Parallel Iterative Edit Models for Local Sequence Transduction

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, Vihari Piratla
Department of Computer Science and Engineering, IIT Bombay

Abstract

We present a Parallel Iterative Edit (PIE) model for the problem of local sequence transduction arising in tasks like Grammatical error correction (GEC). Recent approaches are based on the popular encoder-decoder (ED) model for sequence to sequence learning. The ED model auto-regressively captures full dependency among output tokens but is slow due to sequential decoding. The PIE model does parallel decoding, giving up the advantage of modelling full dependency in the output, yet it achieves accuracy competitive with the ED model for four reasons: 1. predicting edits instead of tokens, 2. labeling sequences instead of generating sequences, 3. iteratively refining predictions to capture dependencies, and 4. factorizing logits over edits and their token argument to harness pre-trained language models like BERT. Experiments on tasks spanning GEC, OCR correction and spell correction demonstrate that the PIE model is an accurate and significantly faster alternative for local sequence transduction. The code and pre-trained models for GEC are available at https://github.com/awasthiabhijeet/PIE.

1 Introduction

In local sequence transduction (LST) an input sequence $x_1, \ldots, x_n$ needs to be mapped to an output sequence $y_1, \ldots, y_m$ where the $x$ and $y$ sequences differ only in a few positions, $m$ is close to $n$, and $x_i, y_j$ come from the same vocabulary $\Sigma$. An important application of local sequence transduction that we focus on in this paper is Grammatical error correction (GEC). We contrast local transduction with more general sequence transduction tasks like translation and paraphrasing which might entail different input-output vocabulary and non-local alignments. The general sequence transduction task is cast as sequence to sequence (seq2seq) learning and modeled popularly using an attentional encoder-decoder (ED) model. The ED model auto-regressively produces each token $y_t$ in the output sequence conditioned on all previous tokens $y_1, \ldots, y_{t-1}$. Owing to the remarkable success of this model in challenging tasks like translation, almost all state-of-the-art neural models for GEC use it (Zhao et al., 2019; Lichtarge et al., 2019; Ge et al., 2018b; Chollet and Ng, 2018b; Junczys-Dowmunt et al., 2018).

We take a fresh look at local sequence transduction tasks and present a new parallel-iterative-edit (PIE) architecture. Unlike the prevalent ED model that is constrained to sequentially generating the tokens in the output, the PIE model generates the output in parallel, thereby substantially reducing the latency of sequential decoding on long inputs. However, matching the accuracy of existing ED models without the luxury of conditional generation is highly challenging. Recently, parallel models have also been explored in tasks like translation (Stern et al., 2018; Lee et al., 2018; Gu et al., 2018; Kaiser et al., 2018) and speech synthesis (van den Oord et al., 2018), but their accuracy is significantly lower than corresponding ED models. The PIE model incorporates the following four ideas to achieve comparable accuracy on tasks like GEC in spite of parallel decoding.

1. Output edits instead of tokens: First, instead of outputting tokens from a large vocabulary, we output edits such as copy, appends, deletes, replacements, and case-changes which generalize better across tokens and yield a much smaller vocabulary. Suppose in GEC we have an input sentence: *fowler fed dog*. Existing seq2seq learning approaches would need to output the four tokens *Fowler, fed, the, dog* from a word vocabulary whereas we
would predict the edits \{\text{Capitalize token 1, Append (the) to token 2, Copy token 3}\}.

2. Sequence labeling instead of sequence generation: Second we perform in-place edits on the source tokens and formulate local sequence transduction as \textit{labeling} the input tokens with edit commands, rather than solving the much harder whole sequence generation task involving a separate decoder and attention module. Since input and output lengths are different in general such formulation is non-trivial, particularly due to edits that insert words. We create special compounds edits that merge token inserts with preceding edits that yield higher accuracy than earlier methods of independently predicting inserts (Ribeiro et al., 2018; Lee et al., 2018).

3. Iterative refinement: Third, we increase the inference capacity of the parallel model by iteratively inputting the model’s own output for further refinement. This handles dependencies implicitly, in a way reminiscent of Iterative Conditional Modes (ICM) fitting in graphical model inference (Koller and Friedman, 2009). Lichtarge et al. (2019) and Ge et al. (2018b) also refine iteratively but with ED models.

