π Parallel Iterative Edit Models for Local Sequence Transduction

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, Vihari Piratla

Correspondence: awasthi@cse.iitb.ac.in @ Awasthi_A_

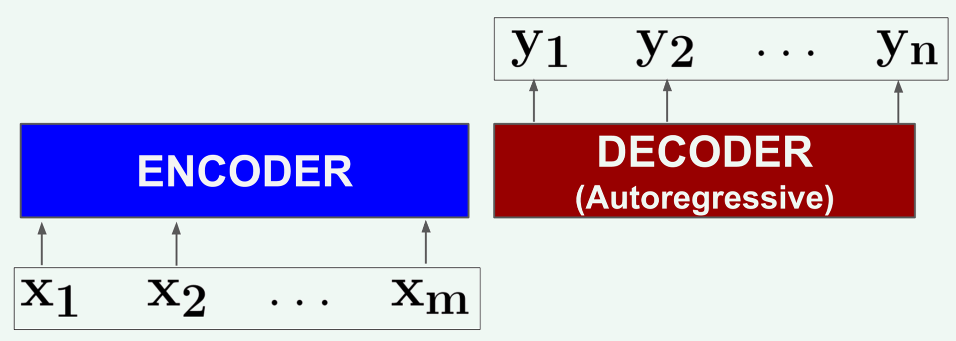
Grammatical Error Correction made 5 to 15 times faster by sequence labeling with π

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 seq2seq 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. Labeling sequences with edits instead of generating sequences, 2. Iterative refinement to capture missed dependencies, and 3. Rewiring a pre-trained language model like BERT for edit predictions. Experiments on tasks spanning GEC, OCR denoising and spell correction demonstrate that the PIE model is an accurate and significantly faster alternative.

Standard Approach

Translate incorrect sequence to correct sequence using auto-regressive encoder decoder models



Why explicitly generate the target sequence from scratch ?

All we need is a few local edits to the input !

Highlights

1. Labeling with edits instead of translation
2. Non-autoregressive, parallel predictions
3. Iterative refinement for capturing missed dependencies
4. Rewiring BERT for sequence editing

Local Sequence Transduction Problems

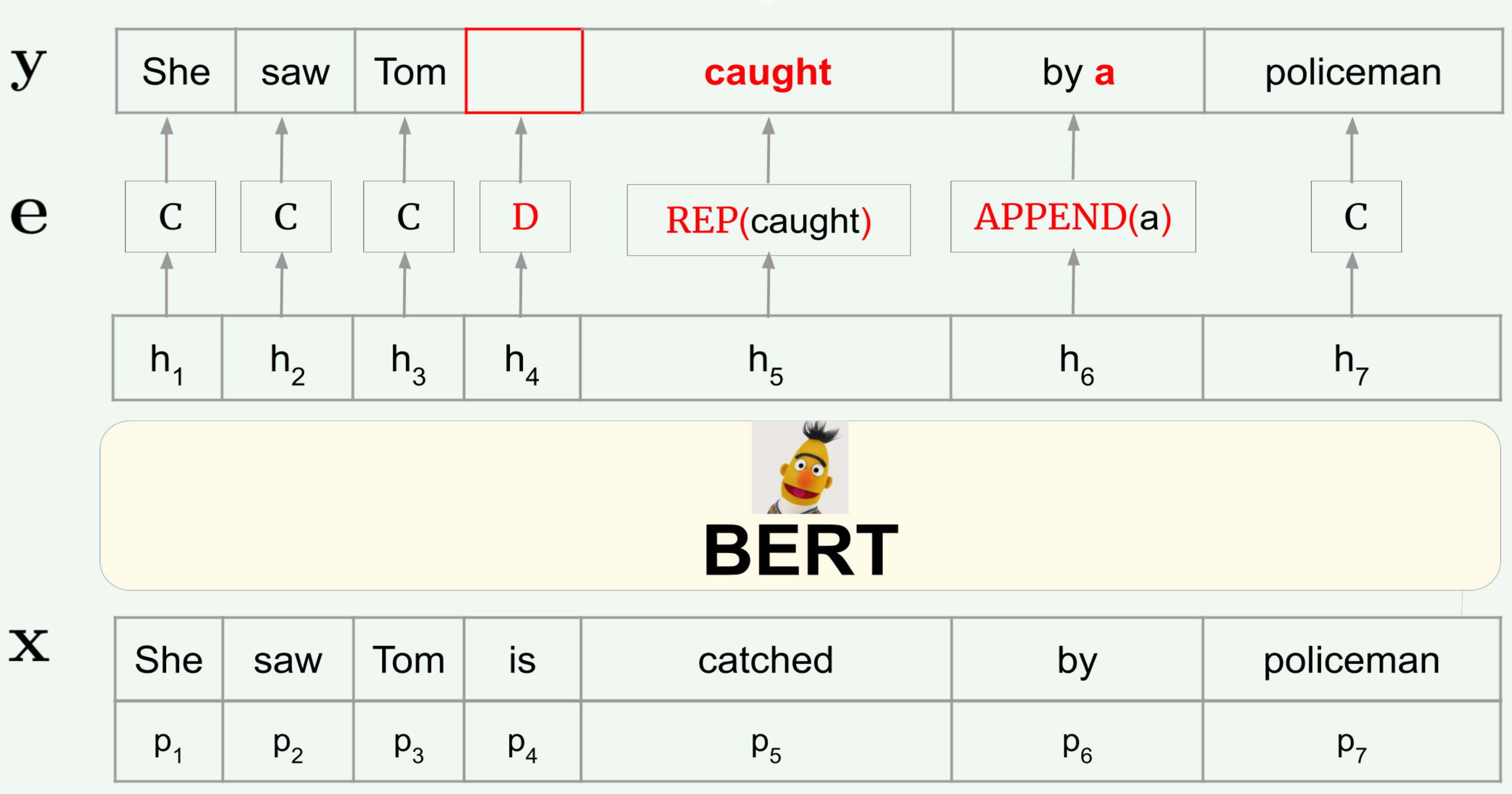
1. Grammatical Error Correction
2. Spell Correction
3. OCR – denoising

