Embedding Structured Dictionary Entries

Steven R. Wilson[1], **Walid Magdy**[1,2], **Barbara McGillivray**[2,3], **and Gareth Tyson**[2,4]
[1]The University of Edinburgh, Edinburgh, UK
[2]The Alan Turing Institute, London, UK
[3]University of Cambridge, Cambridge, UK
[4]Queen Mary University of London, London, UK
steven.wilson@ed.ac.uk, wmagdy@inf.ed.ac.uk
bmcgillivray@turing.ac.uk, g.tyson@qmul.ac.uk

Abstract

Previous work has shown how to effectively use external resources such as dictionaries to improve English-language word embeddings, either by manipulating the training process or by applying post-hoc adjustments to the embedding space. We experiment with a multi-task learning approach for explicitly incorporating the structured elements of dictionary entries, such as user-assigned tags and usage examples, when learning embeddings for dictionary headwords. Our work generalizes several existing models for learning word embeddings from dictionaries. However, we find that the most effective representations overall are learned by simply training with a skip-gram objective over the concatenated text of all entries in the dictionary, giving no particular focus to the structure of the entries.

1 Introduction

While word embedding models are typically trained using large text corpora with objectives based on distributional semantics, recent work has shown how to take advantage of external resources like WordNet (Miller, 1995) and other manually created dictionaries in order to better capture word-level semantic relationships of interest. For example, previous work has used the graph structure of external resources to post-process pre-trained word embeddings, enforcing that the similarity between embeddings reflects the similarity inferred from the graph structure of lexicons like WordNet (Faruqui et al., 2015). Following in a similar principle, others use known synonymy and antonymy relationships between words to adjust the distance between word embeddings (Mrkšić et al., 2016). Other work uses traditional dictionaries to improve the overall coverage of word embedding models by creating embeddings for rare words be leveraging information from their definitions (Bahdanau et al., 2017).

While dictionaries have been shown to be useful, most previous work has focused only on using the text of the definitions in order to learn word representations. However, many dictionaries include additional structural elements such as usage examples, quotations containing the headword, tags, labels, and more. For some online crowd-built dictionaries, information such as the contributing users and even upvotes and downvotes are available.

We conjecture that such meta information may prove useful and, therefore, we seek to leverage all of this additional information to *build improved representations of the words defined in a given dictionary*. To do this, we generalize the Consistency-Penalized Autoencoder (CPAE) (Bosc and Vincent, 2018) to allow for not only the reconstruction of dictionary definitions, but also for making predictions about the other structural elements available, such as usage examples and user-assigned tags.

We make the following contributions in this paper: (1) we propose a flexible, multi-task learning extension to the CPAE model that can be used to produce embeddings from structured dictionary entries, (2) we evaluate the applicability of this extended model to three English-language dictionary datasets, each with their own unique characteristics and sets of structural elements, and (3) we demonstrate the a simple baseline approach for learning word embeddings, based on the popular skip-gram with negative sampling framework, can often lead to representations that better capture word-level semantic similarity according to a range of commonly used evaluation tasks.

2 Data & Baseline

2.1 Structured Dictionary Data

We consider three manually constructed, machine-readable, English-language dictionaries: English

Proceedings of the First Workshop on Insights from Negative Results in NLP, pages 117–125

ISBN 978-1-952148-66-8

Wordnet[1] (Miller, 1995), English Wiktionary[2], and Urban Dictionary (UD)[3], each containing definitions for each word in addition to one or more structural elements such as usage examples, tags, or votes (Table 1). We find that many of the terms that are defined in Urban Dictionary are not commonly used in everyday language, and so we choose to further filter the set of headwords from Urban Dictionary to those that have been used at least 10,000 times in a sample of tweets sampled over a five-year period as identified in (Wilson et al., 2020b).

2.2 Baseline Approach

To provide a simple baseline for later evaluation, we train word embeddings using the entire text of each dictionary, including all structured elements, by treating each structural element as a short document and prepending the entry headword to each. We use a standard skip-gram model with negative sampling (SGNS), trained using the FastText library (Mikolov et al., 2018).

