統計的機械翻訳を用いた自然言語からのソースコード生成

札場 寛之　　小田 悠介　　Graham Neubig　　吉野 幸一郎　　中村 哲

{fudaba.hiroyuki.ev6, oda.yusuke.on9, neubig, koichiro, s-nakamura}@is.naist.jp

1 はじめに

コンピュータの発展とウェブの普及はかつてない機会をもたらしており、コンピュータを自由に操れる者にはアイデアの実現や仕事の効率化がますます容易になっている。しかしこの機会をつかむにはプログラミングを代表とするコンピュータを扱う能力が必要であり、この能力を習得しない限り新技術の恩恵を受けるチャンスを逃してしまう恐れがある。またプログラマでさえ、実装したい機能をすぐにコードに起こせないことがしばしばある。例えば図 1 にある「リストを降順でソートする」という少し複雑な命令を、ドキュメンテーションを見ずに実装できるプログラマはそう多くなかろう。

その一方プログラミング経験の有無に関わらず、ほとんどの人間は日本語や英語といった自然言語が自由に使える。プログラミング言語ではなく、自然言語でコンピュータに話しかけるようにプログラミングができれば、プログラミング言語を学ばずともプログラマ並にコンピュータが操れるようになり、またプログラマでもコードにすぐに起こせない問題に直面した際に、自然言語を介して効率的に実現したい処理をコードに起こせるようになる。

上記の問題を解決するために**自然言語プログラミング**に着目した研究がいくつかある。例えば自然言語の文を独自に定義したプログラミング言語へと変換し、スマートフォンやスプレッドシートを操作する研究 [1, 2] では、ユーザが意図した動作をするプログラムを 90%程度の確率で生成することができた。しかし、操作したい対象ごとに専用のプログラミング言語を定義しなくてはならないという問題がある。また、データを読み込む処理を自然言語で記述された仕様書とサンプルデータから自動生成する研究 [3] が存在する。しかし、実際にプログラムを動作させながら曖昧性を解決するため、パーザに対するサンプルデータが必要となる。

本研究ではこの問題点を解決する方法のひとつとして、既にあるデータから自動的に、自然言語からプログラムを生成するモデルを獲得する**統計的機械翻訳**の技術に着目する。統計的機械翻訳を用いて、自然言語からソースコードではなく、ソースコードから自然言語の擬似コードを生成する関連研究があり、そこで統計翻訳のアプローチの有効性が示されている [4]。しかし曖昧性のないプログラミング言語から自然言語の生成は比較的容易である一方、逆に曖昧性のある自然言語からプログラムの生成には課題が残されている。まず、プログラミング言語は厳密な構文を持つため、自動生成されたソースコードにはプログラミング言語としての**構文的正しさ**が要求される。これに加えて、自動生成されたソースコードはユーザの意図した通りに動作する必要があり、このような**意味的正しさ**についても保証する必要がある。

```
# if n is even,
if n % 2 == 0:
```

```
# sort lst in descending order.
sort(lst, reverse=True)
```

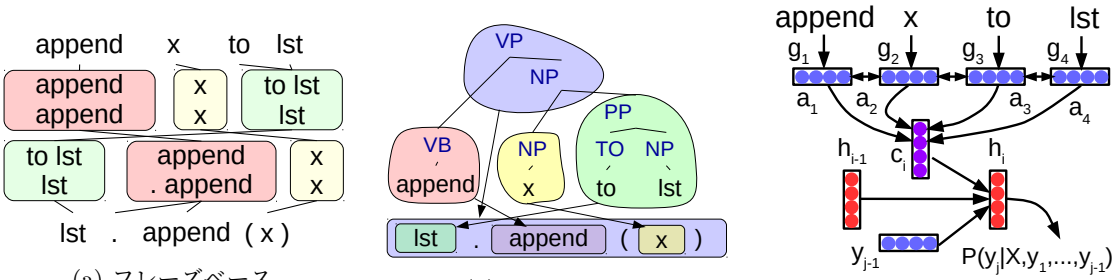
図 1: 自然言語の説明文と対応するソースコードの例。

本研究では、このような自動的かつ高精度なコード生成に向け複数の統計的機械翻訳の枠組みを用いて、自然言語の文から対応するソースコードを生成する。それぞれの手法により生成されたコードを構文的正しさと意味的正しさによって評価し、上記の問題に対する各手法の有効性について検証する。

