academic writing, newspaper articles, fiction, and conversations. The entire dataset has a total of approximately 200,000 words, and each genre is represented by approximately the same

number of words, although the number of texts differs greatly between the genres, with the newspaper genre containing the largest number of texts, 63 in total.

The corpus was manually annotated for comprehensive use of metaphorical language by five analysts using the MIPVU procedure, which is a variation on the MIP procedure with a few adjustments and extensions such as being able to handle metaphoricity through reference and allowing for explicit coding of difficult cases where a group of annotators cannot arrive at a consensus (Steen, 2007). The tagset is organized hierarchically and includes several subtypes of metaphors, as well as other types of annotations such metaphor signals such as like, as, etc.

I follow the guidelines for producing datasets set by previous work. Rather than consider the corpus as a whole, we divide it into distinct genres and address them separately. The intuition is that each genre may use different types of metaphors, and we would like to prevent this information from polluting other genres.

Similarly, I consider all content words with the tag function=mrw (metaphor-related word) as metaphors, and do not experiment based on the finer-leveled seg types.

I use the same training and testing partitions as Klebanov et al. (2014), by randomly sampling approximately 23% of the texts from each genre and setting them aside for testing, while retaining the rest for training and cross-validation. Following Klebanov et al., cross-validation is partitioned differently for each genre based on text fragments, performing 9-fold on Conversation, 10-fold on Newspapers, 11-fold on Fiction, and 12-fold on Academic. This way, all instances from the same text were always placed in the same fold and we retain folds of similar sizes. An interesting disagreement and perhaps error I discovered is that Klebanov et al. (2014) and Klebanov et al. (2016) both state that the total number of texts for the Academic and Conversation genres are 18 and 22 respectively. On the other hand, I found, by scraping the British National Corpus User (BNC) Reference Guide (http://www.natcorp.ox.ac.uk/corpus/baby/thebib.html), that Academic in fact has 16 texts while Conversation has 24. This discrepancy also means that the size of my train/test splits are slightly different to Klebanovs.

Also, unlike Klebanov et al., (2014), I also look at paragraph-level and text-level features, beyond just the sentence-level features, so the information I needed to extract from the provided VUAMC.xml corpus file is slightly different than that of previous works. Although a script for the parsing of the original XML file was provided for the shared task 2018 VUA Metaphor Detection Shared Task in the NAACL 2018 Workshop on Figurative Language Processing (Leong et al., 2018), to facilitate the use of datasets and evaluation, due to the different requirements of my features, I could not use it for parsing. Therefore, I wrote my own scripts to parse the XML file, extract useful tags, partition into reasonable datasets, filter for content words, setup baseline model, evaluate trainings, and so on.

Data	Training		Training	
	#T	#W	#T	#W
Acad.	10	48964	6	17223
Conv.	20	41595	4	6983
Fict.	11	38004	3	6914
News.	49	35740	14	9411

Table 1: Summary of the data. #T = # of texts; #W = # of words.

All the code can be found on GitHub at https://github.com/Sunflower/CPSC-488A, and the process will be elaborated on in the next section.

3 Experimental Setup

3.1 Datasets

Before creating the features, I have to first generate the datasets from the VUAMC XML file and partition them for training and testing.

Dataset Construction: To begin, I download the XML file from http://ota.ahds.ac.uk/headers/2541.xml. Having never encountered the corpus before, I take time to read and understand the structure of VUAMC.xml, looking online for references to the tagset and its attributes.

I decide that my datasets will be in 3 parts, words, sentences, and full texts. I write a Python script for performing these tasks, and the 3 files are saved in CSV format for easy processing with NLP tools in Python.

The *words.csv* file is used primarily for feature creation, and includes the columns word_id, word, lemma, word_type, function, seg_type, sentence_id, and text_id. The lemma is the lemmatized form of the word, provided by the corpus,

but not something that seems to be utilized by the script provided by the Shared Task on Metaphor Detection, as they suggest using auxiliary tools for lemma extraction. The word type indicates the part-of-speech of the word. The function tag indicates whether or not a word is used metaphorically or signals the use of a metaphor, i.e. a metaphor flag, while the seg type, marked by $< seg >$ in the corpus, indicates what type of metaphor or metaphor flag it is.

