Feature-Less End-to-End Nested Term Extraction

Yuze Gao[1] and Yu Yuan[2]

[1] The Institute for Infocomm Research of A*STAR, Singapore
yuze.gao@outlook.com
[2] School of Languages and Cultures, Nanjing University of Information Science and Technology, China
hittle.yuan@gmail.com

Abstract. In this paper, we proposed a deep learning-based end-to-end method on domain specified automatic term extraction (ATE), it considers possible term spans within a fixed length in the sentence and predicts them whether they can be conceptual terms. In comparison with current ATE methods, the model supports nested term extraction and does not crucially need extra (extracted) features. Results show that it can achieve a high recall and a comparable precision on term extraction task with inputting segmented raw text.

Keywords: Term Extraction · Span Extraction · Term Span.

1 Introduction

Automatic Term Extraction (ATE) or terminology extraction, which is to automatically extract domain specified phrases from a given corpus of a certain academic or technical domain, is widely used in text analytic like topic modelling, data mining and information retrieval from unstructured text. To specify, Table 1 shows a simple example of the tasks, the numbers in the brackets (for example [0, 4]) indicate the start and end index of the term in the sentence separately. Given a sentence, the task is to extract the [0, 4], [0, 5], [1, 1] terms which are specific terms in a domain.

Table 1. Terms Example in sentence

Sentence	"Mouse interleukin-2 receptor alpha gene expression"
Terms To Extract	1. [0, 4] –> "#DNA domain or region"
	2. [0, 5] –> "#other_name"
	3. [1, 1] –> "#protein_molecule"

Typically, ACE approaches make use of linguistic information (part of speech tagging, phrase chunking and constituent parsing), extracted features or defined

rules to extract terminological candidates, i.e. syntactically plausible terminological noun phrases (NPs). Furthermore, in some approaches, potential terminological entries are then filtered from the candidate list using statistical, machine learning or deep learning methods. Once filtered, with low ambiguity and high specificity, these terms are particularly useful for conceptualizing a knowledge domain or for supporting the creation of a domain ontology or a terminology base.

2 Related Works

Current methods can mainly be divided into five kinds: rule-based method, statistical method, predominant hybrid-based, machine learning-based and deep learning-based.

Rule-based approaches [1, 2] heavily rely on the syntax information. The portability and extensibility is also low. Error propagation from the syntax information would hamper the accuracy of models. For example, the POS-tag rule-based system [3] suffer from low recall due to erroneous POS-tagging. Moreover, complex structure using modifier always pose parsing challenges for most simple POS-tag rule-based algorithms. *Statistical ATE system* [4] calls for large quality and quantity dataset to make a reasonable statistic of frequency, distribution and etc.. When predicting, the low frequency or new-occur term may be easily neglected. *Predominant hybrid ATE* tries to combine the advantages of both rule-based and statistical approach. Generally, statistical methods are employed to trim the search space of candidates terms that identified by various linguistic heuristic and rules. However, combining the linguistic filters and statistical distribution ranking would lead to a degenerated precision with the increase of recall. *Machine-learning based ATE* [5–8] is to design and learn different features in the raw text or from syntax information, and then integrate these features into a machine learning method (such as conditional random field, supporting vector classifier). However, different domain, especially language shares different feature patterns, making this method specified to one language or domain. *Deep learning-based ATE* like sequence labelling methods [9] are also proposed recent years, but they do not support nested term extraction. [10] also proposed a co-training method using CNN and LSTM, and expand its training data by adding high confidence predicted results, but this method easily leads to error propagation [11].

To overcome some disadvantages of current ATE methods, we proposed another end-to-end method that based on deep learning, it supports nested term extraction and can achieve comparable experiment results without using extra features and syntax information. The **code** and **data** is shared on github[3].

[3] https://github.com/CooDL/NestedTermExtraction

3 Model

In our model, we formulate the term extraction task as a progress of classifications and filtering, which consider all possible spans or segmentation in the sentence and distinguish them whether they can be domain specified terms in the sentence (see Fig. 1 for more details about our model architecture). In the following sections, we will illustrate our classification and ranking-based term extraction system in details.

