Citation prediction challenge - project report

Advanced Learning for Text and Graphs - ALTeGraD

from

Clément Apavou Younes Belkada Arthur Zucker

M2 MVA Mathématiques / Vision / Apprentissage

École Normale Supérieure Paris-Saclay

February 24, 2022

Paris, France

Contents

1 Introduction

1.1 Problem definition

In this challenge, we are given a large scientific citation graph, with each node corresponding to a certain article. The dataset consists of 138 499 vertices i.e articles, with their associated abstract and list of authors. The goal is to be able to predict whether two nodes are citing each other, given all this information. In the next sections, we will try to elaborate on the various intuitions behind our approaches, and present the obtained results as well as some possible interpretations for each observations. The provided code corresponds to the code that we have used for the best model (*i.e* the right commit), the code used for the experiments can be browsed at `https://github.com/younesbelkada/altegrad_challenge`.

2 Dataset

We first extend the provided code by making sure that we do not sample pairs that are linked together when generating negative samples. This process allows to have a clean data that can be used for training and validation. After obtaining the desired pairs (positive and negative), we split the dataset into a training and validation set using a split rate of 20%

3 Features extraction

This step is crucial since we have design the best features possible in order to feed them to the network. In the following section, we will elaborate the attempted approaches to extract meaningful features for this problem. Then our result section5 will present the results and impacts of each method.

3.1 Leveraging the information from the abstracts

3.1.1 SPECTER and SciBERT: learning from textual features

Firstly, we wanted to leverage the information that can be extracted from the abstracts, and then we tried to add more contextual information related to the graph structure. Given two abstracts from two random papers, we first compute the embeddings from the abstracts and use these embeddings as an input to a simple Multi Layer Perceptron.

Beltagy et al. [1] proposed a BERT model trained on scientific literature to perform downstream tasks such as Dependency parsing and Relation Classification. Since this model was trained on a large corpus of scientific papers, we thought that it would be

able to extract meaningful information from the abstracts. We compared the obtained embeddings with a more recent method named SPECTER, introduced by Cohan et al. [2], which is a framework to learn embeddings from scientific papers for link prediction. After comparing SciBERT[1] and SPECTER[2] on the validation set, we have decided to run our experiments using SPECTER[2] only.

3.1.2 Keyword features

Each abstract can be explained through its keywords. If two abstracts share more or less the same keywords, we think that they might be a high chance that one paper cites the other. We can leverage this idea by extracting the top n keywords for each abstract using KeyBERT[3], by selecting SPECTER[2] as a feature extractor for KeyBERT. Many parameters can be tuned such as the size of n-grams for the extracted keywords. Therefore, each abstract can be associated with top n keywords, associated with their embedding obtained from SPECTER[2].

The drawback with this approach is the computational size of the network that grows exponentially with the number of keywords. We aggregated the embeddings from the keywords for each abstract by considering the mean embeddings. In addition to that, we have tried to compute the cosine similarity between each pair of keypoints given two abstracts. This information can be concatenated and used as an input on the model. We also compute the number of common keywords and the number of keywords.

3.2 Leveraging the information from the authors

We propose to leverage the information provided about the authors of each paper. We argue that a famous author may be cited several times, and people who worked together, *i.e* were authors on the same paper, are likely to cite their previous work (extending a previous method). The model will aim to learn unique embeddings (using `torch.nn.Embedding`) for each author that will be aggregated before feeding it to the final model. We deal with the problem of having potential different number of authors per abstract using two techniques: considering the first authors **only** or randomly sampling an author from the possible authors for each abstract. As this appeared to not perform that well on our validation set (see figure 1), we have decided to not investigate this path further.

