Span-Level Model for Relation Extraction

Kalpit Dixit
Amazon AWS AI, USA
kddixit@amazon.com

Yaser Al-Onaizan
Amazon AWS AI, USA
onaizan@amazon.com

Abstract

Relation Extraction is the task of identifying entity mention spans in raw text and then identifying relations between pairs of the entity mentions. Recent approaches for this span-level task have been token-level models which have inherent limitations. They cannot easily define and implement span-level features, cannot model overlapping entity mentions and have cascading errors due to the use of sequential decoding. To address these concerns, we present a model which directly models all possible spans and performs joint entity mention detection and relation extraction. We report a new state-of-the-art performance of 62.83 $F1$ (prev best was 60.49) on the ACE2005 dataset.

1 Introduction

Many NLP tasks follow the pattern of taking raw text as input and then: detecting relevant spans and classifying the relations between those spans. Examples of this include Relation Extraction (Li and Ji, 2014), Coreference Resolution (Ng, 2010) and Semantic Role Labeling (Gildea and Jurafsky, 2002). This class of NLP problems are inherently span-level tasks. This paper focuses on Relation Extraction (RE), which is the task of entity mention detection and classifying the relations between each pair of those mentions. We report a new state-of-the-art performance of 62.83 $F1$ (prev best was 60.49) on the ACE2005 dataset.

Here is a simple example of Relation Extraction for the sentence, "Washington, D.C. is the capital of the USA". Step 1, Entity Mention Detection will detect the spans "Washington, D.C." and "USA" as LOCATIONS. Step 2, Relation Extraction will classify all directed pairs of detected entity mentions. It will classify the directed pair ("Washington, D.C.", "USA") as having the relation IS_CAPITAL_OF. But the directed pair ("USA", "Washington, D.C.") will be classified as having no relation (NONE). In more complex cases, each entity could participate in multiple different relations.

Since (Li and Ji, 2014), work on RE has revolved around end-to-end systems: single models which first perform entity mention detection and then relation extraction. These recent works (Bekoulis et al., 2018; Katiyar and Cardie, 2017; Miwa and Bansal, 2016; Li and Ji, 2014) have used sequential token-level methods for both the steps. Token-level models are primarily constrained by the fact that each token has a single fixed representation while each token is a part of many different spans. To model and extract spans, these token-level models have to resort to approximate span-level features which are increasingly indirect and expensive: Tree-LSTMs (Miwa and Bansal, 2016), CRFs (Bekoulis et al., 2018), Beam Search (Li and Ji, 2014) and Pointer Networks (Katiyar and Cardie, 2017). Their usage of the BILOU (Ratinov and Roth, 2009; Florian et al., 2006) token-tagging scheme makes modelling overlapping entities impossible. In general, these token-level models are sequential in nature and hence have cascading errors.

Another end-to-end approach for RE is to use a simple span-level model. A model which creates explicit representations for all possible spans, uses them for the entity mention detection step and then explicitly compares ordered pairs of spans for the relation extraction step. Such a model is not constrained like the token-level models because it can define direct span-specific features for each span inexpensively. Since each possible span is separately considered, selecting overlapping entity mentions is possible. Predicting one span as an entity no longer blocks another span from being predicted as an entity. This approach models each possible span independently and in parallel i.e. it is not sequential and does not suffer from cascad-

ing errors. Such models have recently found success in similar NLP tasks like Coreference Resolution (Lee et al., 2017) and Semantic Role Labeling (Ouchi et al., 2018). In this paper, we present such a span-level model for Relation Extraction.

We propose a simple bi-LSTM based model which generates span representations for each possible span. The span representations are used to perform entity mention detection on all spans in parallel. The same span representations are then used to perform relation extraction on all pairs of detected entity mentions. We evaluated the performance of our model on the ACE2005 dataset (Doddington et al., 2004) and report a new start-of-the-art $F1$ score of 62.83 for Relation Extraction.

2 Related Work

Given text input, Relation Extraction involves two steps: span detection and classification of the relation between pairs of detected spans. In the RE literature, these are more commonly called Entity Mention Detection and Relation Extraction respectively. An earlier line of research has focused on only the second step, assuming that the arguments of the relations are given by some other system/oracle (Bunescu and Mooney, 2005; Socher et al., 2012; dos Santos et al., 2015).