3.1 Term Spans

First, we would introduce the span or token segment ([12] also used in coreference resolution) used in our model. To specify, given a sentence $\boldsymbol{S}$=$w_1, w_2, ..., w_i, ..., w_n$ with $\boldsymbol{n}$ words, supposing every token sequence fragment $[w_i, w_j](1 \leq i \leq j \leq n)$ is a span candidate, then there will be $\boldsymbol{T} = \frac{\boldsymbol{n(n+1)}}{\boldsymbol{2}}$ term span candidates in a sentence with n words.Refer to Tab. 2 for more details about term span.

Table 2. Term Span Example

Sentence ($\boldsymbol{n}$=6)	"Mouse interleukin-2 receptor alpha gene expression"
True Term Spans	[0, 4], [0, 5], [1, 1]
Model Processed Spans ($\boldsymbol{k}$=5)	[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [2, 2], [2, 3], [2, 4], [2, 5], [3, 3], [3, 4], [3, 5], [4, 4], [4, 5], [5, 5]

3.2 Model Architecture

Briefly, our model can be divided into three parts (see Fig. 1 for more details): First is the feature preparation part (in red rectangle) that builds span representation vectors. Second is the classification part that classifies the span candidates to collect 'true positive spans' (TPS, that are potential to be terms). Finally is the ranking part that ranks the collected TPS based on ranking scores and sift the n-best span candidates. We will elaborate in next two parts (**Sentence Features** and **Span Representation**) for how to build the span representation in details.

Sentence Features This part describes how the sentence sequence features are built from the raw segmented sentence (Refer to the red rectangle part in Fig. 1 for more details). In this step, both char level and word level information are used to build the sequence features, we use the framework of [13] to build the hidden features. Especially, we use the Conventional Neural Network (CNN) [14] on character level feature building, and Long Short Term Memory Neural Network (LSTM) [16] on word level feature building, which is the best combination described in the [13].

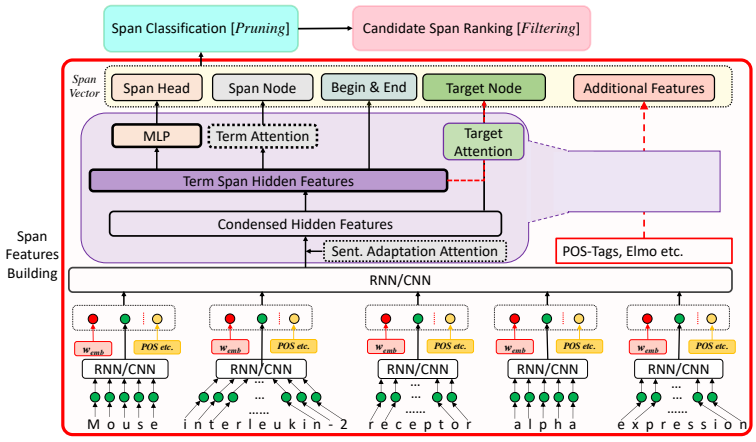

Fig. 1. Model Architecture

Moreover, before using the hidden features, we apply an attention mechanism layer over the sequence features to refine and condense the sequence hidden features. To specify, we use a pre-defined vector $\mathbf{v_s}$ as target vector over the sequence hidden features $H = [h_1, h_2, ..., h_n]$ with dot products. Then, we make a reduce sum operation on the hidden dimension axis and compute a soft-max contribution weight on every token in the sentence. Here, p_i is the probability (contribution weight) to the pre-defined vector $\mathbf{v_s}$.

$$p_i = \frac{h_i \cdot \mathbf{v_s}}{\sum_{k=1}^{n} h_k \cdot \mathbf{v_s}} \quad (1 \leq i \leq n) \tag{1}$$

The probability (contribution weight) of each token in the sentence is applied on its corresponding token.

$$h_{si} = h_i * p_i \quad (1 \leq i \leq n) \tag{2}$$

Here the h_{si} can be seen as the refined feature on terminology. We use the $H_s = [h_{s1}, h_{s2}, ..., h_{si}..., h_{sn}]$ as the sentence sequence features for span representation.

Span Representation Once we get the initial candidate spans in Section 3.1 and the hidden features of the sentences in Eq. 2, we can design feature patterns and build the span representations from H_s (Eq. 2).

