Natural Language Inference (NLI)

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

Abstract

I will present the relevant work in the NLP field and the NLI problem space. Next, I will describe the methodology for achieving the aims of the project and the issues that arose during implementation. Then, I will present the results in various settings and discuss the reasons for them in detail. Lastly, I will discuss the shortcomings of the approaches and how to handle them as part of my future work.

1 Introduction

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. 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].

2 Problem Statement

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 solving NLI. First, build a simple Logistic regression classifier using TF-IDF features with 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].

3 Literature Survey

3.1 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 text 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 can be used in NLI but cannot be used in text generation. Stemming refers to the same process but can go to a form which is not a valid English word. This causes issues when using pre-trained word embeddings.

Stop Words Removal: Stop words are words which are very common in the field of interest and thus do not lend any significant discriminative value to the sentence. Therefore, 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.

Other Pre-processing Steps: One additional step 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 one word or two-word phrases.

3.2 ML Models for NLP

3.2.1 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 then 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, in-part, is 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.

3.2.2 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 vital 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].

Weight Initialisation: Initialisation of weights in any deep neural network is extremely important as it can affect the quality of solution and the rate of convergence and has been studied widely [20]. Random initialisation is better than constant initialisation but there are better heuristics developed. Using Xavier initialisation is better when there the activation function is sigmoid or tanh [21]. Using Orthogonal initialisation has also shown great results in some tasks [22]. As such it is recommended to use Xavier initialisation for weights used for transformation of input and Orthogonal initialisation for weights used for transformation of the hidden state in recurrent networks. Additionally it is recommended to initialise all biases except the forget gate bias (in case of LSTM) to zeros and set the forget gate bias to 1 [10].

3.2.3 Embeddings

The next big storm in NLP, arguably, was word2vec [23, 24], which gave a task-independent way of learning dense representations of a word which represents its context and thus its meaning (in one sense). Many such embeddings have been made since then (e.g., GloVe [25]) 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 [26] and BERT [27] learn contextual meanings of a word and have proved useful in many NLP tasks [28]. Reimers and

Gurevych 2017 [29] 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 [30, 31]. This has inspired people to learn sentence embeddings, structured embeddings etc. [32].

3.3 ML Models for SNLI

In this section, we will look at ML models which have performed well 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 recurrent networks to generate a sentence embedding, one for the premise and one for the hypothesis, which capture information about the sentence, and are 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 [33] proposed an LSTM which uses tree structure to take into account the parsed tree structure of a sentence. Sha et al. 2016 [34] 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. Liu et al. 2019 and Pilault et al. 2020 [35, 36] use a framework in which simultaneously multiple NLP tasks are performed. This helps model learn multi-modal information and hence become better. Zhang et al. 2020 [37] adapted BERT to take into account the semantics of the language.

4 Experiment Methodology

4.1 Exploratory Data Analysis (EDA) & Pre-processing

Class Distribution: Firstly, I checked the output class distribution and found that we are dealing with balanced class distribution (detailed plots can be see in my EDA for SNLI Kaggle notebook). This is good and saves us from the evils of imbalanced class distribution ([38] provides a detailed review of such issues). Additionally, there are samples which do not have a valid gold label. This can be the case when there was no consensus in the humans who evaluated the sentences. Such sentences provide no predictive usability and thus, I have removed such samples from training.

Pre-processing Pipeline: My pre-processing pipeline is motivated from the literature survey. As a first step, I lower cased the sentences. For tokenisation, I did a basic split on the parsed sentence. I removed the tokens which are purely numbers or punctuation or single letters which are meaningless. I chose to do lemmatization and not stemming, so as to give fair comparison with models which use pre-trained word embeddings. I used the `WordNetLemmatizer` from the NLTK library [39] which uses part-of-speech (POS) information to lemmatize more accurately. To get the POS tag for a word, I used `pos_tag` functionality of NLTK, so that no inter-library issues arise. Since we have non-domain specific sentences, we can use a generic stop word list. I used the NLTK stop word list but as a hyper-parameter. All these steps cause a reduction in the number of unique tokens presented to the model. A detailed step-by-step reduction can be seen in my EDA for SNLI Kaggle notebook. Each of the premise and hypothesis sentences will undergo same steps so as to not add any pre-processing related biases separately for the two sentences. For some sentences, the pre-processing method leads to zero tokens. One possible scenario when this happens is if all the words are stop words. Such samples are then discarded. Exactly 6 samples in the training data are such that the hypothesis was an empty string. Such samples are also discarded.

