NHK_STRL at WNUT-2020 Task 2: GATs with Syntactic Dependencies as Edges and CTC-based Loss for Text Classification

Yuki Yasuda* **Taichi Ishiwatari*** **Taro Miyazaki*** **Jun Goto**
NHK Science and Technology Research Laboratories
{yasuda.y-hk,ishiwatari.t-fa,miyazaki.t-jw,goto.j-fw}@nhk.or.jp

Abstract

The outbreak of COVID-19 has greatly impacted our daily lives. In these circumstances, it is important to grasp the latest information to avoid causing too much fear and panic. To help grasp new information, extracting information from social networking sites is one of the effective ways. In this paper, we describe a method to identify whether a tweet related to COVID-19 is informative or not, which can help to grasp new information. The key features of our method are its use of graph attention networks to encode syntactic dependencies and word positions in the sentence, and a loss function based on connectionist temporal classification that can learn a label for each token without reference data for each token. Experimental results show that the proposed method achieved an F1 score of 0.9175, outperforming baseline methods.

1 Introduction

The outbreak of COVID-19 that has occurred since the end of 2019 has greatly impacted our daily lives. In these circumstances, it is important for everyone to understand the situation and grasp the latest information to avoid causing too much fear and panic. Nowadays, social networking sites (SNSs) such as Twitter and Facebook are important information sources because users post information regarding their personal events—including that related to COVID-19—in real time. For this reason, many monitoring systems for COVID-19 have been developed such as The Johns Hopkins Coronavirus Dashboard[1] and the COVID-19 Health System Response Monitor[2]. Many systems use SNSs as resources, but largely depend on manual work such as using cloud sourcing to extract informative posts from massive numbers of uninformative ones. Generally, SNSs contain too much information on miscellaneous topics, so extracting important information is difficult. Therefore, we attempted to develop a method to extract important information.

Our method first embeds each token in the input sentence using BERT (Devlin et al., 2019). Then, the vectors are fed into graph attention networks (GATs) (Veličković et al., 2018) to encode token-to-token relations. Finally, our method classifies each vector into labels using feed-forward neural networks (FFNNs). In the training process, we use a loss function based on connectionist temporal classification (CTC) (Graves et al., 2006). Experimental results show that our method using GATs and the CTC-based loss function achieved an F1 score of 0.9175, outperforming baseline methods.

Our contributions are as follows: (1) We propose a GAT-based network to embed syntactic dependencies and positional features of tokens in an input sentence. (2) We also propose a loss function, which enables to train labels for each token. (3) We confirmed the effectiveness of our proposed methods using the identification of informative COVID-19 English Tweets shared task dataset.

2 Identifying Informative COVID-19 Tweets Shared Task

The identification of informative COVID-19 English Tweets[3] is a shared task held at W-NUT (Workshop on Noisy User-generated Text) 2020 (Nguyen et al., 2020b). The purpose of the task is to identify whether English tweets related to COVID-19 are informative or not. The dataset for the task contains 7,000 tweets for training, 1,000 for validating, and 2,000 for testing. Each tweet in the data, excluding those in the testing data are labelled informative or uninformative. The target metric of the task is the F1 score for informative tweets.

*These authors are equally contributed to this work.
[1] https://coronavirus.jhu.edu/map.html
[2] https://www.covid19healthsystem.org
[3] http://noisy-text.github.io/2020/covid19tweet-task.html

