Learning Representations of Natural Language Edits via Levenshtein Prediction

Edison Marrese-Taylor[1,3], Machel Reid[2,3], Alfredo Solano[3]
[1]AIST, [2]Google Research, [3]The University of Tokyo
edison.marrese@aist.go.jp machelreid@google.com
asolano@weblab.t.u-tokyo.ac.jp

概要

In this paper, we propose a novel approach that employs token-level Levenshtein operations to learn a continuous latent space of vector representations to capture the underlying semantic information with regard to the document editing process. We find that, our proposed method trained on the adversarial paraphrase dataset, PAWS, outperforms a strong RoBERTa baseline, retaining the language understanding performance of its base model.

1 Introduction

Editing documents has become a pervasive component of many human activities [1]. This is, to some extent, explained by the advent of the electronic storage of documents, which has greatly increased the ease with which we can edit them.

From source code to text files, specially over an extended period of time, users often perform edits that reflect a similar underlying change. For example, software programmers often have to deal with the task of performing repetitive code edits to add new features, refactor, and fix bugs during development. On the other hand, right before a conference deadline technical papers worldwide are finalized and polished, often involving common fixes for grammar, clarity, and style [2]. In light of this, it is reasonable to wonder if it would be possible to automatically extract rules from these common edits. This has led researchers to recently propose the task of learning distributed representations of edits [2] using an auto-encoding approach.

Auto-encoding approaches have been used previously in the context of representation learning initially in the visual domain, but more recently have been extended to the natural language and video modalities. These approaches largely form the foundation of "self-supervised learning" which enables the learning of representations via objectives which solely require a source datum. An instance of this relevant to NLP is that of the pre-trained masked language model, BERT [3], in which a source text is initially corrupted with a mask token [MASK] and then reconstructed into the original form with a Transformer encoder.

As an alternative to this approach, other works have instead produced representations of edits in an indirect manner, by instead focusing directly on edit-centric downstream tasks such as edit-based article quality estimation on Wikipedia [4, 5], English grammatical error correction (GEC), and machine translation post-editing.

Machine translation post-editing, where humans amend machine-generated translations to achieve a better final product [6], has more directly addressed the problem of modelling different editing agents [7] Finally, edits are also relevant for GEC, which has attracted recent interest from the research community with several shared tasks being organized in the last years [8].

In this paper, differently from existing prior work, we propose a continued pre-training task, not based on auto-encoding, which aims directly at learning distributed representations of natural language edits. In particular, we look at using the Levenshtein algorithm as a form of supervision to encourage a model to learn to convert a given input sequence into a desired output sequence, namely an edit. In particular, we look to answer the question of whether creating a "neural Levenshtein algorithm" is conducive to improved downstream performance on edit-based tasks, given the edit-centricity of the algorithm.

Our results show that our proposed pre-training technique leads to better performance on the adversarial para-

This work is licensed by the author(s) under CC BY 4.0
(https://creativecommons.org/licenses/by/4.0/).

phrase dataset PAWS, outperforming a strong baseline based on RoBERTa [9] while also retaining the language understanding performance of its base model (RoBERTa).

2 Related Work

[2] was the seminal work in proposing to directly learn edit representations by means of a task specifically designed for this purpose, based on auto-encoding. While their ideas were tested on both source code and natural language edits, the work of [10] proposed a similar approach that is specifically tailored at source code. After that, [11] proposed a variation of this model where a latent variable is introduced as a means to capture properties of Natural Language Edits. Additionally, the authors used the obtained latent vectors to represent edits and tested on a selection or edit-centric tasks.

Different from the above, other works have instead produced representations of edits in an indirect manner, by directly focusing on specific edit-centric downstream tasks. For example, [4] proposed obtaining edit representations that are useful to predict changes in the quality of articles by tackling this task as an edit-level classification problem. Similarly, [5] proposed to improve quality assessment of Wikipedia articles by introducing a model that jointly predicts the quality of a given Wikipedia edit and generates a description of it in natural language. Another related task is machine translation post-editing, where humans amend machine-generated translations to achieve a better final product, where recent work has more directly addressed the problem of modelling different editing agents [7]. Finally, edits are also relevant for English grammatical error correction (GEC), which has attracted recent interest from the research community with several shared tasks being organized in the last years [8].

3 Proposed Approach

We propose a new pre-training task based on self-supervision. In particular, we look at using the Levenshtein algorithm as a means of pushing a model to learn to convert a given input sequence into a desired output sequence.