Sentence Lengths: In my EDA for SNLI Kaggle notebook, we also see that the sentence lengths are not same and the distribution changes between premises and hypotheses. This won't cause a problem for the TF-IDF method as we convert each sentence into a fixed size vector representation (row of TF-IDF matrix). But, this will need to be handled in deep learning methods. We will see how recurrent networks in PyTorch [40] handle it elegantly.

Word2Vec Presence: In my EDA for SNLI Kaggle notebook, we additionally see that a good number of tokens generated are present in the word2vec vocabulary - 80% of premise and 72% of hypothesis tokens of the train data and a whopping 94% of tokens of the test data (both premise and hypothesis).

This shows that we can trust the results of word2vec based models as they are getting sufficient information to learn from.

4.2 TF-IDF with LR

Base Methodology: I used the scikit-learn's TfidfVectorizer to convert the tokens generated by the pre-processing method into a TF-IDF sparse matrix. Two such transformers are used, one for premise and one for hypothesis. This ensures that the data is kept separated. Whether to use IDF or not is tuned as a hyper-parameter. The vectorizer provides many options including methods for pre-processing, but as I am doing all the pre-processing before itself, I instruct the function to use the tokens as it is. The vectors are combined for the two sentences to form a final feature vector which is then fed into the scikit-learn's LogisticRegression model. I have used the multinomial cross-entropy loss for training the model and balanced class weight option which weighs the classes based on their frequency of occurrence (in our case, frequencies are almost same).

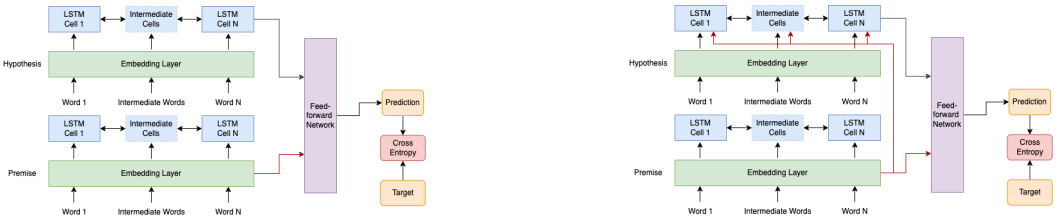
Hyper-parameter Tuning: I used scikit-learn's RandomizedSearchCV to perform hyper-parameter tuning. It has inbuilt support for cross-validation. I chose 4-fold Stratified CV (reason for selecting 4 instead of the "usual" 5 in 6.2). Here, a configuration is chosen uniformly from the given grid configuration, the model is then trained in a cross-validated manner and the average score is reported for that configuration. The hyper-parameters I have chosen to optimise and their domains are shown in table 1. The hyper-parameters related to text processing were applied to both premises and hypotheses. I used 200 iterations for the randomised search procedure to get results at varying values of the regularisation weight.

Table 1: Hyper-parameters for TF-IDF with LR model

Hyper Parameter	Explored Values
Stop Words	Removed or Not
Use IDF	Yes or No
Optimiser	L-BFGS [41], SAGA [42]
Regularisation Method	L2 or None
Regularisation Weight [L2]	loguniform(1, 1e3) [Sampling distribution]

4.3 Deep learning methods

Base Methodology: I used the same pre-processing method as mentioned above for fair comparison of results with the previous approach. My basic architecture is that I have two LSTMs - one for premise and one for hypothesis 1a, inline with most of the models studied in the literature survey section. The idea here is that the respective LSTMs are able to capture the required information of the sentence in the latent space without any interference (other than the ones introduced by back-propagation updates). The sentence representations are then concatenated and fed into a feed-forward network (FFN). FFN makes the class prediction and the whole model is trained using multinomial cross-entropy loss.



