# Legal Text Generation using Recurrent Neural Networks

Dushyant Pathak
DA-IICT
Gandhinagar Gujarat India
201701062@daiict.ac.in

Prasenjit Majumder
DA-IICT
Gandhinagar Gujarat India
p_majumder@daiict.ac.in

## ABSTRACT

This work is an attempt to use Recurrent Neural Networks to automatically generate legal arguments. The reference has been taken from the work on *Automatic generation of Bilateral Investment Treaties* [1]. We have used models of Vanilla RNN, GRU and LSTM(Word Level as well as Char Level), to compare the performance of the models. We have used a set of Legal Trade Agreements [6], split in 80:20 train-test, to train the model. The evaluation has been done using metrics of Accuracy, Jaccard Distance, Perplexity and BERTscore. We observe the relative performance of the models, with different expectations on word formation, logical structure, premise establishment. We are finally successful in generating legal text with a logical structure and word formation. We finally suggest some further steps that may be taken in order to improve upon the performance.

Keywords: [ *RNN, LSTM, GRU, Legal-Text, Cross-Entropy, Keras, Tensorflow, PyTorch, Vanilla RNN* ]

## INTRODUCTION

Recurrent Neural Networks are preferred over Feed forward NNs, due to ability to retain state, which makes them recall the context of a word with respect to the previous word, and hence, commonly used in Natural Language Generation. Legal Text, while having the same basic grammatical and word level rules as natural language, differs on sentence structure; Legal clauses, usually more complex than Natural Language sentences, also have a premise, a reference to a previous argument or clause, that they refer to, build upon, or challenge. Hence, the task in Legal Text Generation, is not only to make structurally meaningful sentences, but also those that can make accurate relations and references.

## DATASET

The ToTA Dataset used to train the model has been obtained by the courtesy of Dr W. Alschner [4]. The Text of Trade Agreements(ToTA), is a set of 400 xml documents of Bilateral and Multilateral trade agreements between various nations, through 1950 to 2000. Each agreement comprises Meta-information regarding the countries involved, the date of agreement and the source URLs. The main body is broken down into chapters, relating to various areas that the agreement honors. The chapters are comprised of articles, each referring to a particular point of legal argument.

The articles have been split into 80:20 train-test. The text from the articles was scraped into one single text file each for train and test.

## MODEL

We have used Gated Recurrent Unit and Bi-directional LSTM models for the implementation of RNN. GRUs have two gates as opposed to three in LSTMs. This makes GRUs easier to train, with an accuracy almost at par with LSTM, for small datasets. However, the LSTMs, due to their additional cell state, have an advantage over larger datasets. We have attempted use of both Character Level RNN as well as Word Level RNN, owing to each of their merits: While Word Level RNNs are better at preserving the context and thus, give more accurate results, whereas Char Level models are a lot easier to train, given that their vocabulary set is a few hundred characters, as opposed to some hundred thousand words in a Word Level model.

## IMPLEMENTATION**

We start with a Vanilla RNN implementation, to set a threshold to performance. We then implement GRU and LSTM Char level models in Tensorflow-Keras. Finally, we implement a word level model in PyTorch, GRU and LSTM, to test performance of Word Level RNNs.

### 1 Vanilla RNN implemented using numpy

We have implemented a Vanilla RNN, using numpy, adapted from this implementation in Torch [5]. While Vanilla RNNs do have a hidden state to maintain context, an advantage over Feed forward Networks, the structure tends to put higher relevance on the more recent information, thus losing out on long term context. This is quite lower than the accuracy of an LSTM/GRU model. Thus, the Vanilla

** All code and data is available on https://github.com/dkp1903/TextGenRNN

RNN sets a lower threshold for performance of our work. We obtain a best perplexity of `725457641.806479`

## 2 Using Tensorflow Keras - Char Level RNN, evaluated on Perplexity, BERTScore, Accuracy and Jaccard distance parameters: Bi - LSTM Model

We store all characters that are part of the train set into a vocabulary. We then vectorize the text using a char to index mapping, to be able to use them as embeddings. We then break it down into sets of sequences. We duplicate the sequences and shift them by one place, mapping them such that each letter is mapped to the letter that follows it in the train data. For each character the model looks up the embedding, runs the GRU one timestep with the embedding as input, and applies the dense layer to generate logits predicting the log-likelihood of the next character, essentially predicting the next character. While the Vanilla RNN predicts the next char only on the basis of the present one, an LSTM model can recall the previous context and thus make more accurate predictions. We use a sequential Keras function to build the model. Each new predicted char acts as the next input to the model, improving the context. This repetitive feeding back is an advantage of an RNN model. We finally calculate the cross entropy loss and Jaccard distance. We have employed Tensorboard to graphically view the analyses.