To specify, given a sentence $S = w_1, w_2, ..., w_i, ..., w_n$, its final feature layer is $H_s = [h_{s1}, h_{s2}, ..., h_{si}..., h_{sn}]$. Suppose a candidate span $Span_m = [i, j], (0 \leq i \leq j \leq n \ and \ j - i \leq \boldsymbol{k})$, $\boldsymbol{k}$ is the maximum length of the terms in the term candidate set $\boldsymbol{T}$. For ***Each Span***, we construct four kind features from H_s:

I). ***Span Node*** is designed to contain the continuous information of the candidate term span for our model. For example the POS-tag sequence of the candidate. For a ***Span***, suppose its continuous hidden feature vectors in H_s are

$H_m = [h_i, h_{i+1}, ..., h_j]$, we first flatted hidden vectors and concatenate them, then we use a Multi-Layer Perceptron (MLP) to refine the the flatten vector to a vector V_n with the same dimension on hidden features. We use the V_n as the span node vector.

$$V_n = MLP([h_i : h_{i+1} : ... : h_j]) \tag{3}$$

II). *Span Head* is designed to contain the head word information if any and whether all the words in span can form a complete Noun Phrase. We use a predefined vector $\mathbf{v_t}$ which has the same dimension with hidden features as term target vector, and apply a term attention over its hidden feature sequence H_m to get its head feature vector V_h of the span. First the vector $\mathbf{v_t}$ is applied on H_m with multiply products to reduce the hidden feature of each word to a logit.

$$P_h^{[x]} = \frac{h_x * \mathbf{v_t^T}}{\sum_{x=i}^{j} h_k * \mathbf{v_t^T}} \qquad (h_x, h_k \in H_m) \tag{4}$$

Then the soft-max score from the logits is applied on their corresponding tokens.

$$V_h = \sum_{x=i}^{j} h_x * P_h^{[x]} \qquad (h_x \in H_m) \tag{5}$$

Here the x means the token in the span token sequence.

III). *Start and End Words* V_{be} is designed to contain the feature information of begin and end word to the model. For example, generally, the term cannot start with a **PREP** word.

$$V_{be} = [h_i : h_j] \tag{6}$$

IV). *Sentence Targeted Attention Node* is designed to embed some feature information like whether the candidate span can express a concept to the complete sentence and leverage the information from the sentence level into term spans. We use the mean vector of a span hidden features as a target vector and apply dot-wise attention over the sentence hidden features. This kind of attention mechanism is widely used in targeted sentiment analysis [17, 18]. To specify, given a span with hidden feature sequence H_m, suppose its mean vector is $\hat{h_m}$. Here $\sum\!\!\!\!\!\!-$ means 'average sum' on token axis.

$$\hat{h_m} = \sum\!\!\!\!\!\!-_{x=i}^{j} h_x \quad (h_x \in H_m) \tag{7}$$

We use $\hat{h_m}$ as a target vector and apply its transposed on sentence level hidden feature sequence H_s with multiply products to reduce the hidden feature of each token to a logit $\hat{h_s}$. The logits will be softmaxed on the token axis in the sentence.

$$P_s^{[x]} = \frac{h_s[x] * \hat{h_m}^T}{\sum_{k=1}^{n} h_s[k] * \hat{h_m}^T} \quad (h_s[x], h_s[k] \in H_s) \tag{8}$$

The computed probability above is applied to its corresponding token in the sentence hiddens H_s, and the weighted sum product V_s is the span targeted attention node.

$$V_s = \sum_{i=1}^{n} h_s[x] * P_s^{[x]} \tag{9}$$

Apart from these four features, we also give each span a length feature vector V_l to indicate the length of the span. So, given a candidate span $Span^M$, it has a span representation S_M, which has a five times dimension with the hidden feature. Here the concatenate operation over the feature dimension axis.

$$S_M = [V_n, V_h, V_{be}, V_s, V_l] \tag{10}$$

Additional Features* Other kind source features such as ELMO [19][4], Part-Of-Speech will also be refined via all the procedures(I, II, III, IV).

