MoRTy: Unsupervised Learning of Task-specialized Word Embeddings by Autoencoding

Nils Rethmeier
German Research Center for AI (DFKI)
Alt-Moabit 91c
10559 Berlin. Germany
nils.rethmeier@dfki.de

Barbara Plank
Department of Computer Science
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen S, Denmark
bplank@itu.dk

Abstract

Word embeddings have undoubtedly revolutionized NLP. However, pre-trained embeddings do not always work for a specific task (or set of tasks), particularly in limited resource setups. We introduce a simple yet effective, self-supervised post-processing method that constructs task-specialized word representations by picking from a menu of reconstructing transformations to yield improved end-task performance (MoRTy). The method is complementary to recent state-of-the-art approaches to inductive transfer via fine-tuning, and forgoes costly model architectures and annotation. We evaluate MoRTy on a broad range of setups, including different word embedding methods, corpus sizes and end-task semantics. Finally, we provide a surprisingly simple recipe to obtain specialized embeddings that better fit end-tasks.

1 Introduction

Word embeddings are ubiquitous in Natural Language Processing. They provide a low-effort, high pay-off way to improve the performance of a specific supervised end-task by transferring knowledge. However, recent works indicate that universally best embeddings are not yet possible (Bollegala and Bao, 2018; Kiela et al., 2018a; Dingwall and Potts, 2018), and that they instead need to be tuned to fit specific end-tasks using inductive bias – i.e., semantic supervision for the unsupervised embedding learning process (Conneau et al., 2018; Perone et al., 2018). This way, embeddings can be tuned to fit a specific single-task (ST) or multi-task (MT: set of tasks) semantic (Xiong et al., 2018).

Fine-tuning requires labeled data, which is often either too small, not available or of low quality and creating or extending labeled data is costly and slow. Word embeddings are typically induced from huge unlabeled corpora with billions of tokens, but for limited-resource domains like biology or medicine, it becomes less clear whether there is still transfer. We set out to create task-specified embeddings cheaply, with self-supervision, that are able to provide consistent improvements, even in limited resource settings.

We evaluate the impact of our method, named MoRTy, on 18 publicly available benchmark tasks developed by Jastrzebski et al. (2017)[1] using two ways to induce embeddings, Fasttext and GloVe. We test them in two setups corresponding to two different overall aims: (a) to specialize embeddings to better fit a *single* supervised task or, (b) to generalize embeddings for *multiple* supervised end-tasks, i.e., to optimize MoRTys for *single* or *multi-task* settings. Since most embeddings are pre-trained on large corpora, we also investigate whether our method further improves embeddings trained on small corpus setups.

Hence, we demonstrate the method's application for single-task, multi-task, small, medium and web-scale (common crawl) corpus-size settings (Section 4). *Learning to scale-up* by pre-training on more (un-)labeled data is both: (a) not always possible in low-resource domains due to lack of such data, and (b) heavily increases the compute requirements of comparatively small supervised down-stream task. This not only leads to high per model-instance costs but also limits *learning to scale-out*, i.e., when combining many smaller models into a larger dynamic model as is desirable in continual learning settings, where models, inputs and objectives may emerge or disappear over time. To provide an alternative in such settings we design MoRTy as a *learning-to-scale-down* approach, that uses less data and compute to achieve a performance improvement despite *forgoing (un-)supervised fine tuning* on target domain data. Consequently, MoRTy uses

[1] https://github.com/kudkudak/word-embeddings-benchmarks

Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019), pages 49–54
Florence, Italy, August 2, 2019.

very little resources,[2] producing a low carbon footprint, especially regarding recent, compute intensive, *scale-up* approaches like ELMo or BERT (Peters et al., 2018; Devlin et al., 2018) which have high hardware and training time requirements and a large carbon footprint as recently demonstrated by Strubell et al. (2019). As a result, we demonstrate a simple, unsupervised scale-down method, that allows further pretraining exploitation, while requiring minimum extra effort, time and compute resources. As in standard methodology, optimal post-processed embeddings can be selected according to multiple proxy-tasks for overall improvement or using a single end-task's development split—e.g., on a fast baseline model for further time reduction.

