8 | Reproducing an experiment in automatic disfluency detection

Frank Grimm[1], David Schlangen[2], Julian Hough[2], Philipp Cimiano[1]

1 – Semantic Computing Group, Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University

2 – Dialogue Systems Group, Faculty of Linguistics & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University

Abstract

In this chapter, we describe an effort to reproduce the main results of the published paper *"Joint, Incremental Disfluency Detection and Utterance Segmentation from Speech"* [1], published as part of the proceedings of the "European Chapter of the Association for Computational Linguistics" (EACL) in 2017. The paper focuses on the task of disfluency detection and utterance segmentation and proposes a simple deep learning system that processes dialogue transcriptions and Automatic Speech Recognition (ASR) output. For this purpose, the Dialogue Systems Group (DSG) at Bielefeld University developed a library that relies on a data model for live ASR data that combines timing and textual information. It utilizes a refined text corpus of open data to demonstrate the feasibility of the system for simultaneously detecting disfluencies and segmenting the individual utterances for use in conversational systems and similar speech related tasks. The code and data for this reproducibility experiment are available at `https://gitlab.ub.uni-bielefeld.de/conquaire/deep_disfluency`.

Keywords

Linguistics, Speech Recognition, Python, Machine learning, LSTM, HMM, RNN, NLTK, Theano, Keras

8.1 Introduction

The Dialogue Systems Group at Bielefeld University, located at the Faculty of Linguistics and Literary Studies and the Cluster of Excellence Cognitive Interaction Technology (CITEC), studies artificial conversational systems. The deep learning based disfluency detection system presented in their paper at the International Conference of the European Chapter of the Association for Computational Linguistics (EACL) aims to improve existing solutions in the field of psychiatric health care delivery by introducing the capability to work on live data. This can facilitate the detection of word repairs for human conversational partners and improve turn taking during dialogues. While currently established systems might make use of disfluency markers in text and segment dialogues into individual utterances already, this is often restricted to processing data offline. As such, a new artificial dialogue system could for example be employed during interview sessions in order to ensure that protocols are followed. They can also augment and assist the human interviewer, since artificial conversational agents have been shown to exhibit many different markers that can be interpreted as psychological distress, such as filled pause or speech rates, as well as other temporal, utterance, and turn-related interactional features [2]. In offline processes, analysing transcripts of such sessions today is often costly and frequently relies on a disconnected utterance segmentation process. In the paper *'Joint, Incremental Disfluency Detection and Utterance Segmentation from Speech'* [1], a more cost-effective process is developed, commencing with directly processing speech data and working with online data as it incrementally becomes available during a conversation. The authors evaluate the full process through multiple metrics to capture how each subtask performs as joint or separate models, in online or offline settings. The specific research objective was to investigate how well a joint deep learning model for incremental disfluency detection and utterance segmentation performs on transcripts and ASR output. The former extends existing work on the pre-segmented utterances of the *Switchboard* (SwDA) corpus [1]. The latter uses an external ASR system (IBM Watson) to incrementally process acoustic data and, thus far, could not achieve comparable performance. While recent advances, particularly regarding lowered Word Error Rates (WER), make hypotheses generation through ASR much more reliable, they traditionally lacked similarly fine-grained annotations on different disfluency types as they were applied to transcripts. The paper in question defines the tasks of (incremental) disfluency detection and utterance segmentation, as well as the joint model. The authors discuss reasonable constraints and develop two tagsets *a)* simple and *b)* complex for different complexities of disfluency types. Three explicit research questions are subsequently developed:

- **Q1**: Given the interaction between the two tasks, can a system which performs both jointly help to improve equivalent systems doing the individual tasks?
- **Q2**: Given the incremental availability of word timings from state-of-the-art ASR, to what extent can word timing data increase performance of

[1]`https://github.com/julianhough/swda`

either task?

- **Q3**: To what extent is it possible to achieve a good online accuracy vs. final accuracy trade-off in a live, incremental, system?

In order to address these questions, the authors of the paper present two deep learning architectures for the technical task of incremental decoding for live predictions, namely Elman Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) based networks. Their experimental protocol evaluates both models in separate and joint task settings to assess whether they can exploit common constraints. The system demonstrates competitive results on different subtasks, verifying its suitability and potential to be used within conversational agents in the domain of psychiatric health.