2 統計的機械翻訳による自動生成

プログラミングにおいては、複数行に渡るひとかたまりの処理を一言で言い表すことができる場合がある。例えば「1 から 10 の間にある偶数の列挙」というのと、「1 から 10 の自然数に対して、小さい順に 2 で割り、その余りが 0 ならばリストに追加する」という説明は、同じ処理を説明していながら抽象度が異なる。あらゆる抽象度の説明文から目的の動作をするソースコードの生成ができることが望ましいが、本研究ではまず比較的抽象度の低い、一行の説明文から一行のコード生成を試みる。

自然言語からのソースコード生成は、自然言語の文 $X = x_1, \ldots, x_I$ が与えられたとき、X の持つ意味的な情報を損なわずにプログラミング言語のトークン列 $Y = y_1, \ldots, y_J$ へ変換することを考える．具体的には、異なる自然言語間の翻訳に用いられる**統計的機械翻訳**に基づく手法で、自然言語の文からプログラミング言語のトークン列を生成する。統計的機械翻訳では、



(a) フレーズベース　(b) Tree-to-string　(c) 注意型ニューラルネット

図 2: 本研究が対象とする翻訳手法

原文である自然言語の文 X が与えられたとき、最も確率の高いプログラミング言語のトークン列

$$\hat{Y} = \underset{Y}{\operatorname{argmax}} P(Y|X)$$

を求める。次節以降、本研究に用いるフレーズベース翻訳 (PBMT)、Tree-to-String 翻訳 (T2S)、注意型ニューラルネット (ANN) について詳しく述べる。

2.1 フレーズベース翻訳

PBMT[5] は図 2a のように、文を翻訳可能な単位に分けて、確率付きの翻訳辞書で各単位に対して翻訳候補の生成を行い、これらを並び替えたものを最終的な翻訳結果とするモデルである。この過程で生成される無数の翻訳候補を確率モデルでスコア付けし、確率最大のものを選択する。

原言語側の文 X が与えられたときに目的言語側の文 Y へ翻訳される確率は、ベイズ則を用いて以下のように変形できる。

$$\begin{aligned}\hat{Y} &= \underset{Y}{\operatorname{argmax}} P(Y|X) \\ &= \underset{Y}{\operatorname{argmax}} P(X|Y)P(Y)\end{aligned}$$

更に、確率は一般的に下記のように、上記の言語モデルと翻訳モデル確率に加えて、単語数やフレーズ数、並べ替え確率等を素性とする対数線形モデルとして表され、翻訳精度が最も高くなるようにパラメータを決定する [6]。

$$\hat{Y} \propto exp(\sum_{f_1}^{F} \boldsymbol{w}_f \phi_f(X,Y)) \quad (1)$$

PBMT の特徴として、自然言語、プログラミング言語のいずれの構文的情報も考慮しないことが挙げられる。PBMT は語順や構造の近い言語同士では翻訳性能が出る。PBMT を用いて Java から C#へのコードからコードへの翻訳の研究もなされている [7]。本研究では原言語の構文的情報を考慮する T2S 翻訳も試みるため、両者を比較することで原言語側の構文情報がどの程度役に立つかを検証する。

2.2 Tree-to-string 翻訳

T2S 翻訳 [8] は図 2b のように、原言語側の文の構文解析木が与えられたとき、その木の部分木をルールに基づいて目的言語側のフレーズに置き換えて翻訳する手法である。機械翻訳において、原言語側の文 X の構造を考慮できる場合、構文を考慮せず単語列やフレーズ列のみを用いて翻訳を行う場合と比べて翻訳精度が向上することが知られている [9]。