2 MoRTy embeddings

Our proposed post-processing method provides a **M**enu **o**f **R**econstructing **T**ransformations to **y**ield improved end-task performance (MORTY).

Approach: The key idea of MORTY is to create a family of embeddings by learning to reconstruct the original pre-trained embeddings space via autoencoders.

The resulting family or representations (post-processed embeddings) gives a "menu" which can be picked from in two ways: (a) standard development set tuning, to gain performance at a *single* supervised task (ST), or (b) via benchmark tasks, to boost performance of *multiple tasks* (MT). The first is geared towards optimizing embeddings for a single specific task (specialization), the latter aims at embedding generalization, that works well across tasks.

In more details, the overall MORTY recipe is: **(1) Train (or take)**: an original (pre-trained) embedding space E_{org} using embedding method f. **(2) Reconstruct E_{org}:** compute multiple randomly initialized representations of E_{org} using a reconstruction loss (mean square error, cf. below). **(3) Pick:** performance-optimal representation for the end-task(s) via a task's development split(s) or proxy tasks, depending on the end-goal, i.e., specialization or generalization. **(4) Gain:** use optimal MORTY (E_{post}) to push relative performance on end task(s).

[2] $<$ 1GB memory including the whole dataset, computes fast on GPU and CPU and inherits FastText's dynamic out-of-vocabulary token embedding generation, which is useful in handling unforeseen words in down-stream tasks.

Which autoencoder variant? For step (2), we found the following autoencoder recipe to work best: A linear autoencoder with one hidden layer, trained via bRMSE (batch-wise root mean squared error), the same hidden layer size as the original embedding model and half of its learning rate[3]– i.e., *a linear, complete autoencoder*, trained for a single epoch (cf. end of Section 3).

We experimented with alternative autoencoders: sparse (Ranzato et al., 2007), denoising, discrete (Subramanian et al., 2018), and undercomplete autoencoders, but found the simple recipe to work best. In the remainder of the paper, we test this 'imitation-scheme' setup recipe.

3 Experiments

With the aim of deriving a simple yet effective 'best practice' usage recipe, we evaluate MORTY as follows: a) using two word embedding methods f; b) corpora of different sizes to induce E_{org}, i.e., small, medium and web-scale; c) evaluation across 18 semantic benchmark tasks spanning three semantic categories to broadly examine MORTY's impact, while assessing both single and multi-task end goals; and finally e) evaluate 1-epoch setups in relation to different corpus sizes.

Embeddings and Corpus Size: We evaluate embeddings trained on small, medium (millions of tokens) and large (billions of tokens) corpus sizes. In particular, we train 100-dimensional embeddings with Fasttext (Bojanowski et al., 2016)[4] and GloVe (Pennington et al., 2014)[5] on the `2M` and `103M` WikiText created by Merity et al. (2016). We complement them with off-the-shelf web-scale Fasttext and GloVe embeddings (trained on 600B and 840B tokens, respectively). This results in the following vocabulary sizes for Fasttext and GloVe embeddings, respectively: on `2M` 25,249 and 33,237 word types. For `103M` we get 197,256 and 267,633 vocabulary words. Public, off-the-shelf – common-crawl trained – Fasttext and GloVe embeddings have very large vocabularies of 1,999,995 and 2,196,008 words.

To account for variation in results, we train both embedding methods *five times each*[6] on the two WikiText corpus sizes. We observed only minor

[3] Original Fasttext and GloVe used $lr = 0.05$, so $lr \approx 0.025$ is a 'careful' rate and used throughout the experiments in this paper.