The research experiment utilizes several machine and deep learning techniques to implement an incremental disfluency detection and utterance segmentation pipeline. Since the live audio recordings used for parts of the original experiment were not available to the Conquaire reproducibility experiment due to licensing, we focused on their tagging system in general and worked with the data that were readily available to reevaluate the published models. All considerations within this chapter refer to the *Deep Disfluency* framework as presented in the git commit identified by the hashcode **4c57a19** [2].

8.2 Methods

Speech recognition (SR), also known as *automatic speech recognition* (ASR), *computer speech recognition* or *speech to text* (STT), is a sub-field within computational linguistics that develops methods and technologies to automate the recognition and translation of spoken language into text by machines.

Human speech patterns vary between individuals and contain complex signals such as nuances and diversity in vocal patterns, aspects which adult humans take into account almost automatically. A machine on the other hand has to explicitly mitigate these aspects in order to gain a more thorough understanding of speech signals, even more so in a conversational context.

In order to suitably train a modern, reliable, speech recognition system, many machine learning algorithms and techniques work in tandem. Since fully training a STT system end-to-end would require large amounts of raw and annotated audio data in various environments, the authors rely on a suitable external system to incrementally generate textual input sequences from audio recordings and focus on the specific aspects of disfluent terms as discussed above. Here, we describe the methods used for these experiments in the Deep Disfluency library and subsequently discuss the reproduction of their results.

[2]available at `https://github.com/d<sg-bielefeld/deep_disfluency/` (4c57a19)

Model Training

The speech models are trained on millions of pre-translated words and phrases from corpora against a live ASR system. For incremental ASR, a free trial version of IBM's *Watson*[3] Speech-To-Text (STT) service was used, which according to the authors works well on noisy input data and also retains some useful artifacts such as disfluency markers (e.g. filler terms like 'uh').

The Deep Disfluency system uses the following input features:

- Words in a backwards window from the most recent word (for transcribed and ASR data, the lack of lookahead capabilities simulates the live influx of speech information).
- Durations of words in the current window, either from manually transcribed data or automatically generated by the ASR system.
- Part-Of-Speech (POS) tags for words in current window. These are either extracted from the transcribed corpus or generated through a Conditional Random Field (CRF) based tagger that was optimized on a domain specific training corpus.

The models of the Deep Disfluency system extract these features in two main experimental settings: *a)* on data generated for Switchboard audio recordings through an external ASR system and *b)* on manually transcribed data from the commonly used and well-annotated Switchboard corpus (SWdA).

For regular usage, the models trained on these corpora can be loaded and are subsequently used to apply the full tagging pipeline to arbitrary input sequences. The pipeline, described in more detail below, consists of the extraction of features as listed above, sequence to sequence tagging through one of the deep neural network architectures and consolidating their output with timing information through a Hidden Markov Model (HMM) to produce a final set of tags for each token in the sequence.

Taggers

The Deep Disfluency tagger accepts input sequences (and optionally, external POS tags and word timings) word-by-word and outputs XML-style tags for each word, symbolising disfluencies in terms of complex repairs or edit terms. The full tagset consists of:

`'<e/>'`	an edit term word, not necessarily inside a repair structure
`'<rms id=N/>'`	reparandum start word for repair with ID number N

[3]`https://www.ibm.com/watson/developercloud/speech-to-text.html`Watson

`‘<rm id=N/>‘`	mid-reparandum word for repair N
`‘<i id=N/>‘`	interregnum word for repair N
`‘<rps id=N/>‘`	repair onset word for repair N (where N is normally the 0-indexed position in the sequence)
`‘<rp id=N/>‘`	mid-repair word for repair N
`‘<rpn id=N/>‘`	repair end word for substitution or repetition repair N
`‘<rpndel id=N/>‘`	repair end word for a delete repair N

Every detected repair (and gold standard entry) will exhibit at least the `rms`, `rpS` and `rpn/rpndel` tags, others might be omitted.