3 Auto-encoding Structured Entries with Multi-task Learning

Next, we present an approach for learning word embeddings that implicitly encode a wide range of the elements that are present in a dictionary entry. Given a word defined in a dictionary, the objective of the model is to accurately recover as much structural information as possible, including the word's definition, usage examples, tags, and authors. We also leverage user provided votes as a means of sorting and filtering the dictionary entries. The model takes a word's definition as input, and learns a transformation from the words in the definition to an embedding that contains features that describe the structural elements of the dictionary entry for the word. We treat the prediction of each type of structural element as a separate task within a multi-task learning framework.

3.1 Model Architecture

Our model (Figure 1; a more formal, detailed description of the model is given in Appendix A) can be seen as a generalization of several others: a simple auto-encoder, Hill's model (Hill et al., 2016), and the consistency penalized auto-encoder (CPAE). In each case, the input for the model is a definition[4] for the target headword, w_h. The input tokens are converted into a sequence of embeddings using a learnable word embedding layer, and these embeddings are passed to the definition encoder, which produces a single embedding, h, which is used as the representation for w_h.

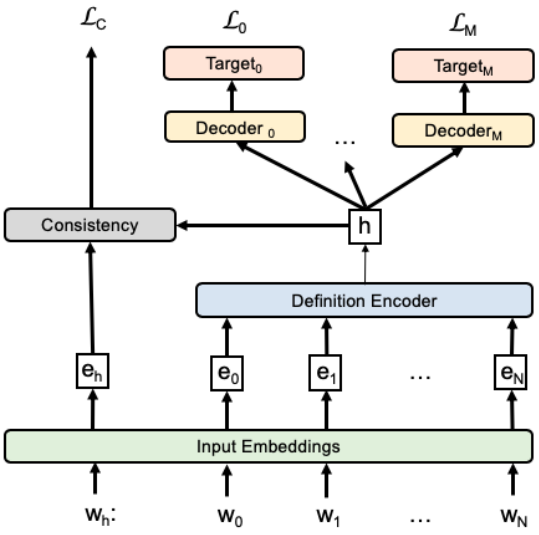

Figure 1: Model architecture for multi-task learning autoencoder for embedding words from their structured dictionary entries. Input tokens are embedding using the Input Embeddings layer, and the n tokens in the definition of headword w_h are passed to the Definition Encoder to produce the definition embedding h. This embedding should be consistent (low distance) with the embedding of the definition headword e_h. M possible output tasks can be used, each with its own decoder which needs to reconstruct the Target.

This embedding is then fed to any number of decoders, each with their own specific objective and loss function (details in the subsections of Appendix A). The goal of each decoder's loss is to influence the weights of the encoder to produce an embedding h that is most useful for capturing a specific structural element of the dictionary entry for w_h, or to retain some other important property of the embedding h. The decoders that we use and their associated losses become components in the overall loss function for our model: $\mathcal{L} = \lambda_0\mathcal{L}_0 + \lambda_1\mathcal{L}_1 \ldots + \lambda_n\mathcal{L}_m$ for up to m objectives, each with its own associated weight term. These weights can be used to control the overall influence of the objective in the final loss computation.

[1]To make our results directly comparable with (Bosc and Vincent, 2018), we use the filtered version of WordNet included at: https://github.com/tombosc/cpae

[2]https://en.wiktionary.org/

[3]https://www.urbandictionary.com/

[4]Or, in the case of polysemous words, the concatenation of all tokens in all definitions, separated by a SEP token

	defs	examples	tags	votes	headwords	definitions	tokens
Wordnet	✓	✓			83K	159K	2.4M
Wiktionary	✓	✓	✓		214K	380K	4.6M
Urban Dictionary	✓	✓	✓	✓	2M	3.5M	195M
UD (Filtered)	✓	✓	✓	✓	22K	104K	59M

Table 1: Structural elements present in three machine-readable dictionaries, and number of headwords, definitions, and total tokens present in each. UD (Filtered) is the filtered version of Urban Dictionary which doesn't contain words that are not commonly used or definitions for which the difference between the number of upvotes and downvotes is negative. This is the version of Urban Dictionary that is used when training our proposed model.

The target of each decoder is dependent on the structural element that it is meant to encoder. For the definitions, the goal of the decoder is to reproduce the definition itself (making the use of this task alone equivalent to a simple autoencoder). For the usage examples and tags, the target task is the predict the context in which the headword appears using a skip-gram learning objective. We also experiment with using the user-provided votes to filter and sort the data, as well as to provide weights for the input definitions.