[4] To train Fasttext we used `https://fasttext.cc`

[5] To train GloVe we used the python glove_python wheel

[6] Fasttext was trained using the implementation's

variations, $< 0.5\%$ between runs for both Fasttext and GloVe, in overall performance Σ – i.e., when summing the scores of all benchmark tasks.

Semantic benchmark tasks: We use a publicly available word embedding benchmark implementation developed by Jastrzebski et al. (2017) – chosen for reproducibility and breadth. The 18 tasks span three semantic categories: (a) word similarity (6 tasks), (b) word analogy (3 tasks), and (c) word and sentence categorization (9 tasks).[7]

Evaluation and Experimental Details For the single-task setup we show MORTY's relative, percentual performance change (`ST % change`) produced by choosing the best MORTY embedding per task – 18 MORTYs. Correspondingly, for multi-task results we show `MT % change` obtained by choosing the MORTY embedding with the best score over all tasks Σ – i.e., one MORTY for all tasks. Performances in Table 1 are averaged over 5 runs each of Fasttext and GloVe per corpus size. To maximize MORTY's usability we evaluate a **1-epoch** training scheme. We test its robustness – *particularly for limited resource use* – by training 1 epoch on three corpus sizes (small to web-scale), using the best multi-task (MT/ Σ) base embedder – see Fasttext Table 1. We again account for variation by using 3 randomly initialized MORTY runs, each over the 5 respective runs per corpus size. In this experiment, a single epoch yielded very stable boosts, that are comparable to multi-epoch training.

4 Results

The main results are provided in Table 1 and Figure 1. There are several take-aways.

f: Fasttext and GloVe: First, regarding the base embeddings (cf. per-category base performance scores in Table 1): i) we notice that Fasttext performs overall better than GloVe; ii) classification and similarity results improve the larger the corpus; consistently over f; and iii) GloVe is better for the analogy tasks on web-scale data.[8]

MORTY for multi-task application: Second, the `MT % change` columns show that a single best MORTY improves overall performance Σ (black row)[9] – the sum of 18 tasks – by 8.9, 5.8 and 3.4 percent compared to Fasttext base. As corpus size increases, there is less space for MORTY to improve Σ scores. What is interesting to note is that MORTY is able to recover analogy performance on 103M (to more than 2M level). This is also reflected in the Google and MSR analogy scores doubling and tripling (middle column). On 2M we also see a modest improvement (6.2) for similarity tasks, while classification on 2M slightly dropped. Regarding GloVe (3 rightmost columns) we notice lower overall performance (black column), which is consistent with findings by Levy et al. (2015). MORTY on GloVe produces lower but more stable improvements for the MT setting (middle column), with analogy and similarity performance noticeably increasing for the small 2M dataset. Generally, we see both performance increases and drops for individual task, especially on 2M and Fasttext, indicating that, a single overall best MORTY specializes the base Fasttext embedding to better fit a specific subset of the 18 tasks, while still beating the base embedders f in overall score (Σ).

MORTY for single-task application: In the `ST % change` columns we see best single task (ST) results for task-specific optimal MORTY embeddings. Both embedders get consistent boosts, with Fasttext exhibiting significantly higher improvement from MORTY on 2M and 103M, despite already starting out at a higher base performance.

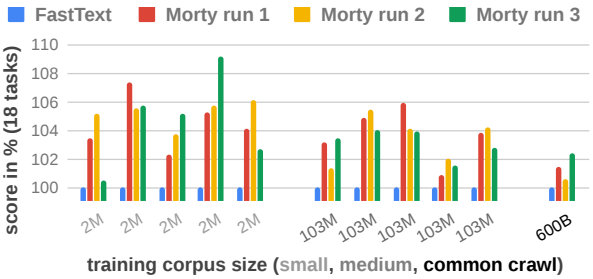

Figure 1: **1-epoch MORTY (MT %) performance change over Fasttext:** Blue bars show Fasttext baseline performance (100%). 3 Morty runs: trained on Fasttext for **1 epoch** (2x5 Fasttext for corpus sizes 2M and 103M and 1x for 600B). Detailed description on next page.

