Visual Tools for Debugging Neural Language Models

Xin Rong RONXIN@UMICH.EDU
School of Information, University of Michigan, 105 S State St., Ann Arbor, MI 48109 USA

Eytan Adar EADAR@UMICH.EDU
School of Information, University of Michigan, 105 S State St., Ann Arbor, MI 48109 USA

Abstract

While neural language models are powerful, they need just the right configuration of hyperparameters and a large amount of training data to perform well. When a model under development or training fails to give expected output, it is difficult for the user to debug, as the model is often a black box to the user. In this work, we discuss a set of visual tools designed to support the debugging process of neural language models. These are implemented in our LAnguage Model Visual Inspector (LAMVI) system, an interactive visual environment for exploring and debugging word embedding models.

1. Introduction

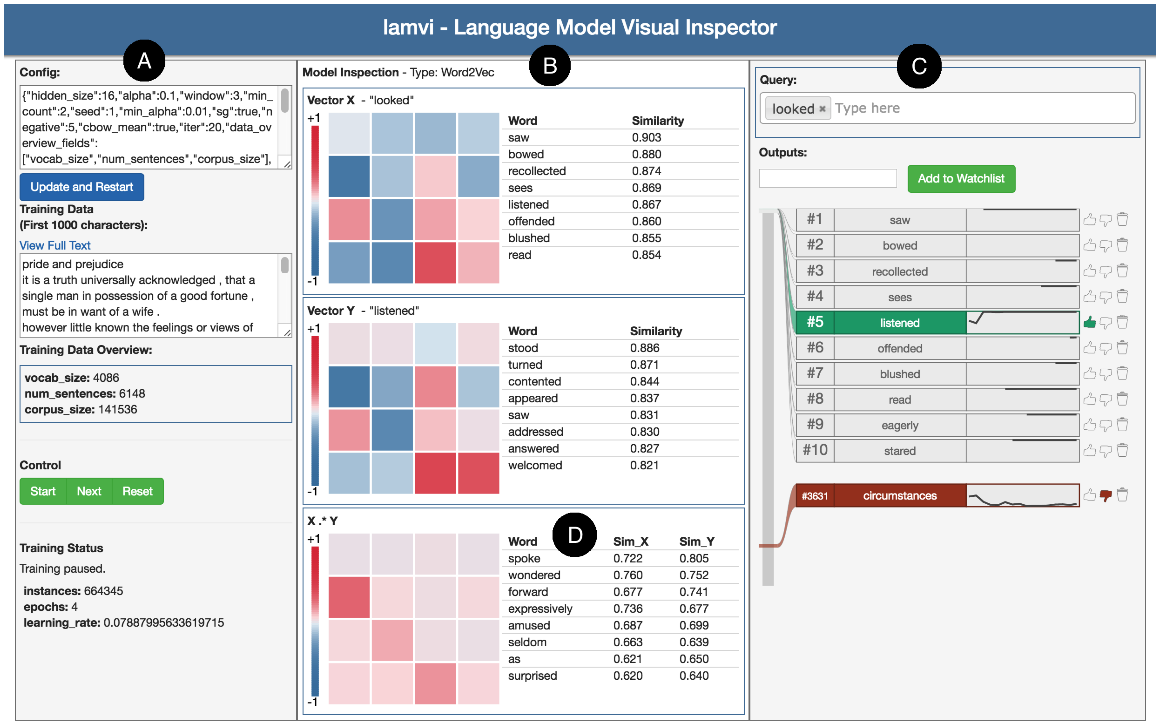
Natural language models based on word embeddings or more complex models (e.g., LSTMs, CNNs) are extremely powerful for representing text. Such models have been trained to generate impressive results ranging from learning linguistic regularities (semantic and syntactic) to higher level tasks such as text generation and summarization (Mikolov et al., 2013; Pennington et al., 2014; Karpathy et al., 2015). With increasing popularity, and new efforts to publish models (e.g., Gitxiv and many open-source tools), there are a variety of off-the-shelf implementations to choose from. While in some cases the pre-trained models are sufficient (and highly performant on the provided application/test scenario), developers must often modify the models in some way to work with the specific application, such as embedding users and products in the space for the purpose of building recommender systems. Tuning these models may require the developer to provide new data, change model parameters, or more significantly change the architecture.