An additional loss term can be used in order to enforce the consistency between the learned embedding h and the input embedding for the headword e_h. This is similar to the main objective of Hill's model (Hill et al., 2016) and is the consistency penalty that is used in the CPAE model (Bosc and Vincent, 2018). This forces the model to produce embeddings for headwords that are consistent to the embeddings produced for the same words when they appear in the definitions of *other* headwords.

4 Evaluation and Results

We evaluate all produced embeddings[5] across a range of intrinsic evaluation tasks as used in (Jastrzebski et al., 2017).[6] For these word-level semantic similarity tasks, the machine generated scores (cosine similarity between the produced word embeddings) are compared against human-labeled similarity scores by computing the correlation between the two sets of scores.

The tasks involved include the Marco, Elia and Nam (MEN) annotated word pairs based on image captioning data (Bruni et al., 2014), the SimVerb (SV) verb similarity dataset (Gerz et al., 2016), both of which have standardized development and testing splits. We use the development splits of these datasets in order to tune our models. The WordSim-353 (WS) dataset contains both similarity (WS-S) and relatedness (WS-R) annotations for the same sets of words, allowing us to examine the ability of our models to capture each of these semantic relations. We also evaluate using the SimLex-999 dataset and a subset of that data, SimLex-333 (SL999 and SL333) (Hill et al., 2015). The SL333 subset contains only the 333 most related pairs according to the human annotations. Stanford's Contextual Word Similarities (SCWS) dataset (Huang et al., 2012), the 65 word pairs studied by Rubenstein and Goodenough (RG65) 1965, the Mechanical Turk (MT) dataset (Radinsky et al., 2011), and the Rare Words (RW) dataset (Luong et al., 2014) round out the rest of our evaluatoin tasks.

For models that use our proposed architecture, we initialize the input embeddings using the baseline pre-trained skip-gram embeddings. We train these embeddings ourselves in the case of WordNet and Wikitionary, and use the `ud-basic` embeddings released by (Wilson et al., 2020a) for Urban Dictionary.[7] Table 2 shows the similarity and relatedness scores achieved when using various combinations of objectives in our model.[8]

We observe that for WordNet, the simple SGNS embeddings are always outperformed by the other approaches, which is in line with the results reported in (Bosc and Vincent, 2018) where the CPAE-P model was found to achieve the best results when using WordNet. We can see that adding structure, which, for the case of WordNet, only includes usage examples, leads to an improvement over the base CPAE-P model in many cases. The overall trend is similar for the Wiktionary data, yet we see a stronger performance from the SGNS baseline. In fact, SGNS achieves the best results for

[5] Details of the experimental setup are in Appendix C.

[6] We used code from the *web* package, located at: https://github.com/kudkudak/word-embeddings-benchmarks to run the intrinsic evaluation tasks.

[7] These embeddings were trained on the entirety of Urban Dictionary rather than just the subset that we use in this study.

[8] Only best performing models are shown; the full set of results can be found in Appendix B.

		dev		*test*									
	Model	MEN	SV	MEN	WS-R	WS-S	SL999	SL333	SV	SCWS	RG65	MT	RW
WordNet	SGNS	58.6	34.7	56.2	45.6	62.9	35.0	20.4	34.4	54.0	63.1	52.8	23.2
	Hill's Model	61.1	45.6	59.9	42.3	59.5	43.8	**35.8**	44.2	59.0	73.6	56.0	30.3
	CPAE-P	**68.3**	49.4	67.3	50.6	66.4	**47.4**	34.1	45.3	**61.5**	76.7	61.4	**31.5**
	+ Structure	68.0	**52.0**	**67.8**	**54.4**	**67.6**	45.8	33.5	**47.4**	61.3	**76.8**	**61.9**	28.5
Wikt.	SGNS	65.1	**43.2**	65.0	56.5	68.7	**42.1**	22.4	38.0	56.6	72.8	**63.3**	25.1
	Hill's Model	63.8	33.5	66.3	53.8	70.6	37.6	**27.4**	33.2	**58.7**	80.7	57.9	**30.3**
	CPAE-P	65.1	33.4	64.1	60.4	**73.0**	35.5	18.4	33.1	56.9	**87.7**	58.2	21.4
	+ Structure	**65.2**	38.0	**67.0**	**61.4**	72.8	39.5	21.6	**38.2**	57.3	85.6	60.4	25.4
UD	SGNS	**79.1**	42.1	**78.0**	**65.7**	**74.2**	**47.9**	30.2	**35.5**	**62.5**	**89.3**	**74.3**	**39.2**
	Hill's Model	72.5	38.9	70.0	61.4	69.6	44.3	32.8	29.5	60.1	75.8	68.4	34.2
	CPAE-P	71.4	38.4	68.5	61.3	69.6	44.7	29.2	30.0	59.2	71.1	65.1	32.7
	+ Structure	74.6	**42.6**	72.3	64.1	71.1	47.8	**33.9**	30.4	60.4	78.9	69.2	34.9

