Transformers and Graph Neural Networks for Spell Checking

Master's thesis presentation

Sebastian Walter

June 27, 2022

Problem: Motivation

- Correct human written text
  - Documents
  - Email
- Preprocessing or postprocessing step for other NLP systems
  - Before using search engine
  - After using OCR system

Problem: Tasks

- Spelling error detection
  - Assign to each word in a text either 0 or 1
  - *This ahs an eror!* $\rightarrow$ *(0, 1, 0, 1)*

Problem: Tasks

- Spelling error detection
  - Assign to each word in a text either 0 or 1
  - *This ahs an eror!* $\rightarrow$ *(0, 1, 0, 1)*
- Spelling error correction
  - Given a misspelled text S predict the correct text S'
  - *This ahs an eror!* $\rightarrow$ *This has an error!*

Problem: Tasks

- Spelling error detection
  - Assign to each word in a text either 0 or 1
  - *This ahs an eror!* $\rightarrow$ *(0, 1, 0, 1)*
- Spelling error correction
  - Given a misspelled text S predict the correct text S'
  - *This ahs an eror!* $\rightarrow$ *This has an error!*

No whitespace errors

For now we assume that the input text contains no whitespace errors.

Problem

Questions?

Approach: Datasets

- Wikidump / Bookcorpus
  - Paragraphs
  - Based on Wikipedia dump and Bookcorpus
  - $\sim$102M samples

Approach: Datasets

- Wikidump / Bookcorpus
  - Paragraphs
  - Based on Wikipedia dump and Bookcorpus
  - ~102M samples
- Neuspell
  - Sentences
  - Based on One Billion Word dataset
  - ~4M samples
  - Finetuning

Approach: Spelling error detection

Main components

- Subword-level input tokenization
- Parallel encoder architectures for fast inference
- Word features
  $\rightarrow$ Strong signals for presence or absence of spelling errors
  - *word in dictionary*
  - *word is punctuation*
  - *word is stop word*
  - ...

Approach: Spelling error detection

1. Transformer^{+}
   - Backbone: Transformer encoder
   - Procedure:
     1. Encode subword sequence using transformer encoder
     2. Aggregate subword representations to word representations
     3. Add word features to word representations
     4. Aggregate and classify word representations

Approach: Spelling error detection

2 GNN$^+$

- Backbone: Attentional graph neural network
- Procedure:
  1. Build word graph from subword sequence
  2. Encode word graph using graph neural network
  3. Aggregate and classify word node representations

+ Learn word and subword representations simultaneously
+ Add word features directly into the graph from the beginning

Approach: Spelling error detection

2 GNN$^+$

- Backbone: Attentional graph neural network
- Procedure:
  1. Build word graph from subword sequence
  2. Encode word graph using graph neural network
  3. Aggregate and classify word node representations

+ Learn word and subword representations simultaneously
+ Add word features directly into the graph from the beginning

GNN and Transformer

Variants without word features for comparison

Approach: Spelling error correction

Main components

- Subword-level input tokenization
- Autoregressive decoder architectures for open vocabulary correction
  $\rightarrow$ Subword-level output tokenization

Approach: Spelling error correction

1. Transformer
   - Backbone: Transformer encoder and decoder
   - Procedure:
     1. Encode input subword sequence using Transformer encoder
     2. Autoregressively decode output subword sequence using Transformer decoder

Approach: Spelling error correction

2 Transformer Word

- Backbone: Transformer encoder and decoder
- Procedure:
  1. Encode input subword sequence using Transformer encoder
  2. Split subword representations into word groups
  3. Autoregressively decode each word group separately by sharing the Transformer decoder

+ Shared decoder allows decoding all words in parallel

Approach: Detection and correction

Pipeline

1. Use detection models to identify spelling errors
2. Apply correction models only to detected spelling errors

Approach: Detection and correction

Pipeline

1. Use detection models to identify spelling errors
2. Apply correction models only to detected spelling errors

Goal

- Improve runtime
- Reduce number of wrong corrections

Approach

Questions?

Evaluation: Benchmarks

- Neuspell
  - 4 benchmarks from literature (Jayanthi, Pruthi, and Neubig, 2020)
  - Spelling errors extracted from real-world GEC data

Evaluation: Benchmarks

- Neuspell
  - 4 benchmarks from literature (Jayanthi, Pruthi, and Neubig, 2020)
  - Spelling errors extracted from real-world GEC data
- Our benchmarks
  - 4 benchmarks
  - Artificial and realistic misspellings into both Wikidump and Bookcorpus

Evaluation: Baselines

- Classical methods
  - Jamspell (n-gram language model)
  - ...

