Triplet Extraction:

Rule-Based VS Machine Learning

Akhil Chaudhary

Syed Mohammad Baqir Husain

Pranav Goel

Department of Computer Science, Dalhousie University

# Abstract:

The most crucial task in any Knowledge Graph [9] creation is triplet extraction, which can be done using machine learning or a rule-based approach.

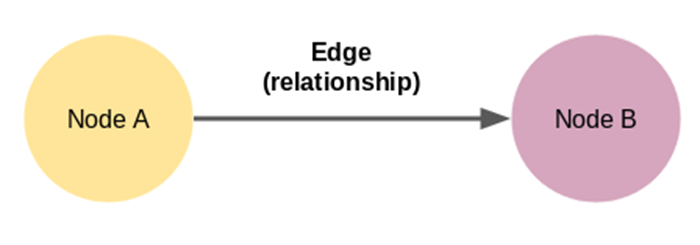
This project used synthetic data collected manually from Wikipedia, examined different deep learning architectures, and developed a reliable sequence tagger that detected entities and relations with 79% accuracy.

# Introduction:

A knowledge graph [9], also known as a semantic network, represents a network of real-world entities—i.e., objects, events, situations, or concepts—and illustrates their relationship. This information is usually stored in a graph database and visualised structure, prompting the term knowledge graph. [9]

A knowledge graph [9] comprises three main components: nodes, edges, and labels. Any object, place, or person can be a node. An edge defines the relationship between the nodes. For example, a node could be a client, like IBM, or an agency. And the connections such as a customer relationship between IBM and Ogilvy. And these three extracted compare ponents called Triplets. [9]

A Knowledge graph [9] is extracted as set of triplets i.e., <<Entity-1> <Relation> <Entity-2>> or <<Subject> <Predicate> <Subject>> from Natural Text Copra and then we can visualize them in a graph using any graph generation library of python or databases like Neo4j.



Extraction of Triplets from natural language expressions that represent the fundamental propositions asserted by a sentence (see Figuis1) is the basis of a Knowledge Graph [9]. They have been used for various tasks, such as textual entailment, question answering, and knowledge base population. However, perhaps with limited data, existing methods use semi supervised-supervisor rule-based algorithms.

For this project, we explored three deep learning architectures specifically for Sequence Tagging (Triplet Extraction), including a baseline Rule-Based Method, an RNN, a Short-Term Long Memory (LSTM) and a Bi-LSTM, to extract the triplets (predict/sequence tag word of a sentence as Relation, Entity1 and Entity2). We also examined various hyperparameters and analysed impacts on model performance. The best model detected Triplets with 79% accuracy.

# Related Work:

OPEN IE has been open problem for many years. To address this problem many attempts have been made. In these attempts earlier rule-based approaches were prominent. But since the language is dynamic and exhaustive rules is almost impossible to pin down. Other approaches have used semi supervised learning [12]. Models have been trained on distantly supervised corpus which contains a lot of noise and have falsely labelled samples [13]. Gabriel Stanovsky et al [12] constructed two datasets from existing datasets, namely, QA-SRL and QAMR and made them. Furthermore, they attempted to model this OPEN IE problem as a sequence labelling problem. However more work is required for making a supervised model that can capture wide variety of relations. Following the same line of thought we have attempted to construct a model that performs this very task. And tested our progress on manually annotated small datasets with different abstractions defined in section 4.

# Datasets:

We used the open-source data: Wiki Sentences and Supervised Open Information Extraction Paper’s Repo[8].

Wiki Sentences data has 4300 raw Wikipedia sentences. We selected 100 to create the Abstract 1 test set and another 100 chosen to develop the Abstract 2 test set and Supervised Open Information Extraction Paper’s trimmed data as the training set.

Few assumptions we considered for all the data were:

1. Sentences length was no more than 20 words.
2. Entity one was followed by relation end entity two respectively.
3. There is only one triplet present in the sentence. i.e., the preposition of the sentence can be represented by one triplet.

