An Exploration of *Unsupervised Learning of Narrative Event Chains*

Arun Kirubarajan
University of Pennsylvania
Computer and Information Science
kiruba@cis.upenn.edu

Abstract

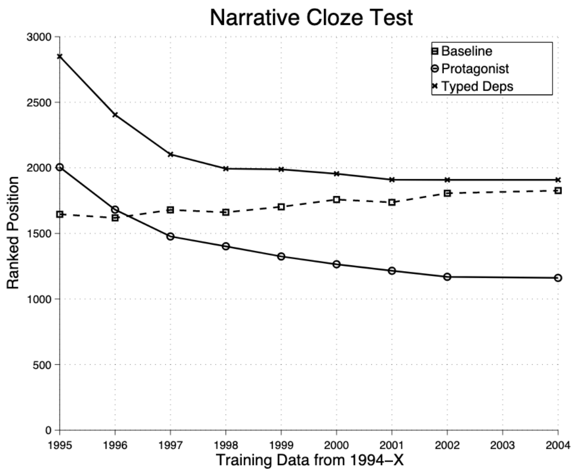
Learning discrete representations of narrative text is an open problem in machine learning for natural language processing. In 2008, Nathanael Chambers and Dan Jurafsky introduced an unsupervised method for learning **Narrative Chains**, a knowledge summary through atomic *Narrative Events* (Chambers and Jurafsky, 2008). Such chains can later be used for inference, due to their capturing of semantic knowledge. Chain generation by greedily selecting events is performed by maximizing an approximation of Pointwise Mutual Information between candidate events. In this independent study, we implement the unsupervised method introduced by Chambers et. al as well as provide improvements such as semantic knowledge from distributional and neural representations.

1 Introduction

This paper serves as a write-up of an independent study project performed at the University of Pennsylvania on unsupervised learning methods for narrative representations. In particular, this project's aim is to re-implement and update the method introduced by Chambers et. al in 2008. This aforementioned method introduces a knowledge representation format named **Narrative Chains**, which is a partially-ordered set of **Narrative Events** that can later be used downstream for inference (e.g. question answering systems).

Chambers and Jurafsky conceptualized Narrative Chains by drawing inspiration from Fillmorean Frames, which were introduced by Filmore et. al in the 1970s as part of a linguistic subfield known as **Frame Semantics** (Fillmore et al., 1976). This cursory approach involved hand-designed and hard-coded *scripts* which included slots for entities that could be replaced as needed. Chambers et. al describe Filmorean Frames as being a ”knowledge backbone” for downstream NLP inference. Prototypical sequence understanding is an efficient method of capturing narrative semantics since ”scripts” can be instantiated in various contexts and arguments can be filled in using grammatical knowledge.

The evaluation task used in the original paper is known as **Narrative Cloze**, which is an extension of the Cloze Task (Taylor, 1953). A narrative chain is provided to the task, and an event is removed in order for the model to perform a prediction to be evaluated on. The aim of the task is to perform a fill-in-the-blanks task, which upon successful completion indicates the presence of coherent narrative knowledge by the model. Chambers and Jurafsky note that this evaluation task considers both syntactic and semantic knowledge. Although the authors also state that this task may not be solvable by humans, it is still a meaningful task for an unsupervised model since it can be employed as a measure of narrative coherence.



The above graphic displays results from Chambers et. al, where the ranked position of a model indicates the average position of the correct event in a ranked list of candidate events. The original paper evaluates the Narrative Cloze evaluation on a baseline model that only considers verb co-occurrence, followed by a full grammatical model that considers typed dependencies as well, which the authors determined to be a more powerful model for performing the Narrative Cloze task. As such, this motivated my use of the typed dependency model in my implementation.

This project is implemented in Python 3.7 and includes a complete implementation suite. This implementation provides custom modules for loading plain-text data, efficient implementations of the models and algorithms introduced in the paper, as well as scripts to run the evaluation task on generated chains. Documentation is available in the code repository `https://github.com/kirubarajan/narrative_chains`.

Primary challenges in successfully completing this project were developing familiarity with prerequisite NLP pipelines (e.g. co-reference resolution and dependency parsing) as well as efficient implementations of unsupervised methods. Complete reflections on completing this independent study (from a personal perspective) are covered in Section 5.

2 Background

This section covers the methods outlined by Chambers at. al, as well as their relevant background information.