Evaluation: Baselines

- Classical methods
  - Jamspell (n-gram language model)
  - ...
- Deep learning methods
  - Neuspell Bert

    Bert encoder with fixed output vocabulary classifier
  - GECToR

    Bert/XLNet-based grammatical error correction model by Grammarly
  - NLMSpell

    Language model to score candidate corrections
  - Google

    Google Docs' integrated spell checker
  - GPT-3

    Large language model with input prompt
    Fix the spelling mistakes: <text> [<output>]

Evaluation: Spelling error detection metric

F_1 score

- Sets:
  - TP: Word with error predicted to be a spelling error
  - FP: Word without error predicted to be a spelling error
  - FN: Word with error predicted to not be a spelling error
- Precision $= \frac{\text{TP}}{\text{TP} + \text{FP}}$
- Recall $= \frac{\text{TP}}{\text{TP} + \text{FP}}$
- $F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Evaluation: Spelling error detection

	neuspell bea60k			neuspell jfleg		
	F_1	Precision	Recall	F_1	Precision	Recall
out of dictionary	91.61	86.70	97.11	87.48	94.23	81.63
jamspell	90.68	91.73	89.65	88.36	96.85	81.23
gector	67.08	58.41	78.78	57.74	49.53	69.23
neuspell bert	88.65	84.23	93.56	**88.37**	86.88	89.91
ours	**92.02**	88.86	95.42	88.23	88.91	87.56

Neuspell benchmarks

	bookcorpus artificial			bookcorpus realistic			wikidump artificial			wikidump realistic		
	F_1	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall
out of dictionary	82.03	92.04	73.99	79.88	89.32	72.24	80.93	84.56	77.61	76.64	79.54	73.95
jamspell	83.70	95.62	74.42	76.71	93.85	64.86	84.29	93.30	76.86	76.92	90.33	66.98
gector	58.01	76.73	46.63	63.38	77.17	53.77	50.18	79.34	36.70	59.78	80.85	47.42
neuspell bert	76.36	94.40	64.11	91.50	94.95	88.30	71.84	92.73	58.63	89.49	93.74	85.61
ours	**96.62**	97.41	95.84	**95.36**	96.56	94.20	**95.52**	97.20	93.90	**95.61**	96.81	94.45

Our benchmarks

Evaluation: Spelling error detection

- On 5 / 6 benchmarks our best model performs best overall

Evaluation: Spelling error detection

- On 5 / 6 benchmarks our best model performs best overall
- GNN outperforms Transformer on 6 / 6 benchmarks
  $\rightarrow$ F_1 0.53 p.p. higher on average

Evaluation: Spelling error detection

- On 5 / 6 benchmarks our best model performs best overall
- GNN outperforms Transformer on 6 / 6 benchmarks
  $\rightarrow$ F_1 0.53 p.p. higher on average
- GNN$^+$ outperforms Transformer$^+$ on 3 / 6 benchmarks
  $\rightarrow$ F_1 0.14 p.p. higher on average

Evaluation: Spelling error detection

- On 5 / 6 benchmarks our best model performs best overall
- GNN outperforms Transformer on 6 / 6 benchmarks
  $\rightarrow$ F_1 0.53 p.p. higher on average
- GNN^+ outperforms $Transformer^+$ on 3 / 6 benchmarks
  $\rightarrow$ F_1 0.14 p.p. higher on average
- On 6 / 6 benchmarks our best model was using word features

Evaluation: Spelling error correction metric

F_1 score (following Hertel, 2019)

- Sets:
  - TP: Word with error properly corrected
  - FP: Word without error changed or word with error not properly corrected
  - FN: Word with error not corrected
- Precision $= \frac{\text{TP}}{\text{TP} + \text{FP}}$
- Recall $= \frac{\text{TP}}{\text{TP} + \text{FP}}$
- $F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Evaluation: Spelling error correction

	neuspell bea322			neuspell bea4660			neuspell bea60k			neuspell jfleg		
	F_1	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall
jamspell	53.74	55.23	52.32	71.70	70.53	72.91	69.87	70.67	69.08	81.54	89.32	75.01
gector	55.49	49.88	62.54	70.34	58.78	87.54	60.31	53.33	69.39	48.29	42.97	55.12
neuspell bert	65.36	59.54	72.45	85.14	78.77	92.61	74.73	71.00	78.87	83.03	81.60	84.52
ours	**68.48**	66.76	70.28	**85.75**	82.27	89.55	**77.43**	78.74	76.16	**85.92**	86.80	85.06