(a) Basic model

(b) Premise-Hypothesis interaction model

Figure 1: Model schematics

Recurrent Processing Units: Many libraries today provide support for ML coding in Python. I have chosen PyTorch [40] due to my familiarity with the library. PyTorch provides support for all the prominent recurrent network models viz., vanilla RNN, LSTM and GRU. I have tried all these as the base recurrent processing unit in my models and chosen the best as part of hyper-parameter tuning.

To keep the comparison fair, the recurrent processing unit type is same for premise and hypothesis. As discussed in the literature survey, I use the `init` module of PyTorch to initialise the weights of the recurrent layer and the FFN. It is worthy to mention that these are the default initialisations used in TensorFlow library [43] and hence makes my models comparable in this sense. Additionally, as seen from literature survey, bidirectional recurrent networks are used almost everywhere. In that spirit, I have also used bidirectional recurrent networks always.

Variable Sentence Length: PyTorch provides a method to deal with variable sequences by using PackedSequence in which we can provide the actual length of each sentence in the batch. This is used to reduce the number of computations and thus is efficient in handling variable length sequences.

Recurrent Unit interface FFN: When using multiple layers and bidirectional recurrent networks, each of the layer provides an output which is the representation of the sentence in some latent space. To make sure I use all the information learnt by the model regarding each sentence, I concatenate the representations of each layer of both the sentences and feed into the FFN.

Word Embeddings: Coming to representation of words for the model, I have tried three approaches of word embeddings. One is to use the word2vec embedding of a word as it is i.e., don't allow training or fine-tuning them. Another is to make them trainable i.e., use them only to initialise the embedding layer and then fine tune them. This is inspired from transfer learning [44]. Last approach is to initialise them randomly and thus learn from scratch.

Missing Words, Padding & All That!: One issue when using pre-trained embeddings is the existence of token in vocabulary of the task on which the embeddings were trained. One solution is to use a randomly initialised vector for the words not present in the embedding vocabulary. Second issue is that not all sentences are of same length. While PackedSequence solve for that at the recurrent network level, one also needs to make sure embedding layers honour that. A special index i.e., 0-index is reserved for indicating "padding". Third issue is that the vocabulary of the validation/test set might have words which the train set doesn't. Some people propose to create a vocabulary which includes all the words. In my opinion, that is wrong, since we need to treat the validation/test set as something that doesn't exist during model training. Another approach is to treat those as padding. This is again wrong, in my opinion, as padding indicates that the data is missing or irrelevant, but here, these words are informative, with the caveat that we do not have that information. A better approach is to treat them as a special index i.e., 1-index and randomly initialise it which I have followed.

FFN Architecture Selection: The FFN can be of any depth, can contain any number of nodes, have different activations etc. Virtually, the search space for optimal FFN is infinite. Size of input to this network = number of recurrent layers * number of recurrent layer units * 2 (bidirectional) * 2 (premise and hypothesis). I have chosen two types of architectures for comparison. One architecture is code named 'ip/2-act-drop-pred' (Type 1). In this, the first layer has number of nodes = size of input to the network / 2. The output is then passed through the activation function, followed by dropout layer if present. This is then passed to the output layer which has 3 nodes (as the output is 3-dimensional vector - 3 classes). The second architecture is code named 'ip/2-act-drop-ip/4-act-drop-pred' (Type 2). Similar to the above but has one more layer which has number of nodes = size of input to the network / 4. Defining architectures in this way helps make models comparable across different hyper-parameter configurations.

Hyper-parameter Tuning: The hyper-parameters I have chosen to tune and their domains are shown in table 2. As can be seen that the number of hyper-parameter combinations grow combinatorially in the number of hyper-parameters. The approach I have used to tune them is to divide the hyper-parameters into logically related sets. Try all the combinations in one set, then choose the one that works the best, fix those hyper-parameters and don't revisit the set again. The set numbers in the order in which I tried can be seen in table 2. Advantages and disadvantages of this method are discussed in 6.3. For each hyper-parameter configuration, I performed a 4-fold CV and obtained the average score for that configuration.

