Recurrent Neural Networks for Incremental Disfluency Detection

Julian Hough and David Schlangen

Dialogue Systems Group
Faculty of Linguistics and Literature
Bielefeld University
julian.hough@uni-bielefeld.de, david.schlangen@uni-bielefeld.de

Abstract

For dialogue systems to become robust, they must be able to detect disfluencies accurately and with minimal latency. To meet this challenge, here we frame incremental disfluency detection as a word-by-word tagging task and, following their recent success in Spoken Language Understanding tasks, we test the performance of Recurrent Neural Networks (RNNs). We experiment with different inputs for RNNs to explore the effect of context on their ability to detect edit terms and repair disfluencies effectively. Although not eclipsing the state of the art in terms of utterance-final performance, RNNs achieve good detection results, requiring no feature engineering and using simple input vectors representing the incoming utterance as their training input. Furthermore, RNNs show very good incremental properties with low latency and very good output stability, surpassing previously reported results in these measures.
Index Terms: disfluency detection, incremental processing, dialogue systems, recurrent neural network

1. Introduction

Disfluencies, such as edit terms and self-repairs, are pervasive in human dialogue. A dialogue system which detects them appropriately is in a strong position to interpret their communicative meaning, such as an indication the speaker is having difficulty continuing their contribution, or finds their original utterance sub-optimal for current purposes and needs to repair it. Additionally, in a practical system, the performance of downstream consuming components to a disfluency detector such as Natural Language Understanding (NLU) can be improved with knowledge of disfluent regions of incoming utterances.

While there has been substantial work on disfluency detection, with state-of-the-art approaches achieving high accuracy [1, 2], much of it has taken transcripts to be texts rather than incoming speech, with the aim of 'cleaning' these texts of disfluent material for subsequent post-processing. This approach is suboptimal if we aim for disfluency detection to compute meaning from repairs and edit terms, which is not only psychologically valid [3] but also practical for a dialogue system aiming to compute deep understanding of user utterances.

The aspect of disfluency detection we focus on in this paper, in line with recent development in dialogue systems [4], is the desideratum to be strongly *incremental*: it must function with minimal latency as it consumes word-by-word input and do so without changing its original hypotheses very often, or at all, avoiding 'jittering' output that is unhelpful to consuming components and outputting its *best predictions as early as possible*. Approaches focussing on incremental performance have been rare: only [5] and [6] report word-by-word incremental performance measures in addition to utterance-final results, so we use these as points of departure.

Here we investigate how the recent advances shown by the use of Recurrent Neural Networks (RNNs) in statistical language modelling [7, 8], and in spoken language understanding (SLU) [9], can be applied to incremental disfluency detection. We train and test a simple Elman RNN [10] for the task, experimenting with different input which varies in terms of the context of the utterance available at a given word position, and different input representations (word vs. word + POS tags) to see how this affects performance on disfluency detection. Now we describe the task in detail with our novel approach.

2. Incremental Disfluency Detection

We focus on detecting speech repairs with the tripartite structure of *reparandum-interregnum-repair* originally proposed by [11], as in (1) and as in the bottom braces in Fig. 1.

$$\text{John} \underbrace{[\ \text{likes}}_{\text{reparandum}} + \underbrace{\{\ \text{uh}\ \}}_{\text{interregnum}} \underbrace{\text{loves}\ ]}_{\text{repair}} \text{Mary} \quad (1)$$

While the majority of approaches since the beginning of the disfluency detection challenge in [12] focus on reparandum word detection, we note there are cases where the entire structure of the repair is needed to calculate meaning – see (2) where access to "the oranges" resolves the anaphoric "them".

(2) "have the engine [take the oranges to Elmira, + { um, I mean, } take them to Corning] " [14]

Furthermore, we wish to detect all *edit terms*, not just those within interregna, which, along with more markedly lexicalised fillers such as 'uh' and 'um', comprise phrasal edit terms such as 'you know' and 'I mean'. We consider their contribution part of an utterance's meaning in conversation [15] and therefore part of the meaning of a user's utterance in a dialogue system setting. While edit terms can form interregna, isolated, non-repair edit terms tend to indicate forward looking trouble from conversation participants [16].