Two example outputs on Switchboard utterances are shown below, where <f/> is the default tag for a fluent word:

```
4617:A:15:h    1   uh          UH     <e/>
               2   i           PRP    <f/>
               3   dont        VBPRB  <f/>
               4   know        VB     <f/>

4617:A:16:sd   1   the         DT     <rms id="1"/>
               2   the         DT     <rps id="1"/><rpn id="1"/>
               3   things      NNS    <f/>
               4   they        PRP    <f/>
               5   asked       VBD    <f/>
               6   to          TO     <f/>
               7   talk        VB     <f/>
               8   about       IN     <f/>
               9   were        VBD    <f/>
               10  whether     IN     <rms id="12"/>
               11  the         DT     <rm id="12"/>
               12  uh          UH     <i id="12"/><e/>
               13  whether     IN     <rps id="12"/>
               14  the         DT     <rpn id="12"/>
               15  judge       NN     <f/>
               16  should      MD     <f/>
               17  be          VB     <f/>
               18  the         DT     <f/>
               19  one         NN     <f/>
               20  that        WDT    <f/>
               21  does        VBZ    <f/>
               22  the         DT     <f/>
               23  uh          UH     <e/>
               24  sentencing  NN     <f/>
```

The authors compare Elman Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) network architectures to train the essential disfluency detection and prediction component of the system. The machine learning library *Theano* is used to implement both networks.

While both architectures are recurrent in nature, LSTMs present a special case of the RNN architecture. Both often exhibit distinct behaviour on different tasks and it is apriori unclear which model would outperform the other. Albeit their similar foundation, the main difference lies in the scope of previously seen information each architecture can take into consideration when predicting the next output. RNNs perform best when the information they require for a prediction is available relatively close to the current input, whereas LSTMs can learn to take information into account that is potentially further away.

All neural network models in the disfluency experiments are trained on identical training sets (within the given experiment), either up to a maximum of 50 epochs or until their parameters converge.

For POS tagging, the system uses the NLTK CRF tagger, which in turn utilizes the *crfsuite* package[4], trained with Limited-memory BFGS (L-BFGS) gradient descent optimization on the training set of the SWdA corpus and evaluated in terms of accuracy against its test split.

8.3 Analytical Reproducibility

In this section, we describe the implementation of the Deep Disfluency library, a software library that is designed for the analysis of voice and textual data. The DSG group were aware of our efforts and interested in the computational reproducibility of their research publications. They made efforts to open up their research by using open toolkits and publishing code and data to a public git repository. The authors publish the library under the permissive free MIT license through an organisational account on GitHub. The project repository contains both the source code for their, Python-based libre, software stack as well as a great deal of raw and intermittent data to reproduce various experiments.

Deep Disfluency

The Deep Disfluency code is packaged and available on a GitHub repository with documentation outlining data availability and the installation process. End users can install the system as a regular Python package via *pip*, the Python installer. The package can be obtained from the public *Python Package Index* (PyPI, using `pip install deep_disfluency`) or locally installed from source using *setuptools* (`pip install setuptools`). At the time of writing, the project depends on Python 2.7 and conveniently documents all packages that were used for

[4]`https://pypi.python.org/pypi/python-crfsuite`

development in a *virtualenv* and *pip* compatible ‘requirements.txt‘ file. This mechanism is used to manage dependencies and retains the exact version of all external library dependencies (`pip install −r requirements.txt`). Some of the *Theano* related requirements, especially for GPU enabled computation, are easier installed through the alternative repositories offered by *conda*, which is part of the *anaconda* platform for data science with Python[5]. Corpus data are either bundled with the package or can automatically be downloaded via an installation script that pulls the data and stores it locally. Internally, the structure of the disfluency system is straightforward and clearly separated into different sections:

- *ASR*, for interacting with the ASR system,
- *Corpus*, for handling different corpora,
- *data*, containing raw data (if allowed by the respective license), as well as intermittent results,
- *experiments*, for reproducing individual experiments and analysis,
- *tagger*, containing the main deep learning model implementation in ‘tagger/deep_tagger.py‘, and
- *decoder*, where a Hidden Markov Model (HMM) is implemented that combines timing information and outputs from the network model in addition to enforcing some model constraints on the output sequence.

