自然言語とソースコード間の対訳コーパス向け Data Augmentation 手法の提案

秋信 有花 [1]　小原 百々雅 [2]　梶浦 照乃 [2]　倉光 君郎 [2]
[1] 日本女子大学大学院 理学研究科　[2] 日本女子大学 理学部
m1616003ay@ug.jwu.ac.jp kuramitsuk@fc.jwu.ac.jp

概要

本論文では，自然言語とソースコード間の対訳コーパス向けの Data Augmentation (DA) 手法と，それを実現する DA ツール Multiese を提案する．近年，深層学習ベースのコーディング支援は関心を集めるテーマとなっているが，学習データとなる NL-code 対訳コーパスの不足は重要な課題である．本研究では，構文木への変換が可能であるというソースコードの形式的な性質を活用した DA 手法を新たに提案する．実験では，Multiese によって増強された NL-code 対訳コーパスを用いることで，自然言語からソースコードを生成するコード生成 AI モデルを，高い精度で構築可能であることを示した．

1　はじめに

社会のデジタル化が進む昨今，プログラミングはより身近な技能へとなってきている．しかし，プログラミングは誰もが自由自在にできるものではなく，プログラムの書き方がわからず，挫折してしまう人も少なくない．そこで，我々は，自然言語を用いたプログラミング支援の実現に向け，日本語記述から Python コードを生成するニューラル機械翻訳モデルであるコード生成 AI モデルの構築に取り組んでいる [1, 2]．ユーザは，プログラムの書き方が思い出せない箇所をコード生成 AI モデルに対して自然言語で与えることで，それに相当する正しいコードを得ることができる．

しかし，コード生成 AI モデルの学習データとなるソースコードとその対訳をペアとした NL-code 対訳コーパスのデータ資源は限られている [3]．先行研究では，ソースコードのデータ資源として，GitHub などのオープンソース上のソースコードが使われてきたが [4, 5]，これらには対訳などは含まれていない．ソースコードの対訳は，ソースコード内のコメントとして付与されることもあるが，その対訳の粒度はコーダーによって様々である．そのため，これらのコメントを抽出し，対訳としてソースコードとマッピングするだけでは，NL-code 対訳コーパスとして不十分である．また，整備済みの NL-code 対訳コーパスもいくつか公開されているが，これらのほとんどは人手で対訳が付与されたものである．そのため，データセット数は数百から数千程度と限られており，深層学習のネットワークモデルに直接適用することは難しい．

そこで，本研究では，NL-code 対訳コーパスの不足を解消すべく，自然言語とソースコード間の対訳コーパス向けの Data Augmentation (DA) 手法を提案する．我々の DA 手法の特徴は，プログラミング言語の構文に基づくソースコードの合成と DA 記法の導入による自然言語表現の増強である．我々は，この二つの特徴を実現する DA ツール Multiese を実装し，人手で作成した NL-code 対訳コーパスのデータセット数を 177 ペアから 14,721 ペアまで増強させることができた．また，実験では，Multiese によって増強された NL-code 対訳コーパスを用いてコード生成 AI モデルを学習し，NL-code 対訳コーパスの合成によるコード生成 AI モデルの精度の変化を確かめた．

本論文では，NL-code 対訳コーパス向け DA 手法と DA ツール Multiese を提案し，実験結果とともに報告する．本論文の残りの構成は以下の通りである．2 節では提案する NL-code 対訳コーパス向け DA 手法について述べる．3 節では実装した DA ツール Multiese について述べる．4 節では実験について述べる．5 節では関連研究を概観し，6 節で本論文をまとめる．