4. Factorize pre-trained bidirectional LMs: Finally, we adapt recent pre-trained bidirectional models like BERT (Devlin et al., 2018) by factorizing the logit layer over edit commands and their token argument. Existing GEC systems typically rely on conventional forward directional LM to pre-train their decoder, whereas we show how to use a bi-directional LM in the encoder, and that too to predict edits.

Novel contributions of our work are as follows:

- Recognizing GEC as a local sequence transduction (LST) problem, rather than machine translation. We then cast LST as a fast non-autoregressive, sequence labeling model as against existing autoregressive encoder-decoder model.

- Our method of reducing LST to non-autoregressive sequence labeling has many novel elements: outputting edit operations instead of tokens, append operations instead of insertions in the edit space, and replacements along with custom transformations.

- We show how to effectively harness a pre-trained language model like BERT using our factorized logit architecture with edit-specific attention masks.

- The parallel inference in PIE is 5 to 15 times faster than a competitive ED based GEC model like (Lichtarge et al., 2019) which performs sequential decoding using beam-search. PIE also attains close to state of the art performance on standard GEC datasets. On two other local transduction tasks, viz., OCR and spell corrections the PIE model is fast and accurate w.r.t. other existing models developed specifically for local sequence transduction.

2 Our Method

We assume a fully supervised training setup where we are given a parallel dataset of incorrect, correct sequence pairs: \(D = \{(x^i, y^i) : i = 1 \ldots N\}\), and an optional large corpus of unmatched correct sequences \(L = \{\tilde{y}^1, \ldots, \tilde{y}^U\}\). In GEC, this could be a corpus of grammatically correct sentences used to pre-train a language model.

Background: existing ED model Existing seq2seq ED models factorize \(Pr(y|x)\) to capture the full dependency between a \(y_t\) and all previous \(y_{<t} = y_1, \ldots, y_{t-1}\) as \(\prod_{t=1}^n Pr(y_t|y_{<t}, x)\). An encoder converts input tokens \(x_1, \ldots, x_n\) to contextual states \(h_1, \ldots, h_n\) and a decoder summarizes \(y_{<t}\) to a state \(s_t\). An attention distribution over contextual states computed from \(s_t\) determines the relevant input context \(c_t\) and the output token distribution is calculated as \(Pr(y_t|y_{<t}, x) = Pr(y_t|c_t, s_t)\). Decoding is done sequentially using beam-search. When a correct sequence corpus \(L\) is available, the decoder is pre-trained on a next-token prediction loss and/or a trained LM is used to re-rank the beam-search outputs (Zhao et al., 2019; Chollampatt and Ng, 2018a; Junczys-Dowmunt et al., 2018).

2.1 Overview of the PIE model

We move from generating tokens in the output sequence \(y\) using a separate decoder, to labelling the input sequence \(x_1, \ldots, x_n\) with edits \(e_1, \ldots, e_n\). For this we need to design a function \text{Seq2Edits} that takes as input an \((x, y)\) pair in \(D\) and outputs a sequence \(e\) of edits from an edit space \(E\) where \(e\) is of the same length as \(x\) in spite of \(x\) and \(y\) being of different lengths. In Section 2.2 we show how we design such a function.
Training: We invoke Seq2Edits on $D$ and learn the parameters of a probabilistic model $\Pr(e|x, \theta)$ to assign a distribution over the edit labels on tokens of the input sequence. In Section 2.3 we describe the PIE architecture in more detail. The correct corpus $\mathcal{L}$, when available, is used to pre-train the encoder to predict an arbitrarily masked token $y_t$ in a sequence $y$ in $\mathcal{L}$, much like in BERT. Unlike in existing seq2seq systems where $\mathcal{L}$ is used to pre-train the decoder that only captures forward dependencies, in our pre-training the predicted token $y_t$ is dependent on both forward and backward contexts. This is particularly useful for GEC-type tasks where future context $y_{t+1} \ldots y_m$ can be approximated by $x_{t+1} \ldots, x_n$.

Inference: Given an input $x$, the trained model predicts the edit distribution for each input token independent of others, that is $\Pr(e|x, \theta) = \prod_{i=1}^{n} \Pr(e_i|x, t, \theta)$, and thus does not entail the latency of sequential token generation of the ED model. We output the most probable edits $\hat{e} = \arg \max_{a} \Pr(e|x, \theta)$. Edits are designed so that we can easily get the edited sequence $y$ after applying $\hat{e}$ on $x$. $y$ is further refined by iteratively applying the model on the generated outputs $y$ until we get a sequence identical to one of the previous sequences up to a maximum number of iterations $I$.