Neuspell benchmarks

	bookcorpus artificial			bookcorpus realistic			wikidump artificial			wikidump realistic		
	F_1	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall
jamspell	47.70	54.48	42.41	48.03	58.76	40.61	49.52	54.81	45.16	46.57	54.69	40.55
gector	34.65	49.46	26.67	33.71	44.60	27.10	35.12	59.03	25.00	35.97	52.52	27.35
neuspell bert	56.79	70.21	47.69	61.82	64.15	59.66	56.01	72.29	45.71	57.67	60.41	55.17
ours	**87.64**	89.50	85.86	**72.69**	75.45	70.13	**88.26**	89.26	87.29	**75.10**	77.89	72.51

Our benchmarks

Evaluation: Spelling error correction

	combined neuspell			combined wikibook		
	F_1	Precision	Recall	F_1	Precision	Recall
gpt3	**89.50**	89.40	89.60	74.13	88.87	63.59
nlmspell	72.68	68.39	77.54	78.86	80.38	77.39
google	71.61	64.05	81.19	58.74	72.78	49.24
ours	81.10	80.35	81.86	**80.28**	81.21	79.37

Combined benchmarks

Evaluation: Spelling error correction

	combined neuspell			combined wikibook		
	F_1	Precision	Recall	F_1	Precision	Recall
gpt3	**89.50**	89.40	89.60	74.13	88.87	63.59
nlmspell	72.68	68.39	77.54	78.86	80.38	77.39
google	71.61	64.05	81.19	58.74	72.78	49.24
ours	81.10	80.35	81.86	**80.28**	81.21	79.37

Combined benchmarks

- On 9 / 10 benchmarks our best model performs best overall
- On 7 / 10 benchmarks our best model is a pipeline

Evaluation: Runtimes

Task	Model	Runtime in s	kB/s
SED	gnn^+	6.8	25.8
SED	gnn	6.6	26.4
SED	transformer^+	3.7	47.6
SED	transformer	3.5	50.6
SEC	transformer	65.1	2.7
SED $\rightarrow$ SEC	$\text{gnn}^+ \rightarrow$ transformer	47.9	3.7
SED $\rightarrow$ SEC	$\text{transformer}^+ \rightarrow$ transformer	45.9	3.8
SEC	transformer word	28.0	6.3
SEC	neuspell bert	18.1	9.6
SED $\rightarrow$ SEC	$\text{gnn}^+ \rightarrow$ transformer word	16.4	10.7
SED $\rightarrow$ SEC	$\text{transformer}^+ \rightarrow$ transformer word	13.3	13.3

Runtimes on runtime benchmark with 1600 samples or 231kB of text

Evaluation: Runtimes

Task	Model	Runtime in s	kB/s
SED	gnn$^+$	6.8	25.8
SED	gnn	6.6	26.4
SED	transformer$^+$	3.7	47.6
SED	transformer	3.5	50.6
SEC	transformer	65.1	2.7
SED $\rightarrow$ SEC	gnn$^+$ $\rightarrow$ transformer	47.9	3.7
SED $\rightarrow$ SEC	transformer$^+$ $\rightarrow$ transformer	45.9	3.8
SEC	transformer word	28.0	6.3
SEC	neuspell bert	18.1	9.6
SED $\rightarrow$ SEC	gnn$^+$ $\rightarrow$ transformer word	16.4	10.7
SED $\rightarrow$ SEC	transformer$^+$ $\rightarrow$ transformer word	13.3	13.3

Runtimes on runtime benchmark with 1600 samples or 231kB of text

- Models with word features are marginally slower than those without

Evaluation: Runtimes

Task	Model	Runtime in s	kB/s
SED	gnn$^+$	6.8	25.8
SED	gnn	6.6	26.4
SED	transformer$^+$	3.7	47.6
SED	transformer	3.5	50.6
SEC	transformer	65.1	2.7
SED $\rightarrow$ SEC	gnn$^+$ $\rightarrow$ transformer	47.9	3.7
SED $\rightarrow$ SEC	transformer$^+$ $\rightarrow$ transformer	45.9	3.8
SEC	transformer word	28.0	6.3
SEC	neuspell bert	18.1	9.6
SED $\rightarrow$ SEC	gnn$^+$ $\rightarrow$ transformer word	16.4	10.7
SED $\rightarrow$ SEC	transformer$^+$ $\rightarrow$ transformer word	13.3	13.3

Runtimes on runtime benchmark with 1600 samples or 231kB of text

- Models with word features are marginally slower than those without
- Transformer Word is faster than regular Transformer

