MORTY Embedding

Improved Embeddings without Supervision

Anonymous xxx conference submission

Abstract

We demonstrate a low effort method that *unsupervisedly constructs task-optimized* embeddings from existing word embeddings to gain performance on a supervised end-task. This avoids additional labeling or building more complex model architectures by instead providing specialized embeddings better fit for the end-task(s). Furthermore, the method can be used to roughly estimate whether a specific kind of end-task(s) can be learned form, or is represented in, a given unlabeled dataset, e.g. using publicly available probing tasks. We evaluate our method for diverse word embedding probing tasks and by size of embedding training corpus – i.e. to explore its use in reduced (pretraining-resource) settings.

1 Introduction

Unsupervisedly pretrained word embeddings provide a low-effort, high pay-off way to improve the performance of a specific supervised end-task by exploiting Transfer learning from an unsupervised to the supervised task. Additionally, recent works indicate that universally best embeddings are not yet possible, and that instead embeddings 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; Kiela et al., 2018a). Hence the established notion, that in order to fine-tune embeddings for specific end-tasks, labels for those end-tasks a required. However, in practice, especially in industry applications, labeled dataset are often either too small, not available or of low quality and creating or extending them is costly and slow.

Instead, to lessen the need for complex supervised (Multi-task) fine-tuning, we explore using *unsupervised fine-tuning* of word embeddings for either a specific end-task (ST) or a set of desired end-tasks (MT). By taking pretrained word embeddings and unsupervisedly postprocessing (fine-tuning) them, we evaluate postprocessing performance changes on publicly available probing tasks developed by Jastrzebski et al. (2017)[1] to demonstrate that widely used word embeddings like Fasttext and GloVe can either: (a) be *unsupervisedly* specialized to better fit a *single* supervised task or, (b) can generally improve embeddings for *multiple* supervised end-tasks – i.e. the method can optimize for *single* and *Multi-task* settings. As in standard methodology, optimal postprocessed embeddings can be selected using 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. Since most embeddings are pretrained on large corpora, we also investigate whether our method – dubbed MORTY – benefits embeddings trained on smaller corpora to gauge usefulness for low-labeling-resource domains like biology or medicine. We demonstrate the method's application for Single-task, Multi-task, small and large corpus-size setting in the evaluation section 3. Finally, MORTY (sec. 2), uses very little resources[2], especially regarding recent approaches that exploit unsupervised pretaining to boost end-task performance by adding complex pretraining components like ELMo, BERT (Peters et al., 2018; Devlin et al., 2018) which may not yet be broadly usable due to their hardware and processing time requirements. As a result, we demonstrate a simple method, that allows further pretraining exploitation, while requiring minimum extra effort, time and compute resources.

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

[2] $<$ 1GB memory and computes fast on GPU and CPU.

2 MORTY embedding

We unsupervisedly create specialized inputs for supervised end-tasks using **M**ultiple **O**rdinary **R**econstructing **T**ransformations to **Y**mprove embedding[3] of the original inputs. Specifically, as seen in the Figure 1, MORTY uses multiple, separate Autoencoders that create new representations by learning to reconstruct the original pretrained embeddings[4]. The resulting representations (post-processed embeddings) can provide both: (a) better performance for a *single* supervised probing task (ST), and (b) boost performance of *multiple tasks* (MT) or overall performance across all probing tasks. To pick an optimal MORTY embedding for single and Multi-task settings, we can either use proxy-tasks or an end-task(s)'s development split(s). In practice, MORTY can be efficiently trained as a (data) hyperparameter to the end or proxy tasks – see details in section 3.