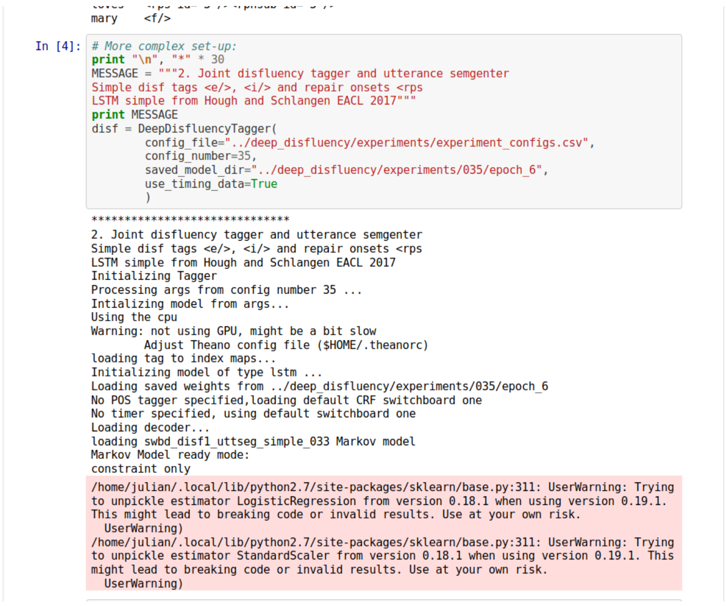
Other auxiliary parts of the system are analogouslyresponsible for one specific subtask only. After the package and its requirements have been installed, the documentation within the repository leads users to either try out a demonstration code or follow the instructions to reproduce individual experiments the system was previously used for. The latter can be used to reevaluate the experiments using RNN and LSTM models, either utilizing the pretrained models provided by the authors or training the full system from scratch. As outlined in figure 3 of the original paper, the system uses Viterbi decoding on a HMM to enforce some constraints on the final output sequence and include timing information from the transcribed corpus and ASR systems. As a crucial input feature, a part-of-speech (POS) tagger for the system was trained on in-domain Switchboard data. The implementation is based on NLTK and the resulting Conditional Random Field (CRF) model is packaged alongside the library.

Many parts of the system are modular and provide sensible defaults, e.g. if no POS tagger is specified, the library will load the default CRF tagger trained on Switchboard data. This allows end users to easily apply the disfluency detection on their own input sequences. The general pipeline that is exposed through the library of the Deep Disfluency system follows figure 3 of the original paper and

[5]`https://www.anaconda.com/`

consists of *a)* input of word embeddings and timing information, *b)* feature extraction (e.g. through POS tagging), *c)* decoding the input through a deep neural network and, optionally, *d)* combining timing information with the output of the neural network in a Hidden Markov model (HMM).

The demonstration code, located in the Jupyter [6] notebook *'demo/demo.ipynb'*, contains a set of concise examples and offers instructions on how to initialize the tagger with different configurations and pretrained models. The code also demonstrates how to tag arbitrary text sequences with the library. Figures 8.1 and 8.2 show the notebook output when the tagger and utterance segmentation system creates repair tags using RNN and LSTM configurations.



```
mary    <f/>
```

```
In [4]: # More complex set-up:
        print "\n", "*" * 30
        MESSAGE = """2. Joint disfluency tagger and utterance semgenter
        Simple disf tags <e/>, <i/> and repair onsets <rps
        LSTM simple from Hough and Schlangen EACL 2017"""
        print MESSAGE
        disf = DeepDisfluencyTagger(
                config_file="../deep_disfluency/experiments/experiment_configs.csv",
                config_number=35,
                saved_model_dir="../deep_disfluency/experiments/035/epoch_6",
                use_timing_data=True
                )
```

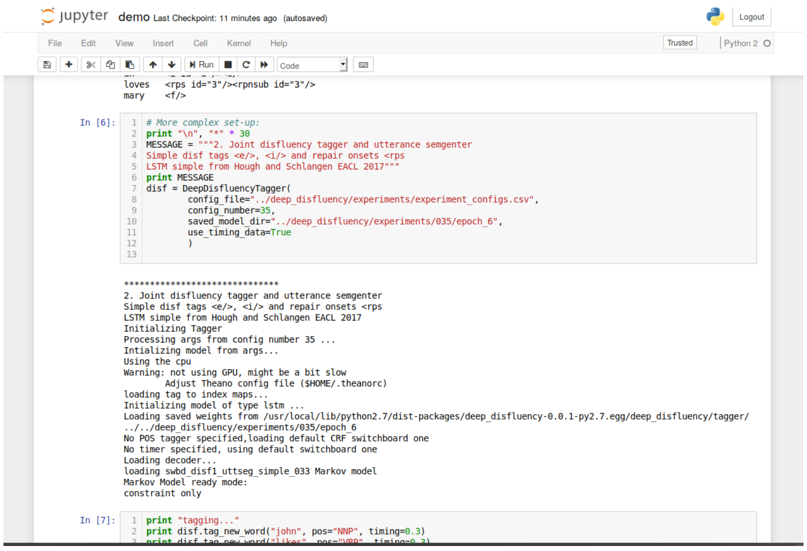
```
******************************
2. Joint disfluency tagger and utterance semgenter
Simple disf tags <e/>, <i/> and repair onsets <rps
LSTM simple from Hough and Schlangen EACL 2017
Initializing Tagger
Processing args from config number 35 ...
Intializing model from args...
Using the cpu
Warning: not using GPU, might be a bit slow
        Adjust Theano config file ($HOME/.theanorc)
loading tag to index maps...
Initializing model of type lstm ...
Loading saved weights from ../deep_disfluency/experiments/035/epoch_6
No POS tagger specified,loading default CRF switchboard one
No timer specified, using default switchboard one
Loading decoder...
loading swbd_disf1_uttseg_simple_033 Markov model
Markov Model ready mode:
constraint only
/home/julian/.local/lib/python2.7/site-packages/sklearn/base.py:311: UserWarning: Trying
to unpickle estimator LogisticRegression from version 0.18.1 when using version 0.19.1.
This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
/home/julian/.local/lib/python2.7/site-packages/sklearn/base.py:311: UserWarning: Trying
to unpickle estimator StandardScaler from version 0.18.1 when using version 0.19.1. This
might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
```