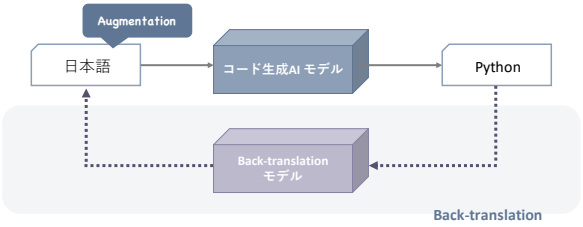

図 1　コード生成 AI モデルと back-translation

2　NL-code 対訳コーパス向け DA

Data Augmentation (DA) は，学習データに加工などの処理を部分的に施すことで，データ量を水増しする手法である．本節では，本研究で着目した DA 手法の一つである back-translation について概説したのち，本研究の提案である NL-code 対訳コーパス向け DA 手法について述べる．

2.1　Back-translation

Back-translation は，機械翻訳向けの DA 手法である．Back-translation では，目的とする翻訳モデルのソース言語とターゲット言語を逆にした逆翻訳モデルを別途学習する．学習された逆翻訳モデルに対して，ターゲット言語の単言語コーパスを入力し，ソース言語の対訳を生成することで，擬似的な対訳コーパスを構築する．コード生成 AI モデルに対して back-translation をナイーブに適用すると，図 1 に示すように，Python コードから日本語記述を生成する back-translation モデルを別途学習することになる．一般的な back-translation と異なる点は，逆翻訳モデルのソース言語が，木構造への変換が可能なプログラミング言語になることである．

2.2　NL-code 対訳コーパス向け DA 手法

本研究では，逆翻訳モデルのソース言語がプログラミング言語である点に着目し，プログラミング言語の性質を活用した DA 手法を新たに提案する．図 2 は，NL-code 対訳コーパス向け DA 手法のコンセプトを示したものである．提案手法の特徴は，ソースコードの任意長の expression 片に対して与えられた対訳から，ボトムアップで NL-code 対訳コーパスを合成する点である．

NL-code 対訳コーパス向け DA 手法では，まず人手で複数の日本語訳を付与した NL-code 対訳コーパスを用意する．このとき，既存の対訳コーパスを再利用することも可能である．

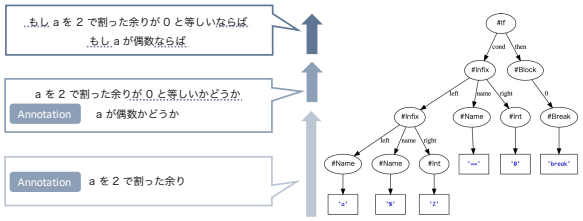

図 2　NL-code 対訳コーパス向け DA 手法

表 1　論理演算式の拡張例

	Python コード	NL 記述
	`x % y == 0`	x を y で割った余りが 0 かどうか
@@not	`not x % y == 0:`	もし x を y で割った余りが 0 でないかどうか
@@if	`if x % y == 0`	もし x を y で割った余りが 0 ならば
@@not.if	`if x % y != 0`	もし x を y で割った余りが 0 でないならば

`x % n`
x を n で割った余り

`x % 2 == 0`
x が偶数かどうか
x は 2 で割り切れるかどうか

次に，用意した NL-code 対訳コーパスに対して，プログラミング言語の文法に基づくアノテーション記法を導入する．Expression 片の前後に来る可能性のある構文を `@@not` アノテーションのように与える．

`x % 2 == 0`　`@@not @@if`
x が偶数かどうか
x は 2 で割り切れるかどうか

すると，アノテーションに基づき，表 1 のように前後の文言が生成される．提案手法では，最初に与えられた expression 片からボトムアップで対訳を合成することになるが，木構造のトップに相当するコードまでは生成しない．これは，Transformer などのネットワークモデルでは，複雑な木構造を扱うことが困難であるという考察に基づいている [6]．

提案手法のもう一つの特徴は，DA 記法の導入による，自然言語表現の増強である．はじめに用意した NL-code 対訳コーパスに対して，以下のような記法を追加することで，自然言語の表現を拡張できるようにしている．

`x % 2 == 0`　`@@not @@if`
@type(x, 変数) が偶数 [かどうか|であるかどうか]
x は 2 で [割り切れる|割ることができる] かどうか