Evaluation: Runtimes

Task	Model	Runtime in s	kB/s
SED	gnn$^+$	6.8	25.8
SED	gnn	6.6	26.4
SED	transformer$^+$	3.7	47.6
SED	transformer	3.5	50.6
SEC	transformer	65.1	2.7
SED $\rightarrow$ SEC	gnn$^+$ $\rightarrow$ transformer	47.9	3.7
SED $\rightarrow$ SEC	transformer$^+$ $\rightarrow$ transformer	45.9	3.8
SEC	transformer word	28.0	6.3
SEC	neuspell bert	18.1	9.6
SED $\rightarrow$ SEC	gnn$^+$ $\rightarrow$ transformer word	16.4	10.7
SED $\rightarrow$ SEC	transformer$^+$ $\rightarrow$ transformer word	13.3	13.3

Runtimes on runtime benchmark with 1600 samples or 231kB of text

- Models with word features are marginally slower than those without
- Transformer Word is faster than regular Transformer
- Pipelines result in overall faster inference

Evaluation: Conclusion

Key takeaways

- Deep learning methods > Classical methods
- GNNs are competitive to Transformers for spelling error detection
- Adding word features improves spelling error detection
- Spelling error detection can improve spelling error correction in runtime and performance

Evaluation

Questions?

References I

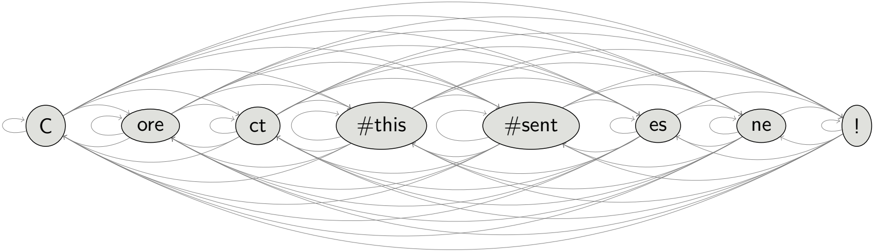
Hertel, Matthias (Dec. 6, 2019). *Neural Language Models for Spelling Correction*.

Jayanthi, Sai Muralidhar, Danish Pruthi, and Graham Neubig (Oct. 2020). "NeuSpell: A Neural Spelling Correction Toolkit". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, pp. 158–164. DOI: 10.18653/v1/2020.emnlp-demos.21.

Appendix: Misspellings

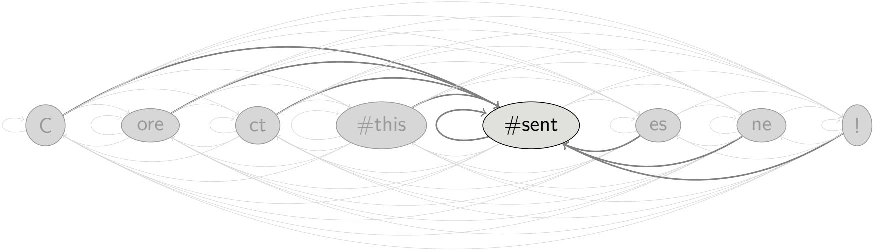
- Artificial
  - Randomly apply character transformations to word
  - Transformations:
    - Insertion (*word* → *wordi*)
    - Deletion (*word* → *wrd*)
    - Transposition (*word* → *wrod*)
    - Replacement (*word* → *worx*)
- Realistic:
  - Randomly replace a words with misspellings from a confusion set
  - Build word confusion sets from a variety of different sources (e.g. Internet, spell checker suggestions, ...):
    2,303,867 misspellings for 119,725 words