2.2 The Seq2Edits Function

Given a $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_m$ where $m$ may not be equal to $n$, our goal is to obtain a sequence of edit operations $e = (e_1, \ldots, e_n)$: $e_i \in \mathcal{E}$ such that applying edit $e_i$ on the input token $x_i$ at each position $i$ reconstructs the output sequence $y$. Invoking an off-the-shelf edit distance algorithm between $x$ and $y$ can give us a sequence of copy, delete, replace, and insert operations of arbitrary length. The main difficulty is converting the insert operations into in-place edits at each $x_i$. Other parallel models (Ribeiro et al., 2018; Lee et al., 2018) have used methods like predicting insertion slots in a pre-processing step, or predicting zero or more tokens in-between any two tokens in $x$. We will see in Section 3.1.4 that these options do not perform well. Hence we design an alternative edit space $\mathcal{E}$ that merges inserts with preceding edit operations creating compound append or replace operations. Further, we create a dictionary $\Sigma_a$ of common q-gram insertions or replacements observed in the training data.

Our edit space ($\mathcal{E}$) comprises of copy ($C$) $x_i$, delete ($D$) $x_i$, append ($A$) a q-gram $w \in \Sigma_a$ after copying $x_i$, replace ($R$) $x_i$ with a q-gram $w \in \Sigma_a$. For GEC, we additionally use transformations denoted as $T_1, \ldots, T_k$ which perform word-inflation (e.g. arrive to arrival). The space of all edits is thus:

$$\mathcal{E} = \{C, D, T_1, \ldots, T_k\} \cup \{A(w) : w \in \Sigma_a\} \cup \{R(w) : w \in \Sigma_a\} \quad (1)$$

<table>
<thead>
<tr>
<th>Example 1</th>
<th></th>
<th>Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>[Bolt can have run race]</td>
<td>[He still won race!]</td>
</tr>
<tr>
<td>$y$</td>
<td>[Bolt could have run the race]</td>
<td>[However, he still won!]</td>
</tr>
<tr>
<td>$\text{diff}(C_i)(C,Bolt)(D,can)(t,can,could)(c,have)(C,run)(l,run,thel)(c,race)(C_i)$</td>
<td></td>
<td>$\text{diff}(C_i)(l,however)(D,he)(t,he,he)(C,still)(C,won)(D,\text{-race})(C_i)$</td>
</tr>
<tr>
<td>$\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow$</td>
<td></td>
<td>$\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow$</td>
</tr>
<tr>
<td>[Bolt can have run race]</td>
<td>[He still won race!]</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Two example sentence pairs converted into respective edit sequences.

We present our algorithm for converting a sequence $x$ and $y$ into in-place edits on $x$ using the above edit space in Figure 1. Table 1 gives examples of converting $(x, y)$ pairs to edit sequences. We first invoke the Levenshtein distance algorithm (Levenshtein, 1966) to obtain $\text{diff}$ between $x$ and $y$ with delete and insert cost as 1 as usual, but...
with a modified substitution cost to favor matching of related words. We detail this modified cost in the Appendix and show an example of how this modification leads to more sensible edits. The \texttt{diff} is post-processed to convert substitutions into deletes followed by inserts, and consecutive inserts are merged into a q-gram. We then create a dictionary $\Sigma$ of the $M$ most frequent q-grams in the training set. Thereafter, we scan the \texttt{diff} left to right: a copy at $x_i$ makes $e_i = C$, a delete at $x_i$ makes $e_i = D$, an insert $w$ at $x_i$ and a $e_i = C$ flips the $e_i$ into a $e_i = \lambda(w)$ if $w$ is in $\Sigma$, else it is dropped, an insert $w$ at $x_i$ and a $e_i = D$ flips the $e_i$ into a $e_i = \tau(w)$ if a match found, else a replace $R(w)$ if $w$ is in $\Sigma$, else it is dropped.

The above algorithm does not guarantee that when $e$ is applied on $x$ we will recover $y$ for all sequences in the training data. This is because we limit $\Sigma$ to include only the $M$ most frequently inserted q-grams. For local sequence transduction tasks, we expect a long chain of consecutive inserts to be rare, hence our experiments were performed with $q = 2$. For example, in NUCLE dataset (Ng et al., 2014) which has roughly 57.1K sentences, less than 3% sentences have three or more consecutive inserts.

