Natural Language Inference (NLI)

Raj V Jain
Computer Science and Automation, Indian Institute of Science
Bengaluru, 560023
`rajv@iisc.ac.in`

Abstract

We will look at relevant work in the NLP field and the NLI problem space. We will also see my current methodology and plan for completing the project. Lastly, I will present some promising preliminary results.

1 Problem Statement

Natural Language Inference (NLI), also known as Recognizing Textual Entailment (RTE) is the task of determining the inference relation between two (short, ordered) texts: entailment, contradiction, or neutral [1]. In this task, we are given two sentences called premise and hypothesis. We should determine whether the "hypothesis" is true (entailment), false (contradiction), or undetermined (neutral) given that the "premise" is true.

The aim of the project is to implement following types of ML models for the solving NLI. First, build a simple Logistic regression classifier using TF-IDF features using scikit-learn [2]. Second, build a deep learning model specific for text like RNN/GRU/LSTM or any of the new approaches like Transformers, BERT etc. We will use Stanford Natural Language Inference (SNLI) dataset [3].

2 Literature Survey

Natural Language Processing (NLP) is a huge field with lots of real-world applications. It deals with modelling human language and can range from speech and text understanding to generation. It has applications in problem spaces like text summarising, image caption generation, document clustering etc. We will now look at the relevant previous work which motivates the methodology of the project.

2.1 Pre-processing

Pre-processing is used in ML to improve the quality of the data being fed to the model. In NLP, this step is even more important because we need to convert text into numbers and make sure the conversion preserves the right information in the right order. The first step when dealing with text is tokenising and represent sentence as a list of ordered tokens. Various tokenisation methods exist and there is no one-size-fit-all solution. Following methods are usually done post or with tokenisation.

Lower casing There are mainly two reasons to do lower casing. One, the casing of the text does not have an effect on the output label. In our task, for example, if the premise and the hypothesis are lower cased, the conclusion is unaffected. Two, this reduces the number of unique words and thus reducing the number of features. There are tasks where casing can be important. For example, in word generation, we need to give the appropriate casing to nouns. One issue with lower casing is that we end up treating nouns like Bush (US President) and bush (grass) same [4]!

Lemmatization Lemmatization aims to remove inflectional endings and return the base or dictionary form of a word, which is known as the lemma, with the use of a vocabulary and morphological analysis of words [4]. In tasks where the word's exact form is not important, lemmatization can significantly reduce noise and the number of features. Thus, this step is applicable to SNLI but may not be in case of text generation like image captioning. Stemming refers to the same process but can go to a form which is not a valid English word. This may cause issues when trying to use pre-trained word embeddings. Hence, we will be using lemmatization and not stemming, so as to give fair comparison with models which use pre-trained word embeddings.

Stop words removal Stop words are words which are very common in the field of interest which do not lend any significant discriminative value to the sentence. Thus, dropping such words can reduce number of features. Because they are domain dependent, there isn't a global stop word list which works for all tasks. Since we have non-domain specific sentences, we use a generic stop word list.

Other pre-processing steps Other steps which can be taken is to expand contractions so that the sentence becomes more plain. Another step is to handle emojis and emoticons by replacing them with equivalent single or two word phrases.

2.2 ML Models for NLP

2.2.1 Statistical methods

Initial approaches for NLP used statistical models for solving problems. One popular approach was bag-of-words in which each sentence is represented as a multiset of words contained in it. Generally, the count of words is used as a feature. Such representations lose grammar and semantic similarity between words. Additionally, they also lose the order of the words. n-grams, which are an ordered set of words generated from the text, overcome this issue to some extent. They also suffer from weighting commonly occurring words too much. This is, in-part, solved by using TF-IDF (term frequency - inverse document frequency), which weighs words inversely proportional to their overall frequency [4]. This yields a matrix in which each row corresponds to a sentence, a column to a token and the cell entry is the TF-IDF weight of that token for that sentence.

2.2.2 Neural network methods

Long short-term memory networks (LSTM) [5] revolutionised the field of NLP by overcoming the problem of vanishing gradients of vanilla RNN [6] by use of gates. Since then, multiple variants of LSTM architecture have come up. Some notable ones are Gated Recurrent Unit (GRU) [7] and LSTM with peephole connections [8]. Multiple studies [9, 10] show that these different variations do not perform significantly differently on many tasks. A very important addition to the family of recurrent networks is the Bi-directional LSTM (Bi-LSTM) [11], which uses information from the past and the future to make predictions. Constructing deep recurrent networks by stacking recurrent layers one above the other has shown phenomenal results on some tasks [12–15]. Problems which use recurrent networks in an encoder-decoder format also have shown good results in NLP tasks [16, 7]. Optimisations like attention have also improved the performance in certain tasks [17–19].