Figure 8.1: Tagger output in the Deep Disfluency demo.ipynb file

Software Toolkit and File Formats

The authors make use of a Free and Open Source Software (FOSS) based Python stack for development consisting of different NLP libraries, like NLTK; with machine learning libraries like Theano (now defunct) for deep learning. The library is currently implemented as a Python package targeting Python 2.7 environments, although a Python 3 port seems to be available. Proper packaging provides some metadata for the code itself and allows end users to install the full library, along with all dependencies, through convenient and well accepted mechanisms.

[6] https://jupyter.org/



```
loves    <rps id="3"/><rpnsub id="3"/>
mary     <f/>
```

In [6]:

```
# More complex set-up:
print "\n", "*" * 30
MESSAGE = """2. Joint disfluency tagger and utterance semgenter
Simple disf tags <e/>, <i/> and repair onsets <rps
LSTM simple from Hough and Schlangen EACL 2017"""
print MESSAGE
disf = DeepDisfluencyTagger(
        config_file="../deep_disfluency/experiments/experiment_configs.csv",
        config_number=35,
        saved_model_dir="../deep_disfluency/experiments/035/epoch_6",
        use_timing_data=True
        )

```

```
******************************
2. Joint disfluency tagger and utterance semgenter
Simple disf tags <e/>, <i/> and repair onsets <rps
LSTM simple from Hough and Schlangen EACL 2017
Initializing Tagger
Processing args from config number 35 ...
Intializing model from args...
Using the cpu
Warning: not using GPU, might be a bit slow
        Adjust Theano config file ($HOME/.theanorc)
loading tag to index maps...
Initializing model of type lstm ...
Loading saved weights from /usr/local/lib/python2.7/dist-packages/deep_disfluency-0.0.1-py2.7.egg/deep_disfluency/tagger/
../../deep_disfluency/experiments/035/epoch_6
No POS tagger specified,loading default CRF switchboard one
No timer specified, using default switchboard one
Loading decoder...
loading swbd_disf1_uttseg_simple_033 Markov model
Markov Model ready mode:
constraint only
```

In [7]:

```
print "tagging..."
print disf.tag_new_word("john", pos="NNP", timing=0.3)
```

Figure 8.2: Tagger output from the local demo.ipynb file

The Deep Disfluency package collectively specifies 85 direct dependencies, most of which are standard libraries commonly used in the NLP space. These dependencies should be readily accessible to all end users. Some relevant examples are listed below, we address the defunct *Theano* dependency in more detail as part of the following section.