上記の例には，二種類の DA 記法を含んでいる．

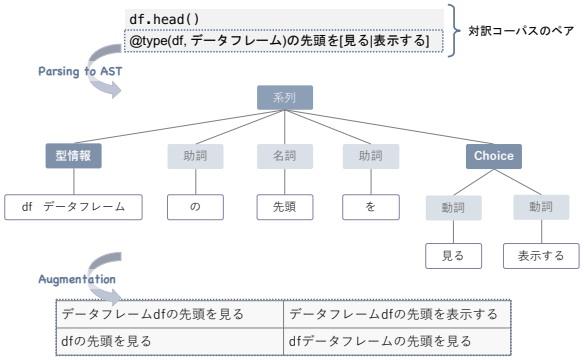

図 3　Multiese による日本語 Data Augmentation

一つ目は，類義語置換である．[**かどうか|であるかどうか**] のように定義された複数の類義語を組み合わせることで，複数の自然言語記述を生成する．二つ目は，変数名やリテラルに対する型情報の付与である．`@type(x, 変数)` のように変数名とその型情報をペアとして明示することで，変数名の前後に型情報を付与，もしくは付与しないパターンを生成する．DA 記法を付与する手間はかかるものの，複数の DA 記法を組み合わせることで，様々な表現パターンへの拡張が実現できる．

3　DA ツール：Multiese

Multiese は，NL-code 対訳コーパス向け DA 手法を実現するための DA ツールである．図 3 は，Multiese が NL-code 対訳コーパスを合成するまでの流れを示したものである．まず，DA 記法を含んだ自然言語記述は，抽象構文木 (AST) へと変換される．その後，構文木のタグ情報に基づき，複数の自然言語表現をルールベースで生成する．最後に，それらを元のソースコードとペアにし，NL-code 対訳コーパスとして出力する．

抽象構文木への変換は，PEG (Parsing Expression Grammar)[7] パーザジェネレータ PEGTree[8] と，形態素解析器 Janome を組み合わせることで実装している．PEGTree で DA 記法に基づいた構文木に変換した後，Janome で品詞タグを付与する．品詞タグを付与することで，語順の入れ替えや助詞の一部変更などの DA 処理を，DA 記法なしに実現している．

4　実験

我々は，Multiese によって増強された NL-code 対訳コーパスを用いてコード生成 AI モデルを構築し，コードの生成例やその精度を確かめた．本節では，コード生成 AI モデル構築の流れとその結果についてまとめる．

表 2　実験に使用したデータセット

コーパス名	DA 手法の適用	学習データ数
Euler	×	210
AOJ	×	694
AOJ-DA	✓	5,199
DS	×	177
DS-DA	✓	14,721

4.1　データセットと前処理

我々は 3 種類の人手で作成した NL-code 対訳コーパスを用意し，そのうちの 2 種類に対して NL-code 対訳コーパス向け DA 手法を適用した．表 2 は，実験で使用した NL-code 対訳コーパスをまとめたものである．Euler[1] は，"Project Euler4" の問題を解くための Python コードに対し，人手で日本語対訳が付与された，Web 上で公開されているデータセットである [9]．AOJ と DS は，コード生成 AI モデルの学習にあたり，我々が人手で作成した NL-code 対訳コーパスである．AOJ は，Aizu Online Judge[2] に提出された Python コードのうち，成績上位 100 人の公開コードを全て収集し，expression 単位のコード片に対して日本語訳を付与したデータセットである．DS は，データサイエンスに関するテキスト教材から，pandas や matplotlib などデータ分析でよく使用されるライブラリの API を採取し，人手で対訳を付与したデータセットである．AOJ-DA と DS-DA は，AOJ，DS に対してアノテーションや DA 記法を加え，Multiese によってデータセット数を増強したものである．

コード生成 AI モデル学習前には，トークナイズと，変数名・リテラルの特殊トークン化への置き換えを前処理として行った．トークナイザは，日本語には Janome を使用し，Python コードはカッコやピリオドなどの記号の前後に空白を挿入した．