Key Property: Source and Target Sequence are generally not too different

Our Approach

Labeling incorrect sequence with edits

Non-autoregressive Parallel Predictions



From translation to sequence labeling with edits

Original Problem: Translation

$\mathbf{x}$ He catched by policeman

$\mathbf{y}$ He was caught by a policeman

Modification: Sequence Editing

$\mathbf{x}$ He catched by policeman $\quad \text{len}(\mathbf{x}) \neq \text{len}(\mathbf{e})$

$\mathbf{e}$ COPY INS(was) REP(caught) COPY INS(a) COPY

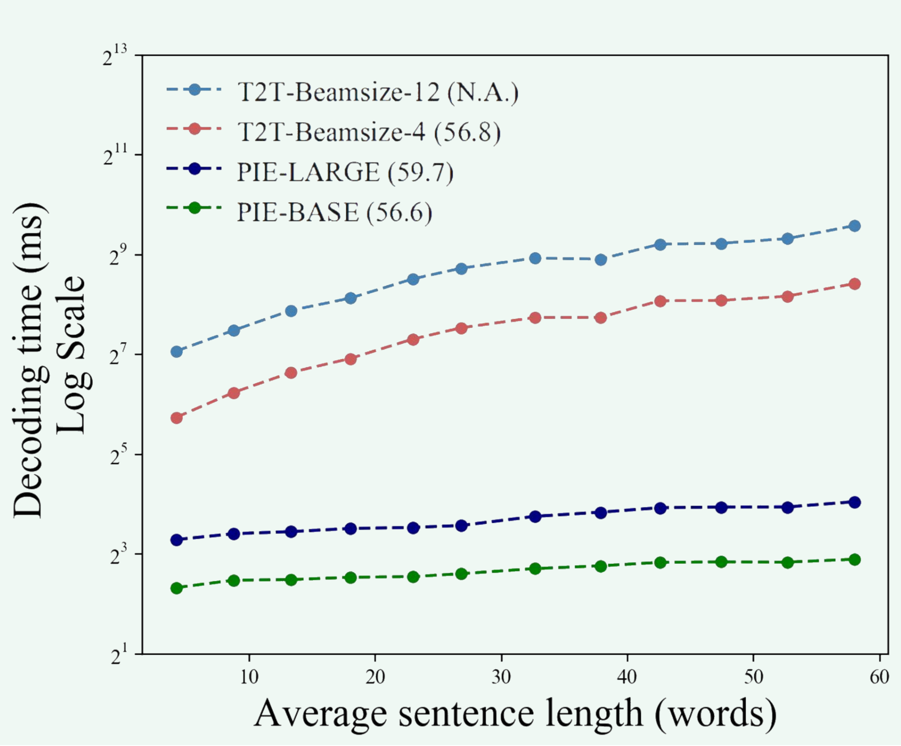
Simplification: Sequence Labeling

Trick: Merge COPY INS(.) to form Append(.) !