Table 2 gives an example of a few rows from *words.csv*.

Both *sentences.csv* and *texts.csv* contain the full sentences and texts respectively constructed while parsing VUAMC.xml. They are mostly so I can interpret experimentation efforts more easily, by having fully-constructed sentences and texts I can read, rather than rows of word features. I also experiment with extracting a file of paragraphs, annotated with $< p >$ tags in the corpus, but problems with index matching made me abandon it for it to be constructed later from the extracted dataset, and make do by including paragraph IDs in the sentences file. Texts in the Conversation genre do not have paragraph IDs.

Dataset Partitioning: Working with Pythons Pandas library, I dive into partitioning the *words.csv* dataset into training and testing sets, following the train/test split scheme taken by prior works (Klebanov et al., 2014), As the raw corpus does not contain information regarding the genres of the texts, I first scrape the BNC Reference Guide web page to get the different text tags for each of the four genres, before assigning them to each word row in my Pandas DataFrame of *word.csv*.

Then, using approximately the ratio given by Klebanov and other papers, I randomly sample text tags. Based on the words assigned text tag, I am thus able to partition and form the train and test sets from the corpus.

3.2 Unigram Contextual Features

Like Klebanov et al., (2014), this project also bases its features on a unigram context model, where each word in a target word is considered as part of the feature. To implement this effective baseline, I first filter the datasets based on content words and stop words. The next step is creating a vocabulary using the content words. The purpose is to use the vocabulary as the unigram feature columns with a one-hot encoding representation. For each word, I use every other word in the sentence it belongs to as the unigram context encoding feature value.

Sparsification is undertaken before feeding the output CSR matrix to the model in order to reduce the huge size of the unigram feature, which would be the number of sentences $\times$ the size of the vocabulary.

After inspection, it is verified that the sparsified feature vectors are valid, and thus they are fed into a logistic regression and evaluated, the process of which is elaborated on in the next section.

3.3 Global Contextual Features

With the underlying unigram model ready, I turn to building global features. Semantic category

Semantic Category: I use as a feature the relative proportion of the target words category with regards to all categories appearing in the global context, which in my implementation is the paragraph. The intuition is that if a target word is used literally, there should be more words in the document that have the same semantic category, and vice versa (Jang et al., 2015).

While Jang uses the SEMAFOR (Chen et al., 2010) tool for parsing FrameNet frames (Baker et al., 1998), their GitHub page states that the tool is outdated and no longer being maintained, and I am pointed to Open-SESAME (Swayamdipta et al., 2017). So, I use this similar, updated system instead.

As this frame-matching framework requires each sentence to be on a separate line, I process my words.csv dataset and save each sentence on a new line, to be used for processing. These sentences are numbered so that they can be matched to the words they contain after Open-SESAME is done with it.

Open-SESAME provides pre-trained models for prediction on unannotated data, however, there is a bug which causes prediction to fail. Therefore, I retrain the models first, before predicting on the sentences.

Using the offset on each sentence given by Open-SESAME in its output, I match the results of the prediction back to the words DataFrame. Open-SESAME was only able to assign frames to 0.22 of all words. Using the formula given for global word category feature in Jang et al. (2015)

$$\frac{\sum_{w \in d} \mathbb{1}(c_w = c_{tw})}{N_d},$$

word_id	word	lemma	word_type	function	seg_type	sentence_id	text_id
147110	are	be	VBB			7903	fet-fragment01
3	reveals	reveal	VVZ	mrw	met	0	a1e-fragment01
1340	like	like	PRP	mFlag	lex	66	a1f-fragment10

Table 2: Example of a few rows in *words.csv*. word_id: ID of the word, lemma: lemmatized form of the word, word_type: part-of-speech of the word, function: indicates whether or not a word is used metaphorically or signals the use of a metaphor, seg_type: subtype of metaphor or metaphor flag, sentence_id: references sentence in *sentences.csv*, text_id: references text in *texts.csv*

I assign a numerical feature value to each word. where c_W is the category of word w, c_{tw} is the category of the target word, and N_d is the number of words in document d. $\mathbb{1}(.)$ is an indicator function that equals 1 when the expression inside is true and 0 otherwise.