2.2.3 Embeddings

The next big storm in NLP, arguably, was word2vec [20, 21], which gave a task-independent way of learning dense representations of a word which represents its context and thus its meaning (in one way). Many such embeddings have been made since then (e.g., GloVe [22]) and are open sourced. One issue with such approaches is the inability to provide different meanings (i.e., different embeddings) based on the context. Embedding approaches like ELMo [23] and BERT [24] learn different contextual meanings of a word and have proved useful in many NLP tasks [25]. [26] found that embeddings can have a huge effect on model performance. One can either use these embeddings or can learn them based on the task that one is trying to solve. Some authors have found that learning from scratch has better model performance [27, 28]. This has inspired people to try to learn sentence embeddings, structured embeddings etc. [29].

2.3 ML Models for SNLI

In this section, we will look at ML models which have good results on SNLI dataset. The SNLI Corpus Page has a list of state-of-the-art (SOTA) models and their respective accuracy. Many of the models mentioned here use RNNs to generate a sentence embedding, one for the premise and one for the hypothesis, which captures information about the sentence and is tuned through back-propagation. This approach was used by the SNLI creators also in their paper [3]. Most approaches have used LSTM or GRU as base with bi-directionality to process words in a sentence. Variations are around the number of dimensions used, which causes changes in the number of parameters. Also popular is the attention mechanism added to these systems.

Some peculiar models are as follows. Choi et al. 2018 [30] proposed an LSTM which uses tree structure to take into account the parsed tree structure of a sentence. Sha et al. 2016 [31] modified LSTM to include the premise representation as an input to all cells of hypothesis LSTM with attention. Also, we see a trend of ensemble methods beating the sentence embedding methods in 2018-19. The current SOTA models use two different techniques. [32, 33] use a framework in which simultaneously multiple NLP tasks are performed. This helps model learn multi-modal information and hence become better. [34] adapted BERT to take into account the semantics of the language.

3 Current method description

3.1 Pre-processing

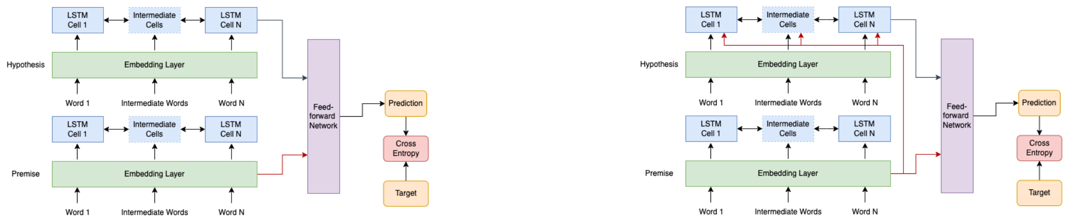
As discussed in the literature survey, I will be lower casing the sentences. For tokenisation, I will do a basic split on the parsed sentence. I will remove tokens which are purely numbers or punctuation or single letters which are meaningless. I will lemmatize tokens using the NLTK library [35] which uses part-of-speech information to lemmatize more accurately. I will use the NLTK stop word list but as a hyper-parameter. I will remove samples which do not have a valid gold label (i.e., there was no consensus in the humans who evaluated the sentence). Each of the premise and hypothesis sentences will undergo same steps so as to not add any bias.

3.2 TF-IDF with LR

I will be using the scikit-learn's TF-IDF functionality to convert the tokens generated by the pre-processing method into a TF-IDF matrix. I will use two different transformers, one for premise and one for hypothesis. This ensures that the data is kept separated. Whether to use IDF or not will be tuned as a hyper-parameter. I will use the multinomial cross-entropy loss for the Logistic regression (LR) model. The model will obtain TF-IDF vectors of premise and hypothesis as features. I will run the algorithm for various regularisation methods (e.g., none, L1 and L2) with varying regularization strengths and optimisers (e.g., SAGA [36] and L-BFGS [37]) and choose the best based on performance on a 5-fold cross-validation on the training data. Once the best hyper-parameters are chosen, I will train a model on the whole training data with these hyper-parameters and report accuracy on test data and generate train and test loss curves.

3.3 Deep learning methods

I will use same pre-processing method to ensure that the results are comparable to the previous model. I will use 2 LSTMs - one for premise and one for hypothesis 1a. This ensures that the LSTMs are able to capture the sentence without any interference. Based on the literature survey, I plan to use at least 2 layer stack and Bidirectional LSTMs. The output of the LSTMs would be fed into a neural network of at least one hidden layer. I will use multinomial cross-entropy loss for training. I plan to use embedding layers in three ways. One is to learn from scratch. Second is to use pre-trained word embeddings. I plan to use word2vec as it has less number of dimensions. Third is to initialise the embedding layer using the pre-trained embeddings but allow fine tuning. I will run the models for various combinations like using GRU / RNN instead of LSTM, number of units for recurrent network, number of stacks, number of layers, number of units in each layer, learning rates, regularisation (e.g., dropout layers, L2, L1 and none), various optimisers (e.g., Adam, L-BFGS, Stochastic Gradient Descent (SGD)) etc. and choose the best based on a 5-fold cross validation on training data. Once the best hyper-parameters are chosen, I will train the model on the whole training data with these hyper-parameters and report the accuracy on test data and generate the train and test loss curves. One additional architecture I want to try is giving the premise LSTM's output to the hypothesis LSTM 1b, inspired by Lei Sha et al. 2016 [31].