Given this motivation, in line with [6], we not only evaluate the task of reparandum word identification, but consider the whole repair structure, focussing on improving the F-score (F1) F_s in (3) for reparandum rm, interregnum i and repair phase words rp. For comparison to previous systems we also report the reparandum word detection F1 which we call F_{rm}. For evaluating edit term detection (which subsumes interregna detection) we use a per-word F1 measure we call F_e. Furthermore, in addition to utterance-final performance, we train and evaluate our systems for their *incremental performance*, with appropriate metrics explained in §4.

$$\text{A flight} \underbrace{[\ \text{to Boston}}_{\text{reparandum}} \underbrace{+\{\ \text{uh I mean}\ \}}_{\substack{e\ \ e\ \ \ e \\ \text{interregnum}}} \underbrace{\text{to}\ \ \text{Denver}\ ]}_{\substack{rm{-}5\ \ \ rpSub \\ \text{repair}}} \text{on Friday}$$

Figure 1: An utterance with our incrementally-oriented tag scheme and the traditional repair disfluency segmentation [11, 17]

$$\text{pr} = \frac{\{rm, i, rp\}^{correct}}{\{rm, i, rp\}^{hyp}} \quad \text{r} = \frac{\{rm, i, rp\}^{correct}}{\{rm, i, rp\}^{gold}}$$
$$\text{F}_s = 2 \times \frac{\text{pr} \times \text{r}}{\text{pr} + \text{r}} \tag{3}$$

We treat disfluency detection as a word-by-word tagging task, where our system outputs the predicted tag for the current word as it consumes utterances strictly word-by-word. Having said this, contrary to the left-to-right order of the structure in (1), we only detect reparandum words upon consuming the *repair onset*, the word after the possibly null interregnum, e.g. 'loves' in (1) or the second 'to' in Fig. 1, rather than predict reparanda before this point. Here our incremental tagging representation tags the repair onset word with the absolute distance back to the start of the reparandum in number of words (see Fig. 1). From this scheme the original tripartite bracketing structure is recoverable incrementally, albeit with the delay in detecting the reparandum (see the bottom of Fig. 1), assuming the following two-part repair detection process:

1. The repair onset is detected, and with it, the distance back to the reparandum onset, solving the 'continuation problem' [18] as to how to integrate previous material in the utterance with the current word. Generally, if the current word continues the previous word fluently it is tagged f,[1] however if it continues a word ordered sequentially n words back, repairing the word sequence from there up to the previous word, then this is tagged $rm{-}[n]$.
2. In the current and subsequent words the decision is made whether a given word is in the middle of the repair phase, tagging it rp, or the final word in the repair, in which case classify it as having substituted (which may mean repeated) the reparandum phase, tagging it $rpSub$, or over-written (deleted) it, tagging it $rpDel$. Once phase 2 is over and a repair end tag has been outputted, the repair is over.[2]

For repair onset tags we use $rm{-}\{1 \ldots 8\}$. Reparandum lengths longer than 8 words are very sparse and so in training we tag these as $rm{-}8$. This assumption does not hinder potential detection performance,[3] whilst greatly cutting down on sparsity of tags. The repair onset tags combine with the repair end tags $rpDel$ and $rpSub$, which, along with a fluent token f and the edit word token e, gives 27 training tags.[4]

3. RNNs for Disfluency Detection

Recurrent neural networks (RNNs, see, e.g., [10]) have been successful in a variety of NLP tasks, yet to our knowledge this is the first time they have been applied to incremental disfluency detection. As repair disfluencies are in general very short, with a power law decay in the distribution of reparandum lengths [19], and frequently with crossed serial dependencies [20], RNNs are well placed for the task: the short-term memory need only last a few words, but effective use of this memory for computing parallelism of substitution repairs is required. The aim of this work is to investigate how far one can succeed without careful feature selection from language models [6] or parsing-based approaches which explicitly string align repairs [20, 5], but learn purely from input contexts and design neural net training regimes to perform best at disfluency detection.

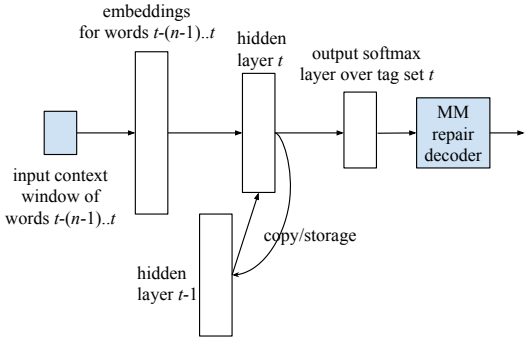

Figure 2: Elman RNN architecture and decoding pipeline

In addition to repair detection, edit term detection should benefit from recent deep learning techniques for word meaning representations. Edit terms such as *uh*, *um* and disfluent *you know*'s are known to have a characteristic vocabulary, so the use (and learning) of embedded vector word representations (as in [21]) in an RNN should result in learning network-internal distinctions between edit terms and fluent words.