3.3 Classification and Ranking

After obtaining the span representations, we use these representations do the classification and ranking steps. First, based on the span representations S_M, we do a binary classification (CLF_{FC}) to classify these span candidates into true and false groups (TF_G).

$$TF_G = CLF_{FC}(S_M) \tag{11}$$

After classification, a scoring step will be applied over the 'true' span group (T_G, $T_G \in TF_G$). Here, we obtain the scores (R_{scores}) from a regression function(REG) which use the span representation S_M as inputs. $S_M^{T_G}$ is the span representation of true group T_G. The regression function(REG) is designed to give each span candidate a score between 0 and 1.

$$R_{scores} = \{REG(S_M^{T_i}), \quad S_M^{T_i} \in S_M^{T_G}\} \tag{12}$$

The scores of the T_G span group are then handed to a ranker as ranking (or confidence) scores. The top-K span results from the ranking step would be thought as the final output (TM_S) of our SCR model, which are with higher ranking scores (or confidence).

$$TM_S = RANKER|_{n=1}^{K}(R_{scores}) \tag{13}$$

Here K is a threshold value, we compute $\mathrm{K} = \alpha \cdot |TotalWords|$. $|TotalWords|$ is the total words currently processed. α is a ratio indicate that how much terms are in a certain number of words. In our model, there are two loss source: one is the classification loss and one is the ranking step loss. We use a two-stage optimization strategy to minimizing the model loss.

$$Loss_{(classifier)} = -(y * log(p) + (1 - y) * log(1 - p)) \tag{14}$$

[4] For ELMO feature, we use the pre-trained model presented by the nlp toklit allennlp: https://github.com/allenai/allennlp/blob/master/tutorials/how_to/elmo.md

$$Loss_{(ranker)} = \sum_{y \in Y_{\{gold\}}} (1 - Sigmoid(y)) + \sum_{y' \in Y_{\{K-gold\}}} Sigmoid(y') \quad (15)$$

4 Experiments and Analysis

Data In our experiments, we use the GENIA3.02 [21][5], a human annotated, and biology corpus for information extraction and text mining systems. There are total 99,111 terms (distribution indicated in the red line) in 18,539 sentences (total 490,766 words). In these terms, 22675 terms are nested in or overlapped with other terms. 76436 terms are independent. We split the total corpus by sentence into Train/Dev/Test parts with a ratio 0.9 : 0.05 : 0.05 and shuffle the Train when training model.
(Hyper) Parameters we used for our experiments are listed in the Table 3 below. The dropout [22] is only applied in the training progress to avoid over-fitting. We would stop the training processing after the evaluating loss has no increasing for a threshold times in *EarlyStop*.

4.1 Results and Analysis

Baselines Currently, most ATE system are not designed to support nested term extraction. As there exists few systems supporting nested term extraction, resulting the weakness in lateral contrast. Here, we list some state-of-the-art ATE systems and their performance on the GENIA corpus.
1. Wang et al. [10], a co-training method that uses minimal training data and achieve a comparable results with the state-of-the-art on GENIA corpus.
2. Yuan et al. [5], a feature-based machine learning method using n-grams as term candidates and 10 kinds features is pre-processed for each candidate.
Our best models for testing are chosen with the loss of development dataset and their performance is list in Tab. 4. In the table, [5] achieves a satisfying result with Random Forest method in their paper, but the feature preparation is complex and time-consuming. The training data is also re-balanced on the positive and negative instances.

For our classifier model, it has a high recall on the extracting terms, however the precision is not satisfying. The pre-trained word embedding [+GloVe] has little contribution, one reason is that there exists nearly 40% of the words in dataset is out of vocabulary. With extra features like POS-tags [+POS-tag], both precision and recall increase, which indirectly gives some evidences that span representations are not precise and concrete as the external POS-tags features on the POS-tag aspect. However, the ELMO features [+ELMO], which we thought should work better than the POS-tag features, do not bring much improvement as it raises pretty little in precision and falls in recall. When we utilize the POS-tag, ELMO feature and GloVe pre-trained embedding together

[5] http://www.geniaproject.org/genia-corpus

Table 3. Hyper-parameters

$DIM_{Word\ Embedding}$	150
$DIM_{POS-tag\ Embedding}$	30
$DIM_{Word\ LSTM}$	150
$DIM_{Span\ Length}$	30
Word LSTM Layers	2
POS-tag LSTM Layers	1 (optional)
Learning Rate	0.01
Batch Size	100
Random Seed	626
Dropout	0.6
Term Ratio	0.23
Early Stop	26