Appendix: Token graph



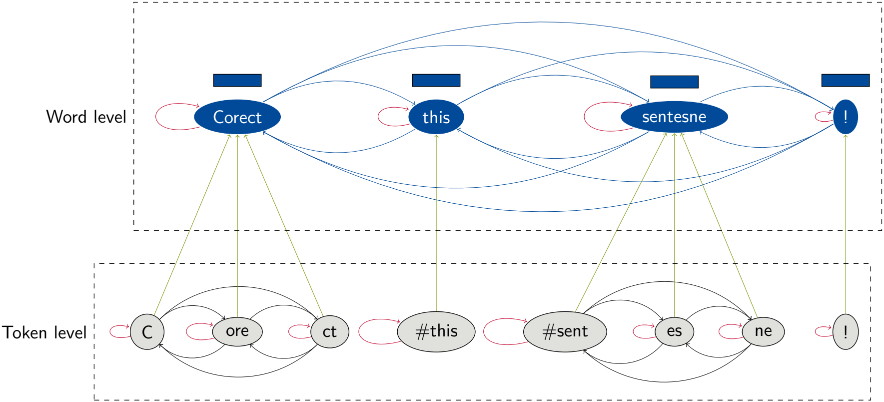
Input: *Corect this sentesne!*

Appendix: Token graph neighborhood



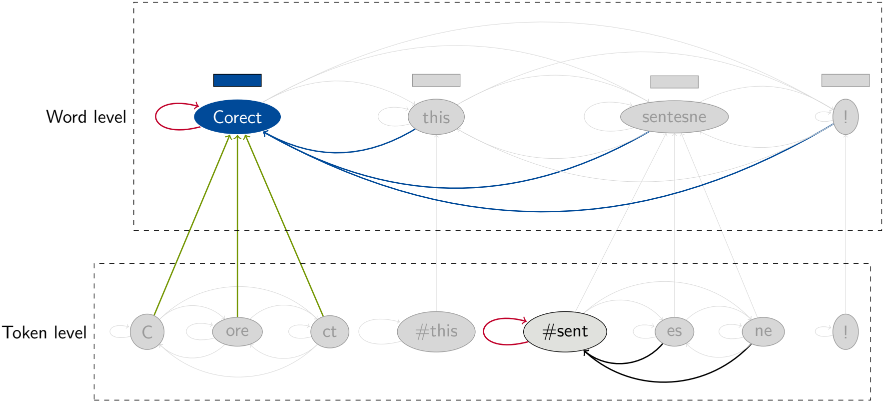
Input: *Corect this sentesne!*

Appendix: Word graph



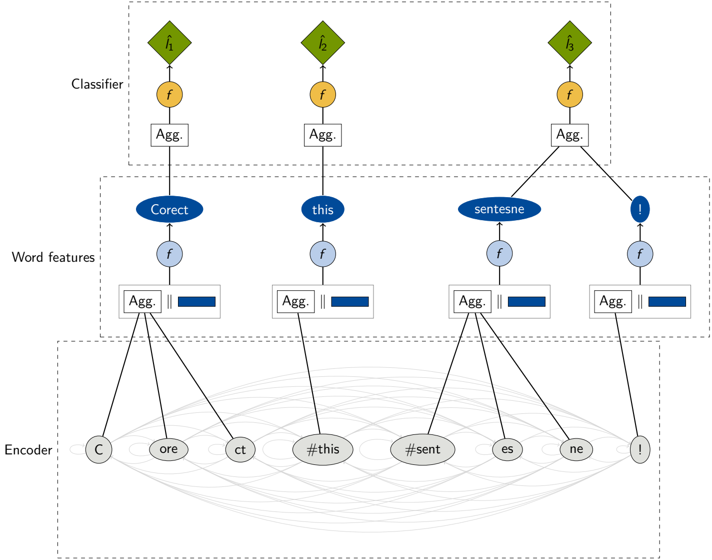
Input: *Corect this sentesne!*

Appendix: Word graph neighborhood



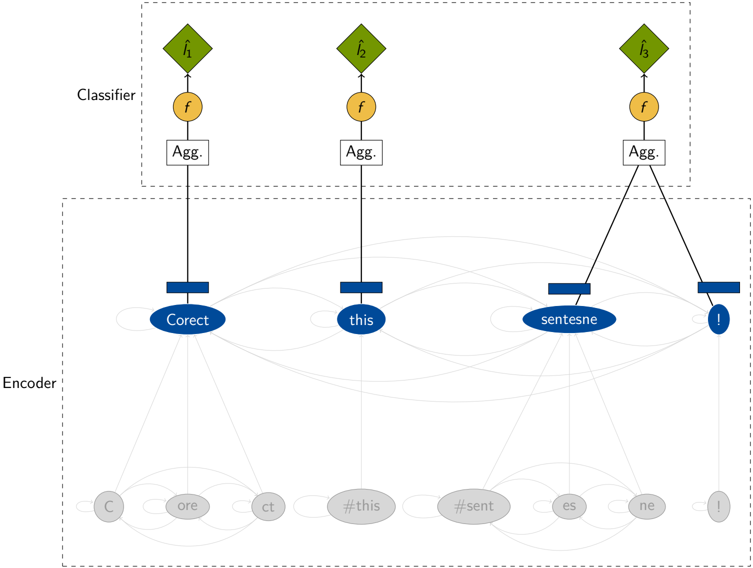
Input: *Corect this sentesne!*

Appendix: Transformer$^+$



Input: *Corect this sentesne!*

Appendix: GNN$^+$

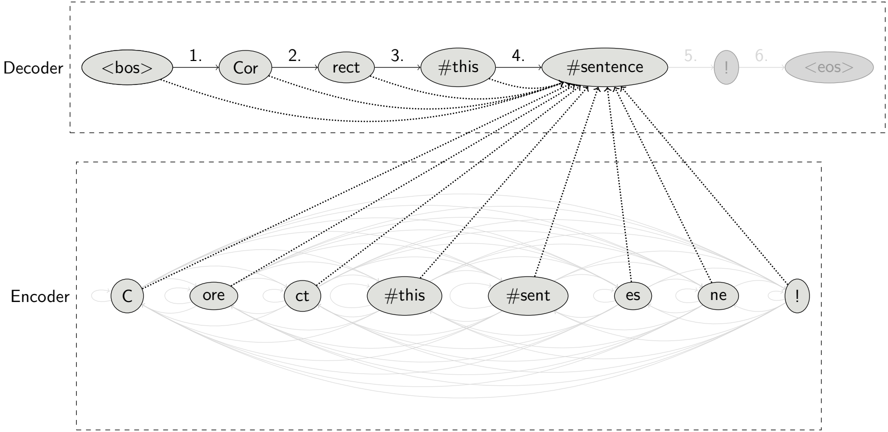


Input: *Corect this sentesne!*

Appendix: Word features

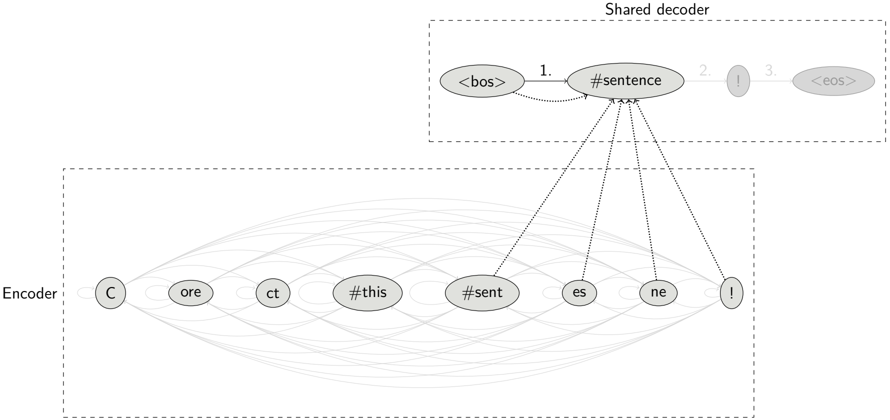
- Improved detection rates (recall)
- Largest improvement for realistic nonword errors
  - *And it's creating a lot of adict people, called "shopalcoholics".*
  - *So in the last century our daily life has changed dramandesly and we have become lazy and our life unpersonal, fast and unromantic.*
  - *The job was hard, but, from my point of view, it was worthful.*

Appendix: Transformer



Input: *Corect this sentesne!*

Appendix: Transformer Word



Input: *Corect this sentesne!*

Appendix: Benchmark statistics

Benchmark	#Sequences	#Words	$\overline{\text{Sequence length}}$	Word errors	Real-word errors	Nonword errors
