大規模疑似データを用いた高性能文法誤り訂正モデルの構築

清野 舜[♢,♠]　鈴木 潤[♠,♢]　三田 雅人[♢,♠]　水本 智也[♣,♢]　乾 健太郎[♠,♢]
[♢]理化学研究所　[♠]東北大学　[♣]フューチャー株式会社
{shun.kiyono, masato.mita, tomoya.mizumoto}@riken.jp
{jun.suzuki, inui}@ecei.tohoku.ac.jp

1 はじめに

文法誤り訂正（Grammatical Error Correction; GEC）は，学習者の書いた文法誤りを含む文を正しい文に自動で訂正するタスクである．最近では誤った文を原言語，正しい文を目的言語とみなし，誤った文を正しい文に**翻訳する**ような機械翻訳タスクとして取り組むのが一般的である．このアプローチにより，機械翻訳の最先端の研究成果を用いることが可能となる．例えば，Encoder-Decoder モデル [17, 18]（EncDec）の導入により，GEC の性能は飛躍的に向上した [6]．

EncDec の持つ問題点の一つとして，大量の訓練データが必要になることがある [8]．しかしながら，GEC の対訳データは限られた規模のものしか存在しておらず，最も規模の大きなものでも 200 万文対 [10] に留まっている．その結果，GEC の研究分野においては，擬似データを生成し活用する研究が活発に行われてきた [5, 7, 9, 20]．

擬似データを EncDec の学習に取り入れる際には，実験設定に関するいくつかの要素を予め定めておく必要がある．代表的なものでは，(1) 擬似データの生成方法，(2) 擬似データの生成元データ，そして (3) 擬似データを用いた最適化手法が挙げられる．しかしながら，これらの要素に関する統一的な知見は，未だ GEC の研究領域において存在しない．例えば，Xie ら [19] は逆翻訳ベースの手法（Backtrans (noisy)）が文法的な文から直接擬似データを生成する手法（DirectNoise）よりも良い性能を示したと報告している．一方で，BEA-2019 で開催された GEC の Shared Task（以下，BEA-2019 と言及する）の優勝システム [5] は DirectNoise ベースの手法を用いている．

本研究の目的は，擬似データに関する上記の要素 (1) – (3) を調べることで，GEC の研究コミュニティがより効果的に擬似データを活用できるような知見を見つけ出すことである．我々はまず，大規模な実験を通して有効な疑似データの設定を見つけ出す．その後，見つけた設定の有効性をベンチマークデータセット上での性能評価によって示す．その結果，EncDec のモデルアーキテクチャに変更を加えることなく，BEA-2019 の優勝システムを超える世界最高性能を達成する．

2 問題設定

本稿で取り扱う GEC タスクを形式的に定義する．$\mathcal{D}$ を GEC モデルの訓練に使う対訳データとし，文法誤りを含む文 $\boldsymbol{X}$ と文法的な文 $\boldsymbol{Y}$ のペアを要素として持つと定義する．つまり，$\mathcal{D} = \{(\boldsymbol{X}_n, \boldsymbol{Y}_n)\}_n$ である．ここで，$|\mathcal{D}|$ は $\mathcal{D}$ に含まれる文ペアの数を表す．

訓練の対象となるモデルパラメータの集合を $\boldsymbol{\Theta}$ で表すことにする．ここで，我々の目的は次の誤差関数 $\mathcal{L}(\mathcal{D}, \boldsymbol{\Theta})$ を最小化するようなパラメータ集合 $\widehat{\boldsymbol{\Theta}}$ の獲得である．

$$\mathcal{L}(\mathcal{D}, \boldsymbol{\Theta}) = -\frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{X},\boldsymbol{Y})\in\mathcal{D}} \log(p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\Theta})), \qquad (1)$$

通常の教師あり学習の枠組みでは，対訳データ $\mathcal{D}$ は**真の対訳データ（genuine data）** $\mathcal{D}_g$ のみから構成される．つまり，$\mathcal{D} = \mathcal{D}_g$ である．一方で，GEC においては文法的な文 $\boldsymbol{Y} \in \mathcal{T}$ から**擬似データ（pseudo data）** $\mathcal{D}_p$ を生成し活用するのが一般的である [5, 19, 20]．ここで，$\mathcal{T}$ は文法的な文の集合であり，以降は生成元コーパスと呼ぶ．