(`fasttext.cc`) default parameters. GloVe was trained with the same parameters as in (Pennington et al., 2014) – Figure 4b. Though, 4a gave the same results.

[7] Jastrzebski et al. (2017) use measures form the dataset literature: Spearman correlation for similarity, 3CosAdd for analogy and accuracy and cluster purity for categorization.

[8] GloVe 3CosAdd matches (Levy and Goldberg, 2014).

[9] Note that, % change for Σ is not the average of the individual task changes, but the % change of the sum of 18 individual scores.

embedder model	Fasttext base performance			MT % change by 1 overall Morty			ST % change by 18 single Mortys			GloVe base performance			MT % change by 1 overall Morty			ST % change by 18 single Mortys		
train size	2M	103M	600B	2M	103M	600B	2M	103M	600B	2M	103M	840B	2M	103M	840B	2M	103M	840B
AP	0.31	0.59	0.68	-6.1	-0.9	-1.5	8.2	5.2	4	0.2	0.43	0.61	2.7	5.6	9.3	13.2	9.2	12.2
BLESS	0.3	0.73	0.84	-2.2	3.8	-3	13	9.7	5.4	0.27	0.51	0.85	1.6	-1.6	-1.8	7.9	7.9	4.7
Battig	0.14	0.32	0.48	-3.6	0.1	-3.7	7	4	0.5	0.1	0.19	0.46	3.5	2	1.9	7.4	5.4	8.5
ESSLI 1a	0.48	0.76	0.77	2.2	4.3	17.6	27.5	10.2	17.6	0.46	0.63	0.75	0	3.1	9.1	8	8.9	12.1
ESSLI 2b	0.63	0.75	0.78	9.2	2.7	0	26.5	11.3	12.9	0.51	0.74	0.75	19.9	-0.5	6.7	23.7	11.7	16.7
ESSLI 2c	0.54	0.54	0.62	-3.7	10.7	-10.7	11	19.7	10.7	0.46	0.54	0.62	2.1	2.7	0	16.9	16.7	10.7
Google	0.06	0.04	0.12	33.6	293.8	187.3	45.3	319.3	217.2	0	0.05	0.58	42.7	13.8	2.8	60.4	18.6	5.9
SEval 12 2	0.11	0.16	0.24	1.6	4.3	-2.8	18.1	14.1	4.8	0.11	0.15	0.2	6.5	2.2	1	11.4	5	2.4
MSR	0.28	0.08	0.18	18.8	246.2	117.1	27.5	267.3	137	0	0.09	0.57	45.6	30.9	-2.4	100.7	38.1	10.1
MTurk	0.24	0.52	0.73	65.6	5.1	1.1	98	12.6	1.5	0.3	0.46	0.69	-22.4	2.6	0.5	1.6	4.2	2.6
RG65	0.29	0.71	0.86	65.2	0.7	2.1	104.7	5.3	5.6	0.15	0.44	0.77	11.6	3.9	-1.3	30.8	10	4
RW	0.21	0.38	0.59	-17.1	-0.8	-2	4.1	2.4	0.9	0.2	0.21	0.46	-2.1	11.8	2	4	19.8	10.3
MEN	0.36	0.71	0.84	13	0.4	-0.4	22	2.3	0.3	0.16	0.51	0.8	3.6	5.6	0.5	15.1	7	7.7
SimLex999	0.18	0.31	0.5	-23.2	3.7	-1.2	7.3	9	3.1	0.03	0.22	0.41	147.8	7.3	3.1	228.3	11.7	9.3
TR9856	0.1	0.13	0.18	2.8	-4.1	-37.1	20.5	17.3	-2.5	0.09	0.08	0.1	13.9	8.9	-4.7	19.8	47.3	36.7
WS353	0.46	0.69	0.79	3.9	1	-1.7	10	2.9	0.6	0.16	0.45	0.74	31.5	7.2	0.7	36.8	8.2	5.6
WS353R	0.35	0.63	0.74	16.4	1.7	-2.8	24.3	4.1	1.6	0.08	0.4	0.69	53.1	6.5	1.1	62	8.2	2.7
WS353S	0.52	0.77	0.84	3.2	0.4	0.6	13.3	3	1.9	0.27	0.58	0.8	15.1	6.5	0.3	20.2	7.6	5.9
∑ tasks	5.55	8.83	10.79	8.9	5.8	3.4	8.9	5.8	3.6	3.56	6.68	10.84	7.8	4.3	1.9	7.8	4.3	1.9
category	2.39	3.7	4.17	-2.1	-0.2	1.8	11.4	4.5	3.1	2	3.04	4.05	3.5	-0.8	2.4	7.3	3.3	5.5
analogy	0.45	0.28	0.55	15.5	115	72.2	24.6	125.2	92.7	0.11	0.29	1.34	7.4	4.2	1.3	12.3	15.8	6.5
similarity	2.71	4.85	6.07	6.2	-0.6	-4.7	17.3	2.2	-0.3	1.45	3.35	5.45	9.2	1.8	0	11	6.3	2.9
legend	<50%	50%	>50%	< -10%		no change		> +10%		<50%	50%	>50%	< -10%		no change		> +10%	