It is often this exercise of modifying existing solutions that creates frustrating challenges to developers. Models can be highly sensitive to numerous factors ranging from preprocessing, to training, to the model configuration itself. These are often non-intuitive to the developer and results in an unguided experimentation with what look like "black-box" models. Often, it is not clear if the progress made in tuning the models is leading to the desired, let alone optimal, outcome. We have experienced these challenges ourselves in a number of applications. While we have developed some "institutional memory" on what works (Rong et al., 2016), this does not have broad coverage or is easy to apply directly.

Our belief is that model tuning and debugging can better be addressed through noval tools. To that end, we have been developing LAMVI (LAnguage Model Visual Inspector), a visualization-driven tool for debugging neural language models. We have specifically opted for a visual interface because of the complexity of the inputs, outputs, and intermediate states of the model. We have found it extremely difficult to debug our models without "summaries" of the data. Visualizations, as cognitive boosts, provide access to the "black-box" without requiring complete transparency. Rather, those aspects of the data/model that are useful in making decisions are provided through visual channels.

In our current instantiation of LAMVI we have focused on a common scenario—where the expected output is largely clear to the developer. For example, given a particular query (e.g., input term), the developer would be able to "grade" the output. Visually providing access to this output, in a way that supports this grading, is a key design requirement we are targeting. Additionally, we would like for the end-user to be able to trace failures of the model in a way that supports tuning decisions. Ideally, the end-user should be able to generate reasonable hypotheses that more training data, different pre-processing, or a different tuning parameter, or even a different architecture, might help. They should then be able to easily test this modification. LAMVI supports this through an interactive, visual front-

International Conference on Machine Learning (ICML) Workshop on Visualization for Deep Learning, New York, NY, USA, 2016.



lamvi - Language Model Visual Inspector

Config:

```
{"hidden_size":16,"alpha":0.1,"window":3,"min_count":2,"seed":1,"min_alpha":0.01,"sg":true,"negative":5,"cbow_mean":true,"iter":20,"data_overview_fields":
["vocab_size","num_sentences","corpus_size"],
```

Training Data (First 1000 characters):

pride and prejudice
it is a truth universally acknowledged , that a single man in possession of a good fortune , must be in want of a wife .
however little known the feelings or views of

Training Data Overview:

vocab_size: 4086
num_sentences: 6148
corpus_size: 141536

Control

Training Status

Training paused.

instances: 664345
epochs: 4
learning_rate: 0.07887995633619715

Model Inspection - Type: Word2Vec

Vector X - "looked"

Word	Similarity
saw	0.903
bowed	0.880
recollected	0.874
sees	0.869
listened	0.867
offended	0.860
blushed	0.855
read	0.854

Vector Y - "listened"

Word	Similarity
stood	0.886
turned	0.871
contented	0.844
appeared	0.837
saw	0.831
addressed	0.830
answered	0.827
welcomed	0.821

X .* Y

Word	Sim_X	Sim_Y
spoke	0.722	0.805
wondered	0.760	0.752
forward	0.677	0.741
expressively	0.736	0.677
amused	0.687	0.699
seldom	0.663	0.639
as	0.621	0.650
surprised	0.620	0.640

Query:

looked × Type here

Outputs:

#1	saw
#2	bowed
#3	recollected
#4	sees
#5	listened
#6	offended
#7	blushed
#8	read
#9	eagerly
#10	stared
#3631	circumstances

Figure 1. Screenshot of LAMVI. Panel A shows model and corpus overview; Panel B supports the inspection of activation levels of hidden-layer units, vector interactions, and influential training instances (not shown here; see Figure 4); Panel C allows the user to enter queries and select words to *watch*; Panel C also visualizes the change of ranks of the watched words over iterations; Panel D shows the interactions between two vectors, as well as the most influential words contributing to their association.

end.

