Baseline: A Library for Rapid Modeling, Experimentation and Development of Deep Learning Algorithms targeting NLP

Daniel Pressel, Sagnik Ray Choudhury, Brian Lester, Yanjie Zhao and Matt Barta

Interactions Digital Roots

{dpressel, schoudhury, blester, yzhao, mbarta}@interactions.com

Abstract

We introduce Baseline: a library for reproducible deep learning research and fast model development for NLP. The library provides easily extensible abstractions and implementations for data loading, model development, training and export of deep learning architectures. It also provides implementations for simple, high-performance, deep learning models for various NLP tasks, against which newly developed models can be compared. Deep learning experiments are hard to reproduce, Baseline provides functionalities to track them. The goal is to allow a researcher to focus on model development, delegating the repetitive tasks to the library.

1 Introduction

Deep Neural Network models (DNNs) now dominate the NLP literature. However, the immense progress comes with some issues. Often the research is not reproducible. Sometimes the code is not open source. Other times, available implementations fail to match the reported performance. When training DNNs, even simple baselines can take a lot of time and resources to reach peak performance (Melis et al., 2017). Additionally, a simple, canonical way to evaluate new models is lacking. Institutional pressures exist to show large relative gains in papers (Armstrong et al., 2009). As a result, new models are often compared with weak baselines.

When software is provided, it is common for authors to provide the source code as a standalone application. These projects include data processing, data cleaning, model training, and evaluation code, yielding an error-prone and time-consuming development process. A complete library should be used to automate the mundane portions of development, allowing a researcher to focus on model improvements. Also, it should be easy to compare the results of a new model across various hyper-parameter configurations and strong baselines.

To solve these problems, we have developed **Baseline**[1]. It has three components.

Core: An object-oriented Python library for rapid development of deep learning algorithms. Core provides extensible base classes for common components in a deep learning architecture (data loading, model development, training, evaluation, and export) in TensorFlow and PyTorch, with experimental support for DyNet. In addition, it provides strong, deep learning baselines for four fundamental NLP tasks – Classification, Sequence Tagging, Sequence-to-Sequence Encoder-Decoders and Language Modeling. Many NLP problems can be seen as variants of these tasks. For example, Part of Speech (POS) Tagging, Named Entity Recognition (NER) and Slot-filling are all Sequence Tagging tasks, Neural Machine Translation (NMT) is typically modeled as an Encoder-Decoder task. An end-user can easily implement a new model and delegate the rest to the library.

MEAD: A library built on Core for fast Modeling, Experimentation And Development. MEAD contains driver programs to run experiments from JSON or YAML configuration files to completely control the reader, trainer, model, and hyper-parameters.

XPCTL: A command-line interface to track experimental results and provide access to a global leaderboard. After running an experiment through MEAD, the results and the logs are committed to a database. Several commands are provided to show the best experimental results under various constraints.

The workflow for developing a deep learning

[1] https://github.com/dpressel/baseline

Proceedings of Workshop for NLP Open Source Software, pages 34–40
Melbourne, Australia, July 20, 2018.

model using Baseline is simple: 1. Map the problem to one of the existing tasks using a <task, dataset> tuple, eg., NER on CoNLL 2003 dataset is a <tagger task, conll> 2. Use the existing implementations in **Core** or extend the base model class to create a new architecture; 3. Define a configuration file in **MEAD** and run an experiment. 4. Use **XPCTL** to compare the result with the previous experiments, commit the results to the leaderboard database and the model files to a persistent storage if desired.

Additionally, the base models provided by the library can be exported from saved checkpoints directly into TensorFlow Serving[2] for deployment in a production environment. the framework can be run within a Docker container to reduce the installation complexity and to isolate experiment configurations and variants. It is open-sourced, actively maintained by a team of core developers and accepts public contributions. While some components from the library can be used for generic machine learning and computer vision tasks, the primary focus of Baseline is NLP: currently the data loaders, models and evaluation codes are provided for NLP tasks only.

2 Related Work

Baseline is not a general toolkit for deep learning (Bergstra et al., 2011; Abadi et al., 2016; Chen et al., 2015; Neubig et al., 2017a; Paszke et al., 2017). Rather, it *builds on* TensorFlow (Abadi et al., 2016), PyTorch (Paszke et al., 2017) and DyNet (Neubig et al., 2017b). Any model used in Baseline will be written in one of the supported, underlying frameworks. Popular NLP libraries such as Stanford CoreNLP (Manning et al., 2014) or nltk (Loper and Bird, 2002) provide abstractions and implementations for different algorithms in a complete NLP architecture: from basic (tokenization) to advanced (semantic parsing) tasks. Baseline serves a complimentary purpose: the models developed here can be used in these architectures.