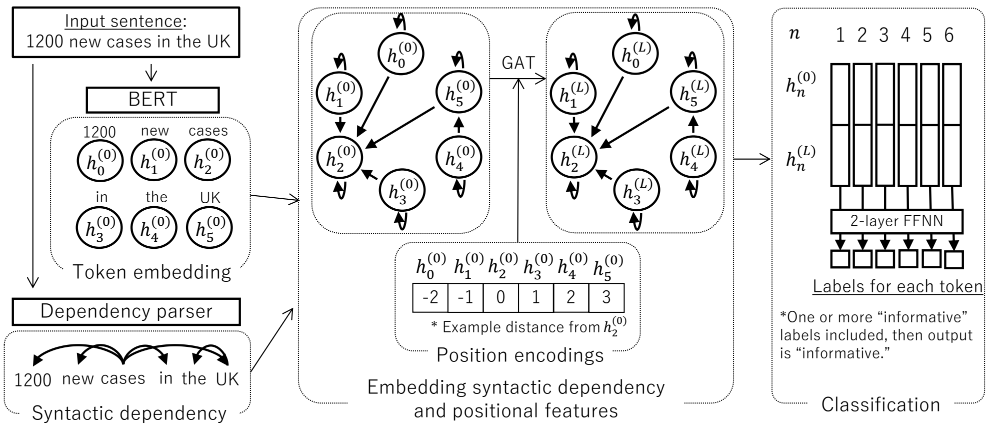

Figure 1: Overview of our method. Our method first embeds each token in an input sentence using BERT. Also, syntactic dependencies are obtained using a dependency parser. Then, our method embeds syntactic features using GATs, by using a graph that has nodes of token-embedding vectors and edges of syntactic dependencies and self-loops. Positional features are also added to the graph. The output vectors of the GATs are concatenated with BERT output vectors, and then fed into 2-layer FFNNs, which classifies each vector into labels. If one or more vectors are labelled as informative, the output class is informative. Note that the arrows in the the dependency parser example connect the head word to the dependent word as to follow a convention. On the other hand, arrows in the GAT example connect the dependent word to the head word, as used in our proposed method.

3 Methods

The overview of our method is illustrated in Figure 1. The key features of our method are embedding syntactic dependencies and positional features using GATs (Veličković et al., 2018), and calculating loss in the training process using a loss function based on CTC (Graves et al., 2006).

We use masked-token estimation as multi-task learning to help improve the generalization capability. We use word-dropout (Sennrich et al., 2016) before BERT, and the "dropped" tokens are used as masked words to be estimated in the training process as a multi-task.

3.1 GATs for encoding token-to-token relations

The BERT model, which we use for token embedding, uses position encoding to consider the position of tokens in the model, but its ability to capture global information including syntactic features is limited (Lu et al., 2020). Therefore, we use GATs with syntactic dependencies as edges of the graph, which enables our method to handle syntactic dependency explicitly. This is inspired from the work of Huang and Carley (2019).

We use all of the universal dependency (McDonald et al., 2013) as a directional edge regardless of dependency type[4]. The tokenizer used in BERT often separates a single word into many tokens. We connect edges from all tokens of a word to all tokens of the head word. For example, if there is a relation between the two words *COVID-19* and *tweet*, and the former word is divided into two tokens *COVID* and *##-19*, our method connects the two edges, *COVID* to *tweet* and *##-19* to *tweet*.

The GAT is based on multi-head attention (Vaswani et al., 2017) among neighbor nodes, with all the connected nodes used as the keys and values of the attention calculation. In many cases, the number of incoming edges for a node is only zero or one if syntactic dependencies are used as edges. Nodes that have no incoming edges cannot update the vector in the GATs. Also, for nodes that have only one incoming edge, the attention weight in the multi-head attention is 1.0, which leads to poor results. To overcome this problem, a self-loop for each node was proposed (Huang and Carley, 2019; Xu and Yang, 2019). Following that, we use a self-loop for each node in the GATs.

Positional features

Many edges are concentrated on the root word of a sentence, so the GATs treats all nodes equally. On the other hand, nearby and distant words are generally more and less related to the root word, respectively. To simulate this, we use positional encoding to our GATs. We use the relative distance between tokens as a parameter, then embed them along with the attention coefficient between nodes

[4]We attempted to use each type separately with the GAT, but the results were worse regardless of dependency types.

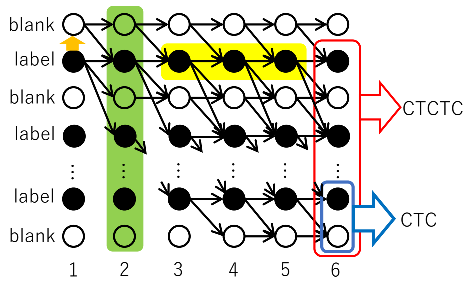

Figure 2: CTC is calculated as the sum of two probabilities in the blue box, while CTCTC is the sum of all probabilities except for the all-blank path as shown in the red box. Green and yellow boxes and orange arrow show the direction of label smoothing, token smoothing, and leaking, respectively.

when calculating the multi-head attention on the basis of the work from Ingraham et al. (2019) and Ishiwatari et al. (2020).

Following the work of Ishiwatari et al. (2020), we compared two types of positional embedding in our experiments, fixed and learned.

For fixed, we use the following representation as a positional embedding between the i-th and j-th tokens of the sentence:

$$PE^{ij}_{\text{fixed}} = L - (i - j) \ , \qquad (1)$$

where L is the number of tokens in the sentence.

For learned, we use a 1-layer FFNN with an input of PE^{ij}_{fixed} as a positional embedding as follows:

$$PE^{ij}_{\text{learned}} = \mathbf{W}_{PE} PE^{ij}_{\text{fixed}} + \mathbf{b}_{PE} \ , \qquad (2)$$

where $\mathbf{W}_{PE} \in \mathbb{R}^{|1\times1|}$ and $\mathbf{b}_{PE} \in \mathbb{1}^{|d|}$ are a learnable weight and bias, respectively.

The positional features are then broadcasted into $PE^{ij} \in |1 \times d|$ where d is the dimension of a GAT layer, and added after calculating the multi-head attention along the edges in the graph.

3.2 CTC for Text Classification (CTCTC)

Most tweets that were labelled as informative contain not only informative phrases but also uninformative parts. To consider this, we propose a new loss function—CTC for Text Classification (CTCTC).

The basis of CTC

Let us consider the input sequence of probabilities $\mathbf{x} \in \mathbb{R}^{|T|\times|L|}$ where $|T|$ denotes the length of the sequence and $|L|$ denotes the number of labels to classify. Note that L includes *blank*, which is a special symbol for CTC labelled for the data in which no labels are aligned. The probability $p_{ctc}(\mathbf{y}|\mathbf{x})$ for input $\mathbf{x}$ and reference data $\mathbf{y} \in \mathbb{1}^{\leq|T|}$ is calculated as follows:

$$p_{ctc}(\mathbf{y}|\mathbf{x}) \ = \sum_{\pi\in\mathcal{B}^{-1}(\mathbf{y})} p(\pi|\mathbf{x}) \ , \qquad (3)$$

where $\mathcal{B}^{-1}$ is the inverse of the many-to-one map $\mathcal{B}$ of all possible labellings from the input to reference data. In generating $\mathcal{B}$, *blanks* are inserted between each label in $\mathbf{y}$, i.e., for $\mathbf{y} = \{y_1, y_2, \cdots, y_{|\mathbf{y}|}\}$, a modified reference $\mathbf{y}' = \{blank, y_1, blank, y_2, \cdots, y_{|\mathbf{y}|}, blank\}$ is used to generate $\mathcal{B}$. In Figure 2, $\mathcal{B}$ is equal to the set of the paths of black arrows that finally reach one of the two dots in the blue box. Then, p_{ctc} represents the probability of the sum of all probabilities of paths that pass all labels with the given order as reference data, which is shown as the sum of two probabiliuties in the blue box in Figure 2.

CTCTC loss

We use a CTC-based loss function that is utilized for text classification. Our loss function accepts the reference data $\bar{y}$, which is a single label for an "informative" or "uninformative" sentence in the task, and assign a label or *blank* for all tokens in the sentence. It works by handling the uninformative parts in informative tweets as *blank* automatically.

Calculating CTCTC is almost the same as CTC, differing only in the construction of the many-to-one map. First, CTCTC arranges a sufficient number of the given reference label $\bar{y}$ and blank, i.e., $\bar{\mathbf{y}}' = \{blank, \bar{y}, blank, \bar{y}, \cdots, \bar{y}, blank\}$. Then, $\bar{\mathcal{B}}$ is generated, which is the set of all possible labellings from the input $\mathbf{x}$ to modified reference data $\bar{\mathbf{y}}'$ regardless of the number of passed labels in $\bar{\mathbf{y}}'$. In Figure 2, $\bar{\mathcal{B}}$ is equal to the set of the paths of black arrows that finally reach one of the dots in the red box. To calculate a CTCTC loss, $\bar{\mathcal{B}}$ is used instead of $\mathcal{B}$ in Equation (3). As a result, the probability p_{ctctc} represents the probability of at least one token in the input sequence being aligned to the label $\bar{y}$, which is illustrated as the sum of all dots in the red box in Figure 2.

Smoothing for CTCTC

CTCTC tends to align most tokens to *blank*, and only one token to the reference label. This is because the probability for *blank* is learned for every sentence in the training data regardless of its label, so the probability tends to be high for all data. To avoid the probabilities of all data being learned as blank, we prepare three types of smoothing.

Label smoothing We use label smoothing (Szegedy et al., 2016), which is a regularization technique to avoid overfitting and overconfidence. This replaces the one-hot reference label l with the smoothed label $l'(k)$ as follows:

$$l'(k) = (1 - \epsilon)\delta_{k,l} + \frac{\epsilon}{K} \ , \tag{4}$$

where $\delta_{k,l}$ is the Dirac delta function, which equals 1 when $k = l$ and 0 otherwise, K is the set of labels to classify, and ϵ is the smoothing rate. The label-wise smoothing is illustrated as a green box in Figure 2.

Token smoothing This is almost the same as label smoothing but differs in the direction of the smoothing—token-wise. It works on the basis that words close together often have similar meanings. We set the max width to 5 to consider this smoothing in the experiments. The token-wise smoothing is illustrated as a yellow box in Figure 2.

Leaking To enable learning the probability for labels instead of *blank*, we use the one-direction smoothing named "leaking." This is calculated as follows:

$$p'_{i,blank} = (1 - \epsilon')p_{i,blank} + \epsilon' p_{i,\bar{y}} \ , \tag{5}$$

where ϵ' is the smoothing rate, $p_{i,blank}$ and $p_{i,\bar{y}}$ are the probabilities for *blank* and the reference label $\bar{y}$ of i-th data of the input sequence, respectively. This is calculated only for the probability of *blank*, and is illustrated as an orange arrow in Figure 2.

4 Experiments

4.1 Experimental settings

Our experiments were based on the identification of informative COVID-19 English Tweets dataset mentioned in Section 2. We conducted two experiments on the basis of the validation and testing data, respectively. For the validation data-based experiment, we used training data contains 7,000 tweets and validating data contains 1,000 tweets for training and testing, respectively. For the testing data-based experiment, we used 8,000 tweets mixed from the training and validating data for 4-fold cross validation. Then, an ensemble of the best model of each fold data were used for testing data.

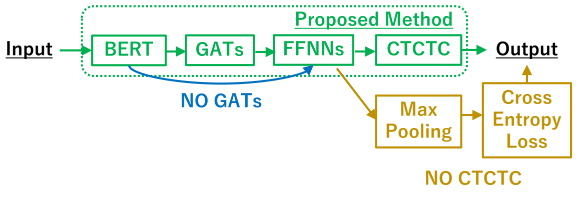

Figure 3: Overview of baseline methods. Green elements show the process of our proposed method. The blue arrow and orange elements show the process of a baseline method that does not use GATs and does not use CTCTC, respectively.

We added the output scores of each model for the model ensemble.

The models were implemented in PyTorch (Paszke et al., 2019), with Transformers (Wolf et al., 2019) and Deep Graph Library (Wang et al., 2019), and learned with the RAdam optimizer (Liu et al., 2020) with a learning rate of 0.0001. We used BERT-base, uncased (Devlin et al., 2019) as a pre-trained model, with fine-tuning and a learning rate of 0.00002. We used spaCy (Honnibal and Montani, 2017) for dependency parsing.

The following hyperparameters were used: number of GATs layers was 2; a mini-batch size of 16; L2 regularization coefficient of 0.1; dropout rate of 0.1; word dropout rate of 0.2; 50 training iterations, with early stopping on the validating data on the basis of the F1 score for the informative class; and smoothing ratio for the three smoothing methods of CTCTC of 0.2.

4.2 Baseline methods

We prepared baseline methods as shown in Figure 3. To confirm the effectiveness of the GATs, a baseline method of "no GATs" that does not use GATs but the output vectors of BERT is directly fed into the FFNNs. Also, to confirm the effectiveness of CTCTC, a baseline of "no CTCTC" that does not use CTCTC but cross entropy loss is used.

4.3 Results

Table 1 shows the results for the validation data-based experiment. The rows in which Use GATs and Use CTCTC are not checked indicate the baselines shown in Section 4.2. F1 score shows the F1 score for the informative class with the mean and standard deviation of five-time trials of the same settings. Our methods using both GATs and CTCTC (# 9 and 10) achieved the top-2 results in the table.

Table 2 shows the results on the test data, which

Table 1: Experimental results on validation data-based experiments.

#	GATs parameters		CTCTC parameters				F1 score
	Use GATs	Positional feature	Use CTCTC	Label smoothing	Token smoothing	Leaking	
1							0.9154 ± 0.0041
2	✓						0.9134 ± 0.0015
3	✓	Fixed					0.9151 ± 0.0026
4	✓	Learned					0.9151 ± 0.0009
5			✓				0.0000 ± 0.0000
6			✓	✓			0.9128 ± 0.0026
7			✓	✓	✓		0.9133 ± 0.0052
8			✓	✓	✓	✓	0.9153 ± 0.0024
9	✓	Fixed	✓	✓	✓	✓	0.9172 ± 0.0027
10	✓	Learned	✓	✓	✓	✓	**0.9175 ± 0.0044**

Table 2: Results on the test data.