Table 2: Correlation (Spearman's ρ) with gold standard similarity and relatedness scores for development and evaluation datasets. Hill's model (Hill et al., 2016) is the structured dictionary encoder with only the consistency penalty, CPAE-P is the Consistency Penalized Autoencoder (Bosc and Vincent, 2018) with pre-trained word embedding targets, and the version with Structure is our proposed extension to the model, making use of additional training objectives based on any available structural elements. SGNS is the skip-gram with negative sampling baseline word embedding model. **Bold** indicates the best result for a given dictionary, **underlined** numbers are also the overall best.

two of the test datasets and achieves competitive results across the board, making it a viable alternative to the more complex dictionary auto-encoding approaches. Finally, for the Urban Dictionary data, we see the baseline SGNS approach overtaking the other methods in almost every evaluation set, also leading to many of the best overall scores found in this study. This shift in performance may be related to the overall size of each dataset: Urban Dictionary dataset contains approximately 200 million total tokens, compared to the 1.7 million in WordNet and 4.6 million in English Wiktionary. Further, as Urban Dictionary's definitions contain a mixture of noisy submissions, jokes, and opinions, they are likely to be less closely tied to the true meanings of the headwords (Nguyen et al., 2018). This could make the auto-encoding objective less useful overall in comparison to learning representations of the words simply based on their usage contexts.

5 Conclusions

We show that the extension of the CPAE model to include additional structural elements can provide some gains in word-level semantic similarity tasks, however, the the extra complexity of this approach is unnecessary for learning useful word embeddings, and in many cases, leads to degradation in the scores across a range of standard word embedding evaluation metrics in comparison to simpler approaches. To build general purpose word embeddings from a sufficiently large dictionary (i.e., containing at least several hundred million tokens of text), our recommendation is to simply concatenate all of the structural elements together as a single text, inserting the entry headword between each element, and applying the widely popular skip-gram architecture to this text to learn traditional distribution embeddings. This approach requires only a single learning objective, trains in much less time, and achieves competitive results in many cases, making it an easier alternative to explicitly leveraging structural information from dictionary entries while still creating useful embeddings.

Future work should explore how these approaches would work when applied to more English dictionaries such as the Oxford English Dictionary[9] in order to better understand the effects of using a more standardized dictionary to learn embeddings. Further, dictionaries in other languages, particularly lower-resource languages, should be considered, since our results suggest that the approaches described in this paper outperform the baseline approach mostly in settings where the total amount of text in the dictionary is small.

Acknowledgments

This work was supported by The Alan Turing Institute under the EPSRC grants EP/N510129/1, and EP/S033564/1. We also acknowledge support via EP/T001569/1.

[9] https://www.oed.com/

References

Dzmitry Bahdanau, Tom Bosc, Stanisław Jastrzebski, Edward Grefenstette, Pascal Vincent, and Yoshua Bengio. 2017. Learning to compute word embeddings on the fly. *arXiv preprint arXiv:1706.00286*.

Tom Bosc and Pascal Vincent. 2018. Auto-encoding dictionary definitions into consistent word embeddings. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1522–1532.

Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. 2012. Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 136–145.

Elia Bruni, Nam-Khanh Tran, and Marco Baroni. 2014. Multimodal distributional semantics. *Journal of Artificial Intelligence Research*, 49:1–47.

Laura Burdick, Jonathan K Kummerfeld, and Rada Mihalcea. 2018. Factors influencing the surprising instability of word embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2092–2102.

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.

Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. 2016. Simverb-3500: A large-scale evaluation set of verb similarity. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2173–2182.

Felix Hill, KyungHyun Cho, Anna Korhonen, and Yoshua Bengio. 2016. Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics*, 4:17–30.

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 873–882.

