UniMelb at SemEval-2018 Task 12: Generative Implication using LSTMs, Siamese Networks and Semantic Representations with Synonym Fuzzing

Anirudh Joshi[1,2] **Timothy Baldwin**[1] **Richard O. Sinnott**[1]
Cecile Paris[2]

[1] The University of Melbourne [2] CSIRO Data61

anirudhj@student.unimelb.edu.au, tb@ldwin.net
rsinnott@unimelb.edu.au, cecile.paris@data61.csiro.au

Abstract

This paper describes a warrant classification system for SemEval 2018 Task 12, that attempts to learn semantic representations of reasons, claims and warrants. The system consists of 3 stacked LSTMs: one for the reason, one for the claim, and one shared Siamese Network for the 2 candidate warrants. Our main contribution is to force the embeddings into a shared feature space using vector operations, semantic similarity classification, Siamese networks, and multi-task learning. In doing so, we learn a form of generative implication, in encoding implication interrelationships between reasons, claims, and the associated correct and incorrect warrants. We augment the limited data in the task further by utilizing WordNet synonym "fuzzing". When applied to SemEval 2018 Task 12, our system performs well on the development data, and officially ranked 8th among 21 teams.

1 Introduction

This paper describes our system for the *Argument Reasoning Comprehension Task of SemEval 2018* (Habernal et al., 2018). The main goal of our system is to learn semantic representations of reasons, claims and warrants upon which vector operations can be applied which encode their interrelationships, whilst sharing encodings.

We train our system over the 1274 candidate reason, claim, first and second warrant examples from the edited *Room for Debate* New York Times dataset provided by the task organizers (Habernal et al., 2018), which we augment to generate a dataset that is used with a multi-task based objective function to learn semantically meaningful representations.

Our system combines these vectors to make a determination, via vector operations and semantic similarity classification, to determine which of the two warrants best fits the reason and claim representations, and best encodes the relationships between them. We replicate this for each sentence type, and mirror it for each warrant. In doing so we were able to achieve strong results on the development set, and ranked 8th overall in the competition.

2 Approach

2.1 Model Overview

As detailed in Figure 1, our system is made up of 3 stacked LSTMs, with one being essentially a Siamese network shared between the two warrants, and the other two encoding the reason and the claim. We take an average pool over the top-layer hidden representations and use them as the semantic feature vectors for each respective sentence. We take these semantic vectors and apply vector operations to them to generate embeddings for each reason, claim and warrant, which we found in practice to perform well. We then do semantic comparisons between the generated and actual encodings, marking as the same those between generated and actual encodings that include the correct warrant (i.e. correct interrelationships) and not the same for those that do not (through a logistic unit). We also do a joint loss on both warrant's logistic output with a softmax to the ground truth (same/not same), and use that as our tracking metric during training.

We tokenize each sentence using Keras (Chollet, 2018) and utilize an embedding layer based on the GloVe 300-dimensional, 840 billion token, uncased word embeddings (Pennington et al., 2014). We then push these through three stacked LSTM networks (Hochreiter and Schmidhuber, 1997) — one each for the reasons, claims and warrants —

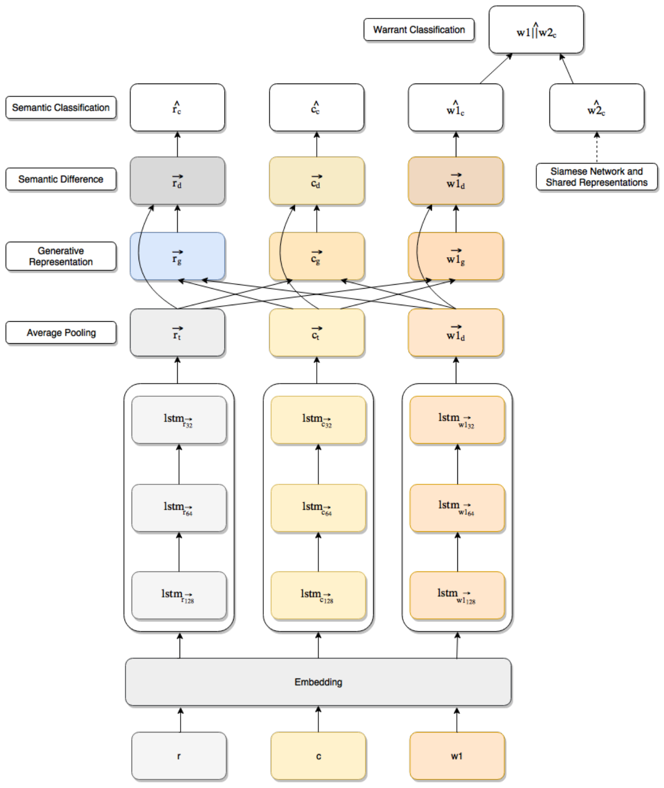

Figure 1: An overview of our model.

to generate the pooled semantic feature vectors of each phrase. Each stacked LSTM network is comprised of three layers, of 128, 64 and 32 nodes, respectively. We do this to force compression and to avoid overfitting on the dataset.