Interaction Model: One diametrically opposite approach to the model that I have, inspired from Lei Sha et al. 2016 [34], involves making a direct connection between premise and hypothesis 1b. The premise sentence representation is fed into the hypothesis recurrent network as part of the input. I tried this model too.

Table 2: Hyper-parameters for deep learning model

Set	Hyper-parameter	Explored Values
1	Optimiser	Adam, SGD
1	Learning Rate	1e-2, 1e-3
2	Regularisation	FFN Dropout, L2 or None
2	FFN Dropout Probability	0.2, 0.5
2	L2 Weight	0.005, 0.0005
3	Layers & Nodes of FFN	Type 1, Type 2 [explained in text]
3	Activation in FFN	Sigmoid, ReLU
4	Recurrent Layer Units	50, 200, 300
4	Number of Recurrent Layers	2, 3
5	Word Embedding	Word2Vec, Word2Vec Trainable, Trainable
6	Base Recurrent Network	Vanilla RNN, LSTM, GRU
7	Stop Words	Removed or Not
8	Recurrent Dropout Probability	0, 0.2

Prediction: The hyper-parameter configuration having highest average accuracy on the validation set was chosen as the best. This was used to train a model on the whole of training data once and then predict the output on the test data.

All code can be found in my ML Course Work GitHub project.

5 Results

I use the Plotly [45] library for all the plots due to its easy-to-understand syntax.

5.1 TF-IDF with LR

Table 3 shows the results of hyper-parameter tuning in the TF-IDF with LR model. The accuracy is calculated on the validation set and averaged across 4-folds of CV. Unexpected result is keeping stop words gave a clear cut advantage causing a jump of about 2%! I expected that IDF would cause a huge shift in accuracy but doesn't seem to be. Instead, not using IDF gave a marginal increase in accuracy! Regularisation systematically has a better accuracy as expected. And among the two optimisers, L-BFGS is better but only marginally. These results are in agreement with the preliminary results shown in the mid-term report. Figure 2 shows the effect of regularisation weight to accuracy. I expected a concave curve where having low and high weights would cause low accuracy and highest accuracy would be somewhere in the middle. Discussion on this in 6.1.

Table 3: Hyper-parameter Tuning Results Summary for TF-IDF with LR model

Stop Words	IDF Used	Mean Acc (%)
Present	False	62.81
Present	True	62.76
Removed	False	60.59
Removed	True	60.45

(a) Set 1 Summary

Regularise	Optimiser	Mean Acc (%)
L2	L-BFGS	62.14
L2	SAGA	61.71
No	L-BFGS	61.82
No	SAGA	61.03

(b) Set 2 Summary

5.2 Deep Learning Models

Figure 3 displays the results obtained for each of the hyper-parameter configuration. As discussed in the methodology section, the configurations are grouped into sets. Each set's best performing combination is mentioned which is then used in all the future models. As can be seen that the best combination obtained was when the optimiser is Adam with learning rate of 0.001, dropout probability of 0.2 is used with 2 layers and ReLU in the FFN, 3 recurrent layers of 200 units are used,

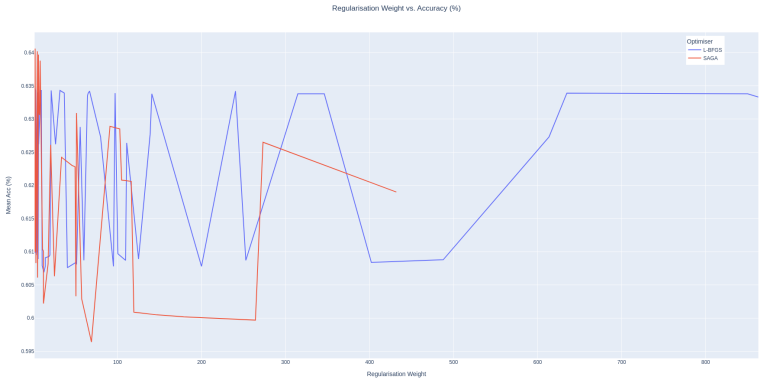


Figure 2: Effect of Regularisation Weight on Validation Accuracy for TF-IDF with LR model