The more interesting problem is joint Entity Mention Detection and Relation Extraction. More interesting because it simultaneously addresses both steps, enriches embeddings from losses related to both sub-tasks and only requires using a single model during test. Past approaches include Integer Linear Programming (Yang and Cardie, 2013) and Probabilistic Graphical Models (Singh et al., 2013). Li and Ji (2014) modeled this joint task as a Structured Prediction problem and since then most work on RE has revolved around end-to-end systems which do the joint task (Miwa and Bansal, 2016; Katiyar and Cardie, 2017; Bekoulis et al., 2018).

A common theme in current end-to-end models is the use of token-level models. For the entity mention detection step, recent works (Miwa and Bansal, 2016; Katiyar and Cardie, 2017; Bekoulis et al., 2018) have used the BILOU (Ratinov and Roth, 2009; Florian et al., 2006) token-tagging scheme. For the relation extraction step there have been a variety of methods tried like Tree-LSTMs (Miwa and Bansal, 2016), sequence labeling (Katiyar and Cardie, 2017) and multi-head selection (Bekoulis et al., 2018). Li and Ji (2014) used semi-Markov chains and the Viterbi algorithm, which is also a sequential token-level approach. This token-level modeling approach has several limitations as highlighted in Section 1.

Recent work using span-level end-to-end models have seen success in NLP tasks following the same pattern as RE (Coreference Resolution (Lee et al., 2017) and Semantic Role Labeling (Ouchi et al., 2018)). In this paper, we adapt (Lee et al., 2017) to create a span-level end-to-end model for RE.

3 Model

Our model consists of three steps which we explain in detail in the next subsections:

1. **Span Representation Generation**

   Use task-agnostic raw token embeddings to create task-specific token embeddings for each token. The task-specific token embeddings are used to generate span embeddings for each possible span.

2. **Entity Mention Detection (EMD)**

   The span embeddings are used to obtain a vector of entity type scores for each span. Each span is assigned the entity type corresponding to its highest entity type score. The spans that are assigned an entity type other than `NONE` are selected for Step 3.

3. **Relation Extraction (RE)**

   For each ordered span-pair (i, j), we obtain a representation by concatenating the respective span embeddings. This representation is defined in an order-sensitive way in Section 3.3 i.e. the span-pair representation of spans (i, j) is different from that of spans (j, i). For each ordered span-pair, its representation is used to obtain a vector of relation type scores. Each ordered span-pair is assigned the relation type of its highest relation type score.

3.1 Step 1: Span Representation Generation

The architecture we use to generate span representations closely follows (Lee et al., 2017).

Given a document D with T tokens, there are $N = \frac{T(T+1)}{2}$ possible spans. span i is defined by all the tokens from START(i) to END(i) inclusive, for $1 \leq i \leq N$. The aim is to obtain a span representation $\mathbf{g_i}$ for each span i.

Raw Token Embeddings We use $\mathbf{x_t}$ to represent the raw token embeddings of token t with $1 \leq t \leq T$. $\mathbf{x_t}$ is a concatenation of the following:

1. Fixed Contextual Word Embeddings
2. Fixed Word Embeddings
3. Trained from scratch Character Embeddings

We use fixed ELMo (Peters et al., 2018) for Contextual Word Embeddings, fixed Senna (Collobert et al., 2011) for Word Embeddings and train Character Embeddings from scratch. The Contextual Word Embeddings for each sentence were computed separately.

In terms of number of free parameters; Contextual Word Embeddings use the most (100's of millions), followed by Word Embeddings (10's of millions) and finally Character Embeddings use by far the least (10's of thousands). The decision to train only the Character Embeddings was based on overfitting concerns given our relatively small dataset.

Bi-LSTM Layers The pretrained Contextual Embeddings we use in $\mathbf{x_t}$ above are obtained by unsupervised task-agnostic training. To obtain task-specific contextualization we use stacked bidirectional LSTMs (Hochreiter and Schmidhuber, 1997) on the raw token embeddings $\mathbf{x_t}$ to obtain $\mathbf{x_t^*}$,

$$\mathbf{x_t^*} = [\overrightarrow{\mathbf{h_t}}, \overleftarrow{\mathbf{h_t}}]$$

where $\overrightarrow{\mathbf{h_t}}$ and $\overleftarrow{\mathbf{h_t}}$ are the hidden states of the last layer of the forward and backward LSTMs respectively. $\mathbf{x_t^*}$ is the concatenation of $\overrightarrow{\mathbf{h_t}}$ and $\overleftarrow{\mathbf{h_t}}$. The bi-LSTMs were run separately on each sentence as that gave better performance.