3.3 Leveraging the information from the graph

3.3.1 Handcrafted features

We have extended some tricks that were provided on the code, by adding the individual degree of each node. Initially, the features were the sum and the difference of number of unique terms of the two nodes' abstracts and the absolute value of difference of number

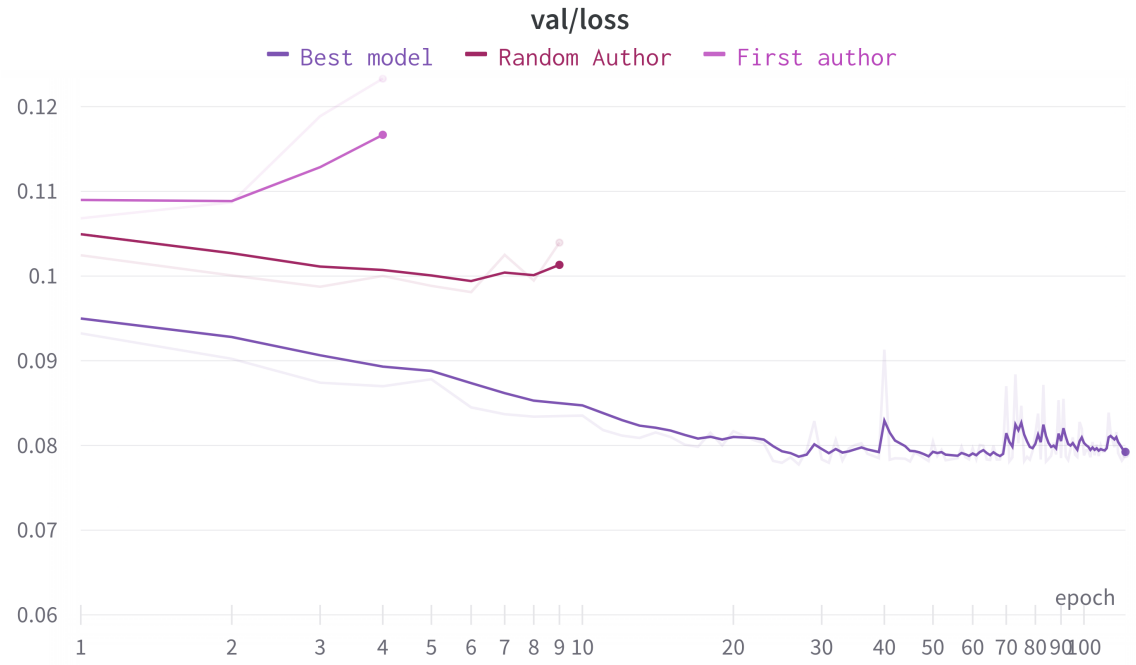

Figure 1: Comparing different authors selection strategy against our best model

of unique terms of the two nodes' abstracts. Here is the list of handcrafted features that we have tried:

- sum of degrees of each node (for each pair)
- absolute difference of degrees of each node (for each pair)
- degree of the first node
- degree of the second node

3.3.2 Mean representation of the neighbors

An idea could be to take the advantage from the neighbors nodes. One can extract the embeddings from the abstracts of the neighbors for each query node and compute the mean of those embeddings. This aggregated embedding should represent a mean representation of the neighbors for each node, therefore provide a strong and meaningful contextual information about the neighbors.

3.3.3 Other features

We also leverage the information from the graph by computing several well-known features that can be computed using the graph information.

- Jaccard index[4]
- Adamic Adar Index[4]
- Soundarajan Hopcoft index[5]
- Preferential attachment [6]
- RA index soundaraja [7]
- Sorenson Index
- Clustering index [8]

These features have been found on the link prediction module of the Networkx library `https://networkx.org/documentation/stable/reference/algorithms/link_prediction.html`.

3.4 Leveraging the information from the neighbors

In addition to the presented features we add an extra information using the neighbors which is the length of the intersection of keywords between the two input nodes. We argue that this information can be crucial since two papers that have a large number of common keywords are most likely to cite each other. On the other hand, if they do not share any keyword in common, they most likely do not cite each other.

We also compute the length of the intersection of the neighbors, which is a measure close to the *Sorenson Index*, which is why we chose it.

3.5 Representational Learning

We were thinking of trying Representational Learning strategies for graphs such as Graph Autoencoders, but due to the huge number of nodes that needs to be represented this was not feasible with the available compute power.

4 Architecture & Training details

Our final architecture is an MLP with 5 layers. You can find it's representation in 2. The input dimension only varies based on the chosen embeddings and is automatically inferred. After various ablation studies, we used a dropout coefficient of 0.6, the `GeLu` activation and the `batchnorm` layer.