I define a document as the paragraph, and for the denominator, only count words that have been assigned a frame in the total number. Due to the scarcity of frames, I have also tried using the full text as document, but the performances were slightly lower. The words that were not matched with frames are assigned -1.

Topic distribution: I use discrepancy between a words topic distribution and the surrounding documents (which in my implementation is the paragraph) topic distribution as another feature. The intuition is that non-literal words tend to have considerably distributions from the global context (Jang et al., 2015).

We start by filtering all non-content words, the intuition being that by only looking at lemmatized content words, it would be easier to generate topics.

I use scikit-learns LatentDirichletAllocation and first fit Latent Dirichlet Allocation (LDA) (Blei et al., 2003) models on each genres training sets with 20 topics for each genre, saving them so they can be used later to transform the test data. Before feeding into the model, I convert the data into a CSR (Compressed Sparse Row) document-word matrix.

As LDA originally computes $P(word|topic)$ rather than $P(topic|word)$, I do additional processing on top, and once $P(topic|word)$ is computed, I find the cosine similarity between $P(topic|word)$ and $P(topic|document)$ and match it to each word in the dataset to use as features that represent the global alignment of topics between the target word and the document.

Lexical chain: I follow the intuition that metaphorical words would not belong to dominant lexical chains (Morris and Hirst, 1991) of the global context. To implement this, I use ELKB toolkit (Jarmasz and Szpakowicz, 2003), which is built on Rogets thesaurus (Davidson, 1982) to find lexical chains for each paragraph.

I look at only the longest chains, and if multiple chains contain an equal number of words, I combine both chains words for use.

The feature value for a word is a boolean of whether or not a word is in the longest lexical chain of its paragraph. It should be noted that some paragraphs can have no lexical chains, in which case all words within have the feature set to False.

3.4 Local Contextual Features

On top of global contextual features, I experiment with local contextual features to explore feature-to-feature interaction, and attempt to enhance system performance with features at a lower context. I use lexical abstractness, which is a good representation of local metaphoricity information (Köper and im Walde, 2017), to aid global features to identify metaphors. Then, I use bring in grammatical dependencies to combine with lexical abstractness, which further induces relational connections between a target word and its contextual information.

Lexical Abstractness/Concreteness: To obtain concreteness for the words in my dataset, I parse Brysbaert's database of concreteness ratings for about 40,000 English words (Brysbaert et al., 2014), combining the average concreteness rating with my dataset.

Lexical AC with Grammatical Dependencies: Grammatical dependencies is used to further enforce the contextual relation between a target word and the surrounding sentence. To implement it requires several steps.

First, I use the Python API for the stanford-corenlp toolkit (Manning et al., 2014) to parse dependency relations on my *words.csv* dataset,

grouped by sentence ID. It is possible to parse the entire dataset at once, however, I ran into many issues with the API, for example, single "words" containing whitespaces may be wrongly delimited as separate words, or perhaps the sentence delimiter may be present in the dataset, causing premature tokenizing of a sentence. To avoid issues like these, I call the API on each sentence individually to guarantee robustness. As this parsing takes a while to run, I save the dependencies when it is done to avoid complications in case of situations like Jupyter Notebook kernel issues.

Afterwards, I mangle the output dependencies using Pandas until it is of the form such that each word has a 43-dimensional feature vector, where each value corresponds to the IDs of words that are in a dependency relation with it, 43 being the number of different types of grammatical dependencies. Due to the stanford-corenlp toolkit only returning sentence IDs and word offsets in their sentences, word ID has to be calculated separately so that the target may be matched to other words in the same sentence which share a grammatical dependency with it.