2.3 The Parallel Edit Prediction Model

We next describe our model for predicting edits $e : e_1, \ldots, e_n$ on an input sequence $x : x_1, \ldots, x_n$. We use a bidirectional encoder to provide a contextual encoding of each $x_i$. This can either be multiple layers of bidirectional RNNs, CNNs or deep bidirectional transformers. We adopt the deep bidirectional transformer architecture since it encodes the input in parallel. We pre-train the model using $\mathcal{L}$ much like in BERT pre-training recently proposed for language modeling (Devlin et al., 2018). We first give an overview of BERT and then describe our model.

**Background: BERT** The input is token embedding $x_i$ and positional embedding $p_i$ for each token $x_i$ in the sequence $x$. Call these jointly as $h_i^0 = [x_i, p_i]$. Each layer $\ell$ produces a $h_i^\ell$ at each position $i$ as a function of $h_i^{\ell-1}$ and self-attention over all $h_j^{\ell-1}$, $j \in [1, n]$. The BERT model is pre-trained by masking out a small fraction of the input tokens by a special \texttt{MASK}, and predicting the masked word from its bidirectional context captured as the last layer output: $h_1, \ldots, h_n$.

![Figure 2: A Parallel Edit Architecture based on a 2-layer bidirectional transformer. Input sequence length is 3. Arrows indicate attention mask for computation of $h_i^1, r_i^1, a_i^1$ at position i for layer l.](image)

2.3.1 A Default use of BERT

Since we have cast our task as a sequence labeling task, a default output layer would be to compute $\Pr(e_i|x)$ as a softmax over the space of edits $\mathcal{E}$ from each $h_i$. If $W_e$ denotes the softmax parameter for edit $e$, we get:

$$\Pr(e_i = e|x) = \text{softmax}(W_e^T h_i)$$

(2)

The softmax parameters, $W_e$ in Equation 2, have to be trained from scratch. We propose a method to exploit token embeddings of the pre-trained language model to warm-start the training of edits like appends and replaces which are associated with a token argument. Furthermore, for appends and replaces, we provide a new method of computing the hidden layer output via alternative input positional embeddings and self-attention. We do so without introducing any new parameters in the hidden layers of BERT.

2.3.2 An Edit-factorized BERT Architecture

We adapt a pre-trained BERT-like bidirectional language model to learn to predict edits as follows. For suitably capturing the contexts for replace edits, for each position $i$ we create an additional input comprising of $r_i^0 = [M, p_i]$ where $M$ is the embedding for \texttt{MASK} token in the LM. Likewise for a potential insert between $i$ and $i + 1$ we create an input $a_i^0 = [M, p_{i+1} + \frac{p_{i+1}}{2}]$ where the second component is the average of the position embedding of the $i$th and $i + 1$th position. As shown in Figure 2, at each layer $\ell$ we compute self-attention for the $i$th replace unit $r_i^\ell$ over $h_j^\ell$ for all $j \neq i$ and itself. Likewise, for the append unit $a_i^\ell$, the self-attention is over all $h_j^\ell$ for all $j$s and itself. At the last layer we have $h_1, \ldots, h_n, r_1, \ldots, r_n, a_1, \ldots, a_n$. Using these we compute logits factorized over edits.
and their token argument. For an edit \( e \in \mathcal{E} \) at position \( i \), let \( w \) denote the argument of the edit (if any). As mentioned earlier, \( w \) can be a q-gram for append and replace edits. Embedding of \( w \), represented by \( \phi(w) \) is obtained by summing up individual output embeddings of tokens in \( w \). Additionally, in the outer layer we allocate edit-specific parameters \( \theta \) corresponding to each distinct command in \( \mathcal{E} \). Using these, various edit logits are computed as follows:

\[
\logit(e_i|\mathbf{x}) = \text{softmax}(\logit(e_i|\mathbf{x})) \quad \text{where} \quad \logit(e_i|\mathbf{x}) = \begin{cases} 
\theta^C_i h_i + \phi(x_i)^\top h_i + 0 & \text{if } e_i = C \\
\theta^A_i(w) h_i + \phi(x_i)^\top h_i + \phi(w)^\top a_i & \text{if } e_i = A(w) \\
\theta^R_i(w) h_i + 0 + (\phi(w) - \phi(x_i))^\top r_i & \text{if } e_i = R(w) \\
\theta^D_i h_i + 0 + 0 & \text{if } e_i = D \\
\theta^T_k h_i + \phi(x_i)^\top h_i + 0 & \text{if } e_i = T_k 
\end{cases}
\]

The first term in the RHS of above equations captures edit specific score. The second term captures the score for copying the current word \( x_i \) to the output. The third term models the influence of a new incoming token in the output obtained by a replace or append edit. For replace edits, score of the replaced word is subtracted from score of the incoming word. For transformation we add the copy score because they typically modify only the word forms, hence we do not expect meaning of the transformed word to change significantly.