Span Representation Syntactic heads obtained from general syntactic parsers are used in many NLP systems. Here we don't use general syntactic parsers but instead use attention (Bahdanau et al., 2015) to create a task-specific span-head feature. This feature vector is computed for each span:

$$\alpha_{\mathbf{t}} = \text{MLP}_\alpha(\mathbf{x_t^*})$$

$$\beta_{\mathbf{i,t}} = \frac{\exp(\alpha_{\mathbf{t}})}{\sum_{k=\text{START}(i)}^{\text{END}(i)} \exp(\alpha_k)}$$

$$\hat{\mathbf{x}}_{\mathbf{i}} = \sum_{k=\text{START}(i)}^{\text{END}(i)} \beta_{\mathbf{i,t}} \mathbf{x_t}$$

where MLP_α is a Multi Layer Perceptron (aka Feed Forward Network). $\hat{\mathbf{x}}_{\mathbf{i}}$ is a weighted sum of fixed word vectors for the tokens in span i. We did experiment with using the weighted sum of the bi-LSTM output ($\mathbf{x_t^*}$) or of the ELMo (Peters et al., 2018) fixed contextual word embeddings but got better results with using fixed word embeddings.

For each span i, its span representation $\mathbf{g_i}$ was defined as:

$$\mathbf{g_i} = [\mathbf{x^*}_{\text{START}(i)}, \mathbf{x^*}_{\text{END}(i)}, \hat{\mathbf{x}}_{\mathbf{i}}, \phi(i)]$$

where $\phi(i)$ encodes the size of span i in number of tokens. Each component of $\mathbf{g_i}$ is a span-specific feature that would be difficult to define and use in token-level models.

3.2 Step 2: Entity Mention Detection (EMD)

In this step, we predict the entity type for each span. This prediction is done identically and parallelly for each span. For each span we compute a vector of entity type scores. The number of entity type scores computed is the number of entity types (including the NONE entity type). For each span, the softmax function is applied to its entity type scores to get a distribution over the entity types. For span i,

$$\begin{aligned} \mathbf{score_i^{ner}} &= \text{MLP}_{ner}(\mathbf{g_i}) \\ \mathbf{p_i^{ner}} &= softmax(\mathbf{score_i^{ner}}) \end{aligned} \quad (1)$$

The output size of MLP_{ner} and hence the size of $\mathbf{p_i^{ner}}$ is equal to the number of NER classes.

The predicted entity type for each span i is the entity type corresponding to span i's highest entity type score i.e. $\max(\mathbf{score_i^{ner}})$. Only spans whose predicted entity type is not NONE are selected for Step 3. Unlike token-level models, overlapping spans can be selected here as each span's selection decision is independent of other spans.

3.3 Step 3: Relation Extraction (RE)

In this paper, we only consider ordered binary relations, the most common setting of RE i.e. only relations between exactly two arguments and where the two pairs (span i, span j) and (span j, span i) are considered different. We consider every ordered pair of selected spans (from Step 2) such that both spans are from the same sentence. For each such pair (span i, span j), we first compute an ordered pair embedding $\mathbf{r_{(i,j)}}$:

$$\mathbf{r_{i,j}} = [\mathbf{g_i}, \mathbf{g_j}, \mathbf{g_i} \circ \mathbf{g_j}]$$

where $\mathbf{g_i}$ and $\mathbf{g_j}$ are the span embeddings of the 1st and 2nd arguments respectively (from Step 1). $\mathbf{g_i} \circ \mathbf{g_j}$ refers to their element-wise product.

We use the ordered pair embedding $\mathbf{r_{i,j}}$ to compute a vector of relation type scores. The number of relation type scores is the number of relation types (including the NONE relation type). For each ordered pair of spans, the softmax function is applied to its relation type scores to get a distribution over the relation types. For pair (span i, span j),

$$\begin{aligned} \mathbf{score^{re}_{i,j}} &= \text{MLP}_{re}(\mathbf{r_{i,j}}) \\ \mathbf{p^{re}_{i,j}} &= softmax(\mathbf{score^{re}_{i,j}}) \end{aligned} \quad (2)$$

The output size of MLP_{re} and hence the size of $\mathbf{p^{re}_{i,j}}$ is equal to the number of RE classes.

