INTERSPEECH 2019
September 15–19, 2019, Graz, Austria

Dual Encoder Classifier Models as Constraints in Neural Text Normalization

Ajda Gokcen[1], *Hao Zhang*[2], *Richard Sproat*[2]

[1]University of Washington, Seattle, WA, USA
[2] Google, New York, NY, USA,

ajdag@uw.edu, {haozhang,rws}@google.com

Abstract

Neural text normalization systems can achieve low error rates; however, the errors they make include not only ones from which the hearer can recover (such as reading *$3* as *three dollar*) but also *unrecoverable* errors, such as reading *$3* as *three euros*. FST decoding constraints have proven effective at reducing unrecoverable errors. In this paper we explore an alternative approach to error mitigation: using *dual encoder* classifiers trained with both positive and negative examples to implement *soft constraints* on acceptability. Since the error rates are very low, it is difficult to determine when improvement is significant, but qualitative analysis suggests that soft dual encoder constraints can help reduce the number of unrecoverable errors.

1. Introduction

Text normalization for speech applications such as text-to-speech synthesis is the problem of converting from text that includes numerical expressions, abbreviations and other *semiotic classes* [1] into a form that specifies how those expressions are to be read. For example, *Train 540 leaves at 4:45* might be read as *train five forty leaves at four forty five*, in American English. One traditional approach to this problem involves hand-built finite-state grammars [2], but more recently there has been interest in neural approaches to the problem [3, 4, 5, 6]. While neural methods perform well overall, they have a tendency on occasion to produce highly misleading output, such as reading *2mA* as *two million liters* [3]. One solution to this problem is to use *covering grammars*, (usually) finite-state models that can constrain the neural models to a reasonable (context-independent) space of options so that *2mA* could be read as *two milliamperes* or *two m a*, but not *two million liters* [3]. These covering grammars can be learned in whole or in part from data [7].

While FSTs provide an obvious way in which to apply constraints, any computational model that can be interpreted as a state machine could be so used. Neural models that assign similarity scores to string pairs are particularly interesting because their scores can be combined with RNN decoder scores rather than used as hard constraints. In Section 2, we formulate the problem of decoding with generalized state-machine-constrained RNNs. From Section 3 to Secton 5, we introduce *dual encoder* classifiers and present some preliminary results on applying them as soft constraints to the text normalization problem.

2. Formulation of State Machine Constraints

The baseline neural textnorm model assumed here is detailed in [3, 6], and is essentially an attention-based RNN [8]. While *tokenization* into *semiotic classes* is part of the full neural model described in [6], we assume for the discussion in this paper that the input has already been tokenized into semiotic class tokens. The task at hand therefore involves mapping the input character sequences of semiotic class tokens to sequences of verbalized output words.

The decoder RNN models a sequence of output words $(y_1, \ldots, y_{T_y})$ as a product of ordered conditionals. At time step t, the conditional is

$$p(y_t|y_1, \ldots, y_{t-1}, c) = \text{softmax}_{y=y_t} g(y_{t-1}, s_t, c)$$

where s_t is the hidden state of the RNN at the time step t; c is a vector representation of the input context, which is embodied as an attention function of the encoder states of the input sequence and the previous RNN state s_{t-1}; and g is a non-linear multi-layer neural network function whose output is the logits distributed over the output vocabulary and is fed to the softmax function.

This interpretation based on the hidden states $s_0, \ldots, s_t$ makes it easy to introduce a constraint state machine that runs in synchronization with the decoder RNN:

$$g'(y_{t-1}, (s_t, t_t), c) = g(y_{t-1}, s_t, c) + h(t_t) \quad (1)$$

Here, $t_0, \ldots, t_{T_y}$ is the state sequence of the constraint state machine, and h is the constraint weight function whose output is a weight vector of the size of the output vocabulary. Similar to s_t, t_t is based on a recurrence:

$$t_t = f(t_{t-1}, y_{t-1})$$

When (f, h) represents a weighted FST, g' represents an RNN constrained by a WFST. In the special case of unweighted FST [3], h is a function that masks out certain logits in the output distribution by outputting $-\infty$.

In the following section, we introduce two novel state machine models based on neural *dual encoder* classifiers.