- **NLTK**: natural language processing for the SwDA corpus readers and CRF implementations
- **gensim**: vector space modeling and topic modeling toolkit
- **Jupyter**: Jupyter notebook used to house parts of the analysis and visualisations
- **Keras**: a neural network library that seems to be used as an initial alternative for the Theano LSTM implementation
- **matplotlib**: a visualisation library
- **numpy**: common data structures and optimized algorithms for mathematical computing
- **pandas**: Library for working with complex data representations such as time series; also includes facilities for data manipulation and analysis
- **scikit-learn**: a machine learning library

- **scipy**: scientific and technical computing algorithms such as optimization, linear algebra, FFT
- **Theano**: (Defunct) Optimization and evaluation of mathematical expressions (including GPU computation); used for the main neural network implementations of the paper

The system makes use of a number of file formats, all of them well documented and accessible through open source frameworks. POS tagged corpora, ASR outputs and Switchboard transcriptions are stored as structured text files, comma-separated values (CSV) in the latter case. Most data artifacts created during experiments, e.g. model weights for the neural network, are serialized using the underlying libraries to create reusable *numpy* matrices. The decoder component forms a notable exception in using the Python package *pickle* for serialization. This format is specific to Python and guaranteed to offer backwards compatibility, enabling portable models between different versions. When used as a library, a convenient Python interface makes all internal file formats transparent and allows users to submit their own pre-segmented tokens for predictions through code instead.

Technical Challenges and Issues

When reproducing the main results of the paper with the Deep Disfluency library, we faced the following problems and challenges:

Dependencies: While most of the dependencies of the project are still under active development and maintenance, two minor issues were noteworthy for future reproductions:

Theano: The neural network component of the Deep Disfluency library is based on *Theano* which has been declared defunct as of 2017, when support ceased following the 1.0 release. The machine learning library originated from the Montreal Institute for Learning Algorithms (MILA), University of Montreal, who ended development and ceased implementing new features. The library shifted to low-maintenance mode, i.e. one should not rely on security bug fixes or patches being implemented at this point. At the time of this writing, a few maintainers seem to still actively commit and merge pull requests (PR) on the GitHub repository. While the deprecation of *Theano* does not, at present, hinder executing the code, it presents a potential danger which affects sustainability and makes it costlier to maintain a dependency to the library. Subsequent work should possibly make an effort to replace the affected parts of the system. Another downside that became apparent when installing the library in an environment where the precompiled dependencies of the *anaconda* repositories were unavailable is that some *Theano* dependencies require rather complicated manual setup routines and compilation on the target architecture.

Python 2.7: The Deep Disfluency library is written for Python 2.7 which has a planned end of life in the year 2020. The library will have to migrate to Python 3.x and potentially be restructured to accommodate a replacement for the machine learning library Theano. While such migrations are no trivial task in terms of time and effort, an open pull request on the GitHub repository indicates that a port to Python 3 is either under active development or already completed.

Original Data: The primary raw data used in the disfluency research project for ASR of live voice recordings was unavailable for the Conquaire reproducibility experiment. Interested parties could acquire the raw audio dataset through a subscription to the LDC Catalog[7]. Since the project retained their output of the ASR component (in 'data/asr_results/'), this does not pose a problem to reproducibility. The process on retraining the system with the original dataset is also preserved and well documented. Furthermore, the authors bundled the corpus of manually transcribed Switchboard data. We focus on this transcription based corpus since it is more readily available and can reproduce the main claims of the original paper.

8.4 Summary of reproducibility experiment

The library was installed from source in an environment equipped with hardware for computation on graphical processing units (GPUs), since the neural networking components within the Deep Disfluency system are capable of taking advantage of such hardware. The setup process through the Python packaging mechanisms did not present any major difficulties and, aside from the environment specific Theano dependency problems described earlier, could be performed just as detailed in the project documentation.

Since it is an isolated compontent that has large effect on data quality within the system, we initially verified the reported performance of the CRF used for POS tagging. The claimed accuracies of 0.915 (overall) and 0.959 (for the *UH* label) on the Switchboard test set could be easily and exactly reproduced. Invoking the feature extraction code[8], with the *TEST* flag set to *True*, loads the pretrained model that was used in the original experiments and evaluates it automatically.

Other parts of the original experiments were then repeated. The authors fortunately aggregate most of the code for the described experiment in *a)* a

[7]`https://catalog.ldc.upenn.edu/`

[8]located at 'deep_disfluency/feature_extraction/POS_Tagging.py '

Python program for training and generating evaluation data on the test sets and *b)* a Jupyter notebook for evaluation of the data generated by the different experiments.

All experiments come with a configuration entry of hyper parameters in the ‘experiment_configs.csv‘ file. This file not only controls the neural network architecture used in an experiment, it also documents important details such as hidden layer sizes and learning rates. This level of documentation and parametrization is vastly conducive to replaying experiments the way they were originally performed.