To train our models, we use a learning rate scheduler to find the optimal learning rate, we train our methods using the Adam optimizer[9] for 30 epochs. We seek to obtain better

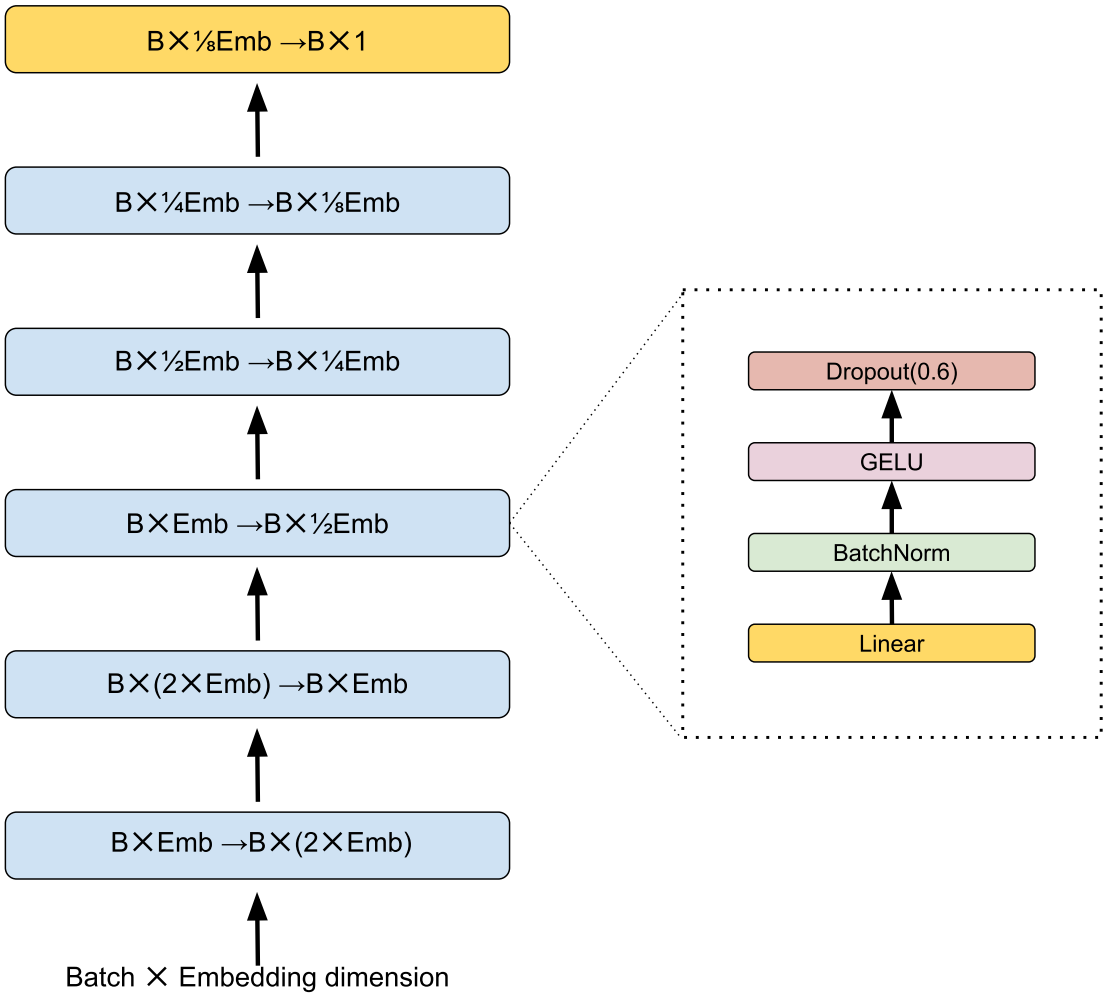

Figure 2: Final MLP architecture wich performed best

generalization at test time by applying several techniques such as weight decay, stochastic weight averaging and dropout during training. We have observed better performance at test time when applying these techniques on top of our methods.

We also used the *reduce on plateau* learning rate scheduler to adjust the learning rate when the validation loss hits a plateau.