3. Dual Encoder Classifier Constraints

Dual encoders (DE) [9, 10] are Siamese models that consist of a pair of encoders that encode pairs of inputs into vectors; and a model to compute the similarity between the two inputs (e.g., a feed-forward multilayer perceptron). DEs have been used in a variety of applications including whether a given answer matches a question [9, 10], whether two queries are the same, or to encode entities for later retrieval. Note that the features used to encode the two inputs need not be the same. In our experiments we typically use character-level ngram features for the input string and word-level ngram features for the output string.

There are two distinct ways to interpret a DE model for our task. The first is as a **string pair classifier** that decides whether a given output (*three kilograms*) is reasonable given a particular input (*3kg*), and then use this to favor or disfavor an output predicted by the neural text normalization model. Since this question only makes sense to ask of a complete prediction, we invoke

http://dx.doi.org/10.21437/Interspeech.2019-1135

Table 1: *Results of preliminary testing on a subset of data including measurement and money class inputs. The baseline represents a contextual model of normalization without additional constraints, while the experiment represents a model with a vocabulary filter DE constraint. Although the two systems perform identically on the measure class inputs, improvements of the experiment model over the baseline are significant for the money class inputs and overall.*

	Frequency	Same	Baseline Better	Experiment Better	Significant	p-value
Overall	212840	99.99%	0.00%	0.01%	yes	0.0006
Measure	151137	100.00%	0.00%	0.00%	no	0.44244
Money	61703	99.98%	0.00%	0.02%	yes	0.00063

the constraint only at a decoding step where the RNN normalization model predicts a STOP symbol. Since the DE score s for a pair is in the range [0.0, 1.0], we would like to favor the verbalization if s is greater than 0.5, and disfavor it otherwise. How much to favor or disfavor can be controlled with a scaling factor λ. Thus we modify the logit l associated with the STOP label so that $\hat{l} = l + \lambda(s - 0.5)$.

Per equation 1, the constraint weight function is

$$h_p(t_t)_{y=y_t} = \begin{cases} \lambda(s-0.5), & \text{if } y_t = \text{STOP} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where $s = \text{DE}(x_0, \ldots, x_{T_x}; y_0, \ldots, y_{t-1})$.

The second way one can interpret a DE for this task is as a **vocabulary filter**: rather than have the model predict the full verbalization for a token, have it instead simply predict which words may occur anywhere in the verbalization. A perfect model would predict that *123*'s verbalization could contain *one*, *two*, *three*, *hundred* and *twenty* but not, say, *four*, *nine* or *zero*. Such a model would not help with all kinds of errors (e.g., cases where a word is duplicated — *one twenty twenty three*) in the full neural system, but it would help to restrict the vocabulary so that unrecoverable readings like *million liters* for *mA* could be avoided. In this case we want to use the constraint to rescore all logits at every decoding step. For words that the model classifies as being good, we want to leave the corresponding logit alone, whereas for bad words, we want to penalize the logit score again possibly with a scaling factor. We subtract 1.0 from the score, since if the word is perfect according to the model we want to retain the logit's value as it is, and thus $\hat{l} = l + \lambda(s - 1.0)$. Expressed following equation 1, the constraint weight function is

$$h_v(t_t)_{y=y_t} = \begin{cases} 0, & \text{if } y_t = \text{STOP} \\ \lambda(s-1.0), & \text{otherwise} \end{cases} \quad (3)$$

where $s = \text{DE}(x_0, \ldots, x_{T_x}; y_t)$.

4. Negative Example Generation

Training such a model requires both positive and negative data. Positive data, of course, can consist of actual examples from training corpora, possibly augmented with examples from other sources.

Note that the positive examples are taken as *context independent*: for a full text normalization system one of course would like to distinguish between *$100* read as *one hundred dollars*, versus *$100 bill*, read as *one hundred dollar bill*. But as with the finite-state covering grammar, here we are interested only in constraining the verbalization to reasonable possibilities independent of context. As with the covering grammar approach, we rely on the neural model to pick the appropriate rendition among the offered set of reasonable cases.

Negative data, on the other hand, must be generated. Two issues arise here, namely how to generate it and how much to generate relative to the amount of positive data. To the former point, we explore using both **mistakes** made by the neural system (which are good examples of the kinds of things we would like to constrain the neural system not to do) and **synthetic** negative data (where we attempt to mimic the kinds of errors the text normalization system makes). The former method is apt to focus on true system errors instead of an engineer's idea of system errors, but since the existing systems have low error rates, the latter can be of use in getting around the issue of data sparsity.