The above equation provides insights on why predicting independent edits is easier than predicting independent tokens. Consider the append edit \( (A(w)) \). Instead of independently predicting \( x_i \) at \( i \) and \( w \) at \( i+1 \), we jointly predict the tokens in these two slots and contrast it with not inserting any new \( w \) after \( x_i \) in a single softmax. We will show empirically (Sec 3.1.4) that such selective joint prediction is key to obtaining high accuracy in spite of parallel decoding.

Finally, loss for a training example \((e, x)\) is obtained by summing up the cross-entropy associated with predicting edit \( e_i \) at each token \( x_i \).

\[
\mathcal{L}(e, x) = -\Sigma_i \log(Pr(e_i|\mathbf{x})) \tag{4}
\]

3 Experiments

We compare our parallel iterative edit (PIE) model with state-of-the-art GEC models that are all based on attentional encoder-decoder architectures. In Section 3.2 we show that the PIE model is also effective on two other local sequence transduction tasks: spell and OCR corrections. Hyperparameters for all the experiments are provided in Table 12, 13 of the Appendix.

3.1 Grammatical error correction (GEC)

We use Lang-8 (Mizumoto et al., 2011), NUCLE (Ng et al., 2014) and FCE (Yannakoudakis et al., 2011) corpora, which jointly comprise 1.2 million sentence pairs in English. The validation dataset comprises of 1381 sentence pairs from CoNLL-13 (Ng et al., 2013) test set. We initialize our GEC model with the publicly available BERT-LARGE model that was pre-trained on Wikipedia (2500M words) and Book corpus (Zhu et al., 2015) (800M words) to predict 15% randomly masked words using its deep bidirectional context. Next, we perform 2 epochs of training on a synthetically perturbed version of the One-Billion-word corpus (Chelba et al., 2013). We refer to this as synthetic training. Details of how we create the synthetic corpus appear in Section A.3 of the appendix. Finally we fine-tune on the real GEC training corpus for 2 epochs. We use a batch size of 64 and learning rate 2e-5.

The edit space consists of copy, delete, 1000 appends, 1000 replaces and 29 transformations and their inverse. Arguments of Append and Replace operations mostly comprise punctuations, articles, pronouns, prepositions, conjunctions and verbs. Transformations perform inflections like add suffixes to, ing, d to s, ed to s, etc. These transformations were chosen out of common replaces in the training data such that many replace edits map to only a few transformation edits, in order to help the model better generalize replaces across different words. In Section 3.1.4 we see that transformations increase the model’s recall. The complete list of transformations appears in Table 10 of the Appendix. We evaluate on \( F_{0.5} \) score over span-level corrections from the MaxMatch (M2) scorer (Dahlmeier and Ng, 2012) on CONLL-2014-test. Like most existing GEC systems, we invoke a spell-checker on the test sentences before applying our model. We also report GLEU+ (Napoles et al., 2016) scores on JFLEG corpus (Napoles et al., 2017) to evaluate fluency.

---

1https://github.com/google-research/bert
<table>
<thead>
<tr>
<th>Work</th>
<th>CONLL-14 P</th>
<th>CONLL-14 R</th>
<th>JFLEG $F_{0.5}$</th>
<th>GLEU $^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhao et al. (2019)</td>
<td>67.7</td>
<td>40.6</td>
<td>59.8</td>
<td>-</td>
</tr>
<tr>
<td>Lichtarge et al. (2019)</td>
<td>65.5</td>
<td>37.1</td>
<td>56.8</td>
<td>61.6</td>
</tr>
<tr>
<td>Chollampatt and Ng (2018a)</td>
<td>69.9</td>
<td>23.7</td>
<td>46.4</td>
<td>51.3</td>
</tr>
<tr>
<td>PIE (This work)</td>
<td>66.1</td>
<td>43.0</td>
<td>59.7</td>
<td>60.3</td>
</tr>
</tbody>
</table>

Table 2: Comparison of non-ensemble model numbers for various GEC models. Best non-ensemble results are reported for each model.