To the best of our knowledge, two other software efforts have similar goals: AllenNLP (Gardner et al., 2017) and CodaLab [3]. AllenNLP is most similar to our work: like Baseline, it provides data and method APIs for common NLP problems and a modular and extensible experimentation framework. The key abstractions of AllenNLP are focused on data representation and conversion. This makes the model development easy for semantic understanding tasks. In Baseline, the key abstraction is an NLP "task", making it very generic. Both Baseline and AllenNLP allow running experiments from JSON configuration files, but AllenNLP currently does not provide utilities to store these configurations or results in a database in order to "track" an experiment (section 4). AllenNLP is built on PyTorch, where Baseline supports PyTorch, TensorFlow and DyNet and has a flexible design to support other frameworks over time. The development of Baseline initially began in early 2016, prior to the development of AllenNLP, and the libraries have been developed in parallel. The idea of reproducible experimentation for machine learning was introduced by CodaLab, but it does not provide abstractions and baseline implementations specific to deep learning or NLP, nor does it provide an extensible architecture with reusable components for building and exploring new deep learning models.

3 Baseline: Core

The top-level module provides base classes for data loading and evaluation. The data loader reads common file formats for classification, CONLL-formatted IOB files for sequence tagging, TSV and standard parallel corpora files for Neural Machine Translation and text files for language modeling. The data is masked and padded as necessary. It is also shuffled, sorted and batched such that data vectors in each batch have similar lengths. For sequence tagging problems, the loader supports multiple user-defined features. Also, the reader supports common formats for pretrained embeddings. The library also supports common data cleaning procedures.

The top-level module provides model base classes for the four tasks mentioned previously. The lower-level modules provide at least one implementation for each task in both TensorFlow and PyTorch. For most tasks, DyNet implementations are also available. The library provides methods to calculate standard evaluation metrics including precision, recall, F1, average loss, and perplexity. It also provides high-level utility support for common architecture layers and paradigms such as attention, highway and skip connections. The default trainer supports multiple optimizers, early

[2] https://www.tensorflow.org/serving/
[3] http://codalab.org/

stopping, and various learning rate schedules.

Model architecture development is a common use-case for a researcher. The library is designed to make this process extremely easy. In Baseline, a new model is known as an *addon*, which is a dynamically loaded module containing user-defined code. The addon code should be written as a python file in the format `<task-name>_<model-name>.py` and should be available to the python interpreter (the location should be in the system PYTHONPATH variable). The model should be written in one of the supported deep learning frameworks. If the task is not one of the existing ones, the `mead.Task` class has to be extended. The methods `create_model()` and `load_model()` have to be overridden and exposed as shown in listing 1. To run the code, the `model-name` has to be passed as an argument to the trainer program. The default trainer and data loaders can be overriden in a similar way.

The following sections describe the tasks and the implemented algorithms.

Classification: For Classification, the input is typically a sequence of words. These words are represented with word embeddings, a composition of character embeddings, or both. Sentences are represented as a combination of word representations using pooling or the final output of an RNN. A final linear layer and softmax is used for classification (Kim, 2014; Collobert et al., 2011). Baseline currently supports these models and an MLP built on pre-trained word embeddings with max-over-time pooling or mean-pooling (Neural Bag of Words) (Kalchbrenner et al., 2014).

Sequence Tagging: For Sequence Tagging, input words are represented similarly to the method used for Classification. State-of-the-art tagging is typically performed using bi-directional LSTMs (BLSTMs) with a Conditional Random Field (CRF) layer on top to promote global coherence (Lample et al., 2016; Ma and Hovy, 2016; Peters et al., 2017). Two common variants of this architecture differ primarily in the treatment of character-composition, either using a convolutional (Dos Santos and Zadrozny, 2014) or BLSTM layer (Ling et al., 2015). The convolutional composition approach is simpler and faster, yet achieves performances similar to the BLSTM (Reimers and Gurevych, 2017). Baseline implements this model and makes the CRF layer optional. It improves this model using multiple parallel convolutional filters and residual connections. It also supports multiple features such as gazetteer information.