## Training Data:

We used the subset data provided in Supervised Open Information Extraction Paper’s Repo; [8] we selected all the sentences less than 20 words and got 900 sentences for final training. We further divided it into training and validation sets, a split of 80:20 per cent.

Training data contains multi worded entities and relations as well as long sentences.

Graphical user interface

Description automatically generated with medium confidenceThere are three columns in training data “Sentence #”, which is the sentence number, “Word”, which is the word of the sentence; and “Tag”, which represents the class of the word.

There are seven classes in the training data and 897 sentences.

Chart, bar chart

Description automatically generated

## Testing Data:

### Abstract 1 Test Set:

Here we will have very simple sentences which contains only one. <<Entity-1> <Relation> <Entity-2>> or <<Subject> <Predicate> <Subject>> triplet, and entities and relation in the triplet are one worded.

Ex:

Graphical user interface, application

Description automatically generated with medium confidenceThere are three columns in training data “Sentence #”, which is the sentence number, “Word”, which is the word of the sentence; and “Tag”, which represents the class of the word. There is a total of 7 classes in the data and 100 sentences.

Chart, bar chart

Description automatically generated

### Abstract 2 Test Set:

Here we will have very simple sentences which contains only one <<Entity-1> <Relation> <Entity-2>> or <<Subject> <Predicate> <Subject>> triplet, but with multi-word entities and relations.

Ex:

Graphical user interface, application

Description automatically generatedThere are three columns in training data “Sentence #”, which is the sentence number, “Word”, which is the word of the sentence; and “Tag”, which represents the class of the word. There is a total of 7 classes in the data and 100 sentences.

## Data Preprocessing:

For data pre-processing, we are performing the below steps:

1. Lower casing all the text.
2. Removing all special characters.
3. Stemming all the words.
4. Removing stop words.

## EDA:

1. We have class distribution as below:

Chart, bar chart

Description automatically generated

1. Sentence length:

Chart

Description automatically generated

# Method:

We explored three different deep learning architectures, a Recurrent Neural Network (RNN)[1], a Long Short-Term Memory (LSTM)[10] and a Bi-LSTM[10] to solve the problem and one baseline Rule-Based Method.

## Baseline Rule-Based Method:

### Entity Extraction:

It is not difficult to extract a single word unit from a sentence. We can easily accomplish this with parts of speech (POS) tags. Our entities would be the nouns and proper nouns.

On the other hand, POS tags are insufficient when an entity spans many words. The sentence's dependency tree must be parsed.

The fundamental concept is to go through a sentence and extract the subject and object as you encounter them. However, there are specific difficulties: an entity can span numerous words, such as "red wine," and dependency parsers only tag the individual words as subjects or objects.

The Algorithm is divided into five steps:

1. Step 1: We defined a few empty variables. The dependency tag of the preceding word in the sentence and the last w phrase itself will be stored previous to dep and before the text, respectively. The prefix and modifier will hold the text linked with the subject or object.[2]

2. We'll then cycle through all the tokens in the statement. We'll start by seeing if the ticket is a punctuation mark. If the answer is yes, we'll disregard it and go on to the next token. We'll preserve the pass in the prefix variable if it's part of a compound term (dependency tag = "compound"). A compound word is made up of many words that are connected to make a new word (for example, "football stadium" and "animal lover").[2]

We'll use this prefix whenever we encounter a subject or an object in the sentence. The modifier words, such as "good shirt," "large house," and so on, will be treated the same way. [2]

3. If the token is the topic, it will be the first entity captured in the ent1 variable. Prefix, modifier, previous to dep, and previous token text variables will be reset. [2]

4. If the token is an object, it will be captured in the ent2 variable as the second entity. Prefix, modifier, previous token dep, and previous token text will be reset. [2]

5. We'll update the previous token and its dependence tag once we've caught the subject and object in the phrase. [2]

### 5.1.2. Relation Extraction:

The task of entity extraction is just half completed. We need edges to connect the nodes (entities) to form a knowledge graph [9]. These edges represent the connections between two nodes. [2]

According to our theory, the predicate is the principal verb of a phrase.