Anticipating the Vanishing Gradient problem (see [8]) whereby sequences longer than a few steps long struggle to make use of information back in time from the current input, we investigate how much short-term memory is required for detecting disfluencies, and what adjustments need to be made to combat the problem. We do that here by investigating incremental context and the extent to which previous words need to be explicitly presented to the network to allow for effective learning and prediction.

3.1. Architecture and parameters

Input and word embeddings Following [9], we use 1-of-N, or 'one-hot', vectors as our raw input to the network, however these one-hot representations are not used by the network in learning and decoding directly but provide unique indices to dense vectors in a word embedding matrix. We experimented with both randomly initialised and pre-learned embeddings which are further updated during training to become more attuned to our detection task. We found using vectors from external sources did not significantly help disfluency detection, however using a word embedding matrix pre-trained from our disfluency detection training data cleaned of disfluencies, fol-

[1] The fluent f tag will largely be suppressed from here, with any words apparently untagged having this tag in practice.

[2] Interspersed with these two tasks is edit term detection, tagging words e, which also serves as interregnum detection when this tag is outputted before a repair onset.

[3] This allows a near perfect recall of all reparandum tokens with a ceiling on F_{rm} at 0.996 on our validation set.

[4] We found rp indicating a mid repair phase word as its own separate tag was unnecessary in training, but can be re-introduced in decoding as will be explained.

lowing the notion of a 'clean' language model by [20], was marginally more effective than random initialisation. We found an embedding dimension of 50 worked best in initial tests.

Backwards-looking context windows Again following [9], we use context windows rather than just inputting one word representation into the network at a time. However, as we are interested in strong left-to-right incrementality we do not allow look-ahead to words beyond the current prefix being consumed, so our contexts are, like n-gram language models, linearly backwards from the current word rather than surrounding it. We experiment with different lengths and report the different outcomes arising from these choices in our results in §6. As in [9], the internal representation of context windows of length n in the network is created through the ordered concatenation of the n corresponding word embedding vectors of size 50, resulting in an input vector to the network of dimension $\mathbb{R}^{50n}$.

Architecture and activation functions In addition to our embedding layer, we use a (recurrent) hidden layer of 50 nodes and an output layer the size of our training tag set (27 nodes). We found in early testing hidden layer sizes of over 50 tended to result in over-fitting on the training data. We use a standard Elman RNN, where the output from the hidden layer at time $t-1$ is stored and then fed as an input vector, along with the word embedding vectors, to the hidden layer at time t. See Fig. 2 for the schematic overall disfluency detection architecture.

Mathematically, the standard Elman-type RNN dynamics in the recurrent hidden layer at time t is as in (4), where the hidden layer $h(t)$ is calculated as the Sigmoid function (5) of the addition of the weight matrix U' applied via dot product to the current input vector $x(t)$ and the weight matrix V' applied via dot product to the stored previous value of the hidden layer at time $t-1$, i.e. $h(t-1)$.

$$h(t) = f(U'x(t) + V'h(t-1)) \tag{4}$$

$$f(x) = \frac{1}{1+e^{-x}} \tag{5}$$

We use the standard softmax function for the node activation function of the output layer at time t: for the output layer $o(t)$ is the softmax of the dot product of the weight matrix W' and the current hidden layer $h(t)$ as in (6), with the standard softmax definition for a given real value x_i in a set of real values $x_1..x_n$ given in (7).

$$o(t) = G(W'h(t)) \tag{6}$$

$$G(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{7}$$

Learning: error function and parameter update As is standard in tagging tasks for RNNs [8] we use negative log likelihood loss (NLL) as a cost function and use stochastic gradient descent over the parameters to be learned (weight matrices U', V' and W' from equations (4) and (6), plus the embedding vectors) to minimize it. The error we are interested in is F-loss (i.e. 1 - F_s) for each batch, in the spirit of [22], however as this is non-differentiable, we use the standard NLL approximation.