The approach introduced by Chambers et. al relies upon the NLP tasks of dependency parsing and coreference resolution. In the 2010s, approaches to these tasks have been significantly overhauled with the reintroduction of artificial neural networks. As such, extensions to the original paper include various neural-based improvements such as updated NLP pipelines and semantic knowledge from distributional learning.

2.1 Narrative Chains

Narrative Chains were introduced by Chambers et. al in 2008 as a discrete representation of narrative knowledge. This respresentation draws from Fillmorean Frames as inspiration, which defines a prototypical architecture for events (e.g. *Goes to War* or *Gets Arrested*). These hard-coded scripts can be instantiated in various contexts by filling in the arguments for each script. Narrative Chains are such scripts whose both arguments *and structure* can be learned by unsupervised methods. For example, the short narrative *"Kevin joined the army. Kevin quit the army"* can be represented using the chain of events: **(joined X), (quit X)**.

The atomic unit in a Narrative Chain is that of a **Narrative Event**, which is a tuple of an event containing a verb and its participants (e.g. dependencies/arguments). Note that narrative induction involves only considering events that share the same protagonist, which explains the absence of a subject in this particular human-readable representation that is common in the original paper. Furthermore, the **X** can indicate either a typed dependency relation or an entity itself (i.e. using coreference resolution). The restriction of considering only a single protagonist is used to induce narrative coherence.

2.2 Neural Networks

Neural Networks experienced a resurgence in the 2010's as a powerful non-linear machine learning model. They can be used as a black-box algorithm for a variety of tasks, including sequence classification, text generation, and language parsing. Their training paradigm involves determining an objective function J whose partial derivative $\frac{\partial J}{\partial w}$ (with respect to a model parameter w) is "backpropogated" throughout the network efficiently. This derivative is used to stochastically update the weights using an algorithm known as Gradient Descent until the model's loss function converges to a minimum.

Recurrent Neural Networks are like vanilla feed-forward networks, except they contain cycles, which allow the network to process sequential data. RNNs do this by maintaining a hidden state $h \in R^d$ that is updated at time-step t, and is later fed back into the network along with the network's previous output at time-step $t+1$. The hidden state h lets the network maintain context while processing the sequence.

Sometimes too much context can be a burden for the network, and results in the vanishing gradient problem where errors are propagated too far and tend to zero. This problem is resolved with models that manage context better, namely selectively remembering and forgetting parts of the context. Examples of these models are LSTMs (Long Short Term Memory) and GRUs (Gated Recurrent Units). The current state-of-the-art in natural language processing tasks involve approaches outlined in these model variations.

2.3 Co-Reference Resolution

In the approach from the original paper on Narrative Chains, the authors consider events involving co-referring entities to share narrative knowledge. Two co-referring entities are two expressions whose syntax/spelling may differ, but whose semantic entailment remains the same. For example, in the phrase *"Kevin yelled at the person bothering him"*, both *"Kevin"* and *"him"* refer to the same entity despite their strings not matching. Humans are able to perform this task with relative ease, however this task is difficult for machines to complete due to various ambiguities that arise from natural language.

Current state-of-the-art approaches in Co-Reference Resolution employ the use of neural networks, specifically Recurrrent Neural Networks with an additional mechanism called Attention (which provides an output similar to a soft-alignment over words). This task is abstracted in the implementation of this project by using third-party libraries.

2.4 Dependency Parsing

The method provided by Chambers et. al to learn Narrative Chains employs the knowledge of grammatical structure to identify narrative events. In particular, the algorithm requires identification of a verb token as well as the grammatical dependencies of the verb token to fill in the arguments of the narrative event. As such, dependency parsing is a core aspect of the approach implemented in this project.

Dependency Parsing is the task of parsing natural language into a structured representation of a syntax tree. Such a representations permits traversal to yield grammatical relationships, such as the one outlined above for narrative events. For example, for the sentence *"Kevin pushed him"*, the word *"him"* is a dependent of the verb *"pushed"*. The dependencies can further be typed (e.g. *dobj, pobj, iobj*) in order to provide additional semantic and syntactic knowledge.

The current state-of-the-art approaches involve the use of Recurrent Neural Networks augmented with an attention mechanism. This is abstracted in the implementation of this project by using third-party libraries.