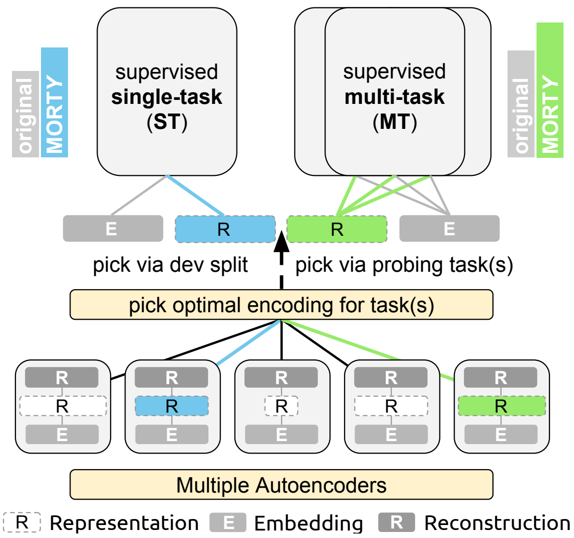

Figure 1: **MORTY steps:** From bottom to top: (1) compute multiple representations of the same embeddings, then (2) pick the best representation for the end-task(s) via its development split(s) or probing tasks to (3) push relative performance (colored, MORTY bar).

Embeddings by corpus sizes: We train 100 dimensional embeddings with Fasttext (Bojanowski et al., 2016)[5] and GloVe (Pennington et al., 2014)[6] on `wikitext-2` and `wikitext-103` created by Merity et al. (2016). By also using public Fasttext and GloVe[7] embeddings we can evaluate MORTY for small to very larger corpora. Both embedding methods are trained *five times each*[8] for the two smaller corpora to be able to capture minor variations[9] in *overall performance* – i.e. performance Σ when summing the scores of all probing tasks. Training Fasttext and GloVe on `wikitext-2` gives a vocabulary of 25249 and 33237 tokens respectively. On `wikitext-103` we get 197256 and 267633 tokens, while the original embeddings have 1999995 and 2196008 tokens. **Embedding task-semantics:** To evaluate MORTY on a variety of end-tasks we use a publicly available word embedding benchmark developed by Jastrzebski et al. (2017). It is split into three semantic categories: (a) word similarity (6 tasks), (b) word analogy (3 tasks), and (c) word and sentence classification/ categorization (9 tasks).

3 Evaluation

In the following we evaluate embedding performance scores for Fasttext and GloVe and their percentual change after postprocessing with MORTY. We evaluate MORTY for *Single-task* (ST) and *Multi-task* (MT) application optimization. Results can be seen in Tables 1 and 2. For the *Single-task* application setting we show MORTY's percentual performance impact in the `ST % change` column, where results are produced by choosing the best MORTY embedding per individual task – 18 MORTYs. For the *Multi-task* application setting the `MT % change` column shows percentual performance impact when choosing the MORTY with the best over-all-tasks score Σ. Finally, we evaluate by smaller (wikitext `2M`, `103M`) and very large (`600B/840B`) *training corpus sizes*, as well as by the *three semantic property categories* described in section 2. Model performances, given in Tables 1 and 2, are 5-run averages of Fasttext and GloVe per corpus sizes 2M and 103M, while the public 600B/ 840B were evaluated once. MORTY's performances on 2M and 103M are given as relative, percentual change, averaged over 5 according base-embedder runs.

3.1 Fasttext and GloVe baselines:

For Fasttext and GloVe – run 5 times on 2M and 103M – we can see in each table's left column that

[3] Y since labels/outputs (embeddings) are reconstructed
[4] Link to code will be made public after publication.
[5] To train Fasttext we used `https://fasttext.cc`
[6] To train GloVe we used the python glove_python wheel
[7] glove.840B.300d.zip, crawl-300d-2M.vec.zip
[8] Fasttext was trained using the implementation's (`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.
[9] $< 0.5\%$ between runs for both Fasttext and GloVe .