and word embedding is word2vec (fixed), the base is GRU with 0.2 recurrent dropout probability. Figures 4, 5, 6, 7 & 8 show learning curves for some of the hyper-parameter settings. 4 and 5 show the typical loss and accuracy curves seen in different hyper-parameter settings. As we can see, it is a clear cut case of overfitting and hence I tried different ways to regularise the models. And since I am taking the decision only on validation accuracies, it becomes a fair comparison. The training and testing loss and accuracy curves of the final model are shown in 6 and 7. As we can see that the accuracies have increased because, in CV, the model sees only 75% of the data, here I re-train the model with all the training data. One notable curve is shown in 8 which is for L2 regularisation with 0.005 regularisation weight. It has successfully stopped the model from overfitting but has even crippled the model from learning useful information from the training data and performs worse than the LR model. All the results can be found in my ML Course Work GitHub project. The predictions generated from the final selected model are also present there. The final test accuracy is **80.49%**.

Hyper-parameter set	Config	Training Acc	Validation Acc	Best
Choosing Optim and Learning rate	Adam with 0.01	78.5	69.9	
	Adam with 0.001	99.6	71.3	Selected
	SGD with 0.01	63.9	62.5	
	SGD with 0.001	44.4	44.2	
Choosing regularisation	Dropout of 0.2	99.6	71.4	Selected
	Dropout of 0.5	99.5	70.6	
	None	99.7	71.3	
	L2 with 0.005	55.0	55.0	
	L2 with 0.0005	76.1	71.6	
Choosing layers nodes activation	One Layer with ReLU	99.6	71.4	
	One Layer with Sigmoid	99.6	69.2	
	Two Layers with Sigmoid	99.6	68.9	
	Two Layers with ReLU	99.6	71.7	Selected
Choosing recurrent hidden units and layers	2 layers of 50 units	99.6	69.6	
	3 layers of 50 units	99.3	70.0	
	2 layers of 200 units	99.6	71.7	
	3 layers of 200 units	99.6	72.0	Selected
	2 layers of 300 units	99.6	72.0	
	3 layers of 300 units	X	X	
Embeddings	Word2Vec	99.5	76.5	Selected
	Word2Vec Trainable	99.7	74.3	
	Trainable	99.6	72.0	
Base model	LSTM	99.5	76.5	
	GRU	99.2	76.8	Selected
	Vanilla RNN	93.9	70.1	
Interaction model	1 layer of 100 units	98.7	74.0	
	2 layers of 50 units	97.4	73.6	
Other Configs	Recurrent Dropout of 0.2	99.1	77.1	Selected
	Removed Stopwords	99.1	74.8	

Figure 3: Hyper-parameter Tuning Results for Deep Learning Models

Figure 4: Typical Loss Curve - Training and Validation

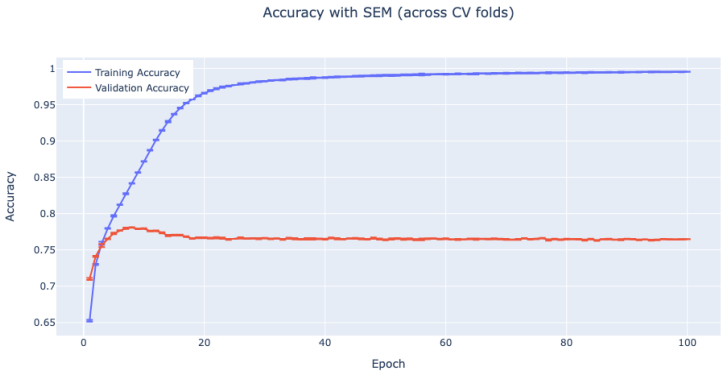


Figure 5: Typical Accuracy Curve - Training and Validation

6 Discussion

6.1 Results Discussion

TF-IDF with LR: It was surprising to see the effect of stop words removal. This could either be because of the generic list that is being used and thus would require a careful calibration of the list or that the word list has some words which are useful in this task. Another surprising result was the effect of regularisation weight. My thought process here is that if we checked much lower values of regularisation, we might the kind of graph that I expect - a concave graph.