T2S 翻訳は原言語側の文の構文解析木 T_X と、目的言語側の文 Y の単語やフレーズを対応付けて翻訳を行うモデルであり、下記のように表せる。

$$\hat{Y} = \underset{Y}{\operatorname{argmax}} P(Y|T_X)$$

T2S に用いる翻訳ルールは PBMT の単語列から単語列へのマッピングとは異なり、原言語側の部分木と目的言語側の変数を含めたフレーズの組の集合であり、これらのルールに基づいて入力された構文木から最適な部分木の置換を求めるモデルである。PBMT と同様に、この最適な組み合わせを対数線形モデル等に基づいて求める。

T2S は文法構造の異なりそうな自然言語からプログラミング言語への変換において PBMT に比べて高い翻訳性能が期待される。一方で目的言語側の構造は考慮していないため、必ずしもプログラミング言語として正しい構文の出力が得られるわけではない。

2.3 注意型ニューラルネットに基づく翻訳

最後に、ニューラル機械翻訳 (NMT) について述べる [10, 11, 12]。NMT では、出力文 Y の確率を各単語 y_j ごとに計算し、単語確率の積で文の確率を計算する。具体的には、下記の通り入力文 X とこれまでに生成された単語 y_1^{j-1} が与えられた確率を用いる。

$$P(Y|X) = \prod_{j=1}^{J} P(y_j|X, y_1, \ldots, y_{j-1}) \quad (2)$$

NMT の手法のひとつである符号化・逆符号化

(encoder-decoder) モデル [10, 11] はまず符号化用のリカレントニューラルネット (RNN) で入力文 X を読み込み、X の内容を表すベクトル表現 $\boldsymbol{h}_0$ を構築する。次に出力の確率をこのベクトルに基づいて計算し、確率最大の単語を選択する。最後に、選んだ単語 y_j と履歴を表すベクトル $\boldsymbol{h}_{j-1}$ を逆符号化用のニューラルネットに入力し、次のベクトル $\boldsymbol{h}_j$ を計算する。

このような符号化・逆符号化に基づく手法は単純でありながら高い翻訳精度を実現する一方、固定長のベクトル $\boldsymbol{h}_0$ で様々な長さの入力文の意味を捉えることが困難であり、特に長い文に対する翻訳性能が低下することも報告されている [13]。この問題を解決する手法として、注意型ニューラルネット (ANN; attentional neural nets) が提案され、翻訳で広く使われている [12]。ANN では、図 2c に示すように、入力文に対して 1 つの実数ベクトルを計算するのではなく、入力文の各単語 $X = x_1, \ldots, x_I$ に対する実数ベクトル $G = \boldsymbol{g}_1, \ldots, \boldsymbol{g}_I$ を計算する。そして、次の単語の確率を計算するときに、現在の潜在状態ベクトル $\boldsymbol{h}_{j-1}$ にもとづいて、注意ベクトル $A = a_1, \ldots, a_I$ を推定し、G の重み付き和に相当する文脈ベクトル $\boldsymbol{c}$ を計算する。

$$\boldsymbol{c} = \sum_{i=1}^{I} a_i \boldsymbol{g}_i \tag{3}$$

このモデルを用いる利点として、生成の都度、その単語の生成に有益な情報を持っている単語に注目して生成するため、文全体を 1 つの固定長のベクトルで格納する必要がなくなり、大幅な翻訳性能向上が報告されている。

NMT は発明当初から様々なタスクで大幅な性能向上を実現しており、機械翻訳研究の中心的な手法の一つになりつつある。その理由の一つには、1 単語 1 単語の対応を陽に記憶する従来法と比べて、実数ベクトルで単語や文脈の類似性を考慮することで、より柔軟に文脈が表現できることが挙がられる。本研究が扱う自然言語とソースコードの対応は通常の自然言語と自然言語の対応とくらべて、単語が 1 対 1 関係になっていないことが多く見受けられ、NMT が有効に働くことが期待できる。