我々は式 1 に紐付いた次の 3 つの要素に関心がある．**要素 (1)** 擬似データ $\mathcal{D}_p$ の生成にあたっては，いくつかの手法が存在する．詳細は第 3 節で述べる．**要素 (2)** 生成元コーパス $\mathcal{T}$ の候補として，膨大な種類のコーパスが存在する．我々の知る限り，生成元コーパスの性質がモデルの性能に与える影響を調べた研究は存在しない．最初の試みとなる本研究では，生成元コーパスとして Simple Wikipedia（SimpleWiki），Wikipedia と Gigaword を比較する．SimpleWiki と Wikipedia は文のドメインは類似しているが，文法的な複雑度が大きく異なる．そのため，これら 2 つのコーパスを比較することで，文法的な複雑度がモデルの性能に与える影響を調べることができる．また，Gigaword の大半は新聞記事であるため，3 つのコーパスの中で文法的に正しい文が最も多いと考えられる．そのため，文の文法性がモデルの性能を改善するか調べることができる． **要素 (3)** 式 1 の最適化に疑似データ $\mathcal{D}_p$ を取り入れるにあたり，最低でも 2 つの設定が存在する．第一に，2 種類のデータを結合（$\mathcal{D} = \mathcal{D}_g \cup \mathcal{D}_p$）し，同時に学習に用いる設定が挙げられる．以降，この設定を Joint と呼ぶ．もう一方の設定として，擬似データ $\mathcal{D}_p$ でモデルを事前訓練した後，真のデータ $\mathcal{D}_g$ で finetune する設定も存在する．以降，この設定を Pretrain と呼ぶ．第 4 節の実験では，上記 3 つの要素の比較を行う．

3 擬似データの生成手法

本節では疑似データの生成手法の詳細を述べる．

■逆翻訳ベースの手法　逆翻訳を用いた EncDec のための擬似データ生成は，Sennrich ら [15] によって提案された．まず，本手法では真のデータ $\mathcal{D}_g$ の入出力を入れ替え，文法的に正しい文から文法誤りを含む文を生成するようなモデルを訓練する．このモデルを逆方向モデルと呼ぶ．その後，逆方向モデ

Copyright(C) 2020 The Association for Natural Language Processing.
All Rights Reserved.

ルに生成元コーパスの文 $\boldsymbol{Y} \in \mathcal{T}$ を入力し，文法誤りを含む文を生成する．得られた出力を入力文と組み合わせて疑似データ $\mathcal{D}_p$ を作成する．

ノイズ付き逆翻訳（以下，BACKTRANS (NOISY) と言及する）は，Xie ら [19] によって提案された逆翻訳の拡張である．本手法では，逆方向モデルのデコード時にビーム内の各仮説のスコアに対してノイズ $r\beta_{\text{random}}$ を加えることで，仮説間の順位のランダムな入れ替えを行う．ノイズにより，通常の逆翻訳よりも多くの文法誤りを生成できることが報告されている [19]．ここで，ノイズ r は範囲 $[0, 1]$ の一様分布からサンプリングされるスカラ値であり，β_{random} はノイズの大きさを制御するハイパーパラメータである．また，$\beta_{\text{random}} = 0$ のとき，BACKTRANS (NOISY) は通常の逆翻訳と等価となる．

■疑似データを直接生成する手法　BACKTRANS (NOISY) が文法誤りを含む文を逆方向モデルによって生成するのに対して，生コーパスの文 $\boldsymbol{Y} \in \mathcal{T}$ に直接ノイズを加える手法も存在する [20]．以降，同手法を DIRECTNOISE と呼ぶ．具体的には，文中の各トークンに対して，次の操作のうち一つを確率的に選択し付与する：(i) 専用のマスクトークン $\langle mask \rangle$ での置換，(ii) 削除，(iii) ランダムなトークンの挿入と (iv) 元トークンの保持[*1]．各トークンへの操作は，カテゴリカル分布 $(\mu_{\text{mask}}, \mu_{\text{deletion}}, \mu_{\text{insertion}}, \mu_{\text{keep}})$ からのサンプリングによって決定する．