(a) Basic model (b) Premise-Hypothesis interaction model

Figure 1: Model schematics

4 Project plan

4.1 Timeline

As of this writing, I have completed the TF-IDF with LR module's cross validation task. By 17th April, I plan to complete the deep learning module's coding and finalise the LR model's outputs and graphs. By 21st April, I plan to complete training and testing the deep learning models and generate required graphs and data points. Lastly, complete the report by 24th April.

4.2 Possible issues & resolution

Data All three classes viz., "entailment", "neutral" and "contradiction" have roughly same number of samples and thus there is no class imbalance problem. I will ignore samples which do not have a gold label as they cannot provide any valuable information. Also, not all sentences will generate the same number of tokens. I will pad the sequences with zeros in the beginning and use PackedSequence concept from PyTorch [38] to inform the models that data is missing.

Processing power As the data is huge, it is possible that I will run into issues related to memory or compute power. Currently I have access to the GPU cluster of CSA and will use it when training the deep learning models. I also have online resources like Kaggle and Colab to turn to.

Word embeddings It is a sure shot scenario that not all the words present in the token list will have an embedding in the pre-trained embeddings. I will handle it by creating a special token and the embedding for this will be learnable.

Hyper-parameters Hyper-parameter tuning is a difficult task not only due to the sheer computation power required but also because we don't generally have an idea what ranges to search for each hyper-parameter. I will be using the recent papers on SNLI and use their ranges as base.

5 Preliminary results

Following are some of the results on the LR model using TF-IDF for hyper-parameter tuning (grid search method) 1. Unexpected result is that removing stop words or using IDF worsens the validation accuracy. Expected result is that regularisation has better accuracy. And L-BFGS has better accuracy compared to SAGA.

Table 1: Hyper-parameter tuning results summary on 5-fold validation data

StopWords Remove	IDF Used	Mean Acc (%)
False	False	60.36
False	True	60.13
True	False	62.67
True	True	62.58

(a) Set 1 Summary

Regularise	Optimiser	Mean Acc (%)
L2	L-BFGS	62.00
L2	SAGA	61.73
No	L-BFGS	61.98
No	SAGA	60.96

(b) Set 2 Summary

References

[1] B. MacCartney and C. D. Manning, "Modeling semantic containment and exclusion in natural language inference," in *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, 2008, pp. 521–528.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "A large annotated corpus for learning natural language inference," *arXiv preprint arXiv:1508.05326*, 2015.

[4] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.

[5] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[6] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[8] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 3. IEEE, 2000, pp. 189–194.

[9] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[10] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International conference on machine learning*. PMLR, 2015, pp. 2342–2350.

[11] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional lstm networks for improved phoneme classification and recognition," in *International conference on artificial neural networks*. Springer, 2005, pp. 799–804.

[12] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[13] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," *Advances in neural information processing systems*, vol. 26, 2013.

[14] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 2013, pp. 6645–6649.

[15] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[16] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[17] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[18] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[22] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[23] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *CoRR*, vol. abs/1802.05365, 2018. [Online]. Available: http://arxiv.org/abs/1802.05365

[24] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[25] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *CoRR*, vol. abs/2111.01243, 2021. [Online]. Available: https://arxiv.org/abs/2111.01243

[26] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep lstm-networks for sequence labeling tasks," *arXiv preprint arXiv:1707.06799*, 2017.

[27] I. Labutov and H. Lipson, "Re-embedding words," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 489–493. [Online]. Available: https://aclanthology.org/P13-2087

[28] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, Jun. 2011, pp. 142–150. [Online]. Available: https://aclanthology.org/P11-1015

[29] F. Almeida and G. Xexéo, "Word embeddings: A survey," *CoRR*, vol. abs/1901.09069, 2019. [Online]. Available: http://arxiv.org/abs/1901.09069

[30] J. Choi, K. M. Yoo, and S.-g. Lee, "Learning to compose task-specific tree structures," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[31] L. Sha, B. Chang, Z. Sui, and S. Li, "Reading and thinking: Re-read lstm unit for textual entailment recognition," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 2870–2879.

[32] X. Liu, P. He, W. Chen, and J. Gao, "Multi-task deep neural networks for natural language understanding," *arXiv preprint arXiv:1901.11504*, 2019.

[33] J. Pilault, A. Elhattami, and C. Pal, "Conditionally adaptive multi-task learning: Improving transfer learning in nlp using fewer parameters & less data," *arXiv preprint arXiv:2009.09139*, 2020.

[34] Z. Zhang, Y. Wu, H. Zhao, Z. Li, S. Zhang, X. Zhou, and X. Zhou, "Semantics-aware bert for language understanding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 9628–9635.

[35] N. Xue, "Steven bird, evan klein and edward loper. natural language processing with python. o'reilly media, inc. 2009. isbn: 978-0-596-51649-9." *Natural Language Engineering*, vol. 17, no. 3, pp. 419–424, 2011.

[36] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," *Advances in neural information processing systems*, vol. 27, 2014.

[37] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on mathematical software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf