# Capstone Project: Data Augmentation in Forte

Yu Hao, Jianfeng Xia, Sixuan Chen

## 1 Introduction

Data augmentation (DA) are techniques that generate a large amount of annotated data from existing data or external data source. It has been widely used in computer vision areas, where augmented data can be easily generated by applying transformation to images. On the other hand, it has been underexplored for NLP areas. Recently, however, there have been great progress in NLP data augmentation, but these techniques have not been adopted by most machine learning libraries and thus require great amounts of effort to implement and test.

The goal of this project is to build an easy-to-use system for a wide range of data augmentation techniques for NLP. We implement the techniques in a generalized way such that they will be applicable to a wide range of tasks. Our contributions are incorporated in the Forte open source project. The code is available at `https://github.com/asyml/forte`.

Our contributions can be summarized into three parts. First, we designed and implemented a data augmentation framework based on Forte's Ontology annotation system. When the user performs data augmentation operations, our system automatically handles changes to the corresponding Ontology annotation data structures based on the transformations of the text.

Second, we provide efficient and easy-to-use implementations of a rich set of data augmentation algorithms, ranging from basic text transformations such as word replacement to advanced techniques such as the Unsupervised Data Augmentation (UDA)[8]. To the best of our knowledge, Forte is the first open-source NLP library to support these algorithms with easy-to-use interfaces.

Third, we conducted experiments on two of the advanced techniques: UDA and RL-based data augmentation. We present the results to demonstrate the effectiveness of these techniques and provide the code examples in the Forte library.

## 2 Processors

### 2.1 Forte's data structures

The data augmentation module is embedded in the Forte[4] open source project. Forte manages the text data with its own data structures. Typically, a Forte pipeline consists of a data reader that loads data from files, and following processors, each of which carries out an NLP operation, such as tokenization, POS-tagging or Named-Entity Recognition. The data flowing in the pipeline is wrapped in DataPacks, or MultiPacks, which contain multiple DataPacks.

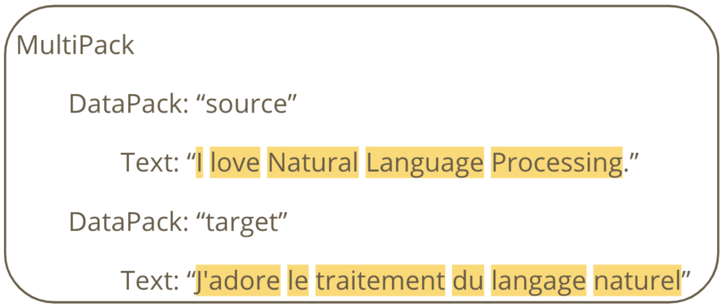

MultiPack
DataPack: “source”
Text: “I love Natural Language Processing.”
DataPack: “target”
Text: “J'adore le traitement du langage naturel”

Figure 1: An example of a Forte MultiPack.

Fig 1 is an example of a MultiPack for machine translation data. It consists of two DataPacks, one for the source language in English, and another one for the target language in French. Each DataPack has its text. For example, the text for source DataPack is ”I love Natural Language Processing.”

The Forte also has Entries, including Annotations, Links, Groups, MultiPackLinks and MultiPackGroups. They are attached to the text and can represent complex structures on the plain text.

The Annotation is a span of text. It is built on the underlying text of DataPack, identified by the start and end index of the span. The Annotation has several sub-classes, such as Token, Sentence and Document. In fig 1, each word with yellow background is a Token. And the whole text is a Sentence.

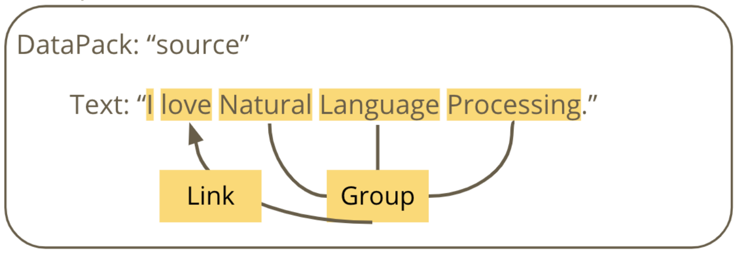


Figure 2: An example of a Link and Group.

Links and Groups can represent the relationship among multiple Entries. The Link is built between a parent and child Entry. It can represent syntactic structures, such as a dependency parsing tree. And the Group is a set of Entries. Fig 2 is an example of Links and Groups. The Group contains three Tokens: ”Natural”, ”Language” and ”Processing”. The Link points from the Group to the Token ”love”.

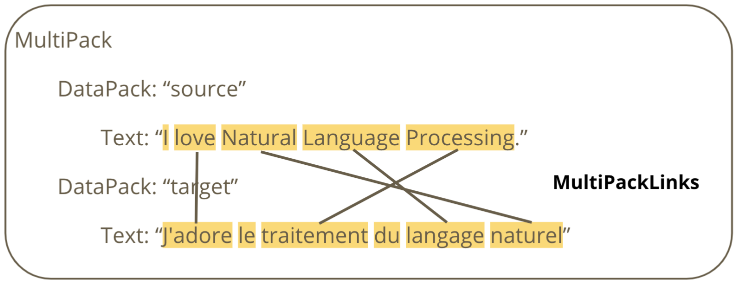

MultiPack
DataPack: “source”
Text: “I love Natural Language Processing.”
DataPack: “target”
MultiPackLinks
Text: “J'adore le traitement du langage naturel”

Figure 3: An example of MultiPackLinks.

The MultiPackLinks and MultiPackGroups can store the relationship among entries across different DataPacks. One of the common use cases for MultiPackLink, for example, is to represent the alignment in machine translation. As fig 3 shows, there are four MultiPackLinks between the ”source” and ”target” DataPack. They capture the 1-on-1 alignment between the two languages, for example, ”Natural” and ”naturel”. The MultiPackGroups work in a similar way as the Groups, but can have children entries across different DataPacks.

## 2.2 Processor

We build a processor for data augmentation that can be plugged into a Forte pipeline. The processor takes a MultiPack as input and outputs the same MultiPack with the original DataPacks in it and the augmented DataPacks. The processor will call data augmentation algorithms to get the augmented texts and create new DataPacks to hold them. It will automatically updates the underlying text and the Forte data structures(Entries).

Common operations of data augmentation includes replacement, insertion and deletion. The processor provides interfaces for the three types of operation and track them to update the entries. For Annotations, the start and end index are updated, because the positions might have been changed due to the augmentation. For Links and Groups, they will be copied and attached to the entries in new DataPacks. The MultiPackLinks and MultiPackGroups are processed similarly.

Fig 4 shows the augmented DataPack with the DataPack in fig 2 as input. We replace the ”I” with ”You”, ”love” with ”like”, delete the ”Processing” and insert a ”too”. After the process, the tokens now have new start and end index attached to the new text. The group now only has two children because one of them is deleted. The link now points to the token ”like”, which replaces the original token ”love”.

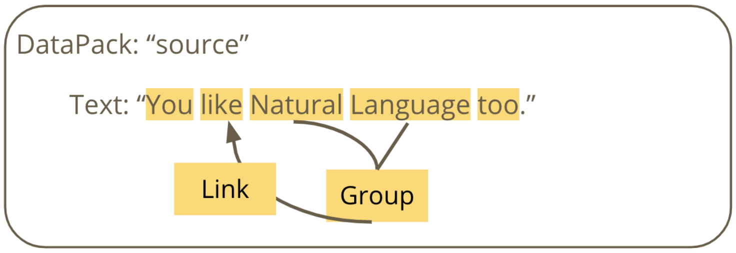


Figure 4: The augmented DataPack.

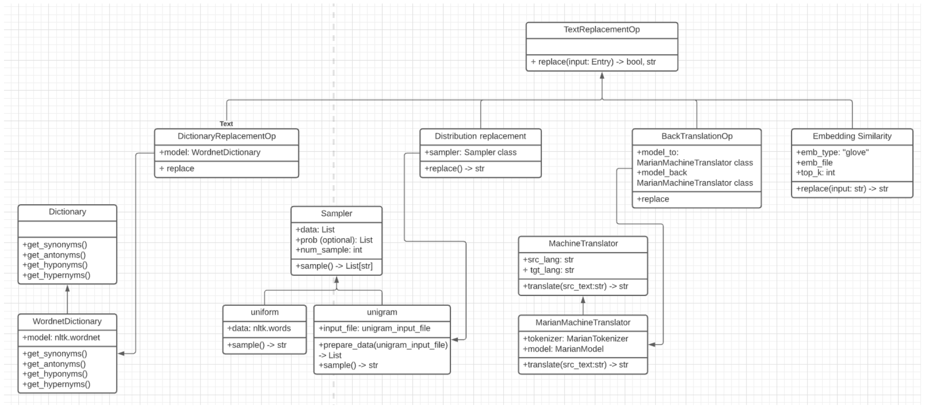


Figure 5: Replacement Ops

## 2.3 Data Selector

Data selector is another important component in our data augmentation system. It is used to pre-select a subset of instances from the training dataset, that may be considered more suitable for data augmentation. This data selector is then connected to the data augmentation processor in the pipeline to perform augmentation and the following tasks.

As illustrated in Figure 6, in order to select data from the training data, there is a database creator that stores all the data for retrieval. Then the selector performs retrieval from it. Specifically, we have two types of selector. The first is a random data selector that randomly select a given number of data from the database. This is useful to reduce the size of the training data to perform data augmentation in low data regime. The second one is an elastic search-based processor, whose selection criteria is a query-based search, where the user provides phrases or sentences as queries, then it performs similarity score computation using algorithms such as BM25 to rank all the documents in the database, and returns those documents according to their relevancy to the queries. In this way, the data selector selects relevant data that matches the input queries for augmentation in only a range of topics.

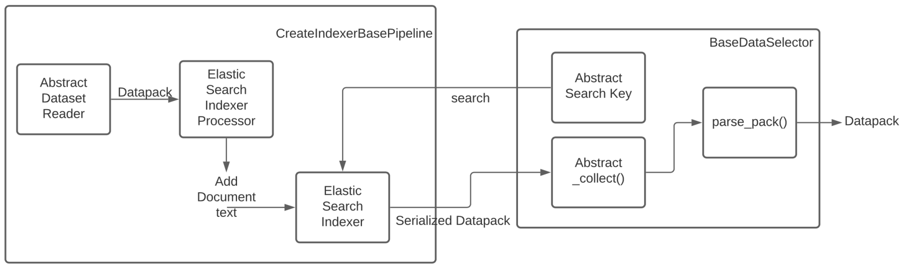


Figure 6: Workflow of data selector

## 2.4 Replacement Ops

Lots of data augmentation are based on text replacement. For example, we can replace a word with synonyms, or replace a sentence with back translation. So we implement the data augmentation algorithms as Replacement Ops. The input to the Ops is an Annotation, and the output is the augmented text string.

They only need to concern the algorithm and leave the manipulation of Forte data structures to the data augmentation processor. The processor will call the Replacement Op to perform the algorithm, and update the text and Entries as we described in section 2.2. We will discuss the Replacement Ops that we've implemented in detail later.

# 3 Algorithms

We have implemented several data augmentation methods, including the replacement-based approaches(Replacement Ops), Easy Data Augmentation(EDA), Unsupervised Data Augmentation(UDA) and Reinforcement-learning-based approach.

## 3.1 Replacement-based methods

The replacement-based methods are implemented as the Replacement Ops we mentioned in 2.4. Depending on the replacement level, we have word replacement and sentence replacement. Given an input sentence "I love NLP", the result of word replacement could be "I like NLP", if we replace the word "love" with "like". The result of sentence replacement could be "I like Natural Language Processing" if we replace the whole sentence.

For word replacement, we have three methods: dictionary-based, distribution-based and embedding-based methods. For sentence replacement, we implemented the back translation. 5 shows the architecture of the Ops.

### 3.1.1 Dictionary Replacement Op

This Op replaces the input word with its synonym, antonym, hypernym or hyponym. To acquire accurate outputs, POS-tags can be passed in to filter synsets. We implement a dictionary class that users could inherit from and use their own dictionary. We also provide a WordNet dictionary for the Op.

### 3.1.2 Distribution Replacement Op

This Op replaces the input word by sampling from a probability distribution over a vocabulary. For example, a uniform distribution indicates a uniformly random pick among the vocabulary, while a uni-gram distribution assigns different probability to different word in the vocabulary. Users can also implement other distributions to support much complex replacement, for example, N-gram probability distribution that can determine the output word with a language model.

### 3.1.3 Embedding Replacement Op

This Op replaces the input with word embeddings. With a set of word embeddings, it will search for similar words by calculating the cosine similarity between the embeddings, and return the top-k similar ones. The Glove[5] embedding is provided, and users can load their own word embeddings for the replacement.

### 3.1.4 Back Translation Op

Back Translation utilizes machine translation for sentence replacement. It translates the input from source language to target language, then translates it back to the source language. Under the assumption of decent translators, the output should have a similar semantic meaning as the input sentence, but possibly in a different presentation.

We wrapped the Marian MT[3] as the translator. It supports a large amount of language pairs, and thus suitable for back translation. Users can also use other translators.

## 3.2 Easy Data Augmentation

The Easy Data Augmentation (EDA)[6] is a set of simple random text transformations inspired by image transformations used in computer vision. They have been extensively tested and shown to substantially improve performance on text classification tasks while still being simple and efficient to implement. EDA consists of 4 different operations: Synonym Replacement (SR), Random Insertion (RI), Random Swap (RS) and Random Deletion (RD). The Synonym Replacement operation can be directly achieved by the Dictionary Replacement Op. The other 3 operations are implemented as separate Forte processors.

### 3.2.1 Random Insertion Processor

The Random Insertion Processor chooses a random token and randomly chooses one of its synonyms to insert to a random location in the input. This operation is repeated $N$ times where $N = \alpha L$, where $L$ is the length of the original input. $\alpha$ serves as a uniform parameter across the 4 operations to control the amount of data augmentation operations.

### 3.2.2 Random Swap Processor

The Random Swap Processor randomly chooses two tokens in the input and swaps there positions. This operations is repeated $N = \alpha L$ times.

### 3.2.3 Random Deletion Processor

The Random Deletion Processor randomly deletes each word in the input with a probability $p = \alpha$.

## 3.3 Unsupervised Data Augmentation

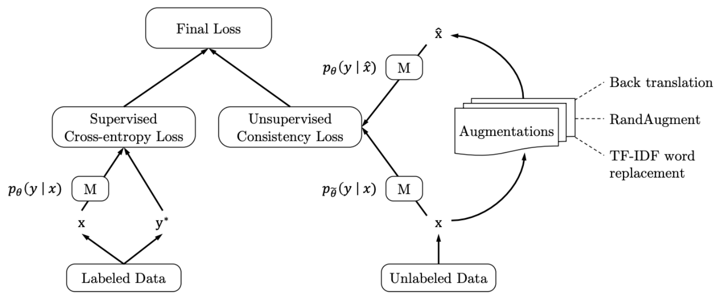


Figure 7: Unsupervised Data Augmentation

The Unsupervised Data Augmentation(UDA)[8] can utilize unsupervised data and incorporate the unsupervised loss function into supervised loss. As fig 7 shows, the final loss consists of two parts, the supervised and unsupervised loss. The supervised loss is the same as the original task, while the unsupervised loss comes from data augmentation.

We use traditional data augmentation techniques on unlabeled data $x$ to get the augmented data $\hat{x}$. Then the same model is applied to $x$ and $\hat{x}$. We expect the output probabilities $p_\theta(y|x)$ and $p_\theta(y|\hat{x})$ to be the same. In other words, the output should remain consistent after the augmentation because the augmentation does not change the semantic meaning of the input.

We implemented the UDA as a data iterator and a loss function. The data iterator combines supervised and unsupervised data within a training batch. After calculating the output probabilities, users may call the consistency loss function(Kullback-Leibler Divergence) to get the unsupervised loss, and add it to the final loss.

### 3.4 Reinforcement Learning based Data Augmentation

There are recent work that connects supervised learning and reinforcement learning. Specifically, it uses reinforcement learning technique to jointly learn the data augmentation model and train the downstream model[2]. Data augmentation is equivalent to different parameterization of the data reward. Data augmentation can be formulated as a reward function, and the objective of this model is to maximize this reward.

The typical maximum likelihood optimization is restricted to uniformly using only the exact training examples $D$. To enable data augmentation, we need to relax the strong restrictions of the reward function and instead use a relaxed reward $R_\phi(x, y|D)$ with parameters $\phi$. This relaxed reward then returns a valid reward even when x matches a data example only in part. Specifically, a sample receives reward of one only when it matches a training example in the dataset, otherwise the reward is negative infinite.

$$R_\phi(x, y|D) = \begin{cases} 1 & \text{if } x \sim g_\phi(x|x*, y), (x*, y) \in D, \\ -\infty & \text{otherwise .} \end{cases} \tag{1}$$

The parameters of the data augmentation model are trained by the EM procedure from reinforcement learning. Assume the data augmentation model parameters are $\phi$. assume the downstream model parameters are $\theta$. In each iteration, we first update the model parameters $\theta$ by optimizing the log-likelihood of the augmented training data.The next step is to optimize $\phi$ in terms of the model validation performance:

$$\theta'(\phi) = \theta - \nabla_\theta L_{train}(\theta, \phi) \tag{2}$$

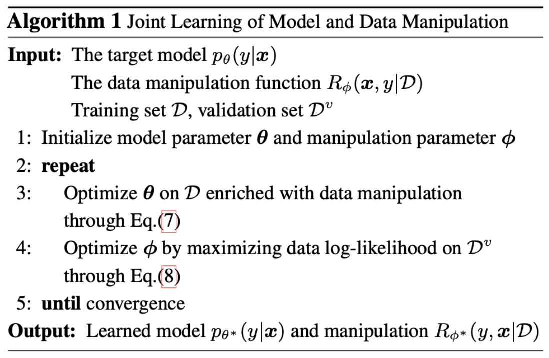
$$\phi = \phi - \nabla_\phi L_{val}(\theta'(\phi)) \tag{3}$$

This algorithm optimizes $\theta$ and $\phi$ alternatingly Figure 8. In this way, we can fine-tune the augmentation model together with the target model.

# 4 Experiment

### 4.1 Unsupervised Data Augmentation

We conduct our Unsupervised Data Augmentation experiment on the IMDB 50000 Movie Reviews dataset. We use a pre-trained BERT base[1] classifier as our baseline model. The IMDB dataset is split into 25000 training examples and 25000 testing examples. It also includes 75000 unlabeled movie reviews, which we will refer to as the unsupervised data. We first perform data augmentation on the unsupervised data using back translation. We use the same translation model as in the original work. The model is trained on both supervised setting



**Algorithm 1** Joint Learning of Model and Data Manipulation

```
Input: The target model $p_\theta(y|\boldsymbol{x})$
       The data manipulation function $R_\phi(\boldsymbol{x}, y|\mathcal{D})$
       Training set $\mathcal{D}$, validation set $\mathcal{D}^v$
1: Initialize model parameter $\boldsymbol{\theta}$ and manipulation parameter $\boldsymbol{\phi}$
2: repeat
3:     Optimize $\boldsymbol{\theta}$ on $\mathcal{D}$ enriched with data manipulation
       through Eq.(7)
4:     Optimize $\boldsymbol{\phi}$ by maximizing data log-likelihood on $\mathcal{D}^v$
       through Eq.(8)
5: until convergence
Output: Learned model $p_{\theta^*}(y|\boldsymbol{x})$ and manipulation $R_{\phi^*}(y, \boldsymbol{x}|\mathcal{D})$
```

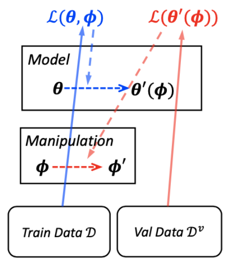

$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$
$\mathcal{L}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$
Model
$\boldsymbol{\theta}$
$\boldsymbol{\theta}'(\boldsymbol{\phi})$
Manipulation
$\boldsymbol{\phi}$
$\boldsymbol{\phi}'$
Train Data $\mathcal{D}$
Val Data $\mathcal{D}^v$

**Figure 1:** Algorithm Computation. Blue arrows denote learning model $\boldsymbol{\theta}$. Red arrows denote learning manipulation $\boldsymbol{\phi}$. Solid arrows denote forward pass. Dashed arrows denote backward pass and parameter updates.

Figure 8: Algorithm for reinforcement learning based data augmentation.

Table 1: Accuracy on IMDB Text Classification.

| Number of Labeled Examples | Max Sequence Length | BERT Base Accuracy | BERT Base + UDA Accuracy |
|---|---|---|---|
| 25000 | 128 | 89.68 | **90.19** |
| 24 | 128 | 61.54 | **84.92** |
| 24 | 512 | 63.88 | **92.54** |

and semi-supervised setting. In the supervised setting, we use the full 25000 labeled training examples for supervised training. In the semi-supervised setting, we randomly choose 24 examples (12 positive and 12 negative) for supervised training. We jointly optimize the supervised training loss and unsupervised consistency loss to train the model.

The experimental results are shown in Table 1. The models trained with UDA outperforms the baseline model in both supervised and semi-supervised settings. In the semi-supervised setting, with only 24 labeled examples, the UDA model outperforms the baseline model substantially and performs close to the models trained with full training dataset. The UDA is more effective in the semi-supervised setting as traditional training overfits with such a small training data size. With the full 25000 training examples, UDA is less effective but still outperforms the baseline model. The max sequence length parameter controls the length of the input sequence to the model. Because the IMDB dataset has very long examples, using longer input sequences naturally leads to better results. As shown in the table, when we increase the max sequence length from 128 to 512, the performance with or without UDA both increased. But interestingly, with a larger sequence length, the UDA makes a larger difference.

This experiment demonstrates UDA's ability to train the model to output predictions that are invariant to noise. The addition of UDA to the Forte NLP

Table 2: Accuracy of Data Augmentation on Text Classification.

| Model | Accuracy |
|---|---|
| pre-trained BERT [1] | 63.55 |
| Fixed BERT LM DA [7] | 63.55 |
| **RL BERT LM DA** | **65.12** |

toolkit makes it available to future NLP research while greatly reducing the implementation cost.

## 4.2 Reinforcement Learning based Data Augmentation

We use a BERT masked language model as the data augmentation model, a BERT classifier as the downstream task, and we adaptively tune the augmentation model with the classifier model. The baseline model is the BERT pre-trained model[1]. We also compare with another similar data augmentation model that uses a fixed conditional BERT language model for data augmentation [7].

Since this method is designed for low data regime, where only a few labeled training examples are available, we use 40 training examples for each class and 5 validation examples for each class. Experimenting with IMDB movie review dataset which has a binary label, there are in total 80 training data and 10 validation data, and we test with the entire 25000 test data.

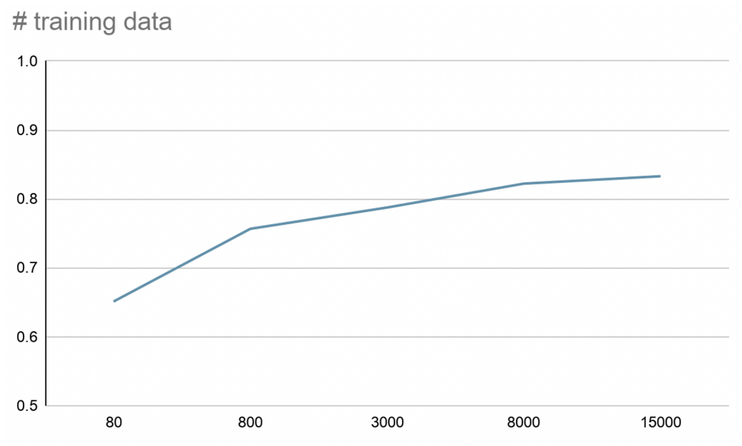


Figure 9: Effect of the Number of Training Data on Classifier Accuracy.

From the experiment results showed in Table 2, this RL-based fine-tuned augmentation model improves the classifier accuracy by 2%, which outperforms both baselines. This result proves that learning-based augmentation has the advantage of adaptively generating useful samples to improve model training.

When we increase the number of training data, the rate of the accuracy increment becomes slow as Figure 9 showed, indicating that data augmentation improves model performance better with a small amount of data available because it is equivalent to generating more training data.

## 5 Conclusion

In conclusion, we have studied and implemented many data augmentation techniques, including replacement-based methods, back translation, easy data augmentation, unsupervised data augmentation, reinforcement learning data augmentation. Some experiment have been conducted with the trainable DA models and the results shows that data augmentation can improve the performance of a downstream task, such as a BERT classifier. Additionally, We finished a data augmentation system in the Forte open source project, which creates a ready-to-use interface for all the above mentioned data augmentation techniques. All these pipelines and modules are very accessible to all users regardless of their knowledge in data augmentation. Following the code instructions and tutorials, users can easily wrap their task and perform data augmentation over theirs to see boost in performance. Last but not least, since we were contributing to an open source project, the code quality is high because we polished all the pull requests according to the reviews in order to merge it into the Forte code base. With our work, data augmentation is made available to all in Forte.

## References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL 2019*, 2019.

[2] Zhiting Hu, Bowen Tan, Ruslan Salakhutdinov, Tom Mitchell, and Eric P. Xing. Learning Data Manipulation for Augmentation and Weighting. In *NeurIPS 2019*, 2019.

[3] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[4] Zhengzhong Liu, Guanxiong Ding, Avinash Bukkittu, Mansi Gupta, Pengzhi Gao, Atif Ahmed, Zhang Shikun, Gao Xin, Swapnil Singhavi, Li Linwei, Wei Wei, Hu Zecong, Shi Haoran, Liang XIaodan, Teruko Mitamura, Eric P. Xing, and Hu Zhiting. A Data-Centric Framework for Composable NLP Workflows. In *EMNLP 2020*, 2020.

[5] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[6] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[7] X. Wu, S. Lv, L. Zang, J. Han, and S. Hu. Conditional bert contextual augmentation. *arXiv preprint arXiv:1812.06705*, 2018.

[8] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation for consistency training. *arXiv preprint arXiv:1904.12848*, 2019.