### 3.1.1 Overall Results

Table 3 compares ensemble model results of PIE and other state of the art models, which all happen to be seq2seq ED models and also use ensemble decoding. For PIE, we simply average the probability distribution over edits from 5 independent ensembles. In Table-2 we compare non-ensemble numbers of PIE with the best available non-ensemble numbers of competing methods. On CoNLL-14 test-set our results are very close to the highest reported by Zhao et al. (2019). These results show that our parallel prediction model is competitive without incurring the overheads of beam-search and slow decoding of sequential models. GLEU $^+$ score, that rewards fluency, is somewhat lower for our model on the JFLEG test set because of parallel predictions. We do not finetune our model on the JFLEG dev set. We expect these to improve with re-ranking using a LM. All subsequent ablations and timing measurements are reported for non-ensemble models.

### 3.1.2 Running Time Comparison

Parallel decoding enables PIE models to be considerably faster than ED models. In Figure 3 we compare wall-clock decoding time of PIE with 24 encoder layers (PIE-LARGE, $F_{0.5} = 59.7$), PIE with 12 encoder layers (PIE-BASE, $F_{0.5} = 56.6$) and competitive ED architecture by Lichtarge et al. (2019) with 6 encoder and 6 decoder layers (T2T, $F_{0.5} = 56.8$ ) on CoNLL-14 test set. All decoding experiments were run and measured on a n1-standard-2$^2$ VM instance with a single TPU shard (v-2.8). We observe that even PIE-LARGE is between a factor of 5 to 15 faster than an equivalent transformer-based ED model (T2T) with beam-size 4. The running time of PIE-LARGE increases sub-linearly with sentence length whereas the ED model’s decoding time increases linearly.

### 3.1.3 Impact of Iterative Refinement

We next evaluate the impact of iterative refinements on accuracy in Table 4. Out of 1312 sentences in the test set, only 832 sentences changed in the first round which were then fed to the second round where only 156 sentences changed, etc. The average number of refinement rounds per example was 2.7. In contrast, a sequential model on this dataset would require 23.2 steps corresponding to the average number of tokens in a sentence. The $F_{0.5}$ score increases from 57.9 to 59.5 at the end of the second iteration.

Table 5 presents some sentences corrected by PIE. We see that PIE makes multiple parallel edits in a round if needed. Also, we see how refinement over successive iterations captures output space dependency. For example, in the second sentence interact gets converted to interacted followed by insertion of have in the next round.

### 3.1.4 Ablation study on the PIE Architecture

In this section we perform ablation studies to understand the importance of individual features of the PIE model.

**Synthetic Training** We evaluate the impact of training on the artificially generated GEC corpus in row 2 of Table 6. We find that without it the $F_{0.5}$ score is 3.4 points lower.

**Factorized Logits** We evaluate the gains due to...
Table 3: Comparison of recent GEC models trained using publicly available corpus. All the methods here except ours perform sequential decoding. Precision and Recall are represented by P and R respectively.

<table>
<thead>
<tr>
<th>Work</th>
<th>Method</th>
<th>CONLL-14</th>
<th>JFLEG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>Zhao et al. (2019)</td>
<td>Transformer + Pre-training + LM + Spellcheck + Ensemble Decoding (4)</td>
<td>71.6</td>
<td>38.7</td>
</tr>
<tr>
<td>Lichtarge et al. (2019)</td>
<td>Transformer + Pre-training + Iterative refinement + Ensemble Decoding (8)</td>
<td>66.7</td>
<td>43.9</td>
</tr>
<tr>
<td>Grundkiewicz and Junczys-Dowmunt (2018)</td>
<td>SMT + NMT (Bi-GRU) + LM + Ensemble Decoding (4) + Spellcheck</td>
<td>66.8</td>
<td>34.5</td>
</tr>
<tr>
<td>Chollampatt and Ng (2018a)</td>
<td>CNN + LM + Ensemble Decoding (4) + Spellcheck</td>
<td>65.5</td>
<td>33.1</td>
</tr>
<tr>
<td>PIE (This work)</td>
<td>Transformer + LM + Pre-training + Spellcheck + Iterative refinement + Ensembles (5)</td>
<td>68.3</td>
<td>43.2</td>
</tr>
</tbody>
</table>

Table 4: Statistics on successive rounds of iterative refinement. First row denotes number changed sentences (out of 1312) with each round on CONLL-14 (test).