2.5 Learning Narrative Chains

Chambers et. al introduce an unsupervised approach to learning Narrative Event Chains that involves the use of similarity measures borrowed from information theory, known as Pointwise Mutual Information. This is done to extract pairwise relationships between events in an unsupervised manner by considering the frequencies of events as well as their co-ocurrence counts.

Given two narrative events $e(w,d)$ and $e(v,g)$ with dependencies d and g, their Pointwise Mutual Information score can be approximated by computing:

$$PMI(e(w,d),e(v,g)) = log\frac{P[e(w,d),e(v,g)]}{P[e(w,d)]\cdot P[e(v,g)]}$$

The marginal probability can be calculated by simple empirical frequency of the event. Let $C(e(w,d),e(v,g))$ be a function that returns the count of that the two events share dependencies for d and g. Then, the joint probability between two events can be computed by marginalizing over the space of verbs and dependencies:

$$P[e(w,d),e(v,g)] = \frac{C(e(w,d),e(v,g))}{\sum_{x,y}\sum_{d,f}C(e(x,d),e(y,f))}$$

Given a current chain c of n events and a candidate event e, we can assign a score to the next most likely event by computing

$$\sum_{i=0}^{n} PMI(e,c_i)$$

and taking the argmax over all events. We can greedily take the top-scoring candidate as the next event in the chain.

3 Implementation

My implementation of the project is split into 5 core modules which were developed from scratch (with various dependencies):

1. `data` : handler for data loader instantiation and text pre-processing of Gigaword Corpus
2. `parse` : pipeline for syntactic and grammatical parsing of input text corpus into dependencies and entities
3. `models` : implementations of probability calculations (and PMI) as well as Narrative Chain modeling helpers

4. `evaluation` : implementation of the Narrative Cloze task (with word embedding extensions/support)

5. `test` : sandbox for above modules, including configurable test runner

3.1 Data Loading

A custom data loading class is defined as `DataLoader` class in the `data` module. The API and usage of this DataLoader class is inspired by the Data Loader modules available in PyTorch, which offer a variety of Pythonic constructs for managing data. In particular, a `DataLoader` instance can be created by calling a helper function `build_loader()` with a path to a plaintext dataset. The `DataLoader` class supports Python magic methods such as `len()`, which returns the number of lines included in the input file. A debugging helper named `sanity_check()` was implemented for displaying a small portion of human-readable training data.

For my testing purposes, I used a subset of the Gigaword Corpus. The compressed subset of the Gigaword Corpus was flattened using the **agiga** maven package, and piping the output (using the *"words"* parser) to a text file. Then, a `DataLoader` object was instantiated, and its helper function `clean_agiga_text()` to remove the **agiga** metadata from the output.

3.2 Python Representation

In order to aid reproducibility, the computations involved in learning narrative events was abstracted into Python classes and modules. In particular, the primary abstracted used was to defined a custom data structure for represented each event. In the `models` module, I defined a custom class `Event`, whose main member variables represent the subject, verb, dependency, and typed dependency of a given event. Furthermore, I implemented several Python magic methods as a quality-of-life measure. First, I implemented the `str()` magic method, in order to display a given event in a human-readable format. Next, I implemented the `hash()` and `eq()` magic methods as a means of consistent event equality. In particular, equality between events is checked through verb/dependency agreement.

The next custom class implemented is the `Document` class, which serves as a representation of the learned models. In particular, it stores a frequency dictionary of events, as well as maintains a vocabulary (set) of possible subjects, verbs, dependencies and dependency types in order to be marginalized upon in modeling events/chains.

3.3 Parsing

The `parse` module contains functions and scripts for constructing structured representations of the syntax tree for a given input corpus. This module relies on the third party libraries of **neuralcoref**, **spacy** and **stanfordnlp** to compute the parse trees. Various implementations of the parse functions are supported to provide options for selecting which parsing library to use (the default being SpaCy for all tasks due to its wide support).

The `parse_stanford()` function relies on the StanfordNLP package to parse the document for dependencies, in order to identify narrative events. However, the `parse_document()` method is the primary implementation for resolving verb/dependency pairs and uses the SpaCy pipeline for simplicity. The output of the parsing method is a `Document` object, whose member value `events` contains a frequency dictionary of observed events with their associated counts.