Table 1: **MORTY on Fasttext and GloVe**: Above are scores for: 18 individual tasks (AP-WS353S), the sum of 18 scores Σ, and scores grouped by semantic: similarity (AP-ESSLI2c), analogy (Google-MSR), classification (MTurk-WS253S). **Left column:** shows absolute scores of the original embedder. **Middle column:** shows % score change after fine-tuning with the MORTY that has the *highest overall score* Σ – i.e., 1 MORTY for all tasks (`multi-task`). **Right column:** shows % score change after applying 18 individually best MORTYs per `single-task` – i.e., 18 MORTYs . Each column is further split by corpus size – 2M, 103M(illion) and 600/840B(illion) tokens. All scores are averages over 5 original embedder scores and respective MORTY changes.

Applying the MORTY 1-epoch recipe So far, we saw MORTYs potential for overall (ST/MT/Σ) performance improvements, but will we observe the same *in the wild*? To answer this question for the MT use-case, we apply a *1-epoch* training only recipe. That is, we train *1-epoch* using a linear, complete autoencoder using *half of the base embedders learning rate* on three randomly initialized MORTYs, and then test them on the 18 task (MT) setup. Figure 1 shows consistent MT/Σ score improvements for each of the 3 MORTY-over-Fasttext runs (red, yellow, green) on 2M, 103M, and 600B vs. base Fasttext (blue 100).

We see that, for practical application, this allows MORTY to boost supervised MT performance even without using a supervised development split or proxy task(s), while also *eliminating multi-epoch tuning*. Both Figure 1 and Table 1 show similar overall (MT) improvements per corpus size, which suggests that 1-epoch training is sufficient and that **MORTY is especially beneficial on smaller corpora** – i.e., in limited resource settings.

5 Related Work

There is a large body of work on information transfer between supervised and unsupervised tasks. First and foremost **unsupervised-to-supervised** transfer includes using embeddings for supervised tasks. However, transfer also works vice versa, in a **supervised-to-unsupervised** setup to (learn to) specialize embeddings to better fit a specific supervised signal (Ruder and Plank, 2017; Ye et al., 2018). This includes injecting generally relevant semantics via retrofitting or auxiliary multi-task supervision (Faruqui et al., 2015; Kiela et al., 2018b). **Supervised-to-supervised** methods provide knowledge transfer between supervised tasks which is exploited successively (Kirkpatrick et al., 2017), jointly (Kiela et al., 2018b) and in joint-succession (Hashimoto et al., 2017).