Deep Learning Models - Hyper-parameter Tuning: We see that the models pretty quickly start to overfit with Adam. SGD's performance of 62 and 44% is just laughable - it is performing worse than the LR model! Using L2 regularisation causes the model to have training and validation accuracy very close to each other as expected, but performs worse than other models overall and hence wasn't selected. Dropout's probability also has an unexpected effect. One explanation is that dropout at this layer which already has the sentence representations may be hurting the model. This is also supported by the closeness in the accuracies of the dropout 0.2 and no regularisation configurations. Sigmoid as an activation does worse than ReLU. Using two layers does not give a significant increase in the accuracy. Similar story with different recurrent hidden units and layers. Almost all have similar accuracies. This can be explained by the training accuracy which is at 99%, basically indicating that all the training information has already been learnt and there is nothing new to be learnt, and thus adding more layers won't be useful. The 3 layers with 300 units model couldn't train due to some error at Kaggle side and since this is an exploratory project, I decided to skip that configuration. The next biggest jump comes with word2vec embeddings. It performs better than trainable word2vec which was a surprise for me. GRU outperforms LSTM by about 0.3%. GRU has significantly less parameters compared to LSTM and thus this result tells us that so many parameters are not required

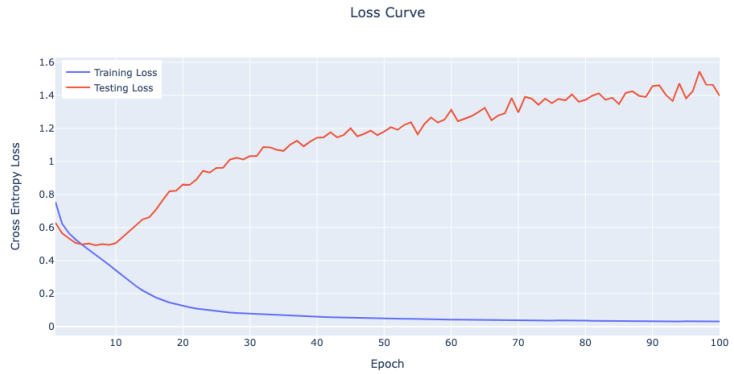


Figure 6: Loss curves of the final selected model

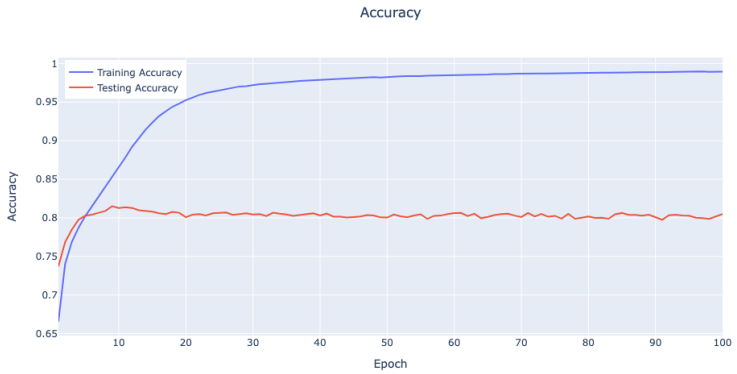


Figure 7: Accuracy curves of the final selected model

reinforcing the observation of overfitting. The interaction model doesn't outperform the normal model. The difference in this result and Sha et al. 2016 [34] could be because they had used attention with their architecture. Removing stop words does not help the model and actually hurts it, in line with the LR model's result. Adding dropout of 0.2 to the recurrent layers helps the model by reducing overfitting and is chosen to be the best.

Deep Learning Models - Learning Curves: To have a fair comparison, I have chosen to compare the models at the end of 100 epochs even though the performance of validation set reduces compared to some epochs earlier. There are multiple ways to implement stopping criteria and mine corresponds to fixed number of epochs.

6.2 Computer Power Issue & Resolution

This is unequivocally the biggest issue I faced and hence deserves a section of its own.