The implementation parses the input text using the SpaCy pipeline and iterates over each token. Upon encountering a **VERB** token, the algorithm parses the dependency tree for arcs that end in a supported dependency tag, which serves as the arguments for the narrative event. Then, a narrative event `Event` is instantiated, and its occurrence is recorded in the `Document` frequency table. Finally, the verb's subject and arguments are recorded in the `Document` vocabulary as well.

Currently, only certain dependency tags (outlined in the SpaCy documentation) are supported as verb arguments. These dependency tags include direct, passive, and indirect objects as well as nominal subjects. Future work will consist of expanding the set of supported dependency tags in order to produce more syntactically-rich models (more in Section 4).

3.4 Narrative Chain Modeling

For the maximum length permitted by the SpaCy pipeline (1 million characters) of a subset of the Gigaword corpus, a total of 1394 unique events were identified, supporting 1393 unique verbs, 1218 unique subjects and 1861 unique dependencies. Examples of identified events are as follows (including subject, verb, dependency and dependency tag):

Science lists breakthroughs dobj
who straddles line dobj
that answers questions dobj
scientists master reprogramming dobj
Astronomers observed planets dobj
we talking years dobj
what causes it dobj
that conducts electricity dobj
lines conduct electricity dobj

Qualitative assessment shows both that the Gigaword corpus has a bias towards direct argument usage (following verbs) and that the parser's limitation of certain dependency tags can overrepresent direct objects in Narrative Event identification. This is most likely due to a small set of supported dependency tags in my implementation. It's possible that the authors of the original paper opted to include more dependency tags to create a syntactically richer model.

In the `models` module, four functions are used to learn narrative chains. The first is `event_prob()`, which computes the probability of a single given event occurring in the document. This marginal probability is calculated in a straightforward fashion, since the `Document` object maintains a frequency dictionary of a given event in the corpus. Thus, the calculation is simply to divide the frequency of the event by the sum of the frequencies in the dictionary.

The second function used to learn narrative chains is the `joint_event_prob()`, which computes the co-occurrence probability of two events sharing the same typed dependencies. This is computed by marginalizing over all possible verb/dependency pairs in the corpus and counting the occurrences of (this counting is abstracted into the `count()` helper function I implemented) each match:

$$P[e(w,d),e(v,g)] = \frac{C(e(w,d),e(v,g))}{\sum_{x,y}\sum_{d,f} C(e(x,d),e(y,f))}$$

The next function I implemented was `pmi_approx()` which approximates the Pointwise Mutual Information between two events using the aforementioned joint event probability function. This function was difficult to implement due to numerical underflow as the text corpus grew in size. To mitigate this, I used logarithm arithmetic using the Python `math` module, which prevented any possible domain errors (e.g. division by zero).

Finally, narrative chains are generated by initializing a chain with a starting event and making repeated calls to `predict_events()`, using a parameter of $n = 1$. Using the greedy algorithm for generating an event chain (highlighted in Section 2.5), an example of generated narrative chains is provided as follows, in the format provided by the original paper, where **X** refers to *"the navy"*:

joined X, disliked X, left X
served X, oversaw X

Qualitative assessment shows narrative and semantic coherence within the chain (e.g. *disliked*, *left*). This is most likely due to the frequency of co-referring arguments for the verb pair (*disliked, left*) as well as for (*served, oversaw*). In various experiments of generating chains, it was observed that shorter chains tended to be more coherent (by virtue of narrative structure, as mentioned by the authors).

The `predict_events()` takes in an optional boolean keyword parameter `embedding`, which defaults to being false. If the parameter is set to true, then the candidate outputs are ranked using the *embedding similarity* between the candidate's and target's verbs. Downstream, this affects the model's position in the Narrative Cloze evaluation task. Additional reference on the use of word embeddings in this project is available in Section 3.6.

3.5 Evaluation Task

The original **Cloze Task** was introduce by Taylor in 1953 to evaluate human language proficiency. The task consists of removing a word from a sentence in order to have the testing subject (or system) predict the missing entity. Depending on what is to be predicted, the Cloze Task can evaluate grammatical, syntactic or semantic knowledge. Chambers et. al introduce a variant of the Cloze Task known as **Narrative Cloze**, in which the testing subject must predict a narrative event e. The authors make the claim that this task may not be solvable by humans, and rather refer to it as a "comparative measure to evaluate narrative knowledge".