3.4 Loss

Two learning signals are provided to train the model: entity type information per span and relation type information per ordered (selected) span pair. Both are provided via CrossEntropy Loss on Equations 1 and 2 respectively. We use $\hat{y}_i^{ner}$ to represent the correct entity type for span i and $\hat{y}_{i,j}^{re}$ to represent the correct relation type for the ordered pair of spans, (span i, span j). $\mathcal{S}$ represents the set of all spans and $\mathcal{S}'$ represents the set of all selected spans (Section 3.2). Then the final training loss is,

$$\text{loss} = \sum_{i \in \mathcal{S}} \mathbf{p^{ner}_i}(\hat{y}_i^{ner}) + \sum_{i \in \mathcal{S}'} \sum_{j \in \mathcal{S}', j \neq i} \mathbf{p^{re}_{i,j}}(\hat{y}_{i,j}^{re})$$

where the first term is a sum over all spans of the entity mention detection loss (eqn 1) and the second term is a sum over all ordered pairs of selected spans of the relation extraction loss (eqn 2).

4 Experiments

Dataset We use the ACE2005 dataset (Doddington et al., 2004). It has 351 documents for train, 80 for validation and 80 for test. There are seven span-level entity types and six ordered span relation types.

Character Embeddings The learned character embeddings are of size 8. 1-dimensional convolutions of window size 3,4,5 are applied per-token with 50 filters of each window size. This is followed by ReLU activation (Nair and Hinton, 2010) and max-pooling over each filter.

Model Size Our stacked bi-LSTMs (Section 3.1) has 3 layers with 200-dimensional hidden states and highway connections. All Multi Layer Perceptrons (MLP) has two hidden layers with 500 dimensions, each followed by ReLU activation.

Feature Encoding Each span gets a span width feature which is a learned 20-dimensional vector representing the number of tokens in that span.

Span Pruning A high number of spans under consideration can lead to memory and speed issues. We only consider spans that are entirely within a sentence and limit spans to a max length of $L = 10$. This choice was based on our Train Set, see Section 5) for a discussion about it. Performance is not affected significantly as very few entity mentions have more than 10 tokens.

Regularization Dropout (Srivastava et al., 2014) is applied with dropout rate 0.2 to all hidden layers of all MLPs and feature encodings, with dropout rate 0.5 to all word and character embeddings and with dropout rate 0.4 to all LSTM layer outputs.

Learning Learning is done with Adam (Kingma and Ba, 2015) with default parameters. The learning rate is annealed by 1% every 100 iterations. Minibatch Size is 1. Early Stopping of 20 evaluations on the dev set is used.

5 Model Complexity

Section 3.1 describes our span generation process and Section 4 describes our algorithmic span pruning process. The algorithmic span pruning process limits our model spans which are entirely within a single sentence and have a max length of $L = 10$ tokens. While our model creates representations for spans (instead of just tokens), it achieves the dual goals of being memory efficient and capturing most (more than 99.95%) entities and relations in the space of the spans considered.

Table 2 shows the model complexity and entity/relation coverage for different policies of span generation on the Train Set of ACE2005. It shows numbers for policies ranging from one which considers all spans across the doc, to a policy that considers only single token spans. It shows that our chosen span generation policy (in bold) is far more memory efficient than a naive search over all possible spans in the input document. Yet our policy still considers more than 99.95% of all entities and relations. Our policy is linear in the document's (sentence) length, not quadratic; because we limit our model to spans that are wholly in a single sentence and have a max length of $L = 10$ tokens.

	Entity Mention Detection			Relation Extraction		
System	P	R	$F1$	P	R	$F1$
(Li and Ji, 2014)	85.2	76.9	80.8	**68.9**	41.9	52.1
(Miwa and Bansal, 2016)	82.9	83.9	83.4	57	54.0	55.6
(Katiyar and Cardie, 2017)	84.0	81.3	82.6	57.9	54.0	55.9
(Sanh et al., 2018) EMD + RE	**86.54**	85.49	**86.02**	68.66	54.05	60.49
(Sanh et al., 2018) multi-task *	85.68	85.69	85.69	68.53	54.48	61.30
our model	85.85	**86.10**	85.98	68.02	**58.38**	**62.83**

Table 1: EMD and RE results on the ACE2005 Test dataset. Our model reports a new state-of-the-art RE performance. Sanh et al. (2018) present several results in their multi-task paper. Results marked with (*) are not fair comparisons here because they use additional signals beyond EMD and RE. Included here for completeness.