4　実験

本節では，第 2 節で述べた要素 (1) – (3) の比較実験を行う．疑似データの効果的な活用方法を探索するという本研究の目的を踏まえると，実験結果から得られる知見が GEC タスク一般に適用可能であることが望ましい．そのため，以下の 2 つの戦略のもと一連の実験を設計する：(i) モデルには**既存の一般的な EncDec のアーキテクチャ**を採用し，GEC タスクに特化した工夫は加えない．(ii) ハイパーパラメータの探索や各要素 (1) – (3) の比較には**開発データ上の性能**を用いる．最終的に，得られた知見を総合し GEC における効果的な疑似データの設定を提案する．その後，**未知のテストデータ**上での性能評価を行い，提案する設定の有効性を示す．

4.1　実験設定

■データセット　訓練データと開発データには BEA-2019 で配布されたものを用いる．以降では，それぞれ BEA-train と BEA-valid と言及する．生成元コーパス $\mathcal{T}$ として SimpleWiki[*2]，Wikipedia[*3]と Gigaword[*4]を用いる．各コーパスに対して，BACKTRANS (NOISY) と DIRECTNOISE をそれぞれ適用して疑似データ $\mathcal{D}_p$ を生成する．

■性能評価　モデルの性能評価には，BEA-valid，BEA-2019 のテストデータ（BEA-test），CoNLL-2014 のテストデータ（CoNLL-2014）[13] と JFLEG のテストデータ（JFLEG）[12] を用いる．ここで，各テストデータのドメインは大きく異なることに注意されたい．例えば，CoNLL-2014 はエッセイのみから構成されるのに対して，BEA-test は手紙，物語や論説のようなより幅広いドメインの文も含んでいる．そのため，テストデータで高い性能を達成することで，我々の見つけた疑似データの設定が GEC 一般に対して頑健であることを示すことができる．

表 1: **各データセットの統計量**: "*" は生成元コーパス $\mathcal{T}$ を表す．

データセット	文（ペア）の数	参照文の数	Split	評価器
BEA-train	561,410	1	train	-
BEA-valid	4,384	1	valid	ERRANT
CoNLL-2014	1,312	2	test	ERRANT & M^2 scorer
JFLEG	1,951	4	test	GLEU
BEA-test	4,477	5	test	ERRANT
SimpleWiki*	1,369,460	-	-	-
Wikipedia*	145,883,941	-	-	-
Gigaword*	131,864,979	-	-	-

表 2: **BEA-valid 上の性能比較**: **太字**は同一カラム内での最高性能を表す．

手法	Prec.	Rec.	$F_{0.5}$
ベースライン	45.2	22.7	37.7
DIRECTNOISE	**48.3**	25.4	**40.9**
BACKTRANS (NOISY)	41.7	**31.2**	39.1

アンサンブルモデルの結果を除いて，報告値は全て 5 つの異なるシード値から得られた結果の平均である．BEA-valid と BEA-test は ERRANT[1] で評価した．BEA-test の参照文は公開されていないため，BEA-2019 のルールに従い CodaLab[*5] 上で評価した．CoNLL-2014 は M^2 scorer [3] で評価した．また，JFLEG の評価には GLEU[11] を用いた．各データセットの詳細な統計量を表 1 に示す．以降，真のデータ $\mathcal{D}_g$ のみを用いて訓練したモデルをベースラインと呼ぶ．

■モデル　既存の EncDec モデルとして，Transformer[18] を用いた．具体的には，Vaswani ら [18] の定義した "Transformer (big)" 設定を用いた．fairseq[*6]の実装を使って実験を行った．

■最適化　JOINT 設定では Adam を用いてモデルを最適化した．PRETRAIN 設定では Adam で事前訓練を行った後，Adafactor[16] で finetune した[*7]．

4.2　要素 (1)：疑似データの生成手法

本節では疑似データの生成手法として BACKTRANS (NOISY) と DIRECTNOISE を比較する．事前実験の結果より，BACKTRANS (NOISY) と DIRECTNOISE のハイパーパラメータはそれぞれ $\beta_{\text{random}} = 6$ と $(\mu_{\text{mask}}, \mu_{\text{deletion}}, \mu_{\text{insertion}}, \mu_{\text{keep}}) = (0.3, 0.25, 0.25, 0.2)$ とした．また，JOINT 設定を使用し，SimpleWiki を生成元コーパス $\mathcal{T}$ として用いた．