Table 4. Results on Test Set

		Precision	Recall	F1
Wang et al. [10]		0.647	0.780	0.707
Yuan et al. [5]		**0.7466**	0.6847	0.7143
Our Model (Classifier)	Random Embedding	0.5044	**0.9639**	0.6622
	GloVe	0.5093	**0.9557**	0.6575
	+ POS-tag	0.5198	**0.9632**	0.6753
	+ ELMo	0.5220	**0.9541**	0.6748
	+ ALL	0.5163	**0.9698**	0.6738
Our Model (Ranker)	Random Embedding	0.7237	0.8343	0.7751
	GloVe	0.7244	0.8356	0.7760
	+ POS-tag	**0.7265**	**0.8375**	**0.7780**
	+ ELMo	0.7252	**0.8386**	0.7778
	+ ALL	**0.7316**	0.8327	**0.7789**

Noted that: Decreasing the term ratio $\boldsymbol{\alpha}$ will increase the precision but degenerate the recall (Fig. **??**). All the results in Tab. 4 are under the the settings in Tab. 3.

[+ALL], we get a further improvement on recall of the classifier. But also cause a sharp increase in the resource consuming. The effect of ELMO feature weakens when we decrease the dimension of ELMO feature.

For the ranker model, it is expected to filter the pruned span candidates, and it gains a better precision score than the classifier, but a low recall score problem due to some loss of true positive terms. The POS-tag features [+POS-tag] bring improvement in both precision and recall but not significant than the classifier. The effect of pre-trained emdedding vector [+GloVe] also vanishes, which can be seen as a fluctuation in error margin. The ELMO features [+ELMO] do increase the recall, but not obviously. Compared with using all the features [+ALL], the ranker model uses POS-tag [+POS-tag] is slight poor in precision. The ELMO feature does not help much to improve both our classifier and ranker model. It makes us doubt that if the 'hard' features work better than the 'soft' features in the ATE task.

4.2 Other Experiments

Span Length: We compare the model performance under different maximum span length (from 1 to 15), and list them on Test set in the Figures (Fig. 2, Fig. 3) on our classifier model and ranker model below.

For the classifier, as we increase the maximum term length, the recall increase to a stable value (approximate 0.96) without dropping, which means the saturation of its recall ability. It is reasonable that the precision decreases to a fluctuated point as the candidates space increases several times when increasing the maximum term length.

For the ranker, the precision and recall increase to a stable state. However, when the length is less than 2, the ranker model has low precision. But, shorter maximum length means lower span candidates space, which means the model should obtain a higher precision. We will explain why in the next part.

Overall, it can be noticed that the final result (from the ranker) has not been influenced too much when increasing max-length, which indicating that the model will maintain stable and can distinguish the true positive instances.

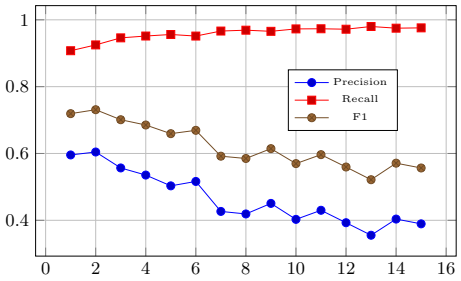

Fig. 2. Classifier on lengths(Testset)

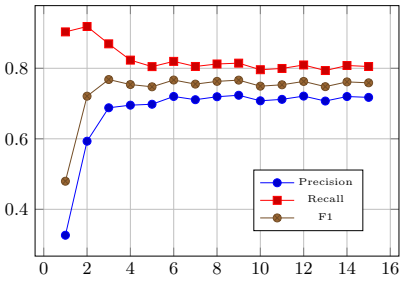

Fig. 3. Ranker on lengths(Testset)

5 Conclusion and Future Work

We proposed a deep learning-based end-to-end term extraction method in this paper. It employs classification and ranking on the span (n-grams) candidates in the sentences. Compared with current methods, it supports the nested term extraction and can achieve a comparable result with merely the segmented raw text as input. Based on the sentence and term span hidden features, four kinds reasonable feature patterns are designed to convey different information. Experimental results show that these features indeed can embed some information. Though the model achieve a satisfying results, the ranker still loss many true positive instances, which decrease the recall score from 0.95 to 0.83. Moreover, threshold-based output is not so applicable on unknown or unfamiliar domain or data as we do not know the term distribution and ratio.