3 実験

3.1 実験設定

対訳コーパスとして、Python のウェブフレームワーク Django の全ソースコードに対して英語でアノテーションされたコーパスを用いた [4]。対訳コーパスは 17357 対の英語とソースコードのペアからなる。実験の評価は機械翻訳の評価尺度である BLEU[14] と RIBES[15] のほかに、Python の ast パーザが解析できる構文的に正しいソースコードの割合と、主観的にユーザが意図した動作をするソースコードの割合によって評価した [16]。PBMT には Moses[17]、T2S には Travatar[18]、ANN には Lamtram[1] を用いた。ANN では、128 と 256 ノードのネットをそれぞれ 3 つ学習し、最終的な出力時にアンサンブルとして用いた。

[1] http://github.com/neubig/lamtram

手法	BLEU	RIBES	構文	意味
PBMT	59.6	82.4	46.0	32
T2S	64.0	88.1	59.0	49
ANN	**79.5**	**95.8**	**79.7**	**62**

表 1: BLEU スコア、RIBES スコア、構文的・意味的正しさによる評価結果 (%)。太字は Bootstrap Resampling による他の手法に対する有意の性能向上を示す ($p < 0.01$)。

	出力	正否
入力 1	reverse element order of dicts, join them in a string, substitute the result for number.	-
PBMT	`reverse(dicts, (1, 1), (=number`	0/0
T2S	`number = in reverse dicts (1)`	0/0
ANN	`number = ''.join(reversed(dicts))`	1/1
入力 2	define the function all_valid with an argument formsets.	-
PBMT	`def all_valid(formsets):`	1/1
T2S	`def all_valid(formsets):`	1/1
ANN	`def all_valid(js):`	1/0

表 2: 生成されたコードの実例と構文/意味の正しさ。

3.2 実験結果

表 1 は実験の評価結果である。ANN による翻訳結果が PBMT と T2S の結果をすべての指標で上回っている。PBMT に比べ、T2S の方が翻訳精度が高いことから原言語側である自然言語の構造情報を考慮することで翻訳精度が向上することが分かる。

ANN の精度が高かった理由を詳しく分析し、その結果得られた実例を表 2 の入力 1 に示す。コーパスの中でも比較的長い入力である入力 1 に対し ANN が構文的/意味的に正しい結果を出力しているのが分かる。PBMT の出力は括弧の対応が取れておらず、T2S の出力は ‘= in‘という Python の構文では許されない形になっている。一方入力 2 に対して ANN は、変数名が異なった出力をしている。これは ANN には変換前の変数名を同一のものにする制約がないことが原因である。これは実用上問題になるため今後の課題である。

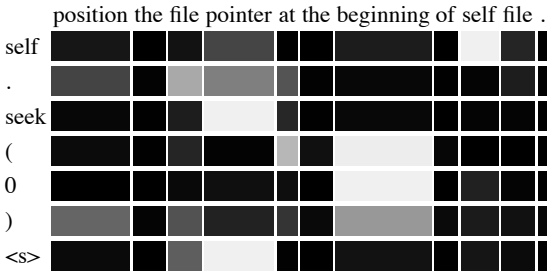


図 3: ニューラルネットの注意行列の例。

図 3 は、ニューラルネットがどの単語に注目して翻訳を行うかを示した例である。beginning や pointer といった翻訳に有益な単語に注目しているのが分かる。さらに、beginning に対応するトークンは 0 であり、また pointer に対応するトークンも seek となっていることから直訳ではなく意訳になっていることが分かる。