Stanisław Jastrzebski, Damian Leśniak, and Wojciech Marian Czarnecki. 2017. How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks. *arXiv preprint arXiv:1702.02170*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. 2014. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*.

Tomáš Mikolov, Édouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

N Mrkšić, D Séaghdha, B Thomson, M Gašić, L Rojas-Barahona, PH Su, D Vandyke, TH Wen, and S Young. 2016. Counter-fitting word vectors to linguistic constraints. In *2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2016-Proceedings of the Conference*, pages 142–148.

Dong Nguyen, Barbara McGillivray, and Taha Yasseri. 2018. Emo, love and god: making sense of urban dictionary, a crowd-sourced online dictionary. *Royal Society open science*, 5(5):172320.

Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. 2011. A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th international conference on World wide web*, pages 337–346.

Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

Steven R. Wilson, Walid Magdy, Barbara McGillivray, Kiran Garimella, and Gareth Tyson. 2020a. Urban dictionary embeddings for slang nlp applications. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 4764–4773.

Steven R. Wilson, Walid Magdy, Barbara McGillivray, and Gareth Tyson. 2020b. Analyzing temporal relationships between trending terms on twitter and urban dictionary activity. In *12th ACM Conference on Web Science*, WebSci '20, page 155–163, New York, NY, USA. Association for Computing Machinery.

Appendix

A Detailed Model Description

Formally, let $\mathcal{D} = \{w_{in}^0, w_{in}^1, \ldots, w_{in}^n\}$ be a sequence of tokens in the definition for w_h. The elements of $\mathcal{D}$ belong to the vocabulary of all words that appear in definitions, $\mathcal{V}_{in}$, and w_h belongs to the vocabulary of all headwords, $\mathcal{V}_h$. In the case of polysemous words which have more than one meaning, we concatenate the tokens from all definitions together into a single sequence, and separate them by a special `SEP` token.

Given the full sequence of input words, $E_{in}(\mathcal{D}) = \{e_{in}^0, e_{in}^1, \ldots, e_{in}^n\}$ is the set of d_{in}-dimensional embeddings representing words in the definition. These embeddings can be learned during training, or pre-initialized and frozen, as discussed later in this section. The embeddings are passed into an encoder layer in order to produce a single d_h-dimensional embedding $h = \text{enc}(E_{in}(\mathcal{D}))$. The encoder can be any type of model that takes a variable-length sequence of embeddings as input and produces a single, fixed-length embedding as output.

This embedding is then fed to any number of decoders, each with their own specific objective and loss function. The goal of each decoder's loss is to influence the weights of the encoder to produce an embedding h that is most useful for capturing a specific structural element of the dictionary entry for w_h, or to retain some other important property of the embedding h. In the following subsections, we describe the decoders that we use and their associated loss, which become components in the overall loss function for our model:

$$\mathcal{L} = \lambda_0\mathcal{L}_0 + \lambda_1\mathcal{L}_1 \ldots + \lambda_n\mathcal{L}_n$$

for up to n objectives, each with its own associated weight term. These weights can be used to control the overall influence of the objective in the final loss computation.

A.1 Definitions as reconstruction targets

The words in a well-formed definition should provide a precise encapsulation of one of the meanings of the headword being defined. So, we expect that a combination of the meanings of the words in the definition should provide a reasonable approximation for the meaning of the word itself. Since the input to our encoder is the set of embeddings of the definition words, a decoder objective based on the intermediate representation, h, will lead to a simple auto-encoder for the definition itself.

The definition decoder with learned parameters θ produces a set of predictions of the words belonging to the original definition $\hat{\mathcal{D}} = \text{dec}_\theta(h)$, and this decoder is used to compute the definition reconstruction loss $\mathcal{L}_R$. We use a simple conditional unigram language modeling loss as our reconstruction loss

$$\mathcal{L}_\mathcal{R} = -\log p(\mathcal{D}|\theta) = \sum_{w \in \mathcal{D}} -\log p(w|\theta)$$

where $p(w|\theta)$ is determined by the decoder dec_θ. For the decoder, the auto-encoder model uses a single linear layer with input size d_h and output size $|\mathcal{V}_{def}|$, followed by a softmax operation, providing a probability $p(w)$ for all words in the output vocabulary $\mathcal{V}_{def}$. The output vocabulary $\mathcal{V}_{def}$ is equal to $\mathcal{V}_{in}$ for the traditional auto-encoder setting, since the objective is to reproduce the set of input words. However, in practice, we can speed up computation with minimal impact on performance by reducing $\mathcal{V}_{def}$ to only contain the m_{def} most common words, and treating all others as out-of-vocabulary. The out-of-vocabulary words are represented by a single token `UNK` which is ignored for the purposes of the loss computation. Including only this objective (which can be achieved by setting $\lambda_t = 0$ for every other task t) is equivalent to a simple definition auto-encoder: given the word in the headword's definition, produce an intermediate embedding h which can then be used to reconstruct the original set of words from the definition.

