**DeepImpact: Impact Factor Predictor from Abstract Text**

By: Xander Krohannon, Gayathri Panangipalli, and Sharmila Selvaraj

**Introduction:**

Contributing to the library of human knowledge is a proud and noble tradition among the world’s scientists; as well as an important milestone for all aspiring researchers. In the modern era, this aspiration maintains all the classic obstacles: sufficient rigor and novelty of the work, mentor sponsorship and approval, acceptance of a journal to publish the work, and incorporating the feedback and criticism from peer evaluations, with the additional wrinkle of selecting the appropriate journal to submit to. Per the 2018 STM Report [1] there are over 33,100 peer-reviewed journals in print, each with their own specific research area and specifications. One heavily weighted metric used by journals is the impact factor, measured by the number of times an article is published within a set duration of its publish date, typically 6 months. As a result, a tool to predict a proposed article’s impact factor is desired, as a means to aid in researchers in selecting journals to submit their article to. Here we present, DeepImpact a deep learning model that takes an article’s abstract, as well as the primary author’s institutional association as features to predict the likely impact factor of that particular work.

**Problem Statement:**

Publishing scholarly works is a rite of passage for graduate students, and often a requirement for their advising faculty. In the modern world, there are a plethora of peer-reviewed journals selecting which 3 million articles will be published this year. Selecting which journal to submit a body of work to, in order to maximize the likelihood of it being published, is a daunting task with few guidelines beyond the intuition of experts, that only comes with experience. DeepImpact aims to serve as a tool to predict an article’s impact factor, thus aiding the aspiring author in determining which journals they should submit their work to.

**Data Source:**

PubMed is a repository for over 30 million biomedical articles from a wide range of journals[2]. To this end, the meta data of a random 5 percent of the articles contained in PubMed were downloaded in the XML format. This data was divided into training data and testing data, 75 percent and 25 percent respectively. An additional 1 percent of the articles were downloaded to serve as an independent validation set.

**Methodology:**

Data preprocessing was performed using an ad hoc Python script (version 3.7.6) utilizing the datetime, networkx, and xmltodict libraries. The XML files downloaded from PubMed were parsed into Python dictionaries using the parse command from the xmltodict library, resulting in each tag becoming a dictionary entry, with enclosed tags as keys, thus retaining the hierarchical structure of the data. The article’s PubMed ID, abstract, and the primary author’s school and department were extracted in the raw text format. The article’s publish date was extracted and stored as a datetime date object. Additionally, a list was composed of the PubMed ID’s of each article referenced by a particular article. This was combined with the previously extracted features and stored as a tuple in a list of all article information.

In order to get the true impact factor of each article the directed network module from networkx was used. Each article was created as a unique node within the network. Then directed edges were created originating at the citing article and pointed towards the article being cited, if the citing article was published within 6 months (180 days) of the article being cited. After all edges had been added, the number of edges facing towards a particular node, the number of other articles that cited that article within 6 months of its publication, was treated as the true impact factor of that article.

**Recurrent Neural network:**

Recurrent Neural Network (RNN) is widely used deep learning algorithm to process sequence data. The recurrent nature of RNN will enable the network to perform the same function for all the input of data. The output from each node is dependent on the previous node’s computation. Therefore, a decision has been made by considering the current input and the output. General structure of RNN includes an input layer, one or more hidden layer and an output layer. The gradients are calculated, and weights are adjusted during the training process, using a back-propagation algorithm. The weight will also be adjusted after the feedback process to accommodate for updation [3,5].

**Long Short-Term Memory (LSTM) Neural Network:**

Monitoring sequence and context information is important while processing text data. LSTM networks, a modified RNN are capable of retaining information and long-term dependencies over a period of time. They eliminate the vanishing gradient problem and similar to RNN, perform training via backpropagation. The architecture of the LSTM network contains an input gate, forget gate and an output gate. The input value is picked in the input gate to modify the memory, the forget gate identifies values to be removed and the output gate takes the processed input and memory to produce outputs ranked in terms of their importance [3].

**Gated Recurrent Unit (GRU) Neural Network:**

GRU networks are yet another commonly used model for capturing logical flow of information from a context. These networks also eliminate the problem of vanishing gradient similar to LSTM networks. The GRU architecture contains two gates, an update gate and a reset gate. These gates regulate the flow of information by deciding on what needs to be retained and forgotten. The gates in the GRU network are trained to keep information relevant to the prediction [4].

**Data Pre-processing:**

The data extracted from the XML format and derived from NetworkX, had columns corresponding to Article ID, Article Abstract, Impact factor and Author Information. The article abstracts and impact factor corresponding to each abstract was considered for training the model. The text in each abstract was processed as follows:

1. Rows having empty values for abstract information and/or impact factor was removed.
2. Duplicate abstracts were removed
3. Non-ascii characters and repeated unnecessary phrases such as ‘formula: see text’ were removed from each abstract.
4. Abstracts with words less than 25 and 50 word count were removed. Model performance was checked for each cut-off.
5. Abstracts with 0 impact factor were removed to improve the model performance at a later stage.

**Model:**

A LSTM RNN model was built with three base layers: Embedding layer, LSTM and a Dense Layer. Dropout layers were included in few models to check for the performance of the model.

Similarly, a GRU neural network was built with the same architecture and tested for performance.