5 Results

In this section, we will quantitatively detail the results obtained with some approaches that we have tried for the challenge. For each method, we report some relevant numbers

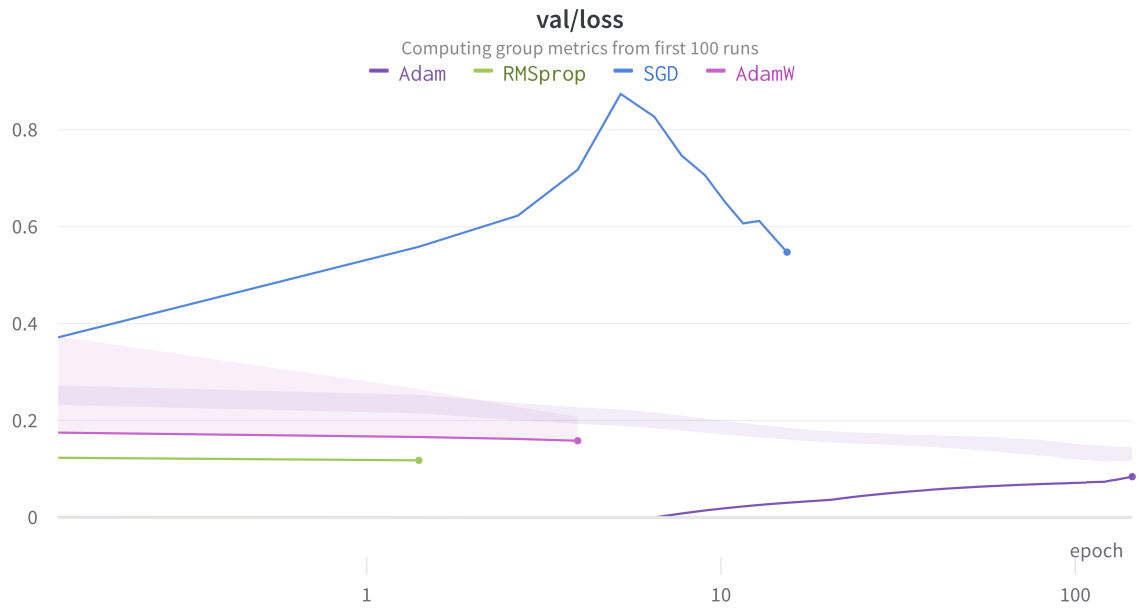

Figure 3: Min validation loss of our experiments w.r.t the optimizers

such as the resulting public BCE Loss score on the testing set obtained from the Kaggle leaderboard or the behavior of the validation curve on our validation set.

5.1 Architecture optimisation

We ran various ablation studies (more than 600 experiments) to find the perfect hyper-parameters. We compared the loss obtained using different optimizers such as *RSProp*, *Adam*, *SGD* and *AdamW*. Tha various results are shown in figure 3. Adam performed best.

We studied the impact of the dropout, and found that a dropout of 0.6 was teh best in allowing the model to generalize and not overfit.

We used a weight decay of $1e^{-}8$, which came out on top from experiments shown in figure 4.

We also studied various MLP architectures, and the one shown in 2 gave us the best results. It has around 44M parameters, depending on the embedding dimension.