A.2 Usage examples and tags as context

While widely used distribution word embeddings rely on examples of words in context in order to learn representations of those words, hundreds of examples of usage of each word are usually required in order to build stable representations (Burdick et al., 2018). We experiment with using only the few prototypical examples that are provided in the dictionary definitions themselves as training samples for the term. This has several advantages: first, no data outside of the dictionary itself is needed to train the embeddings, and second, usage examples should, by nature, be written in a way that a specific meaning of the term is emphasized, providing a potentially stronger semantic signal than randomly sampled occurrences of a term in a text corpus. Usage contexts may help to capture

aspects of meaning that correspond to general semantic relatedness between words. Similarly, tags provide high-level category information related to words, and we expect that words with similar sets of tags will be related in meaning.

To incorporate this information into our model, we use a skip-gram language modeling objective similar to the one used by the word2vec model for learning word embeddings from word-in-context samples. That is, given the embedding for a word, h, we train a new feedforward output layer to predict the set of words that appear in the usage example context *around* the target word, or in the case of tags, the output layer should predict all tags. In the case of the usage examples, we replace the word and its morphological variations with a special `MASK` token so that the model does not learn to simply predict the word itself. Then, we define new vocabularies $\mathcal{V}_{use}$ and $\mathcal{V}_{tag}$ for all words that appear in usage examples in the dictionary and all tags, respectively, and we train linear layers to predict the set of usage words and tags given h. The loss $\mathcal{L}_{use}$ is then the cross-entropy between the predicted distribution over $\mathcal{V}_{use}$ and the equally sized vector of counts representing the number of times each word actually appeared in a usage example, and the same is done with the tag distribution to compute the tag prediction loss, $\mathcal{L}_{tag}$. As with the definition decoder, we allow for the size of the output vocabulary to be restricted to the most common m_{use}/m_{tag} words.

A.3 Consistency between embeddings

The consistency penalized auto-encoder model (CPAE) adapted an additional constraint, based on Hill's model (Hill et al., 2016), to minimize the distance between the input embedding $e_h = E_{in}(w_h)$ and the learned encoder embedding h. To achieve this, the Euclidean distance between the two embeddings is minimized as an additional component of the loss, the consistency penalty: $\mathcal{L}_C = (h - e_h)^2$ which can only be computed for for the set of words which are both *defined* (headwords) and used within definitions of other words, i.e., $\mathcal{V}_h \cap \mathcal{V}_{in}$. When setting $\lambda_t = 0$ for all other tasks t, we can approximately recover Hill's model (Hill et al., 2016). It was previously shown (Bosc and Vincent, 2018) that initializing the weights of the input embeddings E_{in} with pre-trained word embeddings, paired with this type of consistency constraint, can lead to improved performance on a number of word relatedness tasks (we label this setting as CPAE-P).

A.4 Votes as signals of importance

User-provided information can be used in several ways in our method. In our current setup, there may often be too many entries for a given headword to be able to adequately focus on all of them at once using our models which rely on a recurrent encoder for the concatenation of all tokens in all definitions. In Urban Dictionary, we can rely on the signal of user-provided votes, which are applied at the *entry*-level. This information can help sort the set of entries by importance: when training our concatenated lists of definitions, entries, and tags, we try sorting[10] them by their net number of votes (up-votes − down-votes) so that the top scoring entries will be processed by the model first, giving them priority over the other entries. We also remove any entries that received negative net votes from the concatenated list of entries. Empirically, we found that using the voting information in this way resulted in either a minor improvement or no change in the results, and so all results presented reflect the use of votes as signals of importance where votes are available.