model	Fasttext			MT % change			ST % change		
corpus	2M	103M	600B	2M	103M	600B	2M	103M	600B
AP	0.31	0.59	0.68	-6.1	-0.9	-1.5	8.2	5.2	4.0
BLESS	0.3	0.73	0.84	-2.2	3.8	-3.0	13.0	9.7	5.4
Battig	0.14	0.32	0.48	-3.6	0.1	-3.7	7.0	4.0	0.5
ESSLI 1a	0.48	0.76	0.77	2.2	4.3	17.6	27.5	10.2	17.6
ESSLI 2b	0.63	0.75	0.78	9.2	2.7	0.0	26.5	11.3	12.9
ESSLI 2c	0.54	0.54	0.62	-3.7	10.7	-10.7	11.0	19.7	10.7
Google	0.06	0.04	0.12	33.6	293.8	187.3	45.3	319.3	217.2
SEval 12	0.11	0.16	0.24	1.6	4.3	-2.8	18.1	14.1	4.8
MSR	0.28	0.08	0.18	18.8	246.2	117.1	27.5	267.3	137
MTurk	0.24	0.52	0.73	65.6	5.1	1.1	98.0	12.6	1.5
RG65	0.29	0.71	0.86	65.2	0.7	2.1	104.7	5.3	5.6
RW	0.21	0.38	0.59	-17.1	-0.8	-2.0	4.1	2.4	0.9
MEN	0.36	0.71	0.84	13.0	0.4	-0.4	22.0	2.3	0.3
SLex999	0.18	0.31	0.50	-23.2	3.7	-1.2	7.3	9.0	3.1
TR9856	0.10	0.13	0.18	2.8	-4.1	-37.1	20.5	17.3	-2.5
WS353	0.46	0.69	0.79	3.9	1.0	-1.7	10.0	2.9	0.6
WS353R	0.35	0.63	0.74	16.4	1.7	-2.8	24.3	4.1	1.6
WS353S	0.52	0.77	0.84	3.2	0.4	0.6	13.3	3.0	1.9
Σ	5.55	8.83	10.79	8.9	5.8	3.4	–	–	–
category	2.39	3.70	4.17	-2.1	-0.2	1.8	11.4	4.5	3.1
analogy	0.45	0.28	0.55	15.5	115	72.2	24.6	125.2	92.7
similarity	2.71	4.85	6.07	6.2	-0.6	-4.7	17.3	2.2	-0.3

Table 1: **MORTY on Fasttext**: Above are probing task scores for: 18 individual tasks(AP-WS353S), the sum of individual scores Σ, and scores grouped by captured semantics: similarity (AP-ESSLI2c), analogy (Google-MSR), classification (MTurk-WS253S). **Left column:** shows absolute scores of the original embedder. **Middle column:** shows perceptual score change after applying the MORTY embedder with the *best overall score* Σ – i.e. one MORTY embedding trained for application to arbitrary tasks (`Multi-task`). **Right column:** shows perceptual score changes after postprocessing by a best MORTY embedding *per individual task* – i.e. 18 MORTYs optimal for a `Single-task` each. Each column is also split by corpus size – 2M (2 million tokens) for `wikitext-2`, 103M for `wikitext-103` and 600B/840B for the public embedding corpora size. 2M and 103M are averages of 5 runs – or respective MORTY changes.

results for classification (category), similarity and analogy improve expectedly with corpora size.

MORTY for Multi-task application: When looking at the middle columns (`MT % change`) we see that using a single best MORTY improves overall performance Σ[10] – the sum of 18 tasks – by roughly $2-9\%$ compared to base embeddings, especially for smaller corpus sizes. While Fasttext benefits more than GloVe from MORTY, both perform particularly badly for *analogy tasks* on the smaller corpora 2M and 103M where Fasttext beats GloVe, especially after applying MORTY. This is also reflected in the small/medium set Google and MSR analogy scores doubling and tripling (still middle column). However, public GloVe (840B) has the

[10]Note that, percentual change Σ (middle, right column) is not the average of the individual task changes, but the percentual change of the sum of the 18 individual scores.