One class of errors we target with synthetic data is based on common input suffixes. For example, *pho* (an abbreviation for *phosphorus*) is expanded to *number* because of the spurious suffix *o* shared with *no*, a legitimate abbrevation of *number*. To capture such errors, we generate example pairs by creating tables of input suffixes and corresponding outputs based on the full system's training data. We also generate synthetic examples via straightforward transformations on the output strings, including deleting, inserting, and substituting words. Obviously, this latter method only applies to the string pair classifier and not to the vocabulary filter model.

We employ an additional method of generation halfway between the synthetic and observed approaches: random association of outputs to unrelated inputs (e.g. *twenty pounds* associated with *3/4/1929*) to yield noise as negative examples. This does not inherently require hand-engineering, but as noisy sampling seems not to capture difficult cases when performed on the full training set, we also create limited subsets (e.g., only measurements or dates) and sample within those subsets for more targeted examples. For generating examples using observed (namely, incorrect) system output, we use not only the final system output, but also the top-k (e.g., k = 10) output candidates in a similar way. The latter method, like the synthetic methods, combats the sparsity of the low-error system output alone.

We generated training and test sets (the latter for evaluating the DE models on their own) using each of these methods, with additional variants based on negative-to-positive example ratios. Intuitively, a balanced data set is most effective, but in practice we found higher rates of negative examples in training to yield better performance.

5. Experiments

Preliminary experiments showed small but significant gains using a vocabulary filter DE constraint trained and tested on a subset of data including only measurement and money expressions (e.g., *3kg* or *$20*). The improvements, shown in Table 1, were

Table 2: *Error rates, counts, and qualitative breakdowns for a selection of contextual text normalization models with (and without) various DE constraints. Non-errors, such as reading* two thousand and ten *rather than* two thousand ten *are discounted. All systems have the same conditions (constraint or lack thereof) at both training and test time, as omitting constraints at training time has insignificant effect, with use of constraints at both stages faring slightly better.*

-DE: *A system with no DE constraint.* **+seqDE:** *The system with full-sequence classifier DE constraints, trained on a corpus of both synthetic examples and prior system output.* **+vocDE (rand):** *A system with a vocabulary filter constraint, as trained with noisy sampling to obtain "negative" examples.* **+vocDE (top-k):** *A system with a vocabulary filter constraint, as trained on the top 10 hypotheses of a system with no constraints.*

The results here are most comparable to the results reported in Table 7 of [6] (entries for the Kaggle data). But note that those results were for a model that included a neural tokenizer as well, so the overall errors there are a bit higher.

	Error Rate	Error Count	Recoverable Errors	Unrecoverable Errors
-DE	0.67%	2816	2185 (77.59%)	631 (22.41%)
+seqDE	0.66%	2752	2178 (79.14%)	574 (20.86%)
+vocDE (rand)	0.74%	3115	2567 (82.41%)	548 (17.59%)
+vocDE (top-k)	0.74%	3090	2487 (80.49%)	603 (19.51%)

enough to suggest that such methods may prove useful when extended to all input classes.

For our experimental data we used the training and test data for the 2017 Kaggle competition on text normalization [11], which consisted of ten million tokens of English Wikipedia text verbalized using the Kestrel text normalization system [2] for training data, and one million for testing. As described below, for some experiments we trained and tested on just measure and money expressions.

We set up a suite of experiments to test the efficacy of DE constraints with a number of experimental variables. These variables and their possible values include:

Output class: Full string/vocabulary filter

Data source: Synthetic/noisy sample/top-*k* output

Negative-positive ratio: 3-to-1/10-to-1/all-to-all

Constraint coverage: Universal/class-specific

Training conditions: Using constraints during training+test time, or during testing only

Several of the variables were already discussed in Sections 3 and 4. The *all-to-all* ratio is not really a ratio; it means not only that all positive examples are kept in the corpus, but also that all possible negative examples are exhaustively generated and included, regardless of the relative numbers of each. *Constraint coverage* refers to whether we use a single constraint for all inputs, or whether we limit constraints to specific classes of input (e.g., measure expressions) that can be easily detected, and perhaps utilize multiple disjoint constraints at a time.

The experimental variables affect the performance of the DE classifiers in isolation, and while it cannot be guaranteed that a better performing classifier leads to a better system using that classifier as a constraint, we use this as a guideline for the data source and negative-positive ratio variables.