For example, in the statement "Sixty Hollywood musicals were released in 1929," the verb is "released in," which is the predicate for the triple that results from this sentence.

For complex relations, we used Spacy Pattern Match. [2]

Text

Description automatically generated with medium confidence

The pattern defined below looks for the sentence's ROOT word or main verb. The way examines if the ROOT is followed by a preposition ('prep') or an agent word once the ROOT has been recognised. If that's the case, it'll be added to the ROOT word.

## RNN Model:

“A recurrent neural network (RNN) is one of the fundamental architectures. Many of the advanced architectures today are inspired by RNNs. The key feature of an RNN is that the network has feedback connections, unlike a traditional feedforward neural network. This feedback loop allows the RNN to model the effects of the earlier parts of the sequence on the later part of the sequence, which is an essential feature when it comes to modelling sequences[1].

A picture containing text, clock, gauge

Description automatically generated

## LSTM Model:

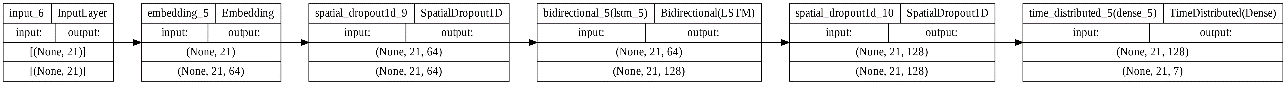
20X1 inputs were fed into Embedding Layer, followed by a spatial dropout layer, an LSTM layer, a second LSTM layer [10], and a time distributed layer before the final output dense layer.[10]

Diagram

Description automatically generated

## Bi-LSTM Model:

20X1 inputs were fed into Embedding Layer, followed by a spatial dropout layer, a Bi LSTM layer, followed by a spatial dropout layer, and a time distributed layer before the final output dense layer.



# Experiments and Results:

## Baseline Experiments:

For the baseline model, we used Spacy and NLTK library for calculating “Source”, “Relation”, and “Target” from each sentence. Rules are defined above in section. 5.1, Baseline Method.

With preprocessing of sentences, we started by removing all the stop words, stemmed the words, lowercased everything, and drew all special characters. After this, Accuracy on the testing set, we were getting “Source Accuracy” as 54%, “Target Accuracy” as 56% and “Relation Accuracy” as 42%, ”Full Match Accuracy” as 23% for the Abstract 1 Test Set. For Abstract 2 Test Set, we were getting accuracies as “Source Accuracy” as 64%, “Target Accuracy” as 70% and “, Relation Accuracy” as 69%,” Full Match Accuracy” as 60%.

## RNN Experiments:

The test size of the data read from the CSV file is changed to account for the excellent training accuracy of the model. The test size should be adjusted so that model doesn’t underfit (due to small training data) and overfit (due to more extensive training data). Various values of test size were used, as shown below. 0.5 was selected as it gave a higher accuracy when the model was tested with test data.

Test size 0.6 Accuracy = 0.334 Loss=3.416

Test size 0.7 Accuracy = 0.331 Loss=4.415

Test size 0.3 Accuracy = 0.2835 Loss=3.66

Test size 0.5 Accuracy = 0.346 Loss=3.711

For the RNN model, I tested by using an RNN layer with 80 and then with 128 units, as well as a dropout layer (p=0.7) and a fully connected layer with 80 and 128 units. It got an accuracy of 43.5% for 80 units, while it was 47.9% for 128 units as shown in the figures below.

The hyperparameters changed are the number of input and output units of the RNN model. Seeing the results further, the batch size and the epochs in the model are adjusted to see the results.