The Narrative Cloze evaluation task is implemented in two functions. The first function is `predict_blank()`, which accepts a narrative chain and an optional `index` parameter to mask the event at the given index to be predicted by the model. If no index is provided, the function randomly removes an event from the chain in order to be predicted. The function outputs the ranked hypothesis events, which is later used to compute the model's overall ranking. The second function is `score_predictions()`, which accepts a list of tuples of "cleaned" chains (chains missing the target event) and the ground-truth event. The function outputs the model average ranking of the correct event. A helper function `get_position()` was also implemented in order to abstract model position calculation in order to be averaged in the `score_predictions()` module.

In my experiments with hand-designed chains, the model position tended to that of the size of the event space. In one subset of the Gigaword corpus consisting of 54 potential events, the model position tended to an average hovering 35. This is most likely due to the events in the test chain not being present in the model's event space. In my implementation, this case is handled by assigning the position to be the last position in the rankings, indicating the lowest score. As a result, it appears that this model suffers from the curse of dimensionality. Due to time constraints, I was not able to partition the Gigaword Corpus into categories, which is most likely how the authors of the original paper were able to constrain the event space such that the model position would be higher.

It is again worth noting that although model performance appears to be low, the Narrative Cloze task may not be solvable by humans and therefore the result from the implementation are only to get a sense of the model's ability to capture narrative coherence. Overall, despite the low model position, the model does in fact appears to learn meaningful narrative relationships between verb/dependency pairs.

3.6 Updates and Extensions

The first notable update to the original paper's implementation was the use of upgraded NLP pipelines. In particular, instead of using the NTLK package to implement NLP tasks such as co-reference resolution and dependency parsing, I opted to use the updated StanfordNLP (Qi et al., 2019) package and the SpaCy NLP pipeline (Honnibal and Montani, 2017). This is a notable update to the existing implementation due to powerful advances in neural network architectures. A Bi-Directional LSTM (Long Short-Term Memory) with Soft-Attention was used by the libraries to resolve co-reference as well as parse dependencies. In addition, these packages were able to leverage GPUs and the CUDA environment in order to parallelize and accelerate computation. This made it significantly faster to prototype implementations. The HuggingFace library NeuralCoRef was used as an extension to the Spacy module in order to provide fast and accurate coreference resolution.

A common trend in natural language processing literature is to employ the use of pre-trained models in order to leverage existing semantic knowledge. As such, I extend the functionality of both the chain generation algorithm as well as the Narrative Cloze evaluation task by using **word embeddings** to capture semantic relationships, on top of grammatical relationships from the original paper. The contribution of this extension is two-fold: 1) the development velocity improvements of leveraging an pre-trained model and 2) utilizing distributional semantics in the modeling of Narrative Event Chains.

I used a pre-trained implementation of Word2Vec (Mikolov et. al) that provides word vectors for every tested word in the vocabulary (barring proper nouns and names) of dimension 300. Word embeddings are often referred to as the "secret sauce" of NLP since they provide a continuous representation of words in vector space, which allows us to compute semantic similarity via cosine similarity:

$$cos(u,v) = \frac{u \cdot v}{|u| \cdot |v|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \cdot \sqrt{\sum_{i=1}^{n} v_i^2}}$$

I used this similarity measure as a potential replacement of the pairwise PMI score between candidates and a given narrative chain for chain generation. In the `predict_events()` function, an optional keyword boolean argument for `embedding` can provided to predict an event using embedding similarity between event verbs. Similarly, in the evaluation module for implementing the Narrative Cloze task, instead of computing position using the Pointwise Mutual Information scores, we can instead rank candidate events using the similarity between event verbs.

Future work in utilizing word embeddings to measure event similarity would be to incorporate some interpolation factor α in order to leverage both PMI for verb/dependcy co-occurrences as well as distributional semantics from models such as Word2Vec. Due to time constraints, ablation experiments were not performed using the embedding-based similarity measures on the Narrative Cloze task, but an example generated chain is as follows:

fell X, **jumped X**, **played X**

Qualitative assessment of the chains shows that although the verbs maintain semantic coherence, the narrative itself does not demonstrate narrative coherence. This is most likely due to an absence of temporal knowledge within the pre-trained embeddings, which the PMI model captures via co-occurring dependencies.