Finally, the aforementioned word-dependencies DataFrame is combined with the lexical concreteness feature, in a way such that each word also takes the concreteness rating of each word in grammatical relations to it as a feature. If there are multiple words in the same sentence with the same grammatical relation to the target word, the mean is taken.

4 Evaluation

4.1 Evaluation Metrics

I report three evaluation metrics: precision, recall, and F-score.

Precision: Precision is the weighted percentage of correctly classified instances among instances assigned to a particular class (metaphor or literal) by the model.

Recall: Recall is the weighted percentage of correctly classified instances among all non-literal or literal instances. Precision and recall are recorded for both metaphorical and literal labels.

F-score: F-score is the weighted harmonic mean of precision and recall.

4.2 Training and Classification

For my classifier, I use scikit-learns Logistic Regression with L1 regularization, the liblinear solver, and an inverse regularization strength of 10. To perform cross-validation training of hyperparameters, I use scikit-learns GridSearchCV function, maximizing f1, with specific folds for each genre as defined in the Data section which follows prior work (Klebanov et al., 2014).

In both the training and testing stages, before using the data, I first preprocess it, filtering for content words and shuffling the data. In the training stage, I also set up cross-validation folds, build the validation labels. Finally, features are created for each experiment, the model is trained by GridSearchCV, and results are evaluated on a test set.

The results are presented in Table 3.

4.3 Baseline

Unigram Model: A logistic regression classifier trained only on one-hot encoding of word lemmas, which is elaborated on in the previous section.

4.4 Discussion

As can be seen in Table 3, both global and local contextual features generally manage to improve on top of the baseline context unigram model when combined, although some local features individually do not give an edge over the unigram model.

Global word category seems to excel, on the other hand, global topic distribution and lexical chain do not seem to perform well on any genres except for Fiction. Further investigation is needed to see why this is the case.

Local contextual features tend to outperform global contextual features individually in all genres except for Conversation, however, when the global contextual features are combined, as in the case of **U+GWC+GT+LC**, it is stronger than lexical abstractness. While using all the features does not guarantee the best performance, we can clearly see that features which implement contextual information are able to contribute to metaphor detection.

The Conversation genre's numbers may look the same, and it is because the texts in that genre are not contained in paragraphs, which is the global context I am work with for the other genres. Further exploration can be done by considering other "global" contextual scopes for Conversation.

		U	U+GWC	U+GT	U+LC	U+GT+LC	U+GWC+GT+LC	U+AC	U+ACDep	All
Academic	P	0.651	0.693	0.651	0.651	0.651	0.692	0.749	0.734	0.736
	R	0.454	0.603	0.454	0.454	0.454	0.589	0.515	0.591	0.593
	F1	0.497	0.634	0.497	0.497	0.497	0.622	0.549	0.626	0.628
Conversation	P	0.823	0.823	0.823	0.823	0.823	0.823	0.851	0.855	0.855
	R	0.695	0.695	0.695	0.695	0.695	0.695	0.514	0.608	0.608
	F1	0.747	0.747	0.747	0.747	0.747	0.747	0.600	0.683	0.683
Fiction	P	0.748	0.751	0.742	0.747	0.743	0.747	0.782	0.793	0.786
	R	0.694	0.694	0.729	0.696	0.735	0.737	0.502	0.623	0.644
	F1	0.717	0.719	0.735	0.719	0.739	0.742	0.564	0.674	0.690
Newspapers	P	0.693	0.717	0.692	0.693	0.694	0.717	0.738	0.759	0.762
	R	0.635	0.631	0.631	0.635	0.632	0.630	0.532	0.683	0.688
	F1	0.659	0.662	0.656	0.659	0.658	0.661	0.576	0.709	0.713

Table 3: Performance on metaphor disambiguation evaluation. (Models) U: context unigram, GWC: global word category, GT: global topic dist., LC: lexical chain, AC: abstractness/concreteness, ACDep: abstractness/concreteness with words in grammatical relations, All: all features combined with context unigram. (Metrics): P: weighted precision on metaphors/literal words, R: weighted recall on metaphors/literal words, F1: weighted F1 score on metaphors/literal words

5 Conclusion

In this project, I experimented with several contextual features, both global and local, in the task of detecting metaphors for all words on running text, which in our case is the VU Amsterdam Metaphor Corpus. Although there were some hits and some misses, in general, we find that contextual information can be good indicators of metaphor, even on unrestricted text.