model	Glove			MT % change			ST % change		
corpus	2M	103M	840B	2M	103M	840B	2M	103M	840B
AP	0.2	0.43	0.61	2.7	5.6	9.3	13.2	9.2	12.2
BLESS	0.27	0.51	0.85	1.6	-1.6	-1.8	7.9	7.9	4.7
Battig	0.1	0.19	0.46	3.5	2.0	1.9	7.4	5.4	8.5
ESSLI 1a	0.46	0.63	0.75	0.0	3.1	9.1	8.0	8.9	12.1
ESSLI 2b	0.51	0.74	0.75	19.9	-0.5	6.7	23.7	11.7	16.7
ESSLI 2c	0.46	0.54	0.62	2.1	2.7	0.0	16.9	16.7	10.7
Google	0.00	0.05	0.58	42.7	13.8	2.8	60.4	18.6	5.9
SEval 12	0.11	0.15	0.20	6.5	2.2	1.0	11.4	5.0	2.4
MSR	0.00	0.09	0.57	45.6	30.9	-2.4	100.7	38.1	10.1
MTurk	0.30	0.46	0.69	-22.4	2.6	0.5	1.6	4.2	2.6
RG65	0.15	0.44	0.77	11.6	3.9	-1.3	30.8	10.0	4.0
RW	0.20	0.21	0.46	-2.1	11.8	2.0	4.0	19.8	10.3
MEN	0.16	0.51	0.80	3.6	5.6	0.5	15.1	7.0	7.7
SLex999	0.03	0.22	0.41	147.8	7.3	3.1	228.3	11.7	9.3
TR9856	0.09	0.08	0.10	13.9	8.9	-4.7	19.8	47.3	36.7
WS353	0.16	0.45	0.74	31.5	7.2	0.7	36.8	8.2	5.6
WS353R	0.08	0.40	0.69	53.1	6.5	1.1	62.0	8.2	2.7
WS353S	0.27	0.58	0.80	15.1	6.5	0.3	20.2	7.6	5.9
Σ	3.56	6.68	10.84	7.8	4.3	1.9	–	–	–
category	2.00	3.04	4.05	3.5	-0.8	2.4	7.3	3.3	5.5
analogy	0.11	0.29	1.34	7.4	4.2	1.3	12.3	15.8	6.5
similarity	1.45	3.35	5.45	9.2	1.8	0.0	11.0	6.3	2.9

Table 2: **MORTY on Glove**: Same as in Table 1 but for GloVe .

best analogy performance, while MORTY further improves analogy scores for both public embeddings – 600B/840B. Additionally, for *similarity* we see decent improvements for the smallest corpus, but not for larger corpora as base Fasttext already has higher performance. Classification exhibits more mixed, smaller, changes. For smaller datasets Fasttext clearly beats GloVe in overall performance (8.83 vs. 6.68). For public embeddings (600B/840B) base scores are equal. GloVe leads analogy. Fasttext leads similarity and improves more from MORTY. However, despite GloVe's significantly lower base performance on smaller datasets, MORTY used on GloVe produces *lower* but more stable improvements for the MT setting (middle column). Generally, we see both performance increases and drops for individual task, especially on the smaller datasets and for Fasttext, indicating that, an overall best MORTY specializes the original Fasttext embedding to better fit a specific subset of the 18 tasks, while still being able to beat base embeddings in overall (Σ) score.

MORTY for Single-task application: When looking at the `ST % change` columns in both tables we see Single-task (ST) results for 18 individually best MORTY embeddings. Both Fasttext and GloVe show consistent improvements from using MORTY, with Fasttext exhibiting more improvement potential on smaller datasets, while GloVe shows more ST improvement on very large datasets, indicating that MORTY benefits both embedding methods. Particularly when base scores for a task were low – e.g. for the analogy tasks – MORTY often improved upon the particular base-embedding's weaknesses.