<table>
<thead>
<tr>
<th>#edited</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>832</td>
<td>64.5</td>
<td>65.9</td>
<td>66.1</td>
<td>66.1</td>
</tr>
<tr>
<td>156</td>
<td>41.0</td>
<td>42.9</td>
<td>43.0</td>
<td>43.0</td>
</tr>
<tr>
<td>20</td>
<td>57.9</td>
<td>59.5</td>
<td>59.7</td>
<td>59.7</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Parallel and Iterative edits done by PIE.

| x | I started involving into Facebook one years ago .  
| PIE1 | I started involving in Facebook one year ago .  
| PIE2 | I started involved in Facebook one year ago .  
| PIE3 | I started getting involved in Facebook one year ago .  
| x | Since ancient times, human interact with others face by face .  
| PIE1 | Since ancient times, humans interacted with others face to face .  
| PIE2 | Since ancient times, humans have interacted with others face to face .  
| x | However, there are two sides of stories always .  
| PIE1 | However, there are always two sides to stories always .  
| PIE2 | However, there are always two sides to the stories .  
| PIE3 | However, there are always two sides to the story .  

3.2 More Sequence Transduction Tasks

We demonstrate the effectiveness of PIE model on two additional local sequence transduction tasks recently used in (Ribeiro et al., 2018).

Spell Correction We use the twitter spell correction dataset (Aramaki, 2010) which consists of 39172 pairs of original and corrected words obtained from twitter. We use the same train-dev-valid split as (Ribeiro et al., 2018) (31172/4000/4000). We tokenize on characters, and our vocabulary Σ and Σₐ comprises the 26 lower cased letters of English.

Correcting OCR errors We use the Finnish OCR data set³ by (Silfverberg et al., 2016) comprising words extracted from Early Modern

³https://github.com/mpsilfve/ocrpp
Methods | P  | R  | $F_{0.5}$ |
---|---|---|---|
1  | PIE       | 66.1 | 43.0 | 59.7 |
2  | – Synthetic training | 67.2 | 34.2 | 56.3 |
3  | – Factorized-logits | 66.4 | 32.8 | 55.1 |
4  | – Append + Inserts | 57.4 | 42.5 | 53.6 |
5  | – Transformations | 63.6 | 27.9 | 50.6 |
6  | – LM Pre-init | 48.8 | 18.3 | 36.6 |
7  | PIE on BERT-Base | 67.8 | 34.0 | 56.6 |

Table 6: Ablation study on the PIE model.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Spell</th>
<th>OCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq2Seq (Soft, lstm)</td>
<td>46.3</td>
<td>79.9</td>
</tr>
<tr>
<td>Seq2Seq (Hard, lstm)</td>
<td>52.2</td>
<td>58.4</td>
</tr>
<tr>
<td>Seq2Seq (Soft-T2T)</td>
<td>67.6</td>
<td>84.5</td>
</tr>
<tr>
<td>Ribeiro2018 (best model)</td>
<td>54.1</td>
<td>81.8</td>
</tr>
<tr>
<td>PIE</td>
<td>67.0</td>
<td>87.6</td>
</tr>
</tbody>
</table>

Table 7: Comparing PIE with Seq2Seq models (top-part) and Ribeiro’s parallel model on two other local sequence transduction tasks.

Finnish corpus of OCR processed newspaper text. We use the same train-dev-test splits as provided by (Silfverberg et al., 2016). We tokenize on characters in the word. For a particular split, our vocabulary $\Sigma$ and $\Sigma_a$ comprises of all the characters seen in the training data of the split.

Architecture For all the tasks in this section, PIE is a 4 layer self-attention transformer with 200 hidden units, 400 intermediate units and 4 attention heads. No $\mathcal{L}$ pre-initialization is done. Also, number of iterations of refinements is set to 1.