Team / Method	F1 score
Ours (#9 in Table 1)	0.8898
Ours (#10 in Table 1)	0.8885
NutCracker	0.9096
NLP_North	0.9096
UIT-HSE	0.9094

are the official results of the shared task and we ranked 21st out of 55 participants[5]. The table also shows the results of the top-3 teams in the shared task.

4.4 Discussion

The results for the methods using GATs with CTCTC (#9 and 10) are better than the others. This is because our CTCTC uses vectors of each token so the performance depends on the quality of the vector of each token. Our GATs work to improve the quality of the vector of each token by using token-to-token relations. Therefore, we believe our GATs and CTCTC work well in combination. On the other hand, GATs without CTCTC cannot make the best use of the improved vectors because they are mixed up vectors of tokens into one vector using max-pooling, so some of the details of the vectors are lost. Also, in using CTCTC without GATs, we observed that the output vectors of each token in the sentence are almost the same. This means that token-level information is lost, so accuracy may be lower for methods using CTCTC in these cases. By using GATs with CTCTC, we can avoid losing the information, which leads to good results.

5 Related Work

There are a number of methods that use GATs with a pre-trained language model. Lu et al. (2020) use a network on a vocabulary graph, which is based on word co-occurrence information, and Huang and Carley (2019) and Xu and Yang (2019) use syntactic features as a graph. Also, there are several methods that use positional encoding into GATs (Ingraham et al., 2019; Ishiwatari et al., 2020). Our method uses GATs to consider syntactic features with positional features in combination, which is distinguishable from conventional methods.

The CTC loss function is widely used for long data sequence with not-one-to-one-aligned reference data such as speech recognition (Graves et al., 2013; Kim et al., 2017), but to the best of our knowledge, no method that uses CTC for text classification tasks exists.

6 Conclusion and Future Work

In this paper, we proposed a GATs-based model that embeds token-to-token relations, and a loss function that can learn classes for each tokens. We conducted evaluations using the identification of informative COVID-19 English Tweets dataset, and confirmed that our proposed methods are effective.

To determine whether CTCTC can work for other tasks especially for the classification into large amount of classes and to exploit pre-trained models other than BERT, especially for tweet-specific models such as BERTweet (Nguyen et al., 2020a) and CT-BERT (Müller et al., 2020), are subjects of as our future work.

Acknowledgement

We would like to thank the identification of informative COVID-19 English Tweets shared task organizers for providing the dataset and opportunity for discussion. We also thank the anonymous reviewers for their helpful comments.

[5]https://competitions.codalab.org/competitions/25845#results

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Binxuan Huang and Kathleen M Carley. 2019. Syntax-aware aspect level sentiment classification with graph attention networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5472–5480.

John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. 2019. Generative models for graph-based protein design. In *Advances in Neural Information Processing Systems*, pages 15820–15831.

Taichi Ishiwatari, Yuki Yasuda, Taro Miyazaki, and Jun Goto. 2020. Relation-aware graph attention networks with relational position encodings for emotion recognition in conversations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*.

Suyoun Kim, Takaaki Hori, and Shinji Watanabe. 2017. Joint CTC-attention based end-to-end speech recognition using multi-task learning. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4835–4839. IEEE.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2020. On the variance of the adaptive learning rate and beyond. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*.

Zhibin Lu, Pan Du, and Jian-Yun Nie. 2020. VGCN-BERT: Augmenting BERT with graph embedding for text classification. In *European Conference on Information Retrieval*, pages 369–382. Springer.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97.

Martin Müller, Marcel Salathé, and Per E Kummervold. 2020. Covid-twitter-bert: A natural language processing model to analyse covid-19 content on twitter. *arXiv preprint arXiv:2005.07503*.

Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. 2020a. BERTweet: A pre-trained language model for English Tweets. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

Dat Quoc Nguyen, Thanh Vu, Afshin Rahimi, Mai Hoang Dao, Linh The Nguyen, and Long Doan. 2020b. WNUT-2020 Task 2: Identification of Informative COVID-19 English Tweets. In *Proceedings of the 6th Workshop on Noisy User-generated Text*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Edinburgh neural machine translation systems for WMT 16. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 371–376, Berlin, Germany. Association for Computational Linguistics.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations*.

Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. 2019. Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv preprint arXiv:1909.01315*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Yinchuan Xu and Junlin Yang. 2019. Look again at the syntax: Relational graph convolutional network for gendered ambiguous pronoun resolution. *GeBNLP 2019*, page 96.