The Levenshtein algorithm has been used in previous work on learning edit-based generative models [12, 13] thanks to its objective of finding the shortest edit path from a given source sequence to a target sequence. However, instead of purposing the algorithm for generation, in this work we look to see if we can include this objective from a natural language understanding (NLU) perspective. In particular, we look to assess whether Transformer encoder representations can be trained to contain information relevant to an edit, which we hypothesize can be achieved by directly predicting relevant operations and their associated tokens — as produced by an oracle Levenshtein algorithm.

Let x_- be the original version of an object, and x_+ its form after a change/edit has been applied. We assume that both x_- and x_+ are sequences of tokens such that $x_- = [x_-^1, \ldots, x_-^n]$ and $x_+ = [x_+^1, \ldots, x_+^m]$. We use a fast implementation of the Levenshtein algorithm to identify spans of tokens that have been replaced, inserted or deleted as a result of the edit, and define token-level edit operation labels to indicate how each token was changed.

Concretely, we first tokenize the pair (x_-, x_+), then use the Levenshtein algorithm to identify the text spans that have changed, and finally further process this output to assign token-level labels capturing the transformations required to convert x_- into x_+. Let $x_-^{i:j}$ be the sub-span on x_- that goes from positions i to j, our post-processing works as follows.

- When a span has been inserted between positions $x_-^{i:j}$, such that it appears in $x_-^{k:j}$, we label the tokens in the latter as INSERTER, and also label token x_-^{k-1}, as INSERT. We do this to provide the model with context of where the insertion was performed, in terms of the x_-.
- Similarly, if the token $x_-^{i:j}$ has been replaced by the span $x_-^{k:l}$, we label the tokens on the respective spans as REPLACE and REPLACER.
- If the span $x_-^{i:j}$ has been removed from the sequence as a result of the edit, we label each token as DELETE.
- Tokens that have not been directly involved in the edit are not labeled (we represent this using the empty label denoted as O).

As a result of our post-processing, each token in both x_- and x_+ is mapped to a single Levenshtein operation label: REPLACE, REPLACER, INSERT or INSERTER, as shown in Figure 1. The end goal of our proposed pre-training task is to predict these token-level Levenshtein operations, which, as explained, encode the operations relevant to transform x_- into x_+.

The input to our model is constructed by first prepending

This work is licensed by the author(s) under CC BY 4.0
(https://creativecommons.org/licenses/by/4.0/).

図 1 Example of model input-output for the edit defined by the sequences "My name is John" and " My last name is Wayne", where the label O denotes tokens that have not been directly involved in the edit.

[CLS]	My	name	is	John	[SEP]	My	last	name	is	Wayne
O	INSERT	O	O	REPLACE	O	O	INSERTER	O	O	REPLACER

the [CLS] token to x_- and x_+, which are separated using the [SEP] token, whose total length we denote as $l = m+n+2$. This input is embedded and then fed to an L-layer Transformer encoder, which can be initialized with a pre-trained model, that returns a sequence of hidden representations $\boldsymbol{h}_0, \ldots \boldsymbol{h}_l$. We add a classification head (a simple linear classifier) and require the model to predict the corresponding label for each token, ignoring tokens that have not been directly involved in the edit (label O), using a cross entropy loss ($\mathcal{L}_{Lev}$).

We also consider an additional mechanism to enrich the quality of the learned representations, which is based on techniques that have proven useful in previous work [11]. Concretely, we note that the vector associated to the [CLS] token ($\boldsymbol{h}_0$) is frequently used to represent the complete model input when using Transformer models such as ours. Since there is no specific token-level Levenshtein label associated to this token, we propose to encourage its representation to contain information about the overall edit. We do this by requiring our model to predict the set of tokens that have been changed in the edit in an unordered fashion, using a separate model head (again, simple linear projection) which receives this representation as input.

$$f = \text{MLP}(\boldsymbol{h}_0) \in \mathbb{R}^{|\mathbb{V}|} \tag{1}$$

As shown in Equation 1 above, we use this additional head to project $\boldsymbol{h}_0$, the hidden representation for the [CLS] token, to $|\mathbb{V}|$, the vocabulary size. We then let our model minimize the loss function defined in Equation 2, where x_Δ is the set of tokens that has been changed (inserted, replaced or removed). Finally, the total loss used to train our model is the simple summation of the above introduced losses, $\mathcal{L} = \mathcal{L}_{Lev} + \mathcal{L}_{x_\Delta}$.

$$\mathcal{L}_{x_\Delta} := \log p(x_\Delta|\boldsymbol{h}_0) = \log \prod_{t=1}^{|x_\Delta|} \frac{\exp(f_{x_t})}{\sum_j^V \exp(f_j)} \tag{2}$$