Graphical user interface, text, application

Description automatically generated

A picture containing application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Fig. 2 Changing output\_dims to 80**

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Graphical user interface, application

Description automatically generated with medium confidenceA picture containing text

Description automatically generated

**Fig. 3 Changing output\_dims to 128**

The RNN model is further improved by changing the units in the RNN layer to 256. Also, a distributed Dense Layer is added to the RNN model to make it more robust. I also changed the batch size to 64 and the number of epochs to 4 when the model tried to fit the data. This helped improve the training accuracy to 87 per cent, while the testing accuracy remained at 50 per cent.

Text

Description automatically generated

Text

Description automatically generated

Chart, line chart

Description automatically generated

Graphical user interface, application

Description automatically generated

**Fig 4. Test size =0.5 epoch= 4 and batch size=64**

Finally, the model was tested by adding the RNN layer and removing the Dense Layer, as shown below. Also, the epochs are changed to 3 and the batch size to 32. The resulting training accuracy reaches above 70, while the testing accuracy is 49, as shown below. This model is chosen as the accuracy of training is not very high. A higher training accuracy can lead to overfitting; hence this model is chosen.

Graphical user interface, text, application

Description automatically generated

Text, letter

Description automatically generated

Chart, line chart

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

**Fig 5. Model having additional RNN layer and removing the Dense Layer.**

## LSTM Experiments:

**Model: 1.** For the LSTM model [3], we started by taking 5 samples from data and tried to overfit the model; for this model, we began by creating manual embedding by taking the embedding layer with input dims as a total number of unique words in the train data, output dim as 256 and sentence length as 14 words which are the max length of sentence in our sample. We took an LSTM layer [4] of 256 units. We made the return sequence parameter to True [5] so that it will return the full input sequence predictions instead of last word predictions, and we can compare it with all the tags of true predicts. There were, in total, 535,532 trainable parameters in this model, and we ran it for 100 epochs. It got an accuracy of 82% for training and 42% for validation, which indicates that our model overfits and lies in a high bias [7] region, and we need to change our strategy.

**Model:2.** To further confirm the high bias [7] issue, we ran the above model by doubling the LSTM Layers [10] and adding one more with 256 units and passing the entire dataset to it with a modified Embedding Layer as on the full dataset after preprocessing the max length of sentence is 20 words. We were getting an accuracy of 57% after 30 epochs and validation accuracy at 55%, which confirmed that the model is overfitting.

**Model:3.** Further, as the above model is clearly in high bias [7], so instead of comparing the complete sequence in the cross-entropy loss, we added a Time Distributed Layer [10] at the end of the model and in the Time Distributed Layer, we said a Dense output layer with seven hidden units which is equal to no of tags/ classes which we are trying to predict using this model and then used Adam optimiser The structure of this model looks like the below,

Table

Description automatically generated

Here we have Input Layer with 20 words, then Embedding Layer, which has an input of 20 output dims of length 20 for all the input words, then we have Spatial Dropout 1D of 0.5, then LSTM Layer [10] with 100 units with return sequence as True and recurrent dropout of 0.5 and the Time Distributed Layer with and Dense Layer inside it with hidden units equal to several classes which are seven and using activation “SoftMax”.

After trying multiple batches, the model performed best with batch size 32, This gives Training Accuracy of 90% and Validation Accuracy of 75%, which creates the model suffers from a high variance issue [7] and can be resolved by adding more data or features.

Accuracy on the testing set, we were getting “Source Accuracy” as 68%, “Target Accuracy” as 10% and “Relation Accuracy” as 9%, and “Full Match Accuracy” as 8%, “Tag Wise Accuracy” is 24% for the Abstract 1 Test Set. For Abstract 2 Test Set, we were getting accuracies as “Source Accuracy” as 74%, “Target Accuracy” as 38% and “Relation Accuracy” as 14%,” Full Match Accuracy” as 23%, “Tag Wise Accuracy” is 27%

Below is the AUC curve for this model for all class predictions on the test set Abstract 1 and Abstract 2, respectively. Some classes are not present in AUC because these tags are not present in this test set.