Batch sizes and back-propagation As our focus is on incrementality, we update our parameters after every word is consumed in training. As an utterance is consumed word by word, the training batch length k goes up from a single window context at the beginning of the utterance (with the window padded by beginning of sentence vectors) to our maximum back-propagation through time distance bp. We found a bp of 9 was useful to avoid the vanishing gradient problem, but also fits with our tag set for repair identification: as explained in §2, limiting this to 8 words back does not hinder performance.

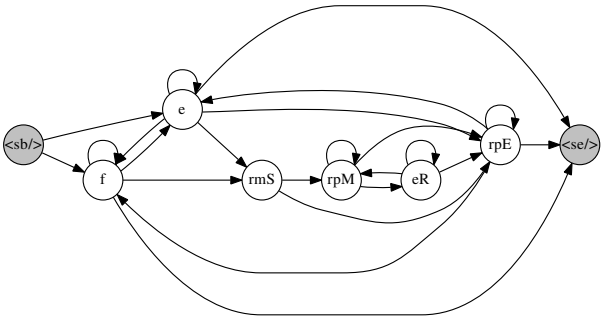

Figure 3: Markov model for repairs, edit terms and fluent words

We optimise the other hyper-parameters of our RNNs on the validation set. We found using a learning rate of 0.005 worked best and we also use L2 regularisation on the parameters to be learned with a weight of 0.0001 to avoid over-fitting.

3.2. Decoding optimization: Repair Markov model

Disfluency detection from the RNN operates locally, just tagging the current word. This means the network can output illegal tag sequences not conforming to the repair structure in Fig. 1. To combat this we add a Viterbi decoder to output the tag sequence that maximises the joint score of the tag's probability in the softmax distribution from the RNN and the probability of the tag given the previous tags as stipulated by the Markov model in Fig. 3. This ensures that for all repair onsets (allowing entry to state rmS in Fig. 3) there must be a repair end tag $rpSub$ or $rpDel$ to enter the state rpE, allowing a legal end of utterance tag, while a mid repair (rpM) or edit term within a repair phase (eR) does not. We do not train the Markov model, however, following the idea of cost functions for this task [2, 6] we stipulate a boosted probability of entering the state rmS, such that for all states that can transition into rmS, that transition is twice as likely as all other transitions out of that state. In other cases we stipulate uniform probabilities in out-going transitions. Using this marginally improved overall accuracy scores and ensured legal repair sequences were outputted.

From an incremental perspective, while Viterbi decoding need not be expensive at each step as the tag sequence hypothesis trellis for the utterance can be stored word-by-word, there is the danger of output 'jitter': undetected repair onsets may have their corresponding repair ends detected, meaning they are subsequently detected, or repair onsets initially hypothesised can be revoked. We investigate this in our evaluation.

4. Evaluation Criteria

Accuracy metrics In addition to reparandum word accuracy $\mathbf{F}_{rm}$ and repair structure accuracy $\mathbf{F}_s$ (see (3) above), we evaluate word tagging accuracy for edit term words $\mathbf{F}_e$, which includes interregna. For repairs we use the incremental metrics in [6], evaluating the *delayed accuracy* (**DA**) [5] on reparandum words detected n words back from the current word where $n \in \{1..6\}$, against the utterance-final gold standard disfluency annotations, using the mean of the 6 F-scores.

Timing metrics We use [5]'s two *time-to-detection* metrics: the two average distances (in numbers of words) consumed before first detection of gold standard repairs, one from the reparandum start, $\mathbf{TD_{rm}}$ and one from the repair start, $\mathbf{TD_{rp}}$.[5]

[5]These measures only apply to repairs detected correctly.

System	Input (context size n)	F_{rm}	F_s	F_{ed}
Elman	word (2)	0.623	0.593	0.872
Elman	word (3)	0.689	0.661	0.873
Elman	word+POS (2)	**0.711**	**0.689**	**0.902**
Elman	word+POS (3)	0.700	0.677	0.899
H&P'14	word	0.741	0.698	0.880
H&P'14	word+POS	0.779	0.735	0.937

Table 1: F1 results for RNNs with different input types

System	Input (context size n)	TD_{rp}	TD_{rm}	DA	EO
Elman	word (2)	**1.01**	**2.37**	0.600	**0.64**
Elman	word (3)	1.02	2.49	0.659	1.03
Elman	word+POS (2)	1.03	2.52	**0.681**	1.30
Elman	word+POS (3)	1.02	2.55	0.674	1.50
H&P'14	word	1.16	2.76	0.662	4.23
H&P'14	word+POS	1.16	2.79	0.698	3.95