結果を表 2 に示す．疑似データを用いることでベースラインから $F_{0.5}$ 値が向上したが，これは先行研究 [19, 20] の結果と一致する．また，DIRECTNOISE と BACKTRANS (NOISY) はそれぞれ Precision と Recall の改善に効果的であるとわかる．$F_{0.5}$ 値では DIRECTNOISE が最も良い性能を示した．

[*1] アルゴリズムの詳細は文献 [7] の Appendix A を参照せよ．
[*2] https://simple.wikipedia.org
[*3] 2019-02-25 のダンプデータを用いた．https://dumps.wikimedia.org/other/cirrussearch/
[*4] English Gigaword Fifth Edition (LDC Catalog No.: LDC2011T07).
[*5] https://competitions.codalab.org/competitions/20228
[*6] https://github.com/pytorch/fairseq
[*7] ハイパーパラメータの詳細は文献 [7] の Appendix D を参照せよ．

Copyright(C) 2020 The Association for Natural Language Processing.
All Rights Reserved.

表 3: 生成元コーパス $\mathcal{T}$ を変化させた場合の **BEA-valid** 上の性能: $|\mathcal{D}_p| = 1.4$M である．

手法	生成元コーパス $\mathcal{T}$	Prec.	Rec.	$F_{0.5}$
ベースライン	N/A	45.2	22.7	37.7
Backtrans (noisy)	Wikipedia	42.8	30.5	39.6
Backtrans (noisy)	SimpleWiki	41.7	31.2	39.1
Backtrans (noisy)	Gigaword	42.2	33.1	40.0
DirectNoise	Wikipedia	47.1	25.8	40.4
DirectNoise	SimpleWiki	48.3	25.4	40.9
DirectNoise	Gigaword	47.3	26.7	41.0

4.3 要素（2）：生成元コーパス

生成元コーパス $\mathcal{T}$ のモデルの性能への影響を調べる．SimpleWiki，Wikipedia，Gigaword を比較した結果を表 3 に示す．ここで，各生成元コーパス $\mathcal{T}$ 間の量の違いが性能に影響するのを回避するため，$|\mathcal{D}_p| = 1.4$M とした．表より，$F_{0.5}$ 値の差の絶対値は 1 ポイント以内と小さいことから，生成元コーパスの種類はモデルの性能に大きく影響しない可能性が示唆される．しかしながら，Gigaword は他 2 つのコーパスよりも一貫して高い性能を示した．特に，DirectNoise と Gigaword の組み合わせが最も良い性能を示した．Gigaword は基本的には新聞記事の集合とみなせるため，近い性質のコーパスは大規模な量を比較的低コストで獲得可能である[*8]．そのため，Gigaword が高い性能を示したことは，GEC の研究領域にとって好ましい結果であると考えられる．

4.4 要素（3）：最適化の設定

本節では最適化の設定として Joint と Pretrain の比較を行う．特に，次の 2 つの状況における性能の違いを比較する．(i) 疑似データの量が真のデータと同程度の場合 ($|\mathcal{D}_p| = 1.4$M) (ii) 疑似データの量が真のデータよりも非常に大きい場合 ($|\mathcal{D}_p| = 14$M)

■Joint 対 Pretrain　表 4 に結果を示す．ここで重要なのは，疑似データの量を増やすことにより，Pretrain ではモデルの性能が向上したのに対して，Joint では性能が変わらなかった点である．例えば，Backtrans (noisy) の結果に着目すると，Pretrain において，疑似データの量 $|\mathcal{D}_p|$ を 1.4M から 14M に増やすことで $F_{0.5}$ 値は 42.3 から 45.6 へと向上した．一方で，Joint では $F_{0.5}$ 値は 40.0 から変化しなかった．この結果から，大量の疑似データを効果的に利用するためには，Pretrain での最適化が必要であると考えられる．

この結果の直感的な解釈は次のとおりである．Joint において，疑似データが真のデータよりも極端に多い場合には，疑似データの教師信号が支配的になってしまう．そのため，モデルの学習において真のデータの影響が薄まってしまう問題がある．一方，Pretrain ではこの問題は生じない．これは，Pretrain においては疑似データの量によらず，最後に真のデータ単体を用いて finetune を行うためである．