Unsupervised-to-unsupervised transfer is less studied. Dingwall and Potts (2018) proposed a GloVe *model-modification* that retrofits publicly available GloVe embeddings to produce specialized domain embeddings, while Bollegala and

Bao (2018) propose *meta-embeddings* via denoising autoencoders to *merge* diverse (Fasttext and GloVe) embeddings spaces. The later, is also a low-effort approach and closest to ours. However, it focuses on embedding merging that they *tuned on a single* semantic similarity task, while MORTY provides an overview of *tuning for 19 different settings*. Furthermore, MORTY requires only a single embedding space, which contributes to the literature by outlining that meta-embedding improvements may partly stem from re-encoding rather than only from semantic merging.

6 Conclusion

We demonstrated a low-effort, self-supervised, *learning scale-down* method to *construct task-optimized* word embeddings from existing ones to gain performance on a (set of) supervised end-task(s) without direct domain adaptation. Despite its simplicity, MORTY is able to produce significant performance improvements for *single* and *multi-task* supervision settings as well as for a variety of desirable word encoding properties while forgoing building and tuning complex model architectures and labeling.[10] Perhaps most importantly, MORTY shows considerable benefits for low-resource settings and thus provides a *learning-to-scale-down* alternative to recent *scale-up* approaches.

7 Acknowledgements

This work was supported by the German Federal Ministry of Education and Research (BMBF) through the project DEEPLEE (01IW17001) and by the European Unions Horizon 2020 research and innovation programme under grant agreement No 780495 (BigMedilytics). We also thank Philippe Thomas and Isabelle Augenstein for helpful discussions.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *CoRR*, abs/1607.04606.

Danushka Bollegala and Cong Bao. 2018. Learning word meta-embeddings by autoencoding. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1650–1661. Association for Computational Linguistics.

[10] Source code at https://github.com/NilsRethmeier/MoRTy

Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco" Baroni. 2018. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Nicholas Dingwall and Christopher Potts. 2018. Mittens: an extension of glove for learning domain-specialized representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 212–217. Association for Computational Linguistics.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615. Association for Computational Linguistics.

Kazuma Hashimoto, caiming xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A joint many-task model: Growing a neural network for multiple nlp tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1923–1933. Association for Computational Linguistics.

Stanislaw Jastrzebski, Damian Lesniak, and Wojciech Marian Czarnecki. 2017. How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks. *CoRR*, abs/1702.02170.

Douwe Kiela, Changhan Wang, and Kyunghyun Cho. 2018a. Context-attentive embeddings for improved sentence representations. *CoRR*, abs/1804.07983.

Douwe Kiela, Changhan Wang, and Kyunghyun Cho. 2018b. Dynamic meta-embeddings for improved sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1477. Association for Computational Linguistics.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Omer Levy and Yoav Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 171–180. Association for Computational Linguistics.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *CoRR*, abs/1609.07843.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics.

Christian S. Perone, Roberto Silveira, and Thomas S. Paula. 2018. Evaluation of sentence embeddings in downstream and linguistic probing tasks.

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

Marc' Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. 2007. Sparse feature learning for deep belief networks. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 1185–1192, USA. Curran Associates Inc.

Sebastian Ruder and Barbara Plank. 2017. Learning to select data for transfer learning with bayesian optimization. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 372–382. Association for Computational Linguistics.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in nlp. In *Proceedings of ACL 2019, Florence, Italy, July 28 - August 2, 2019*. Association for Computational Linguistics.

Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard Hovy. 2018. Spine: Sparse interpretable neural embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Shufeng Xiong, Hailian Lv, Weiting Zhao, and Donghong Ji. 2018. Towards twitter sentiment classification by multi-level sentiment-enriched word embeddings. *Neurocomputing*, 275:2459–2466.

Zhe Ye, Fang Li, and Timothy Baldwin. 2018. Encoding sentiment information into word vectors for sentiment analysis. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 997–1007. Association for Computational Linguistics.