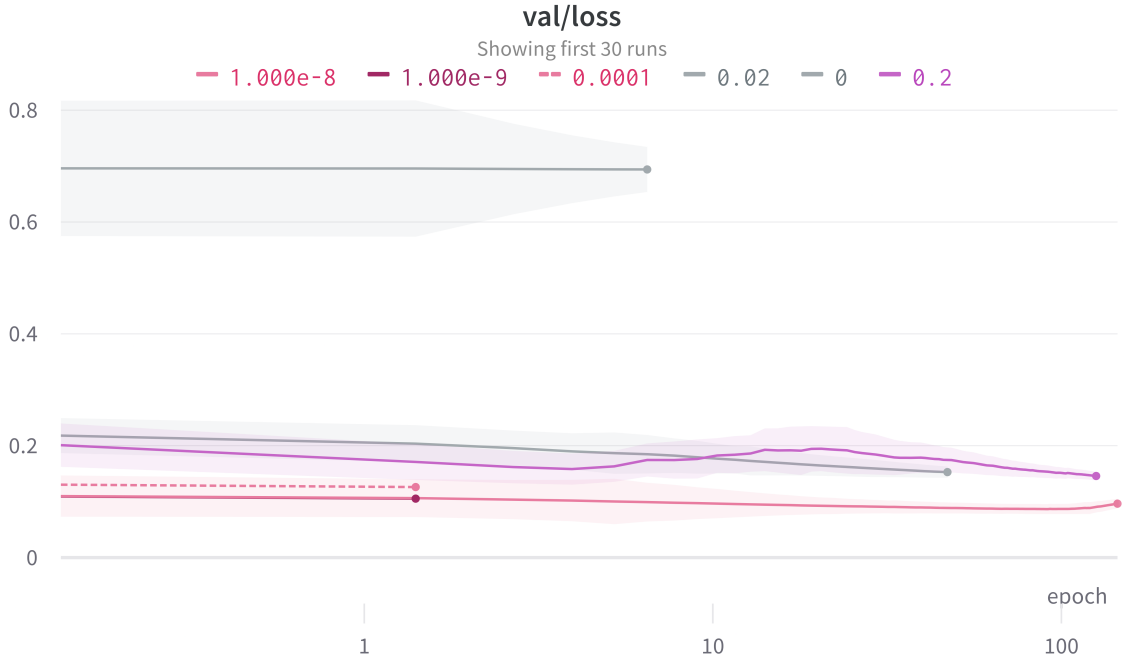

Figure 4: Median validation loss of our experiments w.r.t the weight decay

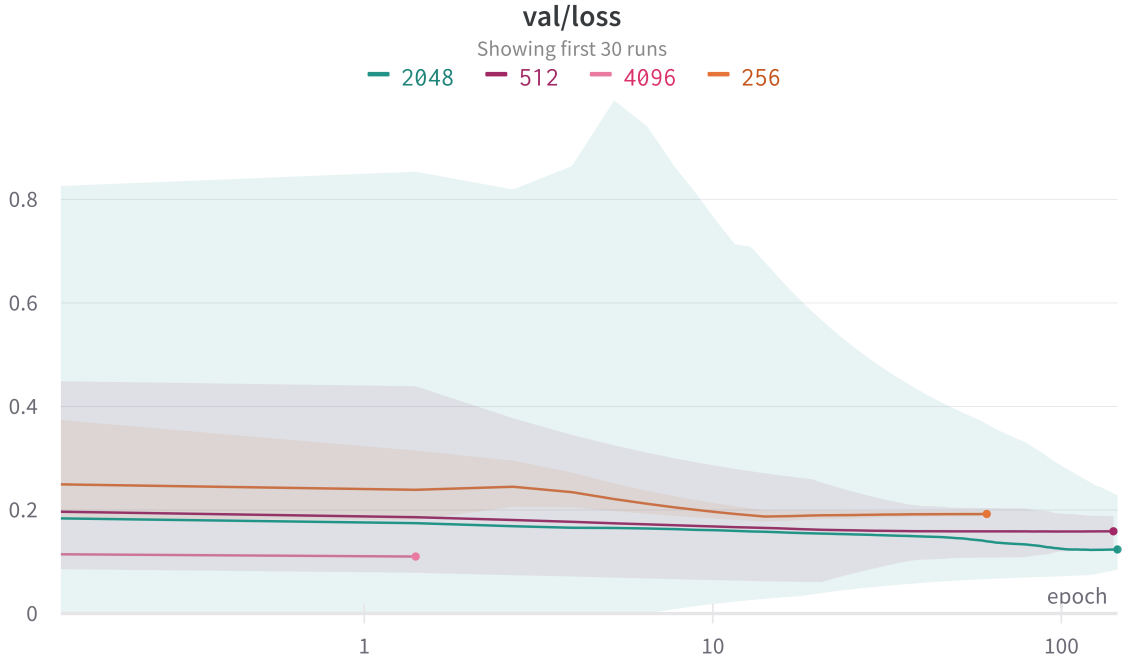

Figure 5: Median validation loss of our experiments w.r.t the batch size. The colorated areas represent the min/max of the val loss. This shows that the most used 2048 batch size obtain the best results. It was also the best compromise as the training speed did not improve for a batch of 4096.