■疑似データの量　表 4 の結果から，Pretrain において疑似データの量を増加させた場合の性能変化を調べた．具体的には，まず DirectNoise と Backtrans (noisy) を用いて疑似データを {1.4M, 7M, 14M, 30M, 70M} ずつ生成した．

[*8]例 : News Crawl `http://data.statmt.org/news-crawl/`

表 4: 最適化の設定を変化させた場合の **BEA-valid** 上の性能: 生成元コーパス $\mathcal{T}$ には Gigaword を用いた．

最適化	手法	$|\mathcal{D}_p|$	Prec.	Rec.	$F_{0.5}$
N/A	ベースライン	0	45.2	22.7	37.7
Pretrain	Backtrans (noisy)	1.4M	49.2	27.2	42.3
Pretrain	DirectNoise	1.4M	47.2	23.3	39.1
Joint	Backtrans (noisy)	1.4M	42.2	33.1	40.0
Joint	DirectNoise	1.4M	47.3	26.7	41.0
Pretrain	Backtrans (noisy)	14M	50.8	32.6	45.6
Pretrain	DirectNoise	14M	48.3	27.7	42.0
Joint	Backtrans (noisy)	14M	40.8	37.1	40.0
Joint	DirectNoise	14M	48.0	25.0	40.5

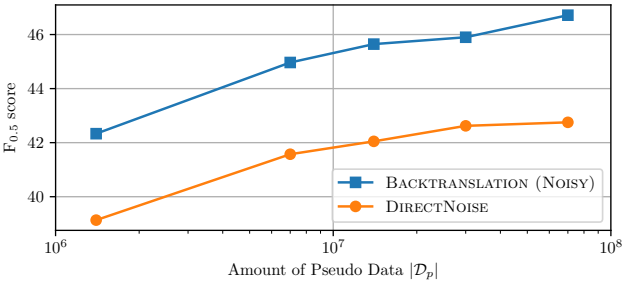

図 1: 疑似データの量を変化させた場合の **BEA-valid** 上での性能変化

その後，Pretrain で最適化を行った．図 1 に結果を示す．Backtrans (noisy) が DirectNoise よりも良いサンプル効率を示したとわかる．ここで，Backtrans (noisy) で生成した 70M のデータと Pretrain との組み合わせが，これまでの実験で最も高い性能（$F_{0.5}$ 値 46.7）を達成した．

4.5 既存研究との性能比較

これまでの実験結果より，疑似データを次のように活用することがモデルの性能向上に有効であることが分かった：(i) DirectNoise で疑似データを生成し Joint で最適化（第 4.2 節）(ii) 生成元コーパス $\mathcal{T}$ には Gigaword を用いる（第 4.3 節）(iii) Joint では，疑似データと真のデータの量を同程度にする（第 4.4 節）(iv) Backtrans (noisy) で大量に擬似データを生成し，Pretrain で最適化（第 4.4 節）．これらの知見のもと，我々は Pretrain と Joint の統合による，疑似データを活用するための最も効果的な設定の構築を試みた．具体的には，Backtrans (noisy) で生成した大規模な疑似データでモデルを事前訓練した後，DirectNoise で BEA-train と同量程度の疑似データを生成し，BEA-train と同時に finetune を行った．しかし，この設定では BEA-valid 上での性能向上は確認できなかった．故に，実験から最も効果的であると分かった設定は，Gigaword から Backtrans (noisy) で生成した 70M のデータでモデルを事前訓練し，その後 BEA-train で finetune するというものとなった．以降，この設定を PretLarge と呼ぶ．

PretLarge の効果を検証するため，テストデータ上での性能を評価し，既存研究との比較を行った．表 5 より，PretLarge のシングルモデルは CoNLL-2014 上で $F_{0.5}$ 値 61.3 を達成した．シングルモデルにも関わらず，この値は Grundkiewicz ら [5] によるアンサンブルモデルを除いた全ての既存研究を既に上回っている．

PretLarge の性能を更に向上させるため，以下の 2 つの手法を追加で適用した．

Copyright(C) 2020 The Association for Natural Language Processing.
All Rights Reserved.