Since the authors included the best performing epochs of their original training, we opted to rerun the evaluation on the test set of the SWdA transcription corpus. Both programs involved in this worked out-of-the-box since the whole codebase makes an effort to use relative paths when referring to data files or cached models. This made switching the Jupyter Notebook used for analysis a matter of pointing a single directory from the original repository data to that of our new run. We then investigated parts of this output in regards to the original outcome.

System	F_{rps}(per word)	F_e(per word)	F_{uttSeg}(per word)	NIST SU
LSTM +timing	**0.693**	0.864	0.654	58.401
LSTM	0.665	0.862	0.666	59.714
LSTM (complex) +timing	0.655	**0.909**	0.680	**56.544**
LSTM (complex)	0.655	0.907	**0.683**	58.231
RNN +timing	0.660	0.839	0.602	68.064
RNN	0.639	0.835	0.607	70.160
RNN (complex) +timing	0.633	0.904	0.653	59.254
RNN (complex)	0.627	0.903	0.662	60.072

Table 8.4: Reproduction of results in table 2 from the original paper.

While our run of the evaluation did not produce the exact results from the original paper, they seem to be close and lead to mostly the same conclusions. The LSTM generally outperforms the RNN architecture as evident in table 8.4, which reproduces parts of table 2 in the original paper. The reported best values on the transcript corpus were $F_e = 0.918$ (LSTM) for repair onsets and $F_{rps} = 0.720$ (LSTM+timing) for editing terms, the reproduced ones reach marginally lower results ($\Delta F_e = -0.09, \Delta F_{rps} = -0.027$). Notable differences are that *a)* the reproduction yields the best F_{rps} score on the LSTM+timing model with complex tags, whereas the original analysis seems to prefer the simple tagset with timings, and *b)* the reproduction seems to exhibit consistently raised utterance segmentation error rates (NIST SU) when compared to the original.

The reproduced results on joint vs. separate tasks are similarly close to the original, see table 8.5 (consistently higher NIST SU error rates remain visible here). These data do not necessarily match the conclusions of the original paper, since the joint task formulation seems to only outperform others in terms of repair onset detection accuracy (F_{rps}) but fails to do so in terms of NIST SU rate, accuracy of edit term words (F_e), and utterance boundary detection (F_{uttSeg}). This might indicate a difference in computing environments rather than wrong results since the variance of results in our reevaluation seems generally higher. This could stem from differences in dependencies that we had to setup manually, or even differences in hardware, especially since GPU acceleration was involved in the reproduction. We executed the evaluation on a node equipped with *nVidia GeForce GTX 1080 Ti* graphic cards, invoked in a cluster environment.

System	F_{rps} (per word)	F_e (per word)	F_{uttSeg} (per word)	NIST-SU
LSTM (uttSeg only)	-	-	**0.720**	**50.222**
LSTM (disf only)	0.658	**0.912**	-	-
LSTM (joint task)	**0.693**	0.864	0.654	58.401

Table 8.5: Reproduction of results in table 3 from the original paper.

Similar small deviations can be observed regarding the re-evaluated data in table 8.6, this corresponds to table 4 of the original paper and presents the performance of incremental results over the transcript corpus. Repair onset detection in terms of words follows the findings of the original publication, with the simple LSTM model outperforming the complex ones. TTD_{rps} measured over time shows more variance than the original data, after corresponding with the authors we suspect this is likely to be an error in how the evaluation scripts aggregate the results. Even with slightly different values, the clear winner in this metric remains the simple LSTM model. In terms of edit overhead (EO) measure, the new evaluation follows the same trends between systems as originally reported. Here, the complex LSTM model that incorporates timing information clearly outperforms the simpler approaches.

The library and data for this project were generally very accessible. The researchers managed to provide an intuitive abstraction layer around their complex system of underlying data models. By bundling not only their final models but also the data used to produce them, they enable other researchers to reproduce results and adapt the system for their own corpora. Free and Open Source Software (FOSS) plays a significant role in reproducing the above results since it enables others to closely match the original environment in which an experiment was performed. We discuss how these aspects affected the reproduction and facilitates data-sharing initiatives in the following section.

System	TTD_{rps} (word)	TTD_{rps} (time in s)	Edit Overhead (word)
LSTM + timing	**0.001**	1.151	10.282
LSTM	**0.001**	**0.763**	10.735
LSTM (complex) +timing	0.104	1.093	**8.577**
LSTM (complex)	0.123	0.855	9.972

Table 8.6: Partial reproduction of the results in table 4 (incremental results for transcript level systems) from the original paper.