$\mathbf{x}$ He catched by policeman $\quad \text{len}(\mathbf{x}) = \text{len}(\mathbf{e})$

$\mathbf{e}$ Append(was) REP(caught) Append(a) COPY

How fast is sequence labeling w.r.t. translation?



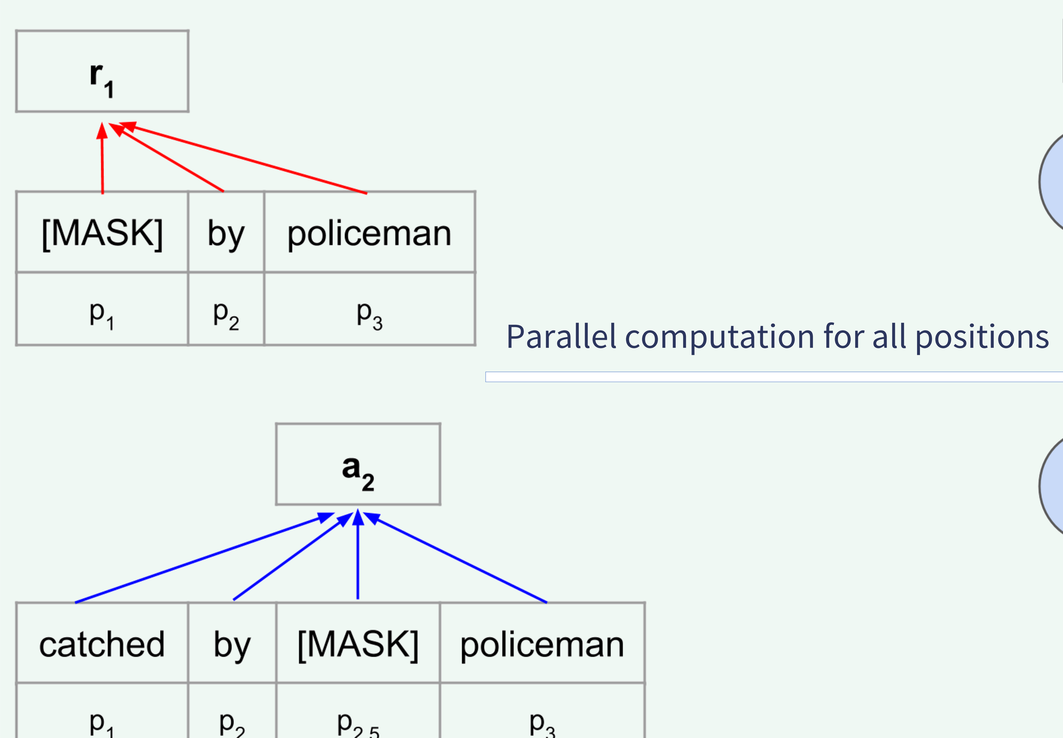
*Comparison of single round non-ensemble models *T2T: Litchage et al. NAACL 2019

How accurate is sequence labeling w.r.t. translation?