bookcorpus artificial	10,000	407,347	193.8	83,124 (20.4%)	19,346 (23.3%)	63,778 (76.7%)
bookcorpus realistic	10,000	407,074	194.3	82,855 (20.4%)	22,868 (27.6%)	59,987 (72.4%)
neuspell bea322	322	5,275	75.9	323 (6.1%)	13 (4.0%)	310 (96.0%)
neuspell bea4660	4,660	136,475	143.4	5,714 (4.2%)	547 (9.6%)	5,167 (90.4%)
neuspell bea60k	63,044	997,600	75.5	70,064 (7.0%)	1,970 (2.8%)	68,094 (97.2%)
neuspell jfleg	1,601	33,414	105.6	2,041 (6.1%)	374 (18.3%)	1,667 (81.7%)
wikidump artificial	10,000	365,829	196.0	73,925 (20.2%)	14,783 (20.0%)	59,142 (80.0%)
wikidump realistic	10,000	365,753	196.2	73,390 (20.1%)	18,760 (25.6%)	54,630 (74.4%)

Word-level spelling error detection and spelling error correction

Appendix: Pipeline

Input: *This sentesne has an eror.*
Tokenization: *(This, #sent, es, ne, #has, #an, #er, or, .)*
Detections: *(0, 1, 0, 0, 1)*

- Transformer
  - Assume parts of input with no error are correct
  - Decode input prefix in order until next whitespace
    1. *(This)* $\rightarrow$ *(This, #sentence)*
    2. *(This, #sentence, #has, #an)*
       $\rightarrow$ *(This, #sentence, #has, #an, #error, .)*
- Transformer Word
  - Correct word groups with spelling errors in parallel