Different models were built keeping in mind the maximum length allowed for an abstract with padding wherever needed. One-hot encoding of abstracts could not be performed as this step is very memory intensive and adequate resources were not available, which lead to the kernel crashing each time. As an alternative, tokenization was performed for each abstract and the resulting arrays were then padded based on the maximum allowed abstract length.

Dropout layers (multiple percentages) were added to prevent overfitting of the model. Activation function for each dense layer was chosen to be sigmoid as it performed better compared to softmax. Optimizers Adam, RMSprop and SGD were tested, with RMSprop performing well. Loss function was kept constant as Binary cross-entropy as this is a widely accepted function for processing text. Although higher batch size leads to quicker convergence, a batch size of 1000 was finalized for quick training of the model with more epochs. Multiple batch sizes were also tested.

Model was built and trained with above considerations and evaluated on a validation set. Best performing model was chosen and tested different train-test split ratios.

**Results**

**Different scenarios:**

**Scenario1**: **Model built with all abstracts (including abstracts having impact factor 0)**A screenshot of a cell phone

Description automatically generated**:**

Abstract with maximum word length was found to be 1560. Abstracts with length lesser than 1560 were padded after tokenization. Below is the table giving the number of parameters and accuracy obtained after running the model on different split ratio, embedding size, optimizers and epochs. All these models have been implemented using an LSTM RNN module.

**Observations made from Scenario 1:**

* Total number of abstracts used for model after processing: 20127 abstracts (before processing: 24566 abstracts).
* Higher accuracies (~54%) was obtained for LSTM models with train-test split ratios of 75:25.
* Embedding size and maximum length of abstract did not have an impact on the performance of the model.
* Change in batch sizes from 32 to 1000 affected the training time of the model but did not affect its performance.
* Softmax activation function at the dense layer resulted in stagnant accuracy of 53% with high loss.
* Each model was evaluated using the validation set and the prediction accuracy was ranging from 39% to ~53%.
* Root mean squared error for the validation set was observed to be 0.79.

A screenshot of a social media post

Description automatically generated A screenshot of a social media post

Description automatically generated

1b

1a

Figure 1a: Model accuracy with increasing epochs; 1b: Loss vs Epochs

**Scenario2: Model built with excluding abstracts having impact factor 0:**

As the above models were performed poorly, it was decided to include only abstracts with impact factor greater than 0. This resulted in a total of 12157 abstracts. The initial LSTM model performed better than the first scenario resulting in ~89% accuracy. Hence, this dataset was further used to test other models with different train-test split ratios. Although LSTM performed well, GRU and SimpleRNN models were also tested for the same dataset. Upon obtaining results from different trials, the below hyperparameters and architecture was maintained:

* Batch size: 1000
* Epochs: 20
* Loss: Binary cross entropy
* Optimizer: RMSprop
* Dense Layer activation: Sigmoid
* Number of units in LSTM/GRU/SimpleRNN: 100
* Dropout probability: 0.3

**Observations made from scenario 2:**

* No drastic differences observed in accuracies between different types of model.
* Model trained with abstracts having minimum length greater than 50 showed higher validation accuracy compared to train. Although the difference is very small, this might be the result of underfitting. To avoid this, further models were built considering LSTM model minimum abstract length of 25 words.
* Taking this into account, the best performing LSTM model with was tested with different train:test split ratios.
* Below are plots showing performance metrics:

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

2b

2a

Figure 1a: Model accuracy with increasing epochs; 1b: Loss vs Epochs

* Accuracy of the LSTM model did not change with increasing epochs whereas the loss decreased with every epoch.
* A screenshot of a cell phone

  Description automatically generatedRoot mean squared error for the validation set was observed to be 0.69.

**Conclusion and Discussion**

Through this project, we aspired to create a new approach for predicting the impact factor of an article that could help researchers in aiming for journals accepting articles with a specific standard. As quoted in the literature and observations from this study, the best performing model for capturing the sequence context from text data is LSTM-RNN. Root mean square error (RMSE) was noted to be higher for the model incorporating all abstracts. This indicates that data processing and cleaning must be implemented at each level to obtain better performance of the model.

Hyperparameter tuning resulted in better accuracy but, there is still scope for improvement in performance by considering more data and deeper network to capture the true dependencies.

**Appendix:** Contribution

Xander Krohannon: Formulated the idea, downloaded the data and parsed it to get relevant fields using NetworkX and xmltodict.

Gayathri Panangipalli: Fixed model performance, brought up the second scenario, tested model performance upon multiple hyperparameter tuning.

Sharmila Selvaraj: Processed the data prior model input, built models and tested model performance upon multiple hyperparameter tuning.

**References:**

1. Johnson et al.(Oct 2018). The STM Report Fifth Ed.STM-Association. https://www.stm-assoc.org/2018\_10\_04\_STM\_Report\_2018.pdf.
2. NCBI Resource Coordinators. Database resources of the National Center for Biotechnology Information. Nucleic Acids Res. 2018;46(D1):D8‐D13. doi:10.1093/nar/gkx1095.
3. Le, X.-H.; Ho, H.V.; Lee, G.; Jung, S. Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting. Water 2019, 11, 1387.Top of FormBottom of Form
4. [Simeon Kostadinov](../../../../../../Users/sharmilaselvaraj/Desktop/Simeon%20Kostadinov), <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>.
5. Nimesh Sinha, <https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47>.