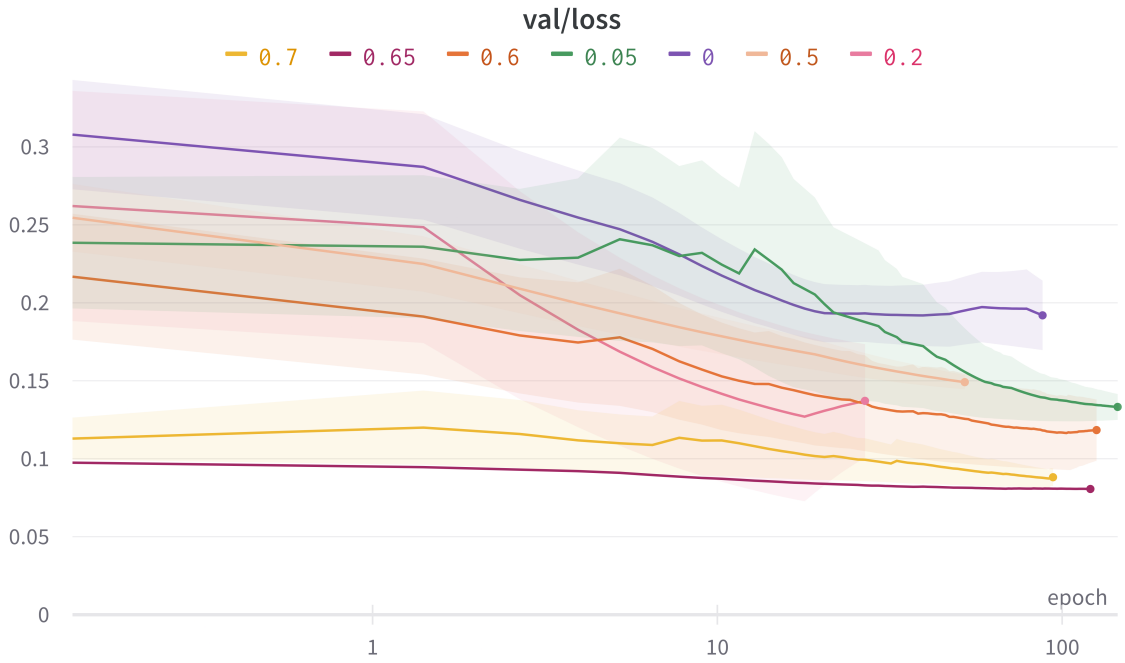

Figure 6: Median validation loss of our experiments w.r.t the dropout. Choosing a dropout between 0.6 and 0.7 yield the best performances.

5.2 Studying the impact of adding graph features

We propose here to compare the public score on the testing set using two different approaches: an only text approach using only the feature vector obtained from SPECTER[2] and a mixed approach that takes both the handcrafted features presented in 3.3.1 and the textual features from SPECTER[2].

Method	Public score on the testing set
SPECTER[2] (text-only)	0.1923
SPECTER[2] and 2 handcrafted graph features	0.1599
SPECTER[2] and 4 handcrafted graph features	**0.1539**

Therefore, using the whole set of handcrafted features helps. We will use this configuration and study the impact of the next methods.

5.3 Studying the impact of adding contextual information

From the previous observation we implement the method explained on the section 3.3 and compare it against the previous number. We use the mean operation as the aggregation operation to aggregate the information from the neighbors embeddings. In addition, for each abstract, we extract the keywords of the latest by applying KeyBERT[3] algorithm with a n-gram between 1 and 2. We extract 10 keywords per abstract and compute the

Method	Public score on the testing set
SPECTER[2] and 4 handcrafted graph features	0.1539
SPECTER[2] and 4 handcrafted graph features + contextual information	0.1009
SPECTER[2] and 4 handcrafted graph features + contextual information (larger MLP)	0.0988
SPECTER[2] and 4 handcrafted graph features + contextual information (larger MLP) + keywords features	**0.0931**

Table 1: Results on the test set for various contextual informations

embedding of these keywords using SPECTER[2]. We concatenate the mean embedding of the keywords to the previous features and obtain the following results:

One can conclude that using the contextual information, *i.e.* a mean embedding representing the neighbors embedding as well as information regarding the keywords that has been used should help for a better prediction.