In practice, DEs trained on data synthetically generated predictably perform worse on classifying actual system outputs (80-85% accuracy at the string level), while those trained on actual system output perform better on held-out data of the same class (85-90% accuracy). The synthetic data alleviates the sparsity of incorrect output, however, as systems trained on composite data perform best (90-95% accuracy). For vocabulary filter classifiers, higher negative-positive ratios, including 10-to-1 and all-to-all, yield systems that rarely overgenerate, – i.e., give "good" scores to "bad" output words – as even systems trained with a 3-to-1 ratio tend to do.

Turning to use in the full system, some of the variables introduced above had no discernable effect: training with the constraint made no difference, nor did the constraint coverage variable. As for the other conditions, while they led to no significant changes in *overall error rates*, manual analysis of the errors to focus on *qualitative* differences, as shown in Table 2, shows beneficial shifts in error composition with DE models. **Recoverable errors** are those that constitute possible verbalizations given a different context: thus *3 kg* could be *three kilograms* or sometimes *three kilogram* or perhaps even *three k g*. **Unrecoverable errors** are those errors that are never licensed for a given input string, even in a different context (*3kg* as *four pounds*).

The lowest error rate is found for the **+seqDE** condition, followed by the unconstrained system, but in general the systems with either full sequence or vocabulary filter constraints have lower rates of unrecoverable errors than the unconstrained system. Those with vocabulary filter constraints make more errors overall, but this largely consists of recoverable errors, and their rate of unrecoverable errors is much lower. Note also the reduction of unrecoverable errors by the **+vocDE (rand)** system, where noisy sampling (Section 4) is used to generate negative errors.

Besides the above-described experimental variables, the scaling factor λ (set to 1.0 in the above) plays an important role in balancing the output distribution of the RNN with that of the dual encoder. In Figure 1, we plot the result of varying $0 \leq \lambda \leq 25$ for a model *trained and tested on only measure and money expressions*. For both semiotic classes, the lowest error rate is obtained when λ is equal to 5.0. We have not fully explored the space of varying this parameter in combination with other variables.

6. Related Work

The tendency of neural machine translation models to favor fluency over adequacy for out-of-domain input has been highlighted as one of the biggest challenges of such models [12]. To attack the problem, translation lexicons [13], finite state machine constraints [14] (effectively covering grammars; see, again, [3, 6]), and attention coverage [15] have been introduced to guide the translation models. To incoporate bias from prior

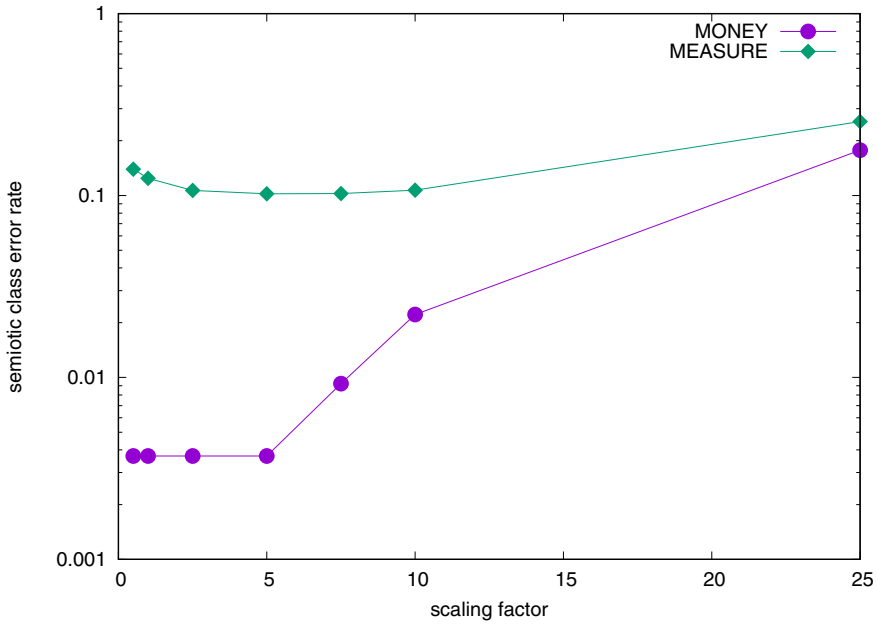

Figure 1: *The effect of the scaling factor λ on the semiotic class error rate. The best results are obtained with $\lambda = 5.0$.*

knowledge, [16] utilize various constraints through posterior regularization of the neural model, inspired by the generic technique of knowledge distillation [17]. In order to combat the tendency to choose high-frequency output words for rare input words, [18] adjust the output layer of the RNN decoder through integration of a feed-forward network of lexical selection. We are the first to integrate a dual encoder with the output layer to influence the output choices.