Chart

Description automatically generatedChart

Description automatically generated

Model:4. As the above model is not very good and suffers from high variance [7], we had two choices to improve our model further: we add more data, which is not possible due to time constraints, and the second is to add more features. So, we tried to add more features by increasing the dictionary size, which we are feeding into the embedding layer; we added all words from the Wiki Sentences dataset, which is based on our training data and used the same model in step 3 with embedding layer output dim as 40 for each word. Now the new model has around 1 million params for training. Still, the output of this activity is not very good as we were getting negligible improvements on this model compared to the model in step 3.

Model:5. So, to further improve our model, we tried to use GLOVE Embeddings [11] with our model instead of training our own embedding to add better and more features.

We imported GLOVE Embedding [11] and extracted the features for all the words in our dictionary from GLOVE embeddings [11] and created an Embedding Matrix to feed into our models Embedding Layer.

For this model, our input layer had a length of 20 words which is the max length of sentence in training data; then, we added an Embedding Layer with input dims equal to a total number of words in the dictionary plus one for padding, output covers as 100 which is equal to the dimension of GLOVE Embedding [11] output for each word. We passed weights as the Embedding Matrix we created and input length of embedding as the max length of sentence and made trainable False. Then we added Spatial Dropout 1D of 0.5, then LSTM layer [10] with 100 units, return sequence as True and recurrent dropout as 0.5. We added another LSTM layer with 50 units, return sequence as True and recurrent dropout as 0.5. Then we added a Time Distributed Layer with Dense Layer inside it with hidden units equal to a number of classes and activation function as SoftMax for the final layer.

We used optimisers as Adam and Sparse Cross-Entropy in all the models and trained this model for 120 epochs.

For this model, we are getting a Training Accuracy of 80% and a Validation Accuracy of 79.2 %, which is our best performance so far.

For this model, we also trained it with balanced class weights, and we are getting a Training Accuracy of 71% and a Test Accuracy of 71%.

Accuracy on the testing set, we were getting “Source Accuracy” as 67%, “Target Accuracy” as 13% and “Relation Accuracy” as 11%, and “Full Match Accuracy” as 10%, “Tag Wise Accuracy” is 25% for the Abstract 1 Test Set. For Abstract 2 Test Set, we were getting accuracies as “Source Accuracy” as 73%, “Target Accuracy” as 50% and “Relation Accuracy” as 15%, “Full Match Accuracy” as 26%, “Tag Wise Accuracy” is 31%

But for the same model trained with balanced class weights, the Accuracy on the testing set, we were getting “Source Accuracy” as 97%, “Target Accuracy” as 25% and “Relation Accuracy” as 22%, and “Full Match Accuracy” as 3%, “Tag Wise Accuracy” is 28% for the Abstract 1 Test Set. For Abstract 2 Test Set, we were getting accuracies as “Source Accuracy” as 97%, “Target Accuracy” as 67% and “Relation Accuracy” as 45%, “Full Match Accuracy” as 79%, “Tag Wise Accuracy” is 26%

Below is the AUC curve for this model for all class predictions on the test set Abstract 1 and Abstract 2, respectively.

Some classes are not present in AUC because these tags are not present in this test set.

Chart, line chart

Description automatically generatedChart

Description automatically generated

Model:6. Our Training Data had some class imbalance to counter that we trained one more model with modified class weights.

Chart, bar chart

Description automatically generated

First, we defined our model, which looks like the image below,

This model had an Embedding layer with input 20 and output dim 20, spatial dropout of 0.5, 2 LSTM layers [10] with 100 and 50 units respectively with recurrent dropouts of 0.5 inside both of them, then Time Distributed Output Layer with Dense layer of units equal to several classes which are 7.

For balancing the classes, we class weights for each category by balancing out each class based on count in training and then fed it to the model fit function of TensorFlow.

After training the model for 50 epochs, we got a Training Accuracy of 83% and a Validation Accuracy of 69 %, which is worse than both above models in step 3 and step 5.