4 関連研究

自然言語の文からプログラムを生成する研究として、スプレッドシートデータに対する操作の自動化のために専用の型付き言語 (ドメイン特化言語、以降 DSL) を用意して問題解決に適用した研究がある [2]。この研究で用意された DSL はスプレッドシートに対する柔軟な操作をサポートしており、ユーザに自然な表現を許している。また、一つの自然言語の文から複数のプログラムを生成し、それらに対してもっともらしさでランクづけを行ったのちにユーザに提示することで最終的に得られるプログラムの精度を高めている。自然言語の文から型付き DSL を用いてプログラムを生成する研究は、上記の他にもスマートフォンの自動化スクリプト生成の研究がある [1]。

プログラムへの入力に対するパーザの自動生成を行った研究もなされている [3]。この研究では、入力として自然言語によって記述されたプログラムへの入力の仕様書とサンプルデータを用いて、パーザに対する入力を適切なデータ構造へと変換するパーザの自動生成を試みている。この研究では仕様書から得られる仕様木と、生成されたプログラムに入力を与えたときに入力が読めるかの情報を利用して生成モデルを学習する。

5 おわりに

本研究では統計的機械翻訳を用いて自然言語の記述からプログラムのソースコードを生成した。ANN によるソースコード生成が他のすべての手法をすべての指標で上回った。T2S が PBMT より高い評価結果であることから、自然言語側の構造を考慮することの有用性も示された。

今後の課題として、string-to-tree 翻訳 [19] 等を用いて、目的言語側の構造を考慮することが挙げられる。目的言語側はプログラミング言語である以上、一意に構文解析結果が定まり、この性質を利用すれば構文的に正しいソースコードの生成が可能になると考えられる。また、本研究では一行の自然言語による記述に対し、一行のソースコードを生成した。より抽象的な自然言語の記述から複数行に渡るソースコードの生成への取り組みも検討していきたい。

謝辞: 本研究の一部は、頭脳循環を加速する戦略的国際研究ネットワーク推進プログラムおよびマイクロソフトリサーチ CORE 連携研究プログラムの助成を受けて行ったものです。

参考文献

[1] Vu Le, Sumit Gulwani, and Zhendong Su. Smartsynth: Synthesizing smartphone automation scripts from natural language. In *Proc. MobiSys*, 2013.

[2] Sumit Gulwani and Mark Marron. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proc. SIGMOD*, 2014.

[3] Nate Kushman and Regina Barzilay. Using semantic unification to generate regular expressions from natural language. In *Proc. NAACL*, 2013.

[4] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Learning to generate pseudo-code from source code using statistical machine translation. In *Proc. ASE*, 2015.

[5] Daniel Marcu Philipp Koehn, Franz Josef Och. Statistical phrase-based translation. *Proc. HLT-NAACL*, 2003.

[6] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proc. ACL*, pp. 160–167, 2003.

[7] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. Lexical statistical machine translation for language migration. In *Proc. FSE*, pp. 651–654. ACM, 2013.

[8] Yang Liu, Qun Liu, and Shouxun Lin. Tree-to-string alignment template for statistical machine translation. In *Proc. ACL*, pp. 609–616, 2006.

[9] Graham Neubig and Kevin Duh. On the elements of an accurate tree-to-string machine translation system. In *Proc. ACL*, 2014.

[10] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proc. EMNLP*, pp. 1700–1709, 2013.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pp. 3104–3112, 2014.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*, 2015.

[13] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proc. EMNLP*, pp. 1724–1734, 2014.

[14] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proc. ACL*, pp. 311–318, 2002.

[15] Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. Automatic evaluation of translation quality for distant language pairs. In *Proc. EMNLP*, pp. 944–952, 2010.

[16] Anh T. Nguyen, Tung T. Nguyen, and Tien N. Nguyen. Divide-and-conquer approach for multi-phase statistical migration for source code. In *Proc. ASE*, 2015.

[17] Philipp Koehn, et al. Moses: Open source toolkit for statistical machine translation. In *Proc. ACL*, pp. 177–180, 2007.

[18] Graham Neubig. Travatar: A forest-to-string machine translation engine based on tree transducers. In *Proc. ACL*, pp. 91–96, 2013.

[19] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *Proc. HLT*, pp. 273–280, 2004.