表 5: **既存研究との性能比較．太字**は同一カラム内での最高性能を表す．G&JD : Grundkiewicz and Junczys-Dowmunt [4]．†: アンサンブルモデルの結果．∗: 提案手法．

モデル	CoNLL-2014			JFLEG	BEA-test		
	Prec.	Rec.	$F_{0.5}$	GLEU	Prec.	Rec.	$F_{0.5}$
G&JD[4]	66.8	34.5	56.3	61.5	-	-	-
Lichtarge et al. [9]	65.5	37.1	56.8	61.6	-	-	-
Lichtarge et al. [9]†	66.7	43.9	60.4	**63.3**	-	-	-
Zhao et al. [20]†	71.6	38.7	61.2	61.0	-	-	-
Grundkiewicz et al. [5]†	-	-	64.2	61.2	**72.3**	60.1	69.5
PretLarge∗	67.9	44.1	61.3	59.7	65.5	59.4	64.2
PretLarge+SSE∗	68.7	45.2	62.2	60.9	66.1	60.9	65.0
PretLarge+SSE+R2L∗†	**72.4**	**46.1**	**65.0**	61.4	72.1	**61.8**	**69.8**

Synthetic Spelling Error (SSE)　SSE は Lichtarge ら [9] によって提案された手法である．SSE のねらいは，疑似データ $\mathcal{D}_p$ の入力側に文字レベルでのノイズを加えることによる，人工的なスペル誤りの生成である．具体的には，各文字について，0.003 の割合で削除，ランダムな文字の挿入，置換，順番の入れ替えを行う．

Right-to-left Re-ranking (R2L)　R2L は機械翻訳の Shared Task で頻繁に用いられるリランキング手法である．先行研究 [14] に従い，文の末尾から先頭に向かって翻訳を行うモデル（right-to-left モデル）を 4 つ訓練した．まず，R2L では順方向に翻訳を行うモデル（left-to-right モデル）4 つのアンサンブルにより幅 n のビームサーチを行い，n 個の候補文とそのスコア（条件付き確率）を出力する．その後，各候補文を right-to-left モデルに入力し，スコアを計算する．最後に 2 つのスコアの合計により上位 n 件のリランキングを行い，最もスコアの高い文を出力する．本研究では $n = 5$ とした．

表 5 に SSE と R2L を適用した結果を示す．PretLarge+SSE+R2L が CoNLL-2014 で $F_{0.5}$ 値 65.0，BEA-test で $F_{0.5}$ 値 69.8 を達成した．これらの値は BEA-2019 の優勝システム [5] を上回る現在の世界最高性能である．

5　分析

本節では，誤りタイプ別の性能を調べることで，疑似データがを用いる手法が有効に作用する，または作用しないような誤りの性質を明らかにする．具体的には，BEA-valid に付与された誤りタイプの情報を用いて，真のデータのみで学習したベースライン，PretLarge と PretLarge+SSE の比較を行った（図 2）．

図 2 より，ベースラインと比較して PretLarge が全ての誤りタイプの性能を大きく改善したことがわかる．さらに，SSE による人工的なスペル誤りの付与が，誤りタイプ “SPELL” だけでなく，ほぼ全ての誤りタイプの性能を PretLarge から改善したことは興味深い結果である．

また，PretLarge+SSE でも解けていない誤りタイプとして，“VERB” や “NOUN” などの内容語の誤りが挙げられる．他の誤りタイプと比較して，これらの性能は低い値に留まっている．同様の傾向は，BEA-2019 のレポートでも報告されていることから [2]，内容語の訂正は一般の GEC モデルにとって難しいと考えられる．これらの性能向上にあたっては，内容語の誤りを集中的に生成するような疑似データ生成の手法の開発が考えられる．

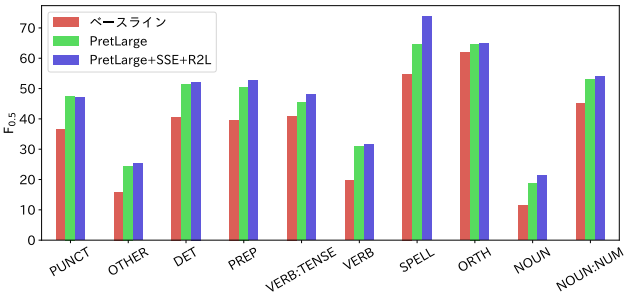

図 2: **BEA-valid における誤りタイプ別の性能**: 頻度上位 10 件の誤りタイプのみを示す．各誤りタイプの定義については，文献 [1] を参照せよ．