B Additional Results

Table 3 shows the full set of results across all three dictionaries using the same evaluation tasks as before. AE/Autoencoder is the simple autoencoder model in which the loss term only consists of the definition reconstruction penalty. CPAE is the Consistency Penalized Autoencoder (Bosc and Vincent, 2018) which is the same as the AE model with the addition of the consistency penalty. Model names ending with "-P" use pre-trained embeddings (the same used for the SGNS baseline) to initialize the input embedding layer of the model. Hill's model (Hill et al., 2016) only uses the consistency penalty and always uses pre-trained embeddings to initialize the input embedding layer. SGNS is the skip-gram with negative sampling baseline, and "+Structure" is the same as the previous row, but using out multi-task learning framework to train the model to use the structural elements available in the dictionary. For the Urban Dictionary data, for models that use pre-trained embeddings to initialize the input layer, "Full" indicates that those embeddings

[10] We also explore using votes to weight the loss for each example, but find no significant differences in the results.

		dev		*test*									
	Model	MEN	SV	MEN	WS-R	WS-S	SL999	SL333	SV	SCWS	RG65	MT	RW
WordNet	Autoencoder	48.9	38.7	49.0	31.3	50.7	36.2	22.4	33.8	54.0	61.4	44.0	21.8
	+ Structure	45.0	42.7	45.7	28.6	41.4	32.9	22.3	38.7	50.1	65.8	40.1	21.9
	CPAE	51.1	41.6	48.9	34.5	47.8	39.7	29.0	37.2	53.2	60.0	40.5	23.4
	+ Structure	51.0	44.1	51.5	36.6	52.2	36.3	23.6	37.6	54.0	63.6	47.0	21.9
	AE-P	55.8	45.9	52.8	39.2	59.7	41.8	28.2	40.4	56.8	68.2	49.4	24.4
	+ Structure	55.8	45.3	55.9	39.9	62.1	42.2	27.2	43.5	57.3	66.8	50.8	23.8
	CPAE-P	**68.3**	49.4	67.3	50.6	66.4	**47.4**	34.1	45.3	**61.5**	76.7	61.4	**31.5**
	+ Structure	68.0	**52.0**	**67.8**	**54.4**	**67.6**	45.8	33.5	**47.4**	61.3	**76.8**	**61.9**	28.5
	Hill's Model	61.1	45.6	59.9	42.3	59.5	43.8	**35.8**	44.2	59.0	73.6	56.0	30.3
	SGNS	58.6	34.7	56.2	45.6	62.9	35.0	20.4	34.4	54.0	63.1	52.8	23.2
Wiktionary	Autoencoder	45.8	27.3	49.2	44.4	60.3	29.4	9.8	25.4	47.9	77.2	45.5	22.4
	+ Structure	43.3	14.1	45.8	35.5	62.3	20.9	-2.9	16.3	42.3	50.7	40.7	17.7
	CPAE	53.3	27.4	53.1	52.4	61.6	34.6	15.8	29.3	54.2	75.2	49.7	16.8
	+ Structure	38.6	18.3	46.1	32.3	53.3	20.2	-1.3	19.9	43.1	56.1	33.6	13.9
	AE-P	51.3	31.2	51.1	43.5	59.8	32.9	15.8	27.5	49.8	82.7	43.5	25.8
	+ Structure	54.0	31.2	54.0	49.2	64.0	33.6	20.3	27.3	49.3	78.1	48.6	24.2
	CPAE-P	65.1	33.4	64.1	60.4	**73.0**	35.5	18.4	33.1	56.9	**87.7**	58.2	21.4
	+ Structure	**65.2**	38.0	**67.0**	**61.4**	72.8	39.5	21.6	**38.2**	57.3	85.6	60.4	25.4
	Hill's Model	63.8	33.5	66.3	53.8	70.6	37.6	**27.4**	33.2	**58.7**	80.7	57.9	**30.3**
	SGNS	65.1	**43.2**	65.0	56.5	68.7	**42.1**	22.4	38.0	56.6	72.8	**63.3**	25.1
Urban Dictionary	Autoencoder	8.7	9.3	13.2	3.4	7.9	10.3	9.3	10.3	25.2	15.8	8.3	4.1
	+ Structure	14.1	11.7	16.0	8.3	20.7	11.2	11.0	8.0	25.2	32.8	10.5	6.2
	CPAE	1.7	7.4	5.9	-2.2	5.6	3.9	-0.1	4.5	23.4	-1.4	0.6	2.8
	+ Structure	20.0	15.8	22.9	17.8	17.4	10.1	9.8	5.5	27.3	21.3	13.1	3.8
	AE-P (Part)	20.9	7.5	16.1	6.7	18.6	10.0	3.6	5.5	31.8	2.5	12.2	7.1
	+ Structure	26.3	9.8	18.6	9.5	22.1	11.0	6.0	4.7	33.1	-3.5	7.0	7.9
	CPAE-P (Part)	54.9	26.1	51.7	42.4	55.6	31.8	23.1	20.4	48.1	47.4	46.9	11.8
	+ Structure	57.1	24.9	52.0	40.5	52.7	30.4	24.5	19.2	48.4	45.7	51.3	11.6
	Hill's (Part)	57.4	24.7	55.1	43.7	57.0	31.4	24.4	19.3	49.9	47.8	49.2	12.7
	SGNS (Part)	65.8	28.2	63.6	51.7	58.2	36.8	20.5	27.6	55.4	50.9	59.6	17.1
	AE-P (Full)	21.1	8.6	17.1	8.0	18.3	10.4	4.8	5.0	30.4	0.3	12.7	12.1
	+ Structure	25.4	9.4	18.4	12.2	20.8	12.0	12.4	6.6	32.2	8.2	11.4	9.3
	CPAE-P (Full)	71.4	38.4	68.5	61.3	69.6	44.7	29.2	30.0	59.2	71.1	65.1	32.7
	+ Structure	74.6	**42.6**	72.3	64.1	71.1	47.8	**33.9**	30.4	60.4	78.9	69.2	34.9
	Hill's (Full)	72.5	38.9	70.0	61.4	69.6	44.3	32.8	29.5	60.1	75.8	68.4	34.2
	SGNS (Full)	**79.1**	42.1	**78.0**	**65.7**	**74.2**	**47.9**	30.2	**35.5**	**62.5**	**89.3**	**74.3**	**39.2**