My code can be found at https://github.com/Sunflower/CPSC-488A.

References

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The berkeley framenet project. In *COLING-ACL*.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022.

Marc Brysbaert, Amy Beth Warriner, and Victor Kuperman. 2014. Concreteness ratings for 40 thousand generally known english word lemmas. *Behavior Research Methods*, 46:904–911.

Desai Chen, Nathan Schneider, Dipanjan Das, and Noah A. Smith. 2010. Semafor: Frame argument resolution with log-linear models. In *SemEval@ACL*.

George W. Davidson. 1982. Roget's thesaurus of english words and phrases.

Ge Gao, Eunsol Choi, Yejin Choi, and Luke S. Zettlemoyer. 2018. Neural metaphor detection in context. In *EMNLP*.

Sam Glucksberg. 2001. Understanding figurative language : from metaphors to idioms.

Ting-Hao Huang. 2013. Social metaphor detection via topical analysis. In *IJCLCLP*.

Hyeju Jang, Seungwhan Moon, Yohan Jo, and Carolyn Penstein Rosé. 2015. Metaphor detection in discourse. In *SIGDIAL Conference*.

Mario Jarmasz and Stan Szpakowicz. 2003. Roget's thesaurus and semantic similarity. In *RANLP*.

Beata Beigman Klebanov, Ben Leong, Michael Heilman, and Michael Flor. 2014. Different texts, same metaphors: Unigrams and beyond.

Beata Beigman Klebanov, Chee Wee Leong, E. Dario Gutiérrez, Ekaterina Shutova, and Michael Flor. 2016. Semantic classifications for detection of verb metaphors. In *ACL*.

Maximilian Köper and Sabine Schulte im Walde. 2017. Improving verb metaphor detection by propagating abstractness to words, phrases and individual senses.

Chee Wee Leong, Beata Beigman Klebanov, and Ekaterina Shutova. 2018. A report on the 2018 vua metaphor detection shared task.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *ACL*.

James H. Martin. 1996. Computational approaches to figurative language.

Jane Morris and Graeme Hirst. 1991. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, 17:21–48.

Jesse Mu, Helen Yannakoudakis, and Ekaterina Shutova. 2019. Learning outside the box: Discourse-level features improve metaphor identification. In *NAACL-HLT*.

Ekaterina Shutova. 2010. Models of metaphor in nlp. In *ACL*.

Ekaterina Shutova and Simone Teufel. 2010. Metaphor corpus annotated for source - target domain mappings. In *LREC*.

Ekaterina Shutova, Simone Teufel, and Anna Korhonen. 2013. Statistical metaphor processing. *Computational Linguistics*, 39:301–353.

Gerard J. Steen. 2007. Finding metaphor in discourse: pragglejaz and beyond.

Gerard J. Steen. 2010. A method for linguistic metaphor identification: From mip to mipvu.

Barbara M. H. Strang, George Lakoff, and Mark K. Johnson. 1982. Metaphors we live by.

Swabha Swayamdipta, Sam Thomson, Chris Dyer, and Noah A. Smith. 2017. Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold. *ArXiv*, abs/1706.09528.

Yulia Tsvetkov, Leonid Boytsov, Anatole Gershman, Eric Nyberg, and Chris Dyer. 2014. Metaphor detection with cross-lingual model transfer. In *ACL*.

Yulia Tsvetkov, Elena Mukomel, and Anatole Gershman. 2013. Cross-lingual metaphor detection using common semantic features.

Peter D. Turney, Yair Neuman, Dan Assaf, and Yohai Cohen. 2011. Literal and metaphorical sense identification through concrete and abstract context. In *EMNLP*.