Low-resource benefits: MORTY seems especially beneficial on the smaller corpora (2M and 103M) for both MT and ST applications as well as for Fasttext and GloVe – indicating that MORTY is well suited for low-resource settings.

MORTY training: Finally, we found optimal parameters for training MORTY to be close to or the same as the original embedding model – i.e. same learning rate, embedding size and epochs. Though we initially experimented with variations such as sparse and denoising, or sigmoid and ReLu activations, we found linear activation, (over)complete Autoencoders trained with bMSE (batch-wise mean squared error) to perform best. In settings, where no supervised, or proxy dataset(s) are available to select the best MORTY embedding we found a practical setting for Fasttext and GloVe that consistently increased overall probing-task performance by simply training with a learning rate $lr = 0.01$[11], for 1 epoch, and a representation size equal to or twice as large as the original embedding – i.e. train an (over)complete representation. When compressing from the original embedding size, e.g. from 100 to 20, space reduction outweighed performance loss – so larger vocabularies are usable at sublinear performance loss[12]. More involved parameter exploration yielded little extra gains.

4 Related Work

Methods of information transfer from or to supervised tasks has been heavily focused in recent Transfer Learning literature, while transfer between unsupervised tasks received less attention. **Unsupervised-to-Supervised:** For word meaning transfer, Word2Vec (Mikolov et al., 2013), Fastext (Bojanowski et al., 2016; Pennington et al., 2014) and GloVe (Pennington et al., 2014) provide unsupervisedly pretrained embeddings that can be used to *generally* improve performance on *arbitrary* supervised end-tasks. **Supervised-to-unsupervised:** However, transfer can also be used vice versa, to (learn to) specialize embeddings to better fit a specific supervised signal (Ye et al., 2018; Ruder and Plank, 2017) or even to enforce that generally relevant semantics are encoded by using auxiliary Multi-task supervision (Kiela et al., 2018b; Faruqui et al., 2015). The approach by Ruder and Plank (2017) is especially interesting since they proposed an *automated method* (Bayesian optimization) for tuning embeddings to a specific end-task. **Supervised-to-supervised:** Another way to realize knowledge transfer is between supervised tasks, that can be exploited successively (Kirkpatrick et al., 2017), jointly (Kiela et al., 2018b) and in joint-succession (Hashimoto et al., 2017) to improve each others performance. **Unsupervised-to-unsupervised:** More recently, Dingwall and Potts (2018) proposed a GloVe modification that retrofits publicly available (external) GloVe embeddings to produce better domain embeddings for a specific end-task.

In contrast, MORTY does not require external (public) embeddings, does not require target domain texts[13], can be applied to embeddings produced by *any* embedding method, and can be used with or without direct supervision by a desired (set of) end-tasks – resulting in low-effort usage. MORTY instead uses unsupervised fine-tuning of embeddings to better fit *one or more desired supervised semantics*. This way, we can avoid manual extensions like complex multitask learning setups or creating potentially hard to come by task-related supervised data sets. Instead MORTY can be optimized as a data-input parameter for a desired (set of) end-tasks or proxy-tasks (proxy-semantics), and shows additional benefits in low-resource settings.

5 Conclusion

We demonstrated a low-effort method to *unsupervisedly construct task-optimized* word embeddings from existing ones to gain performance on a (set of) supervised end-task(s). Despite its simplicity, MORTY is able to produces significant performance improvements for *Single* and *Multi-task* supervision settings as well as for a variety of desirable word encoding properties – even on smaller corpus sizes – while forgoing additional labeling or building more complex model architectures.

[11] This lr is roughly the lr of Fasttext or GloVe times the number of original epochs, thereby increasing the lr to be suitable for training only 1 epoch.

[12] This is an expected, well explored, property of undercomplete encoding, that did not yield interesting insights.

[13] Though MORTY can be applied arbitrary public and domain trained embeddings.

References

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

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.

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

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

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.

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.

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.