4 Experimental Setup

Pre-training Datasets To pre-train our model, we utilize large available corpora containing natural language edits in a variety of topics and domains. We specifically rely on two datasets of edits extracted from Wikipedia, WikiAtomicEdits [14] and WikiEditsMix [11]. As shown in Table 1, we specifically work with two sub-portions of the WikiAtomicEdits, WikiInsertions and WikiDeletions, which respectively contain examples where only additions or deletions are present. These two portions are concatenated and used as a whole. Since pre-training is computationally very expensive, we first use WikiEditsMix, which is much smaller, as a test-bed for ablation experiments regarding the x_Δ loss.

表 1 Details of the data utilized for pre-training.

Dataset	Num. Edits	Avg. Len
WikiInsertions [14]	13.7M	24.5
WikiDeletions [14]	9.3M	25.1
WikiEditsMix [11]	114K	61.6

Downstream Tasks Our pre-training approach aims at generating a generic edit encoder that is useful in a broad variety of situations involving edits. To that end, we select specific datasets to probe the ability of the model to solve edit-related tasks, and also to make sure that our training procedure does not lead to catastrophic forgetting, making the model lose the utility of the representations acquired during the original masked language model's pre-training. For the former, we propose to use PAWS [15], an adversarial dataset for paraphrasing detection, which is strongly correlated to edits, as paraphrases are defined as sentences that are semantically similar to each other. We test both fine-tuning and zero-shot abilities on this model. For the latter, we look at the widely-used GLUE benchmark [16] and select the MNLI dataset to test that language entailment capabilities remain.

Evaluation For evaluation of our pre-training phase, we utilize the per-token classification F1-Score, and also compute the overall F1-Score. Regarding the downstream tasks, we use accuracy, which is the de facto metric for both MNLI and PAWS.

Implementation Details Our model is initialized with RoBERTa [9], which we adopt as our baselines for all of our downstream experiments. We use fairseq [17] to implement our model and perform distributed pre-training

This work is licensed by the author(s) under CC BY 4.0
(https://creativecommons.org/licenses/by/4.0/).

表 2 Performance of our model on our pre-training Levenshtein Prediction task.

Model	Dataset	Performance (F1-score)					
		All	INSERT	INSERTER	REPLACE	REPLACER	DELETE
Full	WIKIEDITSMIX	91.2	87.8	95.6	89.9	88.7	93.5
No x_Δ	WIKIEDITSMIX	91.8	89.4	96.1	90.6	88.6	93.7
Full	WIKIINSERTIONS	79.8	99.9	100	99.8	99.4	-

using 16 NVIDIA V100-16G GPUs, and fine-tuning with a single GPU. We access these by means of nodes on a large cluster, where each node has four GPUs. We use the Adam optimizer with a learning rate of 1e-4 for pre-training, and 1e-3 for fine-tuning on the downstream tasks.

5 Experiments and Results

As can be seen in Table 2, all of our models attain an excellent performance on the pre-training task, with an overall F1-Score of more than 90% across edit labels. We believe this shows our the encoder is capable of successfully predicting the relevant operations generated by our oracle Levenshtein editor, suggesting that the learned representations may indeed contain information relevant to the changes that are introduced.

Regarding the impact of our proposed x_Δ loss during pre-training, as seen on our ablation results performance in Table 2, we see that x_Δ as a positive impact on the overall performance of the model in the WIKIEDITSMIX dataset increasing the F1-Score my more than 0.5 points. This result is consistent with previous work, validating the contribution of this loss applied to our setting. In light of this finding, our final model includes both of our proposed losses and is pre-trained on WIKIINSERTIONS.

Table 3 summarizes our results on the downstream tasks. As can be seen, when fine-tuned, our model is able to outperform our strong RoBERTa baseline in the adversarial paraphrase dataset, PAWS, suggesting that the representations induced by our Levenshtein prediction loss indeed help the model learn relevant information about edits. Moreover, we also observe that the model is able to attain such increased performance while still retaining the language understanding capabilities of its base model, as suggested by the performance on the MNLP dataset, which remains constant.

表 3 Performance of our model and baseline based on RoBERTa on our selected fine-tuning tasks, where ZS stands for Zero Shot.