4.2　コード生成 AI モデルの学習

コード生成 AI モデルの学習には，Transformer を使用した．Transformer の各種パラメータは，エンコーダとデコーダの層数をそれぞれ 6，ヘッド数を 8，バッチサイズを 128，単語エンベッディング次元数と隠れ層次元数をそれぞれ 512 とした．また，損失関数にはクロスエントロピー，オプティマイザに

1）　https://ahcweb01.naist.jp/pseudogen/
2）　https://judge.u-aizu.ac.jp/

表 3　コードの生成例

	入力文	正解コード	生成コード	正誤
Euler	もし<A>が<B>より小さい場合	`if <A> < <B> :`	`if <A> < <B> :`	✓
	<A>の末尾に<B>を追加する	`<A> . append ( <B> )`	`while <A> . append ( <B> )`	×
AOJ	<A>が有限かどうかを求める	`math . isfinite ( <A> )`	`math . isfinite ( <A> )`	✓
	ユーザから入力を数字として受け取られた結果を<A>に代入する	`<A> = int ( input ( ) )`	`sorted ( <A> , input ( ) )`	×
AOJ-DA	<A>の余分な改行を除去する	`<A> . strip ( )`	`<A> . strip ( )`	✓
	入力されたデータをリストに突っ込む	`list ( input ( ) )`	`list ( input ( ) )`	✓
DS	<A>の先頭5行を表示する	`<A> . head ( )`	`<A> . head ( )`	✓
	<A>の形状を取得する	`<A> . dtypes`	`from <A> [ <A> . shape`	×
DS-DA	データフレーム<A>をインデックスで順にソートする	`<A> . sort_index ( )`	`<A> . sort_index ( )`	✓
	<A>データフレーム中の<B>を未記入の値に変え未記入の値がある行を落として，置き換える	`<A> = <A> . replace ( <B> , np . nan ) . dropna ( )`	`<A> . replace ( <B> , np . nan ) . dropna ( inplace = True )`	✓

は Adam を使用した．学習回数は，検証用データの損失値が最小となった時点で打ち切りとした．

4.3　実験結果 (1)：コードの生成例

表 3 は，ホールドアウト法で作成したテストデータの入力文とその正解コード，コード生成 AI モデルによって生成されたコード，それに対する正誤をまとめたものである．はじめに，入力文の自然言語表現を確認すると，AOJ-DA，DS-DA 共に，Multiese によって自然な文章を合成できたことが確認できる．次に，正解コードと生成コードを比べると，DA 適用前の NL-code 対訳コーパスを使用した場合の生成コードは，API 名や構文が間違っているものが多く含まれる．一方，AOJ-DA や DS-DA を使用した場合の生成コードは，AOJ や DS と比べると間違いも少なく，様々な種類の Python 構文の生成を実現した．

4.4　実験結果 (2)：コードの正解率

表 4 はコード生成 AI モデルから生成されたコードの BLEU スコアと正解率をまとめたものである．正解率は，コード生成 AI モデルから生成されたコードと正解コードを比較し，構文的にも解釈的にも正しいコードを人手によって検証し，その比率を示している．テストデータが多い場合は，サンプリングによる統計的解析を行っている．AOJ と AOJ-DA，DS と DS-DA を比較すると，Multiese によって増強した NL-code 対訳コーパスを使用することで，高い精度のコード生成 AI モデルの構築を実現できたことが確認できる．

5　関連研究

Data Augmentation は，深層学習モデルの精度向上に向けて，学習データを水増しする手法である．DA は，元々画像処理の分野で研究されてきた手法であったが [10, 11]，近年では，自然言語処理や音声処理などの様々な分野でも取り入れられている [12]．

表 4　生成されたコードの正解率

コーパス名	学習データ数	BLEU	正解率
Euler	210	72.01	8.70%
AOJ	694	68.41	23.65%
AOJ-DA	5,199	75.00	38.03%
DS	177	65.83	7.89%
DS-DA	14,721	77.87	32.70%