5.4 Final combination of features

Finally, we ran a sweep to find the best combination of the features introduced in 3.3.3. We designed a *weight and biases* sweep file, to test every possible combination. The parameter importance and the parallel graph shown in 8 and 7 give an overview of the most important parameters. We also add the number of intersecting keywords between the two input nodes.

The best combination is thus :

- SPECTER Embeddings
- Contextual information
- Intersection of keywords
- GELU activation function
- 4 handcrafted graph features

We refer this combination as *final combination*

Even after an exhaustive search using wandb sweeps, it appears that using any link prediction coefficient seem to not help on the public test set score. We decided to keep the best submission with these coefficients in case it will be better on the full test set. We

investigated the impact of using various combination of these coefficients on the validation loss using *wandb sweeps*. The results can be observed on the figures 7 and 8.

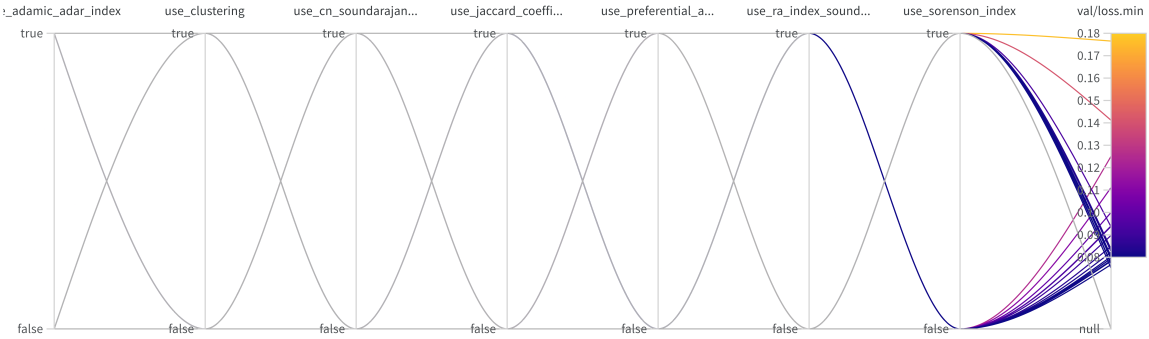

Figure 7: Parallel coordinate plot from wandb. Here we can visualize the various parameters involved in the run. Best viewed on the wandb interface.

6 Discussions

We have managed to successfully achieve a good score on this Kaggle challenge by being ranked 2nd on the public test set. We have learned a lot on how to extract meaningful features from graphs and texts for citation prediction. Even though some results can be intuitively explained, some features, usually handcrafted can achieve a great boost on the public test set and on our validation loss. In the future, we may want to dig more on this direction, and try to formally understand and explain how these features act in this challenge.

Another direction we may want to explore is the exploitation of the information regarding the authors. We argue that this information can be very crucial, since a famous authors could have a high chance to be cited, or colleagues could be easily citing their work. As a possible improvement, one can investigate how to efficiently extract features from the authors (for example by building an authors graph instead of our approach) to use them for the challenge.

References

[1] Iz Beltagy, Kyle Lo, and Arman Cohan. *SciBERT: A Pretrained Language Model for Scientific Text*. 2019. arXiv: `1903.10676 [cs.CL]`.

Method	Public score on the testing set
final combination	**0.082**
final combination + best link prediction coefficient	0.084