Dataset	Model	Accuracy
MNLI	RoBERTa	**87.6**
	EARL	**87.6**
PAWS	RoBERTa	90.6
	EARL	**94.7**
PAWS ZS	RoBERTa	**55.8**
	EARL	44.2

6 Conclusions and Future Work

This paper proposes a novel approach for training a general-purpose edit representation model, which is not based on auto-encoding. Concretely, we propose a predictive task based on token-level Levenshtein operations where the token-level labels encode the set of operations necessary to transform a given input sentence into an output sentence.

We find that a model initialized with RoBERTa and trained with our proposed loss, is able to outperform the baseline on the task of adversarial paraphrasing detection on PAWS, while retaining the language understanding performance of its base model. We think this evidence supports the idea that creating a neural model that implements the Levenshtein algorithm is conducive to improved downstream performance on edit-based tasks, suggesting a potential new path for the future of pre-training. For future work, we are interested in further testing our pre-trained model on more downstream tasks relevant to edits, and in combining our proposed loss with the masked language modelling task for more efficient and effective training.

This work is licensed by the author(s) under CC BY 4.0
(https://creativecommons.org/licenses/by/4.0/).

Acknowlegments

Computational resource of AI Bridging Cloud Infrastructure (ABCI) provided by National Institute of Advanced Industrial Science and Technology (AIST) was used for the experiments in this paper.

参考文献

[1] Anders Miltner, Sumit Gulwani, Vu Le, Alan Leung, Arjun Radhakrishna, Gustavo Soares, Ashish Tiwari, and Abhishek Udupa. On the fly synthesis of edit suggestions. **Proceedings of the ACM on Programming Languages**, Vol. 3, No. OOPSLA, pp. 143:1–143:29, October 2019.

[2] Pengcheng Yin, Graham Neubig, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. Learning to Represent Edits. In **Proceedings of the 7th International Conference on Learning Representations**, 2019.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)**, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[4] Soumya Sarkar, Bhanu Prakash Reddy, Sandipan Sikdar, and Animesh Mukherjee. StRE: Self Attentive Edit Quality Prediction in Wikipedia. In **Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics**, pp. 3962–3972, Florence, Italy, July 2019. Association for Computational Linguistics.

[5] Edison Marrese-Taylor, Pablo Loyola, and Yutaka Matsuo. An Edit-centric Approach for Wikipedia Article Quality Assessment. In **Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)**, pp. 381–386, Hong Kong, China, November 2019. Association for Computational Linguistics.

[6] L. Specia, K. Harris, A. Burchardt, M. Turchi, M. Negri, and I. Skadina. Translation Quality and Productivity: A Study on Rich Morphology Languages. pp. 55–71, 2017.

[7] António Góis and André F. T. Martins. Translator2Vec: Understanding and Representing Human Post-Editors. In **Proceedings of Machine Translation Summit XVII Volume 1: Research Track**, pp. 43–54, Dublin, Ireland, August 2019. European Association for Machine Translation.

[8] Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. The BEA-2019 Shared Task on Grammatical Error Correction. In **Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications**, pp. 52–75, Florence, Italy, August 2019. Association for Computational Linguistics.

[9] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. **arXiv:1907.11692 [cs]**, July 2019.

[10] Rui Zhao, David Bieber, Kevin Swersky, and Daniel Tarlow. Neural Networks for Modeling Source Code Edits. In **Proceedings of the 7th International Conference on Learning Representations**, 2019.

[11] Edison Marrese-Taylor, Machel Reid, and Yutaka Matsuo. Variational Inference for Learning Representations of Natural Language Edits. **Proceedings of the AAAI Conference on Artificial Intelligence**, Vol. 35, No. 15, pp. 13552–13560, May 2021.

[12] Machel Reid and Victor Zhong. LEWIS: Levenshtein Editing for Unsupervised Text Style Transfer. In **Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021**, pp. 3932–3944, Online, August 2021. Association for Computational Linguistics.

[13] Nabil Hossain, Marjan Ghazvininejad, and Luke Zettlemoyer. Simple and Effective Retrieve-Edit-Rerank Text Generation. In **Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics**, pp. 2532–2538, Online, July 2020. Association for Computational Linguistics.

[14] Manaal Faruqui, Ellie Pavlick, Ian Tenney, and Dipanjan Das. WikiAtomicEdits: A multilingual corpus of Wikipedia edits for modeling language and discourse. In **Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing**, pp. 305–315, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[15] Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. PAWS-X: A cross-lingual adversarial dataset for paraphrase identification. 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)**, pp. 3687–3692, Hong Kong, China, November 2019. Association for Computational Linguistics.

[16] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In **Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP**, pp. 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[17] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. Fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)**, pp. 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

This work is licensed by the author(s) under CC BY 4.0
(https://creativecommons.org/licenses/by/4.0/).