自然言語処理における DA 手法の先行研究として，低頻度語や，対訳コーパスのソース文とターゲット文の両方に含まれる語彙を，別の単語で置き換える手法が提案されている．我々の提案手法の特色は，DA 手法をソースコードに対して適用した点と，自然言語表現を多様化するためのルールベースの DA 記法を取り入れた点である．我々の手法は，ソースコードの木構造に変換可能な点を活かした新たなアプローチであるといえる．

6　むすびに

本研究では，NL-code 対訳コーパスの不足を解消すべく，自然言語とソースコード間の対訳コーパス向けの Data Augmentation (DA) 手法を提案した．提案手法を用いることで，人手で作成した NL-code 対訳コーパスのデータ数を 177 ペアから 14,721 ペアまでに増強させた．また，増強された NL-code 対訳コーパスを用いることで，高い精度のコード生成 AI モデルの構築を実現した．

今後は，大きく分けて二つの課題に取り組んでいきたいと考えている．一つ目は，DA 記法を必要としない DA 処理の実現である．類義語などの定義が必要となる部分を，自動で置換可能なアーキテクチャを目指す．二つ目は，ソースコードの構文の多様化である．現状では，最初に与えた expression 単位のコーパスとアノテーション記法に依存しているため，考えられるソースコードの記述パターンに対応しきれない．自然言語記述とソースコード両方を拡張可能な新たな手法を探究していきたい．

参考文献

[1] 秋信有花, 梶浦照乃, 小原百々雅, 倉光君郎. 自然言語を用いたコーディング支援の実現に向けたニューラル機械翻訳ベースのコード生成. 情報処理学会第136回プログラミング研究発表会 (PRO136), 2021.

[2] Yuka Akinobu, Momoka Obara, Teruno Kajiura, Shiho Takano, Miyu Tamura, Mayu Tomioka, and Kimio Kuramitsu. Is neural machine translation approach accurate enough for coding assistance? In **Proceedings of the 1st ACM SIGPLAN International Workshop on Beyond Code: No Code**, BCNC 2021, p. 23–28, New York, NY, USA, 2021. Association for Computing Machinery.

[3] Antonio Valerio Miceli Barone and Rico Sennrich. A parallel corpus of python functions and documentation strings for automated code documentation and code generation. In **Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)**, pp. 314–319, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

[5] Luis Perez, Lizi Ottens, and Sudharshan Viswanathan. Automatic code generation using pre-trained language models. **arXiv preprint arXiv:2102.10535**, 2021.

[6] Zeyu Sun, Qihao Zhu, Yingfei Xiong, Yican Sun, Lili Mou, and Lu Zhang. TreeGen: A tree-based transformer architecture for code generation. **Proceedings of the AAAI Conference on Artificial Intelligence**, Vol. 34, pp. 8984–8991, 04 2020.

[7] Bryan Ford. Parsing expression grammars: A recognition-based syntactic foundation. In **Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages**, POPL '04, pp. 111–122, New York, NY, USA, 2004. ACM.

[8] Kuramitsulab/pegtree: A peg parser generator with tree annotation. https://github.com/KuramitsuLab/pegtree. (Accessed on 01/09/2021).

[9] Hiroyuki Fudaba, Yusuke Oda, Koichi Akabe, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Pseudogen: A tool to automatically generate pseudo-code from source code. In **2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)**, pp. 824–829. IEEE, 2015.

[10] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. **Journal of Big Data**, Vol. 6, No. 1, pp. 1–48, 2019.

[11] Agelos Kratimenos, Kleanthis Avramidis, Christos Garoufis, Athanasia Zlatintsi, and Petros Maragos. Augmentation methods on monophonic audio for instrument classification in polyphonic music. In **2020 28th European Signal Processing Conference (EUSIPCO)**, pp. 156–160. IEEE, 2021.

[12] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. **arXiv preprint arXiv:2105.03075**, 2021.