We take the final 32-dimensional embedding of each reason, claim and warrant, and apply average pooling over time to create robust compressed representations of each.

We take these representations and then perform various vector operations (addition/subtraction) between them. We use these operations as a form of implication to generate a warrant using the reason and claim representations. The generation operation does not necessarily require vector additivity (e.g. operations need not be constrained across tasks). For example the operation to learn the generative representation from a reason and a claim to a warrant can be a simple addition/subtraction, or an affine transformation, or in future scenarios it can be even more complicated dense networks.

For example given a reason and a claim, we wish to combine them together with an operation that creates a generative representation of a warrant (simple vector operations for example), and accordingly generate a representation using the operation $\vec{r}_t - \vec{c}_t = \vec{w}_g$ (where the t subscript entails the true representation, and the g the generated one). We then set up our loss function such that we compare the absolute semantic difference between the true generative encoding (generated with the correct two representations) with that of a query encoding and compare them to determine whether or not they are the are the same (i.e. the correct or incorrect warrant vs. the true generative representation).

We additionally embed both the correct and incorrect warrant for each example, and use these representations to drive multi-task learning using a Siamese network (as each reason, claim and warrant are compared to their true generative representations with only correct triplet combinations being classified as being the same). We use Siamese networks as they have been shown to work well in settings where semantic text document comparisons are involved (Mueller and Thyagarajan, 2016).

Our objective function is intended to minimize the difference between generative and actual warrant representations, and maximize the difference between generative and incorrect warrant representations. We do this via analogy to both DeepFace's semantic similarity classification (Taigman et al., 2014) and FaceNet's anchor representations (Schroff et al., 2015): for the similarity classification we use a logistic regression atop the absolute difference between the generative and the actual representations for the multiple tasks (same/not same), and we combine a joint softmax discriminative function that selects between the candidate warrants on that output, $\widehat{w1 \| w2_c}$, as our tracking objective (for early stopping and checkpointing). In doing so we aim to push the generative and true representations closer together, and push the generative and incorrect representations further apart. In this process we should then be able to perform Generative Implication, where the semantic representations across implication tasks can be shared, but where differing operations between the representations can be used to generate each other.

Due to the small data size (~1k), we explored a variety of approaches to augment the training data. One approach is to use multiple implication tasks to generate representations. Our tasks use the two warrants and reason + conclusion to generate representations of each other. We tried multiple operations including affine transforms and dense networks, as well as vector additivity constraints across tasks (i.e. the projection operations from the shared embeddings must reconcile, cf. Mikolov et al. (2013)). However we found that they led to overfitting, and hence sought to split the implication projection from the shared representations into unconstrained operations (two of which are vector additive, $\vec{r_g}$ and $\vec{c_g}$, with $\vec{w_g}$ being split off, with the split likely acting as a regularizer):

$$\begin{aligned}
\vec{r_g} &= \vec{c_t} - \vec{w_t} \\
\vec{w_g} &= \vec{r_t} - \vec{c_t} \\
\vec{c_g} &= \vec{r_t} + \vec{w_t}
\end{aligned}$$

From this, we have a multi-task output with the generative representation of correct/incorrect warrants being compared to their semantic representations, correct with the true warrant, and incorrect with the wrong warrant.

We also augmented the data by taking random combinations of single synonym fuzzed WordNet (Miller, 1995) sentences using their sampled closest synonyms (using the NLTK toolkit (Bird et al., 2009) and pywsd (Tan, 2014)). Through this method we generated a much larger dataset (~20×, depending on how many synonyms we could sample per sentence). We also swapped the warrants to double the size of the dataset. In doing so we ended up with ~234K examples. We did this to ensure a large enough dataset that included numerous subtleties.

2.2 Training

2.2.1 Optimization and Regularisation

We used heavy dropout (Srivastava et al., 2014), as we found overfitting to be an endemic problem, requiring heavy regularization. Along with the layers being smaller as we go up the stack, we applied progressively reduced dropout rates, from 0.8 at the 128 layer, to 0.6 (64) and 0.4 (32). We found this led to better generalization on the development set. We use Adam as our optimizer (Kingma and Ba, 2014), and checkpointing (with an accuracy max of the joint softmax with $\widehat{w1 \| w2_c}$) and early stopping (Caruana et al., 2001). We trained our models using 3–10 epochs, depending on when they began to overfit.

3 Results

We ranked 8th in the competition (among 21 teams) with a test accuracy of 0.577. Amongst the teams there was a clear outlier in the GIST system (which we note used transfer learning), with the remaining systems incrementally falling from ~0.6 downwards, with the random baseline at 0.527, and the example test system at 0.56 (Habernal et al., 2018).