## 3 Using PyTorch - Word Level RNN

Our next attempt was to create a Word level RNN. RNN implementation word wise means a higher probability to recall the context, than a char level one. However, the vocabulary would involve all distinct words from the dataset, and thus, training is possible on a subset of the dataset. We create trigrams from the text, such that the first two words should predict the third word of the trigram. Using a single layer model with a hidden layer size of 10, we train the model following a similar pattern as in 2.

## EVALUATION

The Tensorflow-Keras models were evaluated on metrics of accuracy, perplexity, Jaccard distance and BERTscore. The LSTM model displayed an accuracy of 81.35% on the train data. This means that given a set of characters, the model is able to predict the next character correctly 81.35% of the time. Perplexity is the exponentiation of the categorical cross-entropy loss. We obtain a best value train loss of 0.5, making the perplexity around 1.8, and a test loss of about 7.8, taking the perplexity to be around 230. We measure the Jaccard distance between the vectorized representations of the output text and test text, to get a value of 0.32. BERTScore[3] computes token wise similarity between test and output text. The BERT-F1 System score for our model was 0.83.

Given the much smaller dataset that the Word Level RNN was trained on, the train loss there was found to be 5, with a perplexity of 35. The other metrics were, expectedly, higher.

We see that given the larger dataset employed in 2, LSTMs have a slightly better performance than GRU

These metrics calculate how close our generated text is, to a human created text. While this approach has widely been adapted, it is not entirely applicable in the context of legal documents, specifically our dataset, which is an unclassified mix of separate documents, evaluated against another document that may not have anything in common to the train document. For instance, one of the train documents might be a treaty between two European countries over the illegal export of cocaine, and this might be tested against a treaty between two Asian countries for spices, which hardly have any contextual similarity. The unclassified, non-annotated nature of the dataset does not allow us to separate the various categories of articles to train separately.

## CONCLUSION AND FURTHER WORK

We have thus, successfully trained a RNN model to automatically generate legal texts. We note that LSTM/GRU's due to their extensive state retention are able to generate much better results than a vanilla RNN.

The training data used here is an unclassified merged text, from a variety of sources, which is why the text generated does not pertain to be specific to an argument, and thus leads to a degradation in performance. Further attempts can be to classify the articles based on normative categories, so as to train the model on topic specific data, leading to better performance. Legal Text Generation evaluation needs a metric that can evaluate the premise relation that one part of a document has to another.

## ACKNOWLEDGEMENT

We are thankful to Dr W. Alschner for his provision of the Text of Trade Agreements dataset, which this work is based on. We are thankful to Mr Apurva Parikh, whose insight on RNN models, and evaluation metrics were used in the projects.

## REFERENCES

[1] Wolfgang Alschner and Dmitriy Skougarevskiy. 2016. **Can Robots Write Treaties? Using Recurrent Neural Networks to Draft International Investment Agreements.** In *Legal Knowledge and Information Systems: JURIX 2016: The Twenty-Ninth Annual Conference (Frontiers in Artificial Intelligence and Applications)*, Floris Bex and Serena Villata (Eds.), Vol. 294. IOS Press, 119--124.

[2] Wolfgang Alschner and Dmitriy Skougarevskiy. 2017. **Towards an automated production of legal texts using Recurrent Neural Networks**, I*CAIL '17: Proceedings of the 16th edition of the International Conference on Artificial Intelligence and Law*.

[3] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT https://arxiv.org/abs/1904.09675

[4] Andrej Karpathy. 2016. *Implementation of Vanilla RNN* https://github.com/karpathy/char-rnn.

[5] Christopher Olah

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] Wolfgang Alschner Text of Trade Agreements (ToTA) : https://github.com/mappingtreaties/tota