Discussion of the reproducibility experiment

Through the public GitHub repository and requirements documentation within the Python ecosystem we were able to reproduce most of the software environment that was used in the original experiments. Some details, such as GPU acceleration and other hardware dependent factors are subject to continuous improvement and cannot be reliably reproduced. By using compatible versions, a best effort was made to get as close as possible to the original setup within the reproduction setting. All major parts of the analytical pipeline were well documented and the authors made visible efforts to comply with many principles of good scientific data management: Findability, Accessibility, Interoperability, and Reusability (FAIR) [9] [3]. The system can be found in a public GitHub repository that presents an aggregation of all the necessary source code, documentation and most of the underlying research data that allows others to use and analyse the system. By packaging their resulting models and exposing a concise Application Programming Interface (API) to their library, the project facilitates re-use of the system as a whole in follow-up and related tasks. The project bundles sufficient instructions and programs to download all external data researchers might need in the context of the original experiments. Much of the raw data that forms the basis of the experiments is widely available. While licensing prevents the project from including the raw voice recordings used to create the ASR models, the dataset is obtainable through reliable sources and the extensive research that has already been performed on it indicates that it will likely remain accessible in the foreseeable future. The authors also provided trained models and the intermittent results they used at the time of publishing, which - in terms of reproducibility - might even be preferable over the raw data due to possible changes in the external ASR system that was used at the time. Additional research is encouraged by maintaining a copy of the Switchboard SWdA corpus itself in a separate repository[10], without having to incorporate the full disfluency system as a dependency. The system allowed us to setup a devel-

[9] `https://www.go-fair.org/fair-principles/`

[10] `https://github.com/julianhough/swda`

opment environment in short time and enabled us to independently reevaluate the models that were generated in the original experiments. The documentation, along with the scientific paper itself, provide sufficient information to gain familiarity with the codebase. While the project does not currently include an explicit description of semantic metadata, the library provides enough of an abstraction to be interoperable with any external data source. This enabled us to exactly verify parts of the original results, namely the performance of the CRF used for domain-optimized POS tagging. The full and more complex experimental settings could so far be partially reproduced through the Deep Disfluency system and original evaluation scripts, which the authors helpfully retained and separated by publication. We have been able to recreate similar results on some of the models, whereas differences in other parts of the results remain open for further investigation. Since the software used for reproduction was almost identical to the original, the reproduction did not have any major problems to re-use even the intermittent data packaged in the repository. Possible explanations for these deviations might include differences in hardware and subsequently different behaviour in terms of numerical processing or similar incompatibilities.

8.5 Conclusion

This chapter showcased a case study from the field of speech recognition and computational linguistics. The particular task was to detect disfluency markers and edit terms in spoken language (or transcriptions). This is used to detect repairs for vocal input or facilitate better detection of turn taking opportunities for subsequent tasks, e.g. in a conversational setting between human and computational agents. We were able to partially reproduce the major claims of the original paper by invoking the system's evaluation scripts on existing data and models in a completely new and independent environment. While comparisons between performance on incremental ASR output and transcription corpora had to be deferred due to licensing constraints, the reproduction could show some of the originally reported behaviour on the transcription corpus itself. The demonstration code for the Deep Disfluency library worked out of the box, enabling future users to adapt the system as a whole for their own corpora. Since the system itself is a complex project with multiple interacting components from data integration to machine learning, we are confident that given enough time and resources the rest of the results could be reproduced in a similar fashion. The research project is already very much aligned with FAIR data principles as it adopts open software practices and makes large parts of the original experiments easily accessible. Overall, this case corresponds to a case of limited reproducibility as the results could be partially reproduced for the offline settings, albeit not exactly.

References

[1] Schlangen D Hough J. Joint, incremental disfluency detection and utterance segmentation from speech. In *Proceedings of the International Conference of the European Chapter of the Association for Computational Linguistics (EACL).*

[2] David DeVault, Kallirroi Georgila, Ron Artstein, Fabrizio Morbini, David Traum, Stefan Scherer, Albert Skip Rizzo, and Louis-Philippe Morency. Verbal indicators of psychological distress in interactive dialogue with a virtual human. In *Proceedings of the SIGDIAL 2013 Conference*, pages 193–202. Association for Computational Linguistics, 2013.

[3] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3, 2016.