Table 2: Results on the test set using the final combination

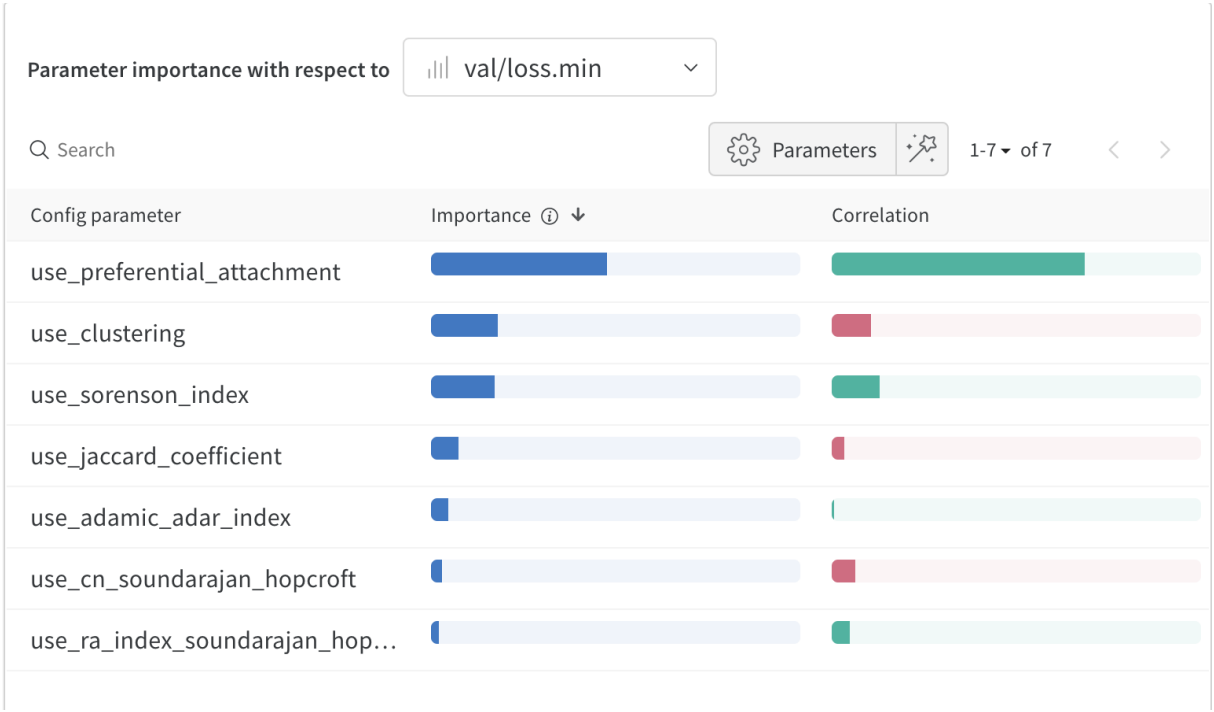

Figure 8: Importance of the parameters with respect to the validation loss. Red means that the parameter has a negative influence, and thus should be used.

[2] Arman Cohan, Sergey Feldman, Iz Beltagy, et al. "SPECTER: Document-level Representation Learning using Citation-informed Transformers". In: *ACL*. 2020.

[3] Maarten Grootendorst. *KeyBERT: Minimal keyword extraction with BERT.* Version v0.3.0. 2020. DOI: 10.5281/zenodo.4461265. URL: https://doi.org/10.5281/zenodo.4461265.

[4] David Liben-Nowell and Jon Kleinberg. "The Link-Prediction Problem for Social Networks". In: *J. Am. Soc. Inf. Sci. Technol.* 58.7 (May 2007), pp. 1019–1031. ISSN: 1532-2882.

[5] Sucheta Soundarajan and John Hopcroft. "Using Community Information to Improve the Precision of Link Prediction Methods". In: *Proceedings of the 21st International Conference on World Wide Web*. WWW '12 Companion. Lyon, France: Association for Computing Machinery, 2012, pp. 607–608. ISBN: 9781450312301. DOI: 10.1145/2187980.2188150. URL: https://doi.org/10.1145/2187980.2188150.

[6] Peng Wang, Baowen Xu, Yurong Wu, et al. *Link Prediction in Social Networks: the State-of-the-Art*. 2014. arXiv: 1411.5118 [cs.SI].

[7] Sucheta Soundarajan and John Hopcroft. "Using Community Information to Improve the Precision of Link Prediction Methods". In: *Proceedings of the 21st International Conference on World Wide Web*. WWW '12 Companion. Lyon, France: Association for Computing Machinery, 2012, pp. 607–608. ISBN: 9781450312301. DOI: 10.1145/2187980.2188150. URL: https://doi.org/10.1145/2187980.2188150.

[8] Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, et al. "Generalizations of the clustering coefficient to weighted complex networks". In: *Physical Review E* 75.2 (Feb. 2007). ISSN: 1550-2376. DOI: 10.1103/physreve.75.027105. URL: http://dx.doi.org/10.1103/PhysRevE.75.027105.

[9] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].