    (#sent, es, ne) $\rightarrow$ *(#sentence)*
    (#er, or, .) $\rightarrow$ *(#error, .)*

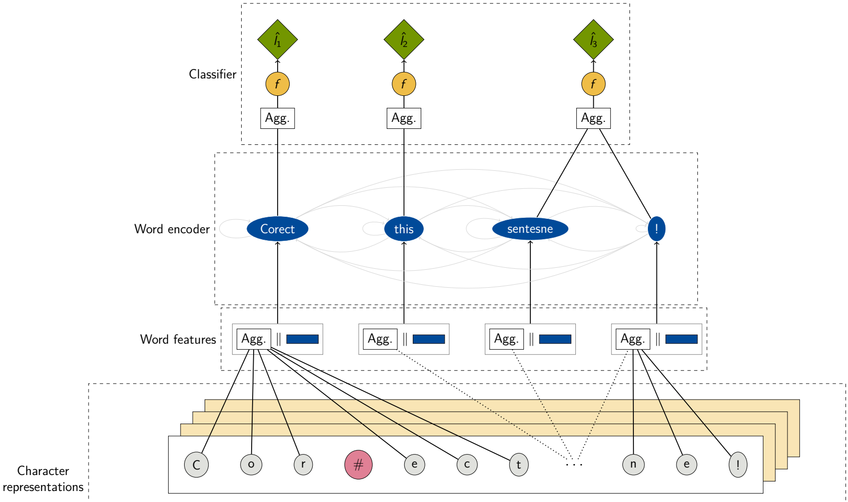
Appendix: Handling whitespaces

Tokenization repair

Task of correcting all whitespace errors in text with or without spelling errors. Can be efficiently handled using character-level Transformer encoder models.

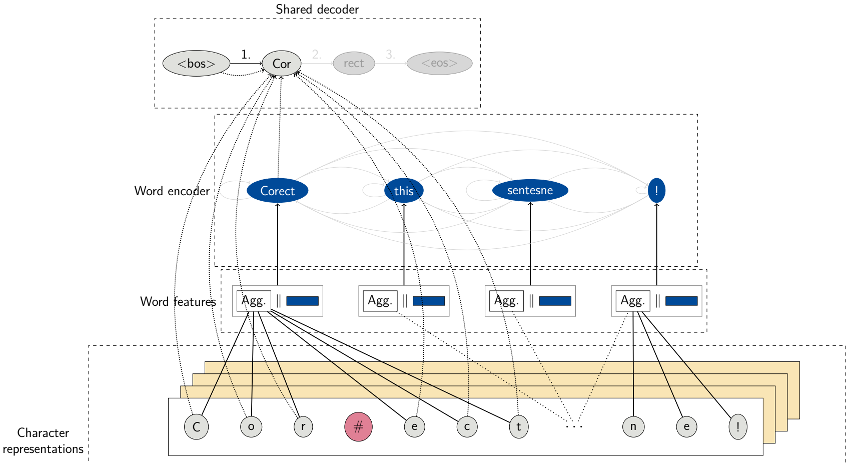
1. Tokenization repair as separate preprocessing step
2. Tokenization repair as feature extraction backbone
3. Sequence-to-sequence transformer to correct whitespace and spelling errors

Appendix: Tokenization repair^{+}



1. Get character representation r_c for all characters C as a weighted average over outputs h of the top n tokenization repair layers:
$$r_c = \sum_{i=1}^{n} w_i h_i^c \text{ for } c \in C \text{ with } \sum_{i=1}^{n} w_i = 1$$
2. Group characters representations into regex-word representations according to tokenization repair ouptut leaving out whitespaces

Appendix: Tokenization repair^{++}



1. Get character representation r_c for all characters C as a weighted average over outputs h of the top n tokenization repair layers:
   $$r_c = \sum_{i=1}^{n} w_i h_i^c \text{ for } c \in C \text{ with } \sum_{i=1}^{n} w_i = 1$$
2. Group characters representations into regex-word representations according to tokenization repair ouptut leaving out whitespaces