Table 2: Incremental results for RNNs with different input types

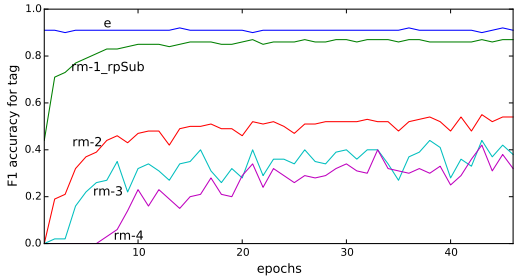

Figure 4: Learning curve progression of F1 accuracy for 5 tags

This gives us an idea of the responsiveness of our detectors, where minimal latency is preferable.

Diachronic metrics To measure stability of disfluency hypotheses over time and the level of output jitter, we use [23]'s *edit overhead* (**EO**) metric. EO measures the proportion of edits (add, revoke)[6] applied to a processor's output structure that are unnecessary– in our case the edits of concern are the word classifications as fluent (f) or reparandum (rm), repair (rp) or edit term (e), derivable from our tag scheme. Rather than evaluating EO against the current gold standard labels, we use a mark-up termed the *incremental repair gold standard* [6]: this does not penalise lack of detection of a reparandum word rm as a bad edit until the corresponding repair start of that rm has been consumed, reflecting the repair processing approach in §2. Also the repair gold standard here is in fact the final output of our RNNs rather than the ground truth– this is done to isolate the evolution of the final hypotheses, rather than evaluate accuracy, which the metrics F_{rm}, F_s, F_e and DA indicate.

5. Experimental Set-up

We experiment with different context input windows (lengths 2 and 3) and different input types (words vs. words + POS tags) to the RNNs as our independent variables. We test RNNs for their accuracy on output representations as in Fig. 1. Our tag set allows incremental derivation of repair structures as explained above when the output is finally computed through our repair Markov model. We implement our RNNs using Theano [24] as an extension to the code in [9].

Data We train on the standard Switchboard disfluency training data (all conversation numbers beginning sw2*,sw3* in the Penn Treebank III release: 100k utterances, 650K words) and use the standard heldout data (PTB III files sw4[5-9]*: 6.4K utterances, 49K words) as our validation set. We test on the standard test data (PTB III files 4[0-1]*) with partial words and punctuation removed from all files for fair comparison to other systems. We compare to the incremental disfluency detector in [6] which we will call H&P'14 from here, running this with and without POS tag input for fair comparison.

Stopping criterion We train all RNNs for a maximum of 50 epochs with a stopping criterion that if there is no improvement on the best F_{rm} score on the validation set after 10 epochs, training is halted. We found in early tests any longer than this did not result in improved performance. The best performing epoch's weights after stopping are used in testing.

6. Results and Discussion

Utterance-final accuracies are shown in Table 1. Our best F_{rm} is reasonably competitive with H&P'14 words-only model, achieving 0.711 on the test set with context size 2 using POS tags.[7] F_s does not degrade significantly compared to this measure, achieving 0.689, showing good accuracy on the repair structures. Performance on editing terms (F_e) was consistently good, being marginally below the H&P'14 model in the without POS tags conditions. As shown in Table 2, incremental performance of our RNNs was better overall than the best utterance-final settings in H&P'14, achieving low EO in our best utterance-final setting at 1.30% and minimal latency, being only marginally above the 1 word minimum achievable for $TD_{\mathbf{rp}}$. In the DA measure our best performing word-only RNN is marginally below H&P'14's best word-only model, and our best word+POS RNN achieves a competitive 0.681.

As for the effect of context size, without POS information inputs of length 2 fare much worse than length 3, however with POS information they outperform the length 3 condition. We observe an over-fitting problem when using the longer context lengths, as the local dependencies within the word window tend to be learned by the network for detecting local repair patterns in the training data which do not generalise to the test data.

As shown in Fig. 4, the best performance was generally reached after a single epoch for the e tag, however $rp{-}[n]$ had steady learning curves, with larger n taking longer to converge. The vanishing gradient problem is evident here for tags requiring longer distance dependencies, as is the problem of sparsity, however the network still manages to improve through training. A challenge we undertook here was investigating how temporal information in an RNN could overcome the need for explicit backtracking (as in H&P'14's model) – for shorter reparanda it is doing this quite successfully.