7. Discussion and future work

A shift in error composition to include more recoverable errors and fewer unrecoverable errors is a qualitative improvement for the systems with DE constraints. The improvement of the vocabulary filter system trained on noisy sampling is particularly interesting, since it counters the intuition that improvements come from targeting common error cases.

That said, there remain obstacles for these systems. A system that utilizes both a full sequence constraint as well as a vocabulary filter constraint may further reduce unrecoverable errors, but both constraints slow down decoding significantly. The vocabulary filter constraint trained on top-k output shows promise when seen as the first iteration of an adversarial set-up (wherein DEs are trained on the output of the prior iterations' normalization systems, and normalization systems are trained iteratively with the DE constraints), but the slow decoding process has thus far been prohibitive.

Finally we should remind the reader that as with any trainable system, a DE constraint model is genre-dependent in that the model will be biased towards the kinds of semiotic tokens it has seen — a limitation that it shares with neural text normalization systems in general. In practice this should not be a serious issue, though, since it is easy to retrain the model with additional training data.

8. Acknowledgements

We thank Alessandro Presta for much help with the DE tools.

9. References

[1] P. Taylor, *Text-to-Speech Synthesis*. Cambridge: Cambridge University Press, 2009.

[2] P. Ebden and R. Sproat, "The Kestrel TTS text normalization system," *Natural Language Engineering*, vol. 21, no. 3, pp. 1–21, 2014.

[3] R. Sproat and N. Jaitly, "An RNN model of text normalization," in *INTERSPEECH*, 2017, pp. 754–758.

[4] S. Pramanik and A. Hussain, "Text normalization using memory augmented neural networks," 2018, arXiv:1806.00044.

[5] S. Yolchuyeva, G. Németh, and B. Gyires-Tóth, "Text normalization with convolutional neural networks," *International Journal of Speech Technology*, vol. 21, pp. 1–12, 2018.

[6] H. Zhang, R. Sproat, A. Ng, F. Stahlberg, X. Peng, K. Gorman, and B. Roark, "Neural models of text normalization," *Computational Linguistics*, vol. 45, p. 293–337, 2019.

[7] K. Gorman and R. Sproat, "Minimally supervised models for number normalization," *Transactions of the Association for Computational Linguistics*, 2016.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[9] Y. Lu, P. Keung, S. Zhang, J. Sun, and V. Bhardwaj, "A practical approach to dialogue response generation in closed domains," 2017, https://arxiv.org/pdf/1703.09439.pdf.

[10] D. Gillick, A. Presta, and G. Tomar, "End-to-end retrieval in continuous space," 2018, https://arxiv.org/abs/1811.08008.

[11] R. Sproat and K. Gorman, "A brief summary of the Kaggle text normalization challenge," 2018, http://blog.kaggle.com/2018/02/07/a-brief-summary-of-the-kaggle-text-normalization-challenge.

[12] P. Koehn and R. Knowles, "Six challenges for neural machine translation," in *Proceedings of the First Workshop on Neural Machine Translation*, 2017, pp. 28–39.

[13] P. Arthur, G. Neubig, and S. Nakamura, "Incorporating discrete translation lexicons into neural machine translation," in *EMNLP*, 2016.

[14] E. Hasler, A. de Gispert, G. Iglesias, and B. Byrne, "Neural machine translation decoding with terminology constraints," in *NAACL*, 2018, pp. 506–512.

[15] H. Mi, Z. Wang, and A. Ittycheriah, "Supervised attentions for neural machine translation," in *EMNLP*, 2016, pp. 2283–2288.

[16] J. Zhang, Y. Liu, H. Luan, J. Xu, and M. Sun, "Prior knowledge integration for neural machine translation using posterior regularization," in *ACL*, 2017, pp. 1514–1523.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[18] T. Q. Nguyen and D. Chiang, "Improving lexical choice in neural machine translation," in *NAACL-HLT*, 2018, pp. 334–343.