TF-IDF with LR model: I am fortunate enough to have a 12 Intel CPU + 16 GB RAM system assigned to me in my lab. I used this system to run all the models pertaining to the LR model. Firstly, to make scikit-learn run faster, I used the Intel®Extension for scikit-learn to accelerate scikit-learn library. Secondly, I had to make some hard choices on the search space for hyper-parameter tuning. I'd like to remind that the result on each hyper-parameter configuration was obtained by averaging results over a 4-fold CV. Optimisation algorithms like `newton-cg` use inverse Hessian which is computationally very expensive. It was taking ~25000 seconds = ~7 hours for one hyper-parameter configuration. L1 regularisation, which finds sparse weights, is not computationally efficient. Hence, I dropped these two settings from my search space. Since this is meant as a project and not as a research paper i.e., it is more about exploring the methods and trying out basic stuff first, I felt that these choices are okay to take. Each configuration of the hyper-parameters listed in 1 takes ~180 seconds = 3 minutes on average to complete. Another issue I faced the issue was with memory.

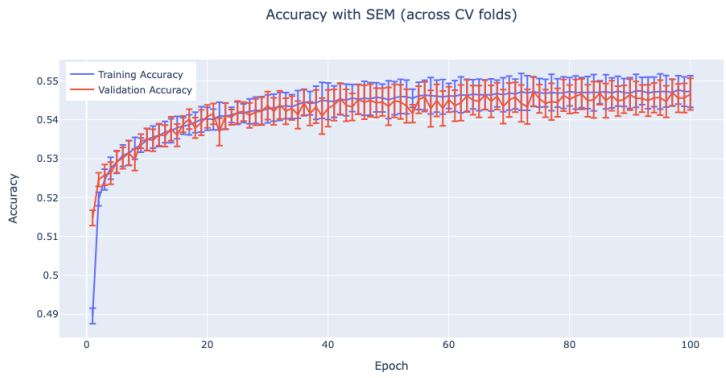


Figure 8: Accuracy Curve when L2 regularisation with weight of 0.005 is used

Within a minute of starting the script, system either hanged or the script was killed by OS due to out-of-memory (OOM). scikit-learn does not have a setting to control the amount of memory but they do provide control over number of CPUs to use by the `n_jobs` parameter. One direct relation is that when more CPUs are engaged, more memory is required. I reduced the `n_jobs` parameter from -1 (use all CPUs) to -3 (use all but 2 CPUs). This setting helped to keep the memory manageable and successful code termination. Lastly, to reduce the computation time even more, I chose not to get the train scores and not to refit the model on the best parameters in the random search method of scikit-learn.

Deep Learning Models: I first ran the basic code (low dimensional) on my 12 CPU system and each epoch took about 11 minutes. I wanted to run 100 epochs for 4-fold CV for about 22 hyper-parameter configurations and it would take me ~67 days! It was very clear that I needed to use GPUs/TPUs. I applied and obtained the access to CSA GPU cluster but it was always very busy. I turned to online platforms like Google Colab and Kaggle. Both platforms provide limited GPU and TPU resources for free users. I chose Kaggle because all the data is already present and can be imported easily. This involves even the word2vec embeddings. A year ago, TPUs were available for everyone in Kaggle, but now there is a queue system and thus you are not guaranteed to get TPUs. Hence, I chose GPUs as my processing resource. The basic model was taking about 70 seconds per epoch which comes to ~7.8 hours per hyper-parameter configuration. Kaggle has limit on how long a GPU notebook can run, which is 15 hours, and total GPU usage per user is capped at 30 hours per week. Thus, I still couldn't have completed all my configurations. I then turned to my friends and family who are fellow Kagglers to run my code. By taking help to get access to more number of hours on GPUs, I could solve my computer power issue and complete the project in time. One good thing was that GPU / RAM memory was never breached and thus I didn't need to solve for that. The reason for choosing 4-fold instead of 5-fold also came from this issue. Using 5-fold would mean a 25% increase in time. 4-fold CV has better generalisation performance measure because each fold sees only 75% of data compared to 80% in 5-fold CV. For these reasons, I chose 4-fold CV and to keep results comparable, I use 4-fold CV even in the LR model.