Permitted Spans	# Spans	% Entities Covered	% Relations Covered
all spans across doc	45,836,252	100.00	100.00
only spans within single sentence	1,894,256	100.00	100.00
+ max length L = 10	**1,079,150**	**99.99**	**99.96**
+ max length $L = 5$	632,477	99.92	99.60
+ max length $L = 2$	279,515	98.02	94.92
+ max length $L = 1$	144,783	89.13	78.87

Table 2: Numbers are for the Train Set (351 docs) of ACE2005, where each Relation is between exactly two Entities. Dev and Test Sets follow the same trends. Each row is a different policy for span generation and our chosen policy is bolded. "*# Spans*" is the number of spans considered by the policy. "*% Entities Covered*" is the percentage of entities in the dataset that are considered by that policy. "*% Relations Covered*" is the same thing for Relations (i.e. a Relation is covered if both entities of the Relation are covered). Note how our chosen policy is more than 40x more memory efficient than a policy which considers all spans in the doc. And yet, our method covers 99.99% and 99.96% of all Entities and Relations respectively in the Train Set of ACE2005.

6 Results

Table 1 shows the results for RE. For the joint task, we compare entity mention detection performance and relation extraction performance. Our proposed model achieves a new SOTA on RE with a $F1$ of 62.83, more than 2.3 $F1$ above the previous SOTA. Our proposed model also beats a multi-task model Sanh et al. (2018) which uses signals from additional tasks by more than 1.5 $F1$ points.

For both tasks, our model's Precision is close to and Recall is significantly higher than previous works. The Recall gains for RE (4.3 absolute points) are much higher than for EMD (0.6 absolute points). The gains in EMD Recall highlights the effectiveness of our span representations (Section 3.1). The disproportionate gains in RE Recall cannot be fully explained by the relatively lower gains in EMD Recall. Thus, our large gains in RE Recall (and $F1$) showcase the effectiveness of our simple modeling of ordered span pairs for relation extraction (Section 3.3).

7 Conclusions

We present a neural span-level end-to-end model for joint entity mention detection and relation extraction. In contrast with existing token-level models: our model is able to use span-specific features, allows for overlapping entity mentions and does not use sequential decoding. Our proposed model achieves a new state-of-the-art RE performance on the ACE2005 dataset. The gains are driven by improvements in Recall for both tasks.

Acknowledgements

We would like to thank He He for her guidance in writing this paper and comments on multiple manuscripts. We would also like to thank the anonymous reviewers of ACL 2019, their comments and suggestions have helped us focus more on the important aspects of this paper.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by

jointly learning to align and translate. *CoRR*, abs/1409.0473.

Giannis Bekoulis, Johannes Deleu, Thomas Demeester, and Chris Develder. 2018. Joint entity recognition and relation extraction as a multi-head selection problem. *Expert Syst. Appl.*, 114:34–45.

Razvan C. Bunescu and Raymond J. Mooney. 2005. A shortest path dependency kernel for relation extraction. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 724–731, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537.

George R Doddington, Alexis Mitchell, Mark A Przybocki, Lance A Ramshaw, Stephanie Strassel, and Ralph M Weischedel. 2004. The automatic content extraction (ace) program-tasks, data, and evaluation. In *LREC*, volume 2, page 1.

Radu Florian, Hongyan Jing, Nanda Kambhatla, and Imed Zitouni. 2006. Factorizing complex models: A case study in mention detection. In *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Comput. Linguist.*, 28(3):245–288.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Arzoo Katiyar and Claire Cardie. 2017. Going out on a limb: Joint extraction of entity mentions and relations without dependency trees. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 917–928. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 188–197. Association for Computational Linguistics.

Qi Li and Heng Ji. 2014. Incremental joint extraction of entity mentions and relations. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 402–412. Association for Computational Linguistics.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using lstms on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1116. Association for Computational Linguistics.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA. Omnipress.

Vincent Ng. 2010. Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1396–1411, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hiroki Ouchi, Hiroyuki Shindo, and Yuji Matsumoto. 2018. A span selection model for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1630–1642. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 147–155, Stroudsburg, PA, USA. Association for Computational Linguistics.

V. Sanh, T. Wolf, and S. Ruder. 2018. A hierarchical multi-task approach for learning embeddings from semantic tasks.

Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2015. Classifying relations by ranking with convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 626–634. Association for Computational Linguistics.

Sameer Singh, Sebastian Riedel, Brian Martin, Jiaping Zheng, and Andrew McCallum. 2013. Joint inference of entities, relations, and coreference. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, AKBC '13, pages 1–6, New York, NY, USA. ACM.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1201–1211, Stroudsburg, PA, USA. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Bishan Yang and Claire Cardie. 2013. Joint inference for fine-grained opinion extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1640–1649. Association for Computational Linguistics.