Encoder-Decoders: Encoder-Decoder frameworks are used for Machine Translation, Image Captioning, Automatic Speech Recognition, and many other applications. Sequence-to-Sequence models are Encoder-Decoder architectures with an input sequence and an output sequence, which typically differ in length. We implement the most common version of this architecture for NLP: a stack of recurrent layers for the encoder and the decoder. We support multiple types of RNNs, including GRUs and LSTMs, as well as bidirectional encoders, multiple mechanisms of bridging the input encoder to the output decoder, and the most common types of global attention. We also provide a beam-search decoder for testing the model on standard tasks such as NMT.

Language Modeling: Language Modeling with RNNs operate at the word level or at the character level. Most baseline implementations use word-based models following (Zaremba et al., 2014) but character-compositional models (Kim et al., 2016; Józefowicz et al., 2016) may significantly reduce the number of parameters. Baseline has implementations for both word-based models and character-compositional models, closely following the model architecture of Kim et al. (2016).

3.1 Dataset and Results

Table 3.1 summarizes the TensorFlow implementation results. The PyTorch results are equivalent. Detailed results and hyper-parameter configurations are maintained in the library codebase.

For **Classification**, we use three public datasets: 2-class Stanford Sentiment Treebank (SST2) (Socher et al., 2013), TREC QA (Voorhees and Tice, 2000), and DBpedia (Auer et al., 2007). The **Sequence Tagger** has been tested on several problems including NER (CoNLL 2003 (Sang and Meulder, 2003), wnut17 (Derczynski et al., 2017) datasets), POS Tagging (Twitter dataset (Gimpel et al., 2011): TwPOS) and Slot Filling (ATIS dataset (Dahl et al., 1994)). The CRF layer is critical for NER but not necessary for ATIS and the Twitter POS corpus. For **Language Modeling**, our model improves on Zaremba et al. (2014) using pre-trained word embeddings. The state-of-the-art for Language Modeling has changed sig-

```
def create_model(embeddings, labels, **kwargs):
        return MyModel.create(embeddings, labels, **kwargs)

def load_model(modelname, **kwargs):
        return MyModel.load(modelname, **kwargs)
```

Listing 1: Methods to override and expose in a user-defined model

nificantly since our implementation and we anticipate releasing a new baseline model in the future (Yang et al., 2017). Our **Encoder-Decoder** model is tested on English-Vietnamese Translation Task on TED tst2013 corpus (Cettolo et al., 2015) and achieves strong results.

Apart from these base models, we provide implementations of more advanced models that can demonstrate the software architecture and provide a stepping stone for researchers developing their own models. Some of these implementations show better results than the existing state of the art. For example, using pre-trained ELMo embeddings from TensorFlow Hub (Peters et al., 2018), our tagger has 42.19% F1 for the wnut17 dataset, which is better than the last reported highest score (41.86%) on the dataset (Derczynski et al., 2017). The repository [4] is updated continually with the list of available implementations.

4 MEAD and XPCTL

DNNs are heavily dependent on hyper-parameter tuning, yet many papers do not report the exact hyper-parameters. This often leads to non-reproducible research. To solve this problem, we have developed **MEAD** and **XPCTL** to track hyper-parameters, model architecture, and results of a deep learning experiment.

MEAD: MEAD provides a driver program that runs experiments from a configuration file (in JSON/YAML format). We define a problem as a <task, dataset> tuple. For any task, the configuration file should contain 1. The dataset name, 2. Embedding type, 3. Reader type, 4. Model type, 5. Model hyper-parameters (number of layers, convolution filter size), 6. Training parameters (number of epochs, optimizers, optimizer specific parameters, patience for early stopping), 7. Pre-processing information. Reasonable default values are provided where possible. Thus, the whole experiment including hyper-parameters is uniquely identified by the sha1 hash of the configuration file. An experiment produces comprehensive logs including step-wise loss on the training data and task-specific metrics on the development and test sets. The reporting hooks support popular visualization frameworks including Tensorboard logging and Visdom. The model is persisted after each epoch, or, when early-stopping is enabled, whenever the model improves on the target development metric. The persisted model can be used to restart the training process or perform inference.

XPCTL: XPCTL (experiment control) can be used to track and compare the results with the previous ones by providing a command line interface to a database. The current implementation supports common databases including MongoDB, PostgreSQL and SQLite. The existing base classes can be extended to support other databases if needed. The configuration file, logs, and results can be stored in the database through a command. XPCTL also helps to maintain a leaderboard for these tasks. The results for a problem (<task, dataset>) can be sorted by any evaluation metric, filtered for particular users and limited by a number. Configuration files can be downloaded and the model files can be stored in a persistent storage location using the same utility.