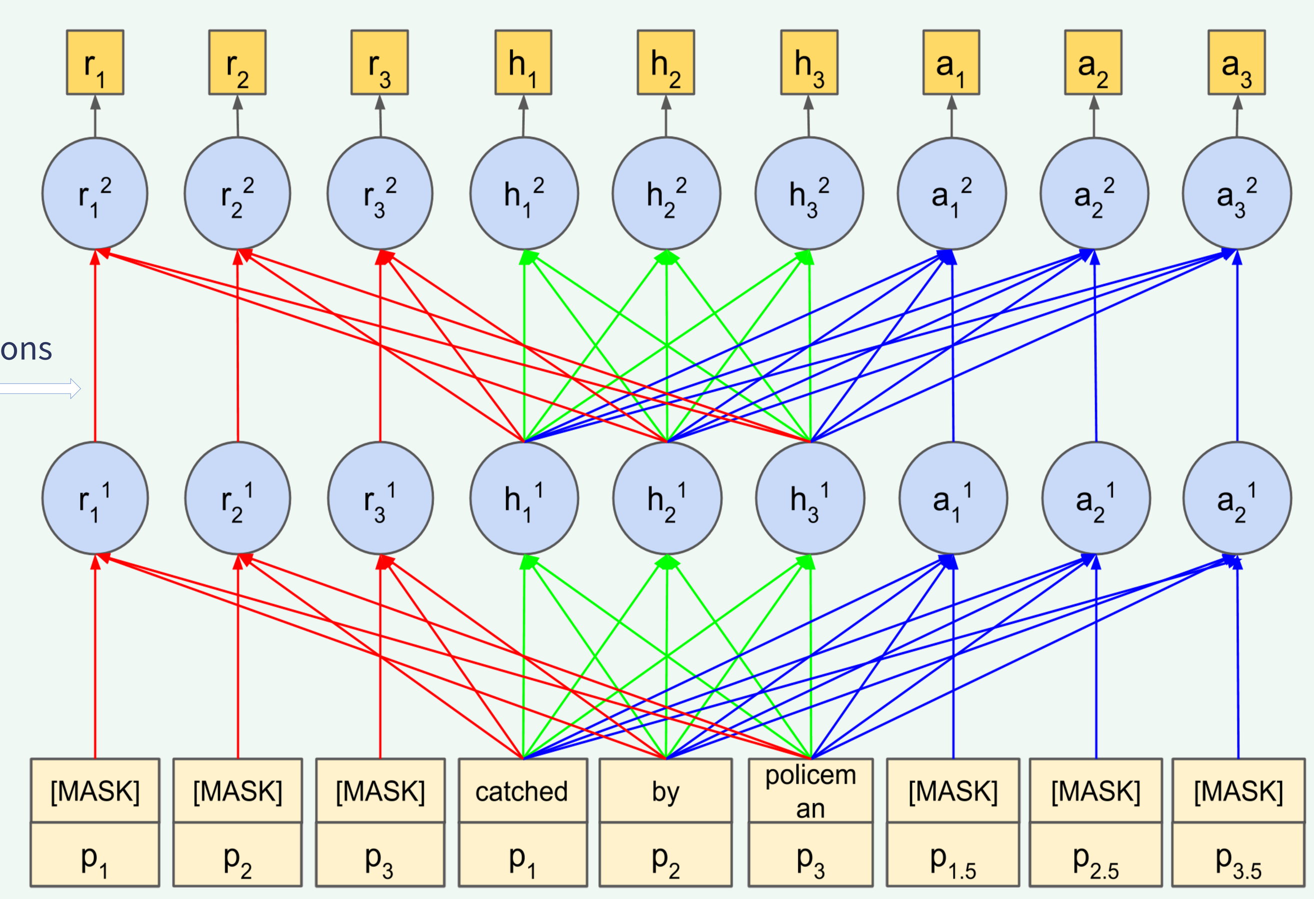
Work	Architecture	$F_{0.5}$
Lichtarge et al. NAACL 2019	Seq2Seq, Transformers	60.4
Zhao et al. NAACL 2019	Seq2Seq, Transformers	61.2
PIE (This work) EMNLP 2019	Sequence Labeling, Transformers	61.2
Kiyono et al. EMNLP 2019	Seq2Seq, Transformers + Better Pseudo data	65.0 *

* PIE is expected to provide similar accuracy gains when pre-trained with pseudo data of Kiyono et al.

Rewiring (without retraining) BERT for Sequence Editing



Utilizing BERT's ability to fill in the blanks for guiding Appends and Replace edits



Factorizing logit scores over edits and token arguments

$$\Pr(e_i|\mathbf{x}) = \text{softmax}(\text{logit}(e_i|\mathbf{x})) \quad \text{where}$$

$$\text{logit}(e_i|\mathbf{x}) = \begin{cases} \theta_{\text{C}}^{\intercal}\mathbf{h}_i + \phi(x_i)^{\intercal}\mathbf{h}_i + 0 & \text{if } e_i = \text{C} \\ \theta_{\text{A}(w)}^{\intercal}\mathbf{h}_i + \phi(x_i)^{\intercal}\mathbf{h}_i + \phi(w)^{\intercal}\mathbf{a}_i & \text{if } e_i = \text{A}(w) \\ \theta_{\text{R}(w)}^{\intercal}\mathbf{h}_i + 0 + (\phi(w) - \phi(x_i))^{\intercal}\mathbf{r}_i & \text{if } e_i = \text{R}(w) \\ \theta_{\text{D}}^{\intercal}\mathbf{h}_i + 0 + 0 & \text{if } e_i = \text{D} \end{cases}$$

Iterative Refinement of parallelly edited sentences

Input : However , there are two sides of stories always .

Iter1: However , there are always two sides to stories ~~always~~ .

Iter2 : However , there are always two sides to the stories .

Iter3 : However, there are always two sides to the story .

Acknowledgements : Google's TFRC program for TPUs, Google PhD Fellowship in Machine Learning, Microsoft Student Travel Grant