Appendix: Whitespace benchmarks

	whitespace high-high F_1	Precision	Recall	whitespace high-low F_1	Precision	Recall	whitespace low-high F_1	Precision	Recall	whitespace low-low F_1	Precision	Recall
transformer with tokenization repair	88.95	87.17	90.81	85.38	84.25	86.55	94.30	93.09	95.55	91.53	89.35	93.82
transformer with tokenization repair$_{\text{beam}}$	**89.28**	87.54	91.10	**85.93**	84.85	87.04	**94.44**	93.27	95.63	91.77	89.64	94.00
tokenization repair^{++}	85.90	84.14	87.73	83.64	83.34	83.95	94.08	93.10	95.09	92.81	91.92	93.71
tokenization repair$^{++}_{\text{beam}}$	86.02	84.29	87.83	83.85	83.56	84.14	94.08	93.09	95.09	**92.88**	91.99	93.80
eo medium $\rightarrow$ transformer^{+} $\rightarrow$ transformer	85.71	83.84	87.66	83.98	83.88	84.09	94.03	93.05	95.02	92.67	91.72	93.63
eo medium $\rightarrow$ transformer^{+} $\rightarrow$ transformer word	85.70	83.89	87.58	83.84	83.62	84.06	93.96	92.95	95.00	92.63	91.69	93.59
tokenization repair^{+} $\rightarrow$ transformer	85.89	84.09	87.77	84.22	84.21	84.23	94.06	93.10	95.05	92.74	91.94	93.55
tokenization repair^{+} $\rightarrow$ transformer word	85.85	84.09	87.69	84.05	83.90	84.19	94.02	93.04	95.03	92.73	91.95	93.53

Whitespace benchmarks: F_1

	whitespace high-high Improvement	MNED	whitespace high-low Improvement	MNED	whitespace low-high Improvement	MNED	whitespace low-low Improvement	MNED
do nothing	-	0.1885	-	0.0900	-	0.1496	-	0.0464
transformer with tokenization repair	-84.0%	0.0301	-69.2%	0.0277	-91.8%	0.0123	-77.8%	0.0103
transformer with tokenization repair$_{\text{beam}}$	**-84.7%**	0.0288	**-70.9%**	0.0262	**-92.2%**	0.0116	**-79.0%**	0.0098
tokenization repair^{++}	-78.9%	0.0399	-66.9%	0.0297	-91.1%	0.0133	-78.2%	0.0101
tokenization repair$^{++}_{\text{beam}}$	-79.1%	0.0395	-67.4%	0.0293	-91.1%	0.0133	-78.5%	0.0100
eo medium $\rightarrow$ transformer^{+} $\rightarrow$ transformer	-77.9%	0.0416	-65.7%	0.0309	-90.8%	0.0137	-76.7%	0.0108
eo medium $\rightarrow$ transformer^{+} $\rightarrow$ transformer word	-78.1%	0.0412	-65.7%	0.0308	-91.3%	0.0130	-77.9%	0.0103
tokenization repair^{+} $\rightarrow$ transformer	-77.5%	0.0424	-64.4%	0.0320	-89.8%	0.0153	-73.7%	0.0122
tokenization repair^{+} $\rightarrow$ transformer word	-77.6%	0.0422	-64.3%	0.0321	-90.4%	0.0143	-75.2%	0.0115

Whitespace benchmarks: Mean normalized edit distance

Appendix: Runtimes

Task	Model	Runtime in s	kB/s
TR*	eo large	3.5	50.0
TR*	eo medium	4.2	41.7
TR	tokenization repair^{+}/tokenization repair^{++}	4.2	41.7
TR*	eo small	5.8	30.2
SEDS/SEDW†	tokenization repair^{+}/tokenization repair^{++}	8.7	20.1
SEDS/SEDW†	gnn^{+}	6.8	25.8
SEDS/SEDW†	gnn	6.6	26.4
SEDS/SEDW†	transformer^{+}	3.7	47.6
SEDS/SEDW†	transformer	3.5	50.6
SEC	transformer	65.1	2.7
SEDW → SEC	gnn^{+} → transformer	47.9	3.7
SEDW → SEC	transformer^{+} → transformer	45.9	3.8
SEC	transformer word	28.0	6.3
SEC	neuspell bert	18.1	9.6
SEDW → SEC	gnn^{+} → transformer word	16.4	10.7
SEDW → SEC	transformer^{+} → transformer word	13.3	13.3
TR & SEC	transformer with tokenization repair	71.1	2.5
TR → SEDW → SEC	eo medium → gnn^{+} → transformer	52.3	3.3
TR → SEDW → SEC	eo medium → transformer^{+} → transformer	52.3	3.3
TR → SEC	tokenization repair$^{++}_{\text{w/o detection}}$	40.6	4.3
TR → SEDW → SEC	eo medium → gnn^{+} → transformer word	21.4	8.2
TR → SEDW → SEC	eo medium → transformer^{+} → transformer word	21.4	8.2
TR → SEDW → SEC	tokenization repair^{++}	19.0	9.2

* Ported models from https://github.com/ad-freiburg/trt, shown here for reference
† The overhead of converting word level detections into sequence level detections is negligible

Model runtimes