7. Conclusion

We have presented incremental disfluency detection as a word-by-word tagging task and test the performance of RNNs. While we do not eclipse state-of-the-art results, the RNNs here have had no feature engineering, and show very good incremental properties with low latency and good output stability. In future we wish to experiment with other deep learning architectures, such as the LSTM, to improve the recognition of disfluencies in the strongly time-linear way we present here.

Acknowledgments This work is supported by the Deutsche Forschungsgemeinschaft (DUEL project, grant SCHL 845/5-1) and the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277) at Bielefeld University.

[6] We do not use substitution edits here, but substitutions are implicitly realized by a revoke followed by an add for a given word tag.

[7] We achieve 0.730 on the validation set with this system.

8. References

[1] M. Honnibal and M. Johnson, "Joint incremental disfluency detection and dependency parsing," *Transactions of the Association of Computational Linugistics (TACL)*, vol. 2, pp. 131–142, 2014.

[2] X. Qian and Y. Liu, "Disfluency detection using multi-step stacked learning," in *Proceedings of NAACL-HLT*, 2013, pp. 820–825.

[3] S. Brennan and M. Schober, "How listeners compensate for disfluencies in spontaneous speech* 1," *Journal of Memory and Language*, vol. 44, no. 2, pp. 274–296, 2001.

[4] D. Schlangen and G. Skantze, "A General, Abstract Model of Incremental Dialogue Processing," *Dialoge & Discourse*, vol. 2, no. 1, pp. 83–111, 2011.

[5] S. Zwarts, M. Johnson, and R. Dale, "Detecting speech repairs incrementally using a noisy channel approach," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 1371–1378.

[6] J. Hough and M. Purver, "Strongly incremental repair detection," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 78–89.

[7] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 2010, pp. 1045–1048.

[8] W. De Mulder, S. Bethard, and M.-F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech & Language*, vol. 30, no. 1, pp. 61–98, 2015.

[9] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding." in *INTERSPEECH*, 2013, pp. 3771–3775.

[10] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[11] E. Shriberg, "Preliminaries to a theory of speech disfluencies," Ph.D. dissertation, University of California, Berkeley, 1994.

[12] E. Charniak and M. Johnson, "Edit detection and parsing for transcribed speech," in *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*. Association for Computational Linguistics, 2001, pp. 1–9.

[13] H. H. Clark, *Using Language*. Cambridge University Press, 1996.

[14] M. G. Core and L. K. Schubert, "A syntactic framework for speech repairs and other disruptions," in *Proceedings of the 37th annual meeting of the ACL*. Stroudsburg, PA: ACL, 1999, pp. 413–420.

[15] H. Clark and J. Fox Tree, "Using uh and um in spontaneous speaking," *Cognition*, vol. 84, no. 1, pp. 73–111, 2002.

[16] J. Ginzburg, R. Fernández, and D. Schlangen, "Disfluencies as intra-utterance dialogue moves," *Semantics and Pragmatics*, vol. 7, no. 9, pp. 1–64, 2014.

[17] M. Meteer, A. Taylor, R. MacIntyre, and R. Iyer, "Disfluency annotation stylebook for the switchboard corpus. ms." Department of Computer and Information Science, University of Pennsylvania, Tech. Rep., 1995. [Online]. Available: ftp://ftp.cis.upenn.edu/pub/treebank/swbd/doc/DFL-book.ps

[18] W. Levelt, *Speaking: From Intention to Articulation*. MIT Press, 1989.

[19] J. Hough and M. Purver, "Modelling expectation in the self-repair processing of annotat-, um, listeners," in *Proceedings of the 17th SemDial Workshop on the Semantics and Pragmatics of Dialogue (DialDam)*, Amsterdam, Dec. 2013, pp. 92–101.

[20] M. Johnson and E. Charniak, "A TAG-based noisy-channel model of speech repairs," in *ACL*, 2004, pp. 33–39.

[21] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[22] S. Zwarts and M. Johnson, "The impact of language models and loss functions on repair disfluency detection," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, ser. HLT '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 703–711.

[23] T. Baumann, O. Buß, and D. Schlangen, "Evaluation and optimisation of incremental processors," *Dialogue & Discourse*, vol. 2, no. 1, pp. 113–141, 2011.

[24] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a cpu and gpu math expression compiler," in *Proceedings of the Python for scientific computing conference (SciPy)*, vol. 4. Austin, TX, 2010, p. 3.