The current implementation delegates the database creation and maintenance to the user. In future, we plan to maintain a global database accessible to all users. The user can also have her own local database, and push to the global leaderboard as needed. XPCTL will provide command line utilities for this purpose.

5 Exporting Models

DNNs can be deployed within a model serving framework such as **TensorFlow Serving**, a high-performance and highly scalable deployment architecture. All of the baseline implementations have exporters, though some customization may be required for user-defined models, for which we provide interfaces. The exporter transforms the model to include pre-processing and service-

[4] https://github.com/dpressel/baseline/tree/master/python/addons

Problem	Dataset	Algorithm	Metric	Score
Tagging	ConLL 2003	CNN-BLSTM-CRF	F1	90.88
	wnut17	CNN-BLSTM-CRF		39.19
	TwPOS	CNN-BLSTM		88.91
	ATIS	CNN-BLSTM		96.74
Classification	SST2	CNN	accuracy	87.9
		LSTM		87.1
	TREC-QA	CNN		93.2
		LSTM		91.8
	DBpedia	CNN		99.05
		LSTM		98.95
Language Modelling	PTB	RNN	perplexity	77.22
Encoder-Decoder	TED tst2013	Seq2Seq	BLEU	25.21

Table 1: Results for TensorFlow implementations in Baseline

consumable output.

6 Conclusion and Future Work

We have presented Baseline, a library for reproducible experimentation and fast development of DNN models in NLP. Our goal is to help automate the frustrating parts of the process of model development and deployment to allow researchers to focus on innovation. The library is currently used in a production environment for various problems, attesting to the fact that it is suitable for a complete research-to-deployment pipeline. Currently, the library has implementations for many strong baseline models which we will continue to update and improve as the state-of-the-art changes. Future versions of the software will attempt to improve the development process further by assisting with automatic parameter tuning and model selection, support for more deep learning frameworks, improved visualization, and a simpler cross-platform deployment mechanism.

Acknowledgments

We thank Patrick Haffner, Nick Ruiz and John Chen for their valuable feedback. Funding for this research was provided by Interactions LLC.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 265–283.

Timothy G. Armstrong, Justin Zobel, William Webber, and Alistair Moffat. 2009. Relative significance is insufficient: Baselines matter too.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg. Springer-Verlag.

James Bergstra, Olivier Breuleux, Frederic Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2011. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, volume 1.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, Roldano Cattoni, and Marcello Federico. 2015. The iwslt 2015 evaluation campaign. In *IWSLT 2015, International Workshop on Spoken Language Translation*.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274.

Ronan Collobert, Jason Weston, Lon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Deborah A. Dahl, Madeleine Bates, Michael Brown adn William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, Elizabeth Shriberg, John S. Garofolo, Jonathan G. Fiscus, Denise Danielson, Enrico Bocchieri, Bruce Buntschuh, Beverly Schwartz, Sandra Peters, Robert Ingria, Robert Weide, Yuzong Chang, Eric Thayer, Lynette Hirschman, Joe Polifroni, Bruce Lund, Goh Kawai, Tom Kuhn, and Lew Norton. 1994. Atis3 training data ldc94s19.

Leon Derczynski, Eric Nichols, Marieke van Erp, and Nut Limsopatham. 2017. Results of the WNUT2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text, NUT@EMNLP 2017, Copenhagen, Denmark, September 7, 2017*, pages 140–147.

Cícero Nogueira Dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1818–II–1826. JMLR.org.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA - Short Papers*, pages 42–47.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland. Association for Computational Linguistics.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2741–2749.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 260–270.

Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1520–1530.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany. Association for Computational Linguistics.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 55–60.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2017. On the state of the art of evaluation in neural language models. *CoRR*, abs/1707.05589.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017a. Dynet: The dynamic neural network toolkit. *CoRR*, abs/1701.03980.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal

Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017b. Dynet: The dynamic neural network toolkit. *CoRR*, abs/1701.03980.

Adam Paszke, Sam Gross, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1756–1765.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning, CoNLL 2003, Held in cooperation with HLT-NAACL 2003, Edmonton, Canada, May 31 - June 1, 2003*, pages 142–147.

Richard Socher, A. V. Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods in Natural Language Processing*.

Ellen M. Voorhees and Dawn M. Tice. 2000. Building a question answering test collection. In *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 24-28, 2000, Athens, Greece*, pages 200–207.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2017. Breaking the softmax bottleneck: A high-rank RNN language model. *CoRR*, abs/1711.03953.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR*, abs/1409.2329.