Future works may focus on is to immigrate the architecture on a better feature extracting model (such as BERT [23] or GPT2.0 [24]). The ranking step should be degisned more reasonable to pick up the outputs. More reasonable feature patterns can be designed to convey useful information.

References

1. Stankovi Ranka, Krstev Cvetana, Obradovi Ivan, Lazi Biljana and Trtovac Aleksandra: "Rule-based automatic multi-word term extraction and lemmatization." Proceedings of LREC 2016.
2. Katerina Frantzi, Sophia Ananiadou and Hideki Mima: "Automatic recognition of multi-word terms: the c-value/nc-value method." IJDL, 3(2):115130. 2000.

3. Ziqi Zhang, Jie Gao, and Fabio Ciravegna: "Jate 2.0: Java automatic term extraction with apache solr." Proceedings of the 10th LREC, 2016.
4. Lishuang Li, Yanzhong Dang, Jing Zhang and Dan Li: "Domain term extraction based on conditional random fields combined with active learning strategy." JICS.
5. Yu Yuan, Jie Gao, and Yue Zhang: "Supervised learning for robust term extraction." 2017 International Conference on Asian Language Processing (IALP). IEEE, 2017.
6. GuoDong Zhou and Jian Su: "Exploring deep knowledge resources in biomedical name recognition." Proceedings of the IJWNLP. 2004.
7. Rogelio Nazar and Maria Teresa Cabre: "Supervised learning algorithms applied to terminology extraction." Proceedings of the 10th TKEC, 2012.
8. Merley da Silva Conrado, Thiago A. Salgueiro Pardo and Solange Oliveira Rezende: "A machine learning approach to automatic term extraction using a rich feature set." Proceedings of the NAACL HLT 2013 Student Research Workshop
9. Maren Kucza, Jan Niehues and Sebastian Stker: "Term Extraction via Neural Sequence Labeling a Comparative Evaluation of Strategies Using Recurrent Neural Networks." Proceedings pf Interspeech. 2018.
10. Rui Wang, Wei Liu, and Chris McDonald: "Featureless domain-specific term extraction with minimal labelled data." Proceedings of the Australasian Language Technology Association Workshop 2016. 2016.
11. Nigam, Kamal, and Rayid Ghani. "Analyzing the effectiveness and applicability of co-training." Cikm. Vol. 5. 2000.
12. Kenton Lee, Luheng He, Mike Lewis and Luke Zettlemoyer: "End-to-end neural coreference resolution." arXiv preprint arXiv:1707.07045 (2017).
13. Jie Yang, Yue Zhang: "NCRF++: An Open-source Neural Sequence Labeling Toolkit." Proceedings of ACL (2018), 2018
14. Krizhevsky, Alex, Ilya Sutskever and Geoffrey E. Hinton: "Imagenet classification with deep convolutional neural networks." ANIPS. 2012.
15. Yuze Gao, and Tong Xiao: "A Comparison of Pruning Methods for CYK-based Decoding in Machine Translation." Proceedings of CWMT. 2015.
16. Hochreiter Sepp and Jrgen Schmidhuber: "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.
17. Jiangming Liu and Yue Zhang: "Attention modeling for targeted sentiment." Proceedings of the 15th Conference of the EACL, Short Papers. 2017.
18. Yuze Gao, Yue Zhang and Tong Xiao: "Implicit Syntactic Features for Targeted Sentiment Analysis." Proceedings of IJCNLP (2017), 2017
19. Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer: "Deep contextualized word representations." arXiv preprint arXiv:1802.05365 (2018).
20. Kingma Diederik P. and Jimmy Ba: "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
21. Jin-Dong Kim, Tomoko Ohta, Yuka Teteisi and Junichi Tsujii: "Genia corpusa semantically annotated corpus for bio-textmining." Bioinformatics (Oxford), 2003.
22. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov: "Dropout: a simple way to prevent neural networks from overfitting." JMLR, 15(1), 1929-1958. 2014
23. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
24. Radford Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language models are unsupervised multitask learners." OpenAI Blog 1, no. 8 (2019).