Accuracy on the testing set, we were getting “Source Accuracy” as 89%, “Target Accuracy” as 16% and “Relation Accuracy” as 27%, and “Full Match Accuracy” as 14%, “Tag Wise Accuracy” is 34% for the Abstract 1 Test Set. For Abstract 2 Test Set, we were getting accuracies as “Source Accuracy” as 92%, “Target Accuracy” as 43% and “Relation Accuracy” as 35%, “Full Match Accuracy” is 48%, “Tag Wise Accuracy” is 35%

Below is the AUC curve for this model for all class predictions on the test set Abstract 1 and Abstract 2, respectively.

Some classes are not present in AUC because these tags are not present in this test set.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Conclusion for LSTM Experiments:

1. Best Model in Training and Testing Accuracy:

Model 5 (Training = 80, Testing=79) > Model 3 (Training = 90, Testing=75) > Model 6 (Training = 83, Testing=69)

1. Best Model in Abstract 1 and Abstract 2 Test Sets:
   1. Abstract 1:

Model 5 (Full Match Accuracy = 31%) > Model 5 (Full Match Accuracy =14%) > Model 3(Full Match Accuracy =10%)

* 1. Abstract 2:

Model 5 (Full Match Accuracy = 79%) > Model 5 (Full Match Accuracy =48%) > Model 3(Full Match Accuracy =23%)

Overall, Model 5 is the best performing model, as it’s able to extract triplets with partial match from 79 sentences out of 100 of the test set.

A picture containing graphical user interface

Description automatically generated

## Bi-LSTM Experiments:

Diagram

Description automatically generated

**Understanding the Data**

Before constructing the model, it is necessary to understand the data and correspondingly perform preprocessing the data. Key observations pertaining to EDA and preprocessing were as follows (The performance of the final model depended on these findings)

1. Class distribution:

The distribution of class was highly imbalanced. The “o” tags specifically had close to 7000 examples.

1. Stop words removal:

When removing the stop words, important tags for entity were lost. For Example, the entity name “The United States of America” contains stop words (e.g., “the”) but we cannot remove them to correctly extract this entity.

1. Special Characters:

Special characters like full stop and comma were part of the sampled labelled as “o”.

Once the preprocessing was performed the distribution of the tags changed from figure A to figure B. (Please note the scale of Y axis)

Chart, bar chart

Description automatically generatedChart, bar chart

Description automatically generated

**Model Construction:**

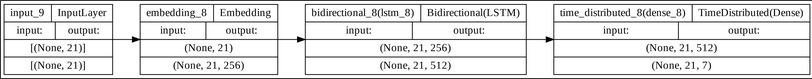
Choosing a recurrent neural network for sequence labelling was an obvious choice given its ability to unfold over time, incorporate context in understanding textual data and our available resource for training the data..

Let’s see in progression how and why model was constructed and improved.

Initially the model consisted of the following layers:

Input Layer 🡺 Embedding Layer 🡺 LSTM Layer 🡺 Time Distributed dense Layer 🡺 SoftMax Layer

This model is expected to produce a probabilistic distribution over all the classes for each word. So, to compare the predicted probability distribution to the true target probability distribution we used the loss: cross entropy.

With Model structure that looked like: 

Within 50 epochs we were getting the training accuracy of 87% and validation accuracy as 63%. On further tweaking several hyperparameters such as batch size, train test split, the size of embedding layer, the no of LSTM layers and hidden units in those layers etc. The training accuracy of 90% and validation accuracy as 73%.

Now we addressed the issue of overfitting. We used **recurrent dropouts** and **spatial dropouts**. This led to the increase of validation accuracy to 79%.

Graphical user interface, chart, histogram

Description automatically generatedOn plotting our graph plot, we observed something interesting:

* **Overfitting Persisted**

That the model was learning some parameters that were relevant for the training set but not great for generalization. This resulted in overfitting.