6　おわりに

本研究では，GEC への疑似データの適用における 3 つの要素について比較検討を行った．実験結果から，次の設定が有効であることがわかった．(i) 疑似データの生成元コーパスとして Gigaword 用いる（ii）ノイズ付き逆翻訳によって大規模な擬似データを生成し，モデルを事前訓練する．これらの知見のもと，我々は効果的な疑似データの活用手法を提案し，複数のベンチマークデータセット上で世界最高性能を達成した．

謝辞．本研究は JSPS 科研費 JP19H04162 の助成を受けたものである．

参考文献

[1] C. Bryant et al. “Automatic Annotation and Evaluation of Error Types for Grammatical Error Correction”. In: *ACL*. 2017, pp. 793–805.
[2] C. Bryant et al. “The BEA-2019 Shared Task on Grammatical Error Correction”. In: *BEA*. 2019, pp. 52–75.
[3] D. Dahlmeier and H. T. Ng. “Better Evaluation for Grammatical Error Correction”. In: *NAACL*. 2012, pp. 568–572.
[4] R. Grundkiewicz and M. Junczys-Dowmunt. “Near Human-Level Performance in Grammatical Error Correction with Hybrid Machine Translation”. In: *NAACL*. 2018, pp. 284–290.
[5] R. Grundkiewicz et al. “Neural Grammatical Error Correction Systems with Unsupervised Pre-training on Synthetic Data”. In: *BEA*. 2019, pp. 252–263.
[6] M. Junczys-Dowmunt et al. “Approaching Neural Grammatical Error Correction as a Low-Resource Machine Translation Task”. In: *NAACL*. 2018, pp. 595–606.
[7] S. Kiyono et al. “An Empirical Study of Incorporating Pseudo Data into Grammatical Error Correction”. In: *EMNLP-IJCNLP*. 2019, pp. 1236–1242.
[8] P. Koehn and R. Knowles. “Six Challenges for Neural Machine Translation”. In: *WNMT*. 2017, pp. 28–39.
[9] J. Lichtarge et al. “Corpora Generation for Grammatical Error Correction”. In: *NAACL*. 2019.
[10] T. Mizumoto et al. “Mining Revision Log of Language Learning SNS for Automated Japanese Error Correction of Second Language Learners”. In: *IJCNLP*. 2011, pp. 147–155.
[11] C. Napoles et al. “Ground Truth for Grammatical Error Correction Metrics”. In: *ACL-IJCNLP*. 2015, pp. 588–593.
[12] C. Napoles et al. “JFLEG: A Fluency Corpus and Benchmark for Grammatical Error Correction”. In: *EACL*. 2017, pp. 229–234.
[13] H. T. Ng et al. “The CoNLL-2014 Shared Task on Grammatical Error Correction”. In: *CoNLL*. 2014, pp. 1–14.
[14] R. Sennrich et al. “Edinburgh Neural Machine Translation Systems for WMT 16”. In: *WMT*. 2016, pp. 371–376.
[15] R. Sennrich et al. “Improving Neural Machine Translation Models with Monolingual Data”. In: *ACL*. 2016, pp. 86–96.
[16] N. Shazeer and M. Stern. “Adafactor: Adaptive Learning Rates with Sublinear Memory Cost”. In: *ICML*. 2018, pp. 4603–4611.
[17] I. Sutskever et al. “Sequence to Sequence Learning with Neural Networks”. In: *NIPS*. 2014, pp. 3104–3112.
[18] A. Vaswani et al. “Attention Is All You Need”. In: *NIPS*. 2017, pp. 5998–6008.
[19] Z. Xie et al. “Noising and Denoising Natural Language: Diverse Backtranslation for Grammar Correction”. In: *NAACL*. 2018, pp. 619–628.
[20] W. Zhao et al. “Improving Grammatical Error Correction via Pre-Training a Copy-Augmented Architecture with Unlabeled Data”. In: *NAACL*. 2019.

Copyright(C) 2020 The Association for Natural Language Processing.
All Rights Reserved.