Table 3: Full similarity and relatedness evaluation results for each dictionary.

have been trained on the entirety of Urban Dictionary, while "Part" means that the embeddings were only trained on the filtered subset (only commonly used words, no words receiving negative total votes) of Urban Dictionary that was used to train the main dictionary embedding model.

The models presented in the Results section of the main paper are those that achieved the best result for at least one evaluation task for any dictionary dataset. However, from this full set of results, we can observe that the addition of structural elements through multi-task learning does lead to improvements in some cases, especially for the Urban Dictionary. This may be due to the fact that the definitions in Urban Dictionary are not always strictly providing direct meanings of the words and sometimes include jokes and opinions (Nguyen et al., 2018), so the usage examples and tags can be used to help provide more useful signals when training the model. This phenomenon can be seen even more closely from the very poor results achieved by the plain autoencoder and CPAE models on Urban Dictionary, which achieve no better than random results on some of the evaluation tasks, indicating that the signal from the definitions in Urban Dictionary is extremely noisy, even within the filtered subset of the dictionary.

C Experimental details

We train our models for a maximum of 150 epochs, implementing early-stopping using two of the intrinsic evaluation tasks which have readily available development sets: MEN (Bruni et al., 2012) and SimVerb-999 (Hill et al., 2015). When the model average performance on these two tasks does not increase for 10 epochs in a row, we stop training and save the embeddings produced by the model which achieved the maximum average score on these development tasks. We initialize our input embeddings with the baseline FastText embeddings trained on the concatenation of all structural dictionary elements treated as plain text.

For our definition encoder, we use a 300-dimensional, bidirectional GRU[11] layer followed by a single feedforward layer. We set the dimension d_h to match the size of whichever pre-trained embeddings we use with that model (usually 300) so that the consistency penalty can be properly computed. We limit the size of each output vocabulary to the most common 10,000 words, and we limit the size of the input vocabulary to the most common 50,000 words.

We use Adam (Kingma and Ba, 2014) as the optimizer with a learning rate of 3×10^{-4}. Following (Bosc and Vincent, 2018), we set the λ value for the reconstruction task to 1 and modify the other weights proportionally. Given the previously reported importance of the consistency penalty, we set this to 64. In order to focus our search on the possible combinations of objectives, we also leave the λ values for the tags and examples at 1.

[11] We also experimented with several other simple encoder types, including the LSTM that was used in (Bosc and Vincent, 2018), but found the bi-GRU to give consistently better or equal results with a smaller number of parameters.