4 Future Work

The first area of improvement is to expand the grammatical parsing capabilities of the current implementation. As seen in Section 3.4, there is an overrrepresentation of direct and passive objects as dependencies in the Narrative Events. This is because I chose to identify arguments satisfying the prototypical arguments in the SpaCy documentation, since the authors did not include what dependency tags were selected in the original paper. These additional dependency tags can be identified by running a POS-tagger or dependency parser on the gold-standard Narrative Chains released by Chambers and selecting for them in the `parse` module.

Additional future work involves the task of generating Narrative Chains. Recall that the objective demonstrated by the authors of the original paper to generate chains is to maximize the sum of Pointwise Mutual Information between an event candidate e and an event chain c. In modern generation literature, this is referred to as *Greedy Decoding*, using the summed PMI score as a generation signal. I hypothesize that as the training corpus grows (thereby increasing the size of the vocabulary and, in turn, the event space), alternative decoding strategies can result in more diverse and semantically coherent outputs. For example, as chain lengths get larger (to represent more intricate narratives), decoding algorithms such as Beam Search, a search algorithm which maintains a set of top candidate options, can be used in order to mitigate the shortcomings of Greedy Decoding.

An interesting direction to take the work presented in this project is through the lens of natural language generation (NLG). Since the narrative event chains provide an accurate and simple discrete representation of a narrative, I hypothesize that it is possible to use the narrative chains as a model's internal representation of a narrative body of text. In modern Seq2Seq models, a given network's internal representation is a continuous vector that is maintained by the Encoder network before being passed to the decoder network. Although this representation is concise, it can offer suffer from the curse of dimensionality that comes from continuous space. This could be mitigated by the much smaller (by comparison) space of possible discrete events from a given corpus. Furthermore, using Narrative Event Chains as a representation is much more human-interpretable than a continuous vector. This means that it will be easier for humans to annotate/summarize narrative text into this representation as a potentially powerful dataset that can assist in the tasks of machine summarization, natural language understanding and natural language generation.

5 Discussion and Reflections

This project was challenging, but fun! I think the most challenging aspect of implementing this paper was getting up to speed with dependency parsing and coreference resolution. These topics were covered in CIS 530 (Computational Linguistics), but I wasn't able to implement or use any existing pipelines (since they weren't assignments). Getting my hands dirty with these libraries and concepts took up most of the development time, since I had to learn the various dependency types as well as APIs for the NLP engine. Thankfully, the SpaCy and StanfordNLP documentation were very helpful. Implementing the probability functions and defining certain algorithm behaviour was also difficult since the original paper wasn't as verbose as I would have liked. I ended up taking a couple of ”academic liberties” in implementing certain functionality that I believed was true to the original paper and was efficient to program.

My favourite part about this project was being able to manually inspect the identified events and chains. During training, my personal pick for "Most Memorable Event" is definitely **(scientists, master, reprogramming)**, since I feel it captures my essence re-implementing this paper. I felt the chains weren't as coherent as I would have liked, but the neural network extensions I outlined in the Future Work section should be an interesting place to pick up from after this semester is officially over. My initial disappointment in poor performance was assuaged by the author's note that the Narrative Cloze evaluation task is rather challenging for both models and humans.

Overall, I feel like I learned a lot, not just about PMI or dependency parsing, but mainly the process involved in conducting research. Decisions have to be made and trade-offs are always apparent, especially when working with large datasets. Moving forward from what I've learned in this project, I'm excited to tackle additional narrative-related problems with a new perspective in CIS 700-8!

Acknowledgments

Special thank you to Chris Callison-Burch for introducing me to this (really cool) project and for being flexible with the project timeline.

Additional acknowledgements for Daphne Ippolito for helping me navigate the various stages of implementing my first research paper.

This work could not be done without the original paper published by Nathanael Chambers and Dan Jurafsky, and Chambers' gold-standard chains being released online.

References

Nathanael Chambers and Dan Jurafsky. 2008. Unsupervised learning of narrative event chains. In *Proceedings of ACL-08: HLT*, pages 789–797, Columbus, Ohio. Association for Computational Linguistics.

Charles J Fillmore et al. 1976. Frame semantics and the nature of language. In *Annals of the New York Academy of Sciences: Conference on the origin and development of language and speech*, volume 280, pages 20–32.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D Manning. 2019. Universal dependency parsing from scratch. *arXiv preprint arXiv:1901.10457*.

Wilson L Taylor. 1953. cloze procedure: A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433.