In terms of the different systems we trained, our top three would have ranked 4th in the competition, two of which used more aggressive dropout, and one of which used a combination of features identified above (including attention between the other two sentences). The development to test set gap was wide, indicating generalization was extremely hard (similarly with others in the competition). In terms of performance, we found early on that constrained (i.e. vector additive operations) led to lower development performance, which continued on to the test set. With the remaining systems we used the operations as defined previously (unconstrained). In general we found that regularization dominates performance measures, above and beyond operations.

System	Development	Test
Higher Starting Dropout (0.9)	0.658	0.597
Lower Dropout Reduction Between LSTM Layers (0.1)	0.658	0.592
Combination	0.668	0.592
Equal Dropout on each LSTM Layer (0.5)	0.646	0.583
Extra Layer (4 Layer, 256 Base Latent LSTM)	0.652	0.579
Test System	**0.671**	**0.577**
Attention (with Double Batch Size for speed)	0.639	0.574
Test System (Constrained)	0.665	0.568
Large Batch Size (1024 minibatch)	0.617	0.538
Noise Semantic Output Layers (Gaussian)	0.627	0.527

Table 1: Accuracy over the development and test sets for various system configuration and our official submission ("Test System"), sorted by test set accuracy. "Combination" = attention, equal dropout, 4-Layer, noise semantic layer (with double batch size for speed)

4 Discussion

A few things helped performance (specifically generalization): (1) average pooling, most likely by making the overall meaning of the sentence more stable; (2) WordNet fuzzing and the resultant data augmentation; (3) progressively reduced dropout; (4) adding layers somewhat improved performance; (5) using progressively smaller LSTM layers; (6) using uncased, larger token size GloVe vectors, likely due to the larger coverage and more specific embeddings; and (7) multi-task learning.

We did not find that using dense networks for generative semantic operations worked well, as overfitting was endemic. We tried to enforce constrained vector additivity between tasks, but found that this harmed performance, and instead we took the path of multiple unconstrained tasks projected from the shared embeddings. In future, these operations should be fully-fledged generative functions, to account for the inherent complexity of the task itself. Adding noise to the embeddings (e.g. Gaussian) as a form of data augmentation also did not aid performance. Larger batches in general did speed up training, but in general harmed overall performance. We experimented with L1/L2 regularization, but found dropout to be far more robust. We attempted shared LSTM layers (between reason, claim, warrants), but again, found it be to detrimental. We tried BiLSTMs (which did not improve performance), and GRUs (which took longer to train with comparable performance). We also tried attention (e.g. a warrant on its respective reason and claim), but this too did not improve performance.

4.1 Future Directions

There are numerous future directions from this work, mostly in the integration of transfer learning, more complicated generation functions, and low resource learning. As we found overfitting to be an endemic problem across approaches, we believe that aggressive use of transfer learning of higher-level concepts from parallel domains will likely be of use. The operations for the different tasks were originally trained to be vector additive, but we found in practice that they harmed performance. Instead, complex embedding generation will almost certainly require more complex operations than the simple ones we found to work well in this work. This opens up new directions in terms of classification and text generation within argument mining, such as the generation of implicit warrants between reasons and claims, or the detection of reasoning triplets for dataset generation. We also believe aggressive use of low resource (e.g. few-shot) learning mechanisms will be beneficial in the future.

5 Conclusion

In this paper we demonstrated a system that attempts to learn a form of generative implication from sets of reasons, claims and warrants. There was a large generalization gap between the development and test test results for both of the tested systems, as well as the competition as a whole, which highlights how large an issue overfitting is for problems based on small datasets. We demonstrated our tested models' performance on both the development and test sets, with our final submission coming in 8th (among 21).

Acknowledgments

This research is based upon work supported in part by CSIRO Data61 and the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research projects Activity (IARPA), under Contract [2017-16122000002]. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. "O'Reilly Media, Inc.".

Rich Caruana, Steve Lawrence, and C Lee Giles. 2001. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In T K Leen, T G Dietterich, and V Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 402–408. MIT Press.

François et al Chollet. 2018. keras [software]. `https://github.com/keras-team/keras`. Accessed: 2018-2-17.

Ivan Habernal, Henning Wachsmuth, Iryna Gurevych, and Benno Stein. 2018. The argument reasoning comprehension task: Identification and reconstruction of implicit warrants. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, page to appear, New Orleans, USA. Association for Computational Linguistics.

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

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Tomas Mikolov, Wen-Tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751.

George A Miller. 1995. WordNet: a lexical database for english. *Commun. ACM*, 38(11):39–41.

Jonas Mueller and Aditya Thyagarajan. 2016. Siamese recurrent architectures for learning sentence similarity. In *AAAI*, pages 2786–2792. aaai.org.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543. aclweb.org.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

Nitish Srivastava, Geo Rey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Yaniv Taigman, Ming Yang, Marc'aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708. cv-foundation.org.

Liling Tan. 2014. Pywsd: Python implementations of word sense disambiguation (WSD) technologies [software]. `https://github.com/alvations/pywsd`.