* **Accuracy Constant, Loss Increasing**

On investigating we discovered that the model’s bad predictions keep getting worse (e.g., a cat image whose prediction was 0.2 becomes 0.1). Due to this the loss increased whereas the accuracy remained the same as the prediction was already wrong. Furthermore, we observed that the model was predicting most of the words with “o” class. In other words, it was **overfitting a class** with large representation in the dataset.

To solve this problem, we employed some preprocessing to improve our class imbalance. Furthermore, we modified the loss function using **class weights**.

n\_samples / (n\_classes \* np.bincount(y))

Using the above formula, we found out how class weights. Based on the said weights, the loss function penalized the model’s bad predictions. Now the problem of overfitting a certain class due to class imbalance was reduced if not removed.

Chart

Description automatically generatedDespite all the efforts the difference between the training-validation accuracy and loss was huge. We were getting the accuracy of training to be above 90% and test accuracy around 78%. To address this, we employed **early** **stoppage** and reduce the parameters of our model from to 256,903 from 1,814,023. The early stoppage based on minimum loss and maximum accuracy. The results were as follows: The best model extracted from this training had training accuracy being 87% and the test accuracy being 73% and the loss being 0.4 and 0.8 respectively.

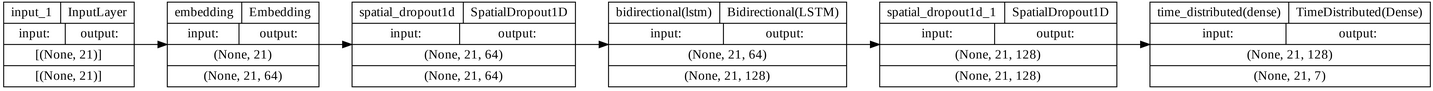
Even after the following steps:

1. Different Modeling methods
2. Targeted preprocessing
3. Employing Class weights
4. Tuning hyperparameters
5. Early stoppages
6. Dropouts
7. Etc.

...the model is still overfitting. To further improve the performance of the model we need to perform two tasks

1. Increase the dataset so that the variance due to smaller dataset can be removed
2. Increase the number of features. Using existing tools, it is easy to figure out the class of words like noun or verb etc. Since the entities are in the noun phrases and relationship usually belong to verb phrases.
3. Also a form of transfer learning, as employed with the LSTM, pretrained embedding like glove and word2vec could improve our model by a lot.

**Further testing:** (best performing Bi-LSTM shown below)

****

We manually extracted data from Wikipedia and made our custom dataset to see If the model can figure out entities and relations from a text that comes from a different source consisting of different types of relations which model might not have seen before. In this abstracted data as we explained before maintained two abstractions. The first abstraction has simpler sentences with single worded entities and relations and the second contained more complex multiword entities and relations.

* **Abstract 1**: Full Match accuracy: **19%**
  + **Source accuracy** (Accuracy of tagging entity 1): 79%
  + **Relation accuracy** (Accuracy of tagging relation): 19%
  + **Target Accuracy** (Accuracy of tagging entity 2): 10%
* **Abstract 2**: Full Match accuracy: **40%**
  + **Source accuracy** (Accuracy of tagging entity 1): 88%
  + **Relation accuracy** (Accuracy of tagging relation): 35%
  + **Target Accuracy** (Accuracy of tagging entity 2): 26%

# Conclusion:

We explored 4 deep learning architectures, a Recurrent Neural Network (RNN), a Long Short-Term Memory (LSTM) and a Bi-LSTM and examined them with different hyperparameters.

For models without adding external features, Bi-LSTM [10] with balanced class weights got the best result of 40% accuracy for Abstract 2 Test set compared to baseline rule-based 60, for Abstract 1 Test set LSTM [10] with balanced class got best results of 20% compared to baseline rule-based 40.