Other Improvements: Other things that I chose to do to reduce the time taken are as follows. I chose to use sparse matrices in LR model as sparse matrix computations are much faster. In deep learning models, I chose to work with `float` datatype instead of `double` datatype to reduce computation at GPU side. The loss of precision is negligible which justified the choice.

6.3 Other considerations

Hyper-parameter Tuning: One of the methods which is dicey is the hyper-parameter tuning approach taken in the deep learning models. Although my method has close to linear number of hyper-parameter configurations and has logical separation, it isn't the best as it misses many combinations altogether. Many frameworks like Optuna [46] have ways of pruning the search tree and efficiently find the best hyper-parameter configuration. But such systems require dedicated hardware as they can take a lot of time to come to a solution. And this can be an issue in online free compute platforms as they generally have a limit on the number of hours a single code can run.

Reproducibility: Reproducibility, in general, in science is difficult [47] and is the same case with ML. The "irreproducibility" can come from various factors like library changes, random algorithms etc. One step I have taken to ensure reproducibility is to use a seed for all the computations which use a random algorithm. This has added benefit that the models across different configurations are more comparable as they would have the same initialisation and data presentation patterns.

7 Future Work

Pre-processing: In pre-processing steps, I would like to see if expanding contractions can help. This would certainly increase the average sentence length but we might see an increase in the number of words matching the word2vec vocabulary and hence can be beneficial.

Character Models: I want to explore how character level models would perform in this task. Few advantages are that we won't have to deal with issues like out-of-vocabulary words, the input space is much more cleaner etc. We do lose out the goodness of word embeddings but there have been tasks where character level models have done reasonably well and thus, it is worth a try.

Data Augmentation: Data augmentation is a huge sub-field of ML in itself as we need to find good ways of increasing data size without tampering the actual underlying distribution of it. One approach here could be that we could interchange the hypothesis and premise and keep the same label. It may seem unreasonable in logical sense, but I believe the model will be able to learn better patterns.

Advanced Techniques: Techniques like attention, memory augmentation and symbolic AI could be tried and compared to this relatively simpler model.

Word Embeddings: Currently, I have used only word2vec. One could use GloVe embeddings or embeddings which use context e.g., ELMo or BERT etc. I predict that using context-aware embeddings should give better performance.

Other Variations: Some variations worth trying are as follows. One is to use same embedding layer for both premise and hypothesis. Here, the underlying assumption is that a word is drawn from a language not from premises and hypotheses separately. Another variation is to provide only the last layer output of recurrent networks to the FFN. This helps in reducing dimensions and may help in validating whether we need all intermediate layers or not. Lastly, we could try using batch normalisation which has shown good results in some tasks.

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] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: https://proceedings.mlr.press/v28/sutskever13.html

[21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[22] W. Hu, L. Xiao, and J. Pennington, "Provable benefit of orthogonal initialization in optimizing deep linear networks," *CoRR*, vol. abs/2001.05992, 2020. [Online]. Available: https://arxiv.org/abs/2001.05992

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

[24] 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.

[25] 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

[26] 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

[27] 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

[28] 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

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

[30] 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

[31] 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

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

[33] 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.

[34] 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.

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

[36] 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.

[37] 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.

[38] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *CoRR*, vol. abs/1106.1813, 2011. [Online]. Available: http://arxiv.org/abs/1106.1813

[39] 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.

[40] 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

[41] 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.

[42] 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.

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[44] S. Bozinovski, "Reminder of the first paper on transfer learning in neural networks, 1976," *Informatica*, vol. 44, no. 3, 2020.

[45] P. T. Inc. (2015) Collaborative data science. Montreal, QC. [Online]. Available: https://plot.ly

[46] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," *CoRR*, vol. abs/1907.10902, 2019. [Online]. Available: http://arxiv.org/abs/1907.10902

[47] M. R. Munafò, B. A. Nosek, D. V. Bishop, K. S. Button, C. D. Chambers, N. Percie du Sert, U. Simonsohn, E.-J. Wagenmakers, J. J. Ware, and J. Ioannidis, "A manifesto for reproducible science," *Nature human behaviour*, vol. 1, no. 1, pp. 1–9, 2017.