Results Table 7 presents whole-word 0/1 accuracy for these tasks on PIE and the following methods: Ribeiro et al. (2018)’s local transduction model (described in Section 4), and LSTM based ED models with hard monotonic attention (Aharoni and Goldberg, 2017) and soft-attention (Bahdanau et al., 2015) as reported in (Ribeiro et al., 2018). In addition, for a fair decoding time comparison, we also train a transformer-based ED model referred as Soft-T2T with 2 encoder, 2 decoder layers for spell-correction and 2 encoder, 1 decoder layer for OCR correction. We observe that PIE’s accuracy is comparable with ED models in both the tasks. Table 8 compares decoding speed of PIE with Soft-T2T in words/second. Since more than 90% of words have fewer than 9 tokens and the token vocabulary $\Sigma$ is small, decoding speed-ups of PIE over ED model on these tasks is modest compared to GEC.

### 4 Related Work

Grammar Error Correction (GEC) is an extensively researched area in NLP. See Ng et al. (2013) and Ng et al. (2014) for past shared tasks on GEC, and this⁴ website for current progress. Approaches attempted so far include rules (Felice et al., 2014), classifiers (Rozovskaya and Roth, 2016), statistical machine translation (SMT) (Junczys-Dowmunt and Grundkiewicz, 2016), neural ED models (Chollampatt and Ng, 2018a; Junczys-Dowmunt et al., 2018; Ge et al., 2018a), and hybrids (Grundkiewicz and Junczys-Dowmunt, 2018). All recent neural approaches are sequential ED models that predict either word sequences (Zhao et al., 2019; Lichtarge et al., 2019) or character sequences (Xie et al., 2016) using either multi-layer RNNs (Ji et al., 2017; Grundkiewicz and Junczys-Dowmunt, 2018) or CNNs (Chollampatt and Ng, 2018a; Ge et al., 2018a) or Transformers (Junczys-Dowmunt et al., 2018; Lichtarge et al., 2019). Our sequence labeling formulation is similar to (Yannakoudakis et al., 2017) and (Kaili et al., 2018) but the former uses it to only detect errors and the latter only corrects five error-types using separate classifiers. Edits have been exploited in earlier GEC systems too but very unlike our method of re-architecting the core model to label input sequence with edits. Schmaltz et al. (2017) interleave edit tags in target tokens but use seq2seq learning to predict the output sequence. Chollampatt and Ng (2018a) use edits as features for rescoring seq2seq predictions. Junczys-Dowmunt et al. (2018) use an edit-weighted MLE objective to emphasise corrective edits during seq2seq learning. Stahlberg et al. (2019) use finite state transducers, whose state transitions denote possible edits, built from an unlabeled corpus to constrain the output of a neural beam decoder to a small GEC-feasible space.

<table>
<thead>
<tr>
<th></th>
<th>PIE</th>
<th>Soft-T2T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spell</td>
<td>80.43</td>
<td>36.62</td>
</tr>
<tr>
<td>OCR</td>
<td>65.44</td>
<td>43.02</td>
</tr>
</tbody>
</table>

Table 8: Wall-clock decoding speed in words/second of PIE and a comparable Seq2Seq T2T model.

⁴https://nlpprogress.com/english/grammatical_error_correction.html

Local Sequence Transduction is handled in Ribeiro et al. (2018) by first predicting insert slots in \( x \) using learned insertion patterns and then using a sequence labeling task to output tokens in \( x \) or a special token delete. Instead, we output edit operations including word transformations. Their pattern-based insert pre-sloting is unlikely to work for more challenging tasks like GEC. Koide et al. (2018) design a special edit-invariant neural network for being robust to small edit changes in input biological sequences. This is a different task than ours of edit prediction. Yin et al. (2019) is about neural representation of edits specifically for structured objects like source code. This is again a different problem than ours.

5 Conclusion

We presented a parallel iterative edit (PIE) model for local sequence transduction with a focus on the GEC task. Compared to the popular encoder-decoder models that perform sequential decoding, parallel decoding in the PIE model yields a factor of 5 to 15 reduction in decoding time. The PIE model employs a number of ideas to match the accuracy of sequential models in spite of parallel decoding: it predicts in-place edits using a carefully designed edit space, iteratively refines its own predictions, and effectively reuses state-of-the-art pre-trained bidirectional language models. In the future we plan to apply the PIE model to more ambitious transduction tasks like translation.

Acknowledgements This research was partly sponsored by a Google India AI/ML Research Award and Google PhD Fellowship in Machine Learning. We gratefully acknowledge Google’s TFRC program for providing us Cloud-TPUs. We thank Varun Patil for helping us improve the speed of pre-processing and synthetic-data generation pipelines.

References


Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. ICLR.


Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y. Ng. 2016. Neural language correction with character-based attention. CoRR, abs/1603.09727.