We added external features (Glove Embedding) to one of the LSTM models [10] with 2 LSTM layers [10] of 100 and 50 respectively and recurrent dropout of 0.5 on both layers and the final layer is a Time Distributed layer with a Dense layer in it with hidden units equal to the number of classes and SoftMax activation and balancing the class weights, and it outperformed all other trained models as well as the baseline with an accuracy of 79% on Abstract 2 and 41 on Abstract 1 Test sets. Which is considerably higher than baseline if we consider the size of our training data which is just 900 samples.

We discussed that the reason that LSTM with GLOVE [11] and Bi-LSTM stands out among the four architectures is that the prediction task (sequence tagging) which requires the model to learn long references, complex relationships and large vocabulary which can only be achieved by having good embeddings or model architecture which can handle long sequences. LSTM does a good job of extracting features from the sentences.

With such great accuracy on our very small dataset, it's very promising and can be further extended for Triplet Extraction tasks.

Moreover, the experiments and comparisons done in this project can also be helpful in similar sequence tagging.

# References:

1. https://www.sciencedirect.com/topics/computer-science/recurrent-neural-network#:~:text=A%20recurrent%20neural%20network%20(RNN)%20is%20a%20special%20kind%20of,next%20word%20of%20a%20sentence.
2. [Knowledge Graph & NLP Tutorial-(BERT,spaCy,NLTK) | Kaggle](https://www.kaggle.com/pavansanagapati/knowledge-graph-nlp-tutorial-bert-spacy-nltk)
3. [named-entity-recognition-template/ner.ipynb at master · floydhub/named-entity-recognition-template · GitHub](https://github.com/floydhub/named-entity-recognition-template/blob/master/ner.ipynb)
4. [GitHub - BeiqiZh/Named-Entity-Recognition: Named Entity Recognition (NER) using LSTMs with Keras](https://github.com/BeiqiZh/Named-Entity-Recognition)
5. [tensorflow - why set return\_sequences=True and stateful=True for tf.keras.layers.LSTM? - Stack Overflow](https://stackoverflow.com/questions/55296013/why-set-return-sequences-true-and-stateful-true-for-tf-keras-layers-lstm)
6. [Text classification with an RNN  |  TensorFlow](https://www.tensorflow.org/text/tutorials/text_classification_rnn)
7. <https://www.coursera.org/lecture/machine-learning/diagnosing-bias-vs-variance-yCAup>
8. <https://github.com/gabrielStanovsky/supervised-oie>
9. [What is a Knowledge Graph? | IBM](https://www.ibm.com/cloud/learn/knowledge-graph#:~:text=A%20knowledge%20graph%2C%20also%20known,the%20term%20knowledge%20%E2%80%9Cgraph.%E2%80%9D)
10. <https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM>
11. <https://nlp.stanford.edu/projects/glove/>
12. Gabriel Stanovsky, Julian Michael, Luke Zettlemoyer, and Ido Dagan. 2018. [Supervised Open Information Extraction](https://aclanthology.org/N18-1081). 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 885–895, New Orleans, Louisiana. Association for Computational Linguistics.
13. Tapas Nayak, Navonil Majumder, Pawan Goyal, and Soujanya Poria. 2021. Deep neural approaches to relation triplets extraction: A comprehensive survey. Cognitive Computing

# Contribution:

* Akhil Chaudhary:
* Written the baseline model code.
* Written code and generated logic to calculate the accuracies of different models on Abstract 1 and Abstract 2 test set such as Total Match, Total Partial Match, Source Accuracy, Target Accuracy, Relation Accuracy, etc.
* Written code for LSTM and its experiments section 6.3.
* Written data preprocessing pipeline.
* Written code to generate AUC curves, confusion matrix, etc.
* Syed Mohammad Baqir Husain
  + Problem statement formulation
* Written code for LSTM and its experiments section 6.4.
* Implemented code and communicated insights to handle imbalanced class weights and recomputed the new balanced weights.
* Written code for constructing dataset.
* Improved upon data preprocessing pipeline.
* Pranav Goel
  + Constructed Abstract 1 and abstract 2 dataset.
  + Written RNN code and conducted experiments in section 6.2.