Though our long term goal is to support many different kinds of model failures, we have particularly focused on two in our initial implementation: (1) insufficient signal in the input corpus, and (2) the training process was not properly configured. LAMVI allows the developer to see the input corpus, configure the model, and iteratively execute the training. During this training, the developer can inspect the hidden layers of the model, observe how they change, and "probe" them for specific query terms. More critically, the system allows for inspection of *pairs* of words (e.g., query and result), their joint hidden layers, features, and actual training instances (examples pulled from the input corpus). Various visualizations (e.g., heatmaps, and 2D projections) allow the developer to better "understand" the underlying model. Additionally, LAMVI allows a developer to see if model training is progressing in a desired direction. Given an input query (or queries), the developer can track the "similarity" to a set of words through a visual ranking which displays trend information.

This paper describes our current work-in-progress implementation. The tool, which runs in a browser, is currently configured for word2vec, a popular word embedding model (Mikolov et al., 2013). However, we have architected it to work with other models such as GloVe (Pennington et al., 2014), DeepWalk (Perozzi et al., 2014), or LINE (Tang et al., 2015). We describe our current implementation and the design decisions we have made that we believe are useful beyond our current system. Our goal is to provide a user-friendly tool for allowing developers to determine if their system is working, and if not, to support changes that lead to effective solutions.

2. Related Work

There is a large amount of work on use of visualization to inspect text, some with a machine learning focus. For example, AntConc provides concordances and other visual toolkits to support corpus linguistic analysis, such as word frequencies and collocation inspection (Anthony, 2011). Chuang et al. use association matrices and alignment charts to investigate latent topic coverage of a large collection of model variants (Chuang et al., 2012; 2013). LDAvis employs interactive visualizations to facilitate the user to interpret the content and inter-relationships of different latent topics learned by a topic model (Sievert & Shirley, 2014). Other examples related to topic visualization include (Liu et al., 2009; Cui et al., 2014). Kulesza et al. use simple bar graphs of feature importance to provide answers to the user's *why*-questions regarding text message classification in an email client application (Kulesza et al., 2011). The visual tools presented in these studies greatly improve the interpretability of their corresponding language models, and many of the techniques, such as concordances, are borrowed in our LAMVI implementation. LAMVI, however, is particularlly focused on the training process of neural network models.

Using visualization techniques to improve understandability of neural network (NN) models has also drawn a great deal of attention. Many NN visualization projects are focused on computer vision (Yosinski et al., 2015; Samek et al., 2015). There are also several interactive visualization projects developed for educational purposes, such as the recent Tensorflow Playground.[1]

Compared to images, videos, or quantitative multidimensional datasets, words and sentences lack natural visual representation and can be harder to interpret. The required transformation of text to data that can be visually encoded presents unique challenges. To improve understandability of natural language representations learned by neural network models, existing techniques for visualizing high-dimensional data are often borrowed, such as principal component analysis (PCA) (Jolliffe, 2002), multidimensional scaling (Kruskal, 1964), or t-SNE (Van der Maaten & Hinton, 2008). However, these methods are not neural-network-specific, and do not specifically improve the understandability of the learning process of neural networks. In comparison, a recent work by Karpathy et al. employ multiple views of activation levels to illustrate how recurrent neural networks capture patterns in text sequences, such as long-range dependencies (Karpathy et al., 2015). We have previously released Word Embedding Visual Inspector (WEVI) (Rong, 2014),[2] an interactive educational tool that lets the user play with a toy word2vec model in the browser. In comparison, the presented LAMVI system is not just a tool that supports understanding the underlying neural network model, but provides an interactive debugging environment for model development and deployment as well.

3. Skip-gram Model with Negative Sampling

Before introducing LAMVI, we provide a brief review of the word2vec skip-gram model (Mikolov et al., 2013), the underlying neural language model that LAMVI currently supports. The model can be efficiently trained on large linguistic corpora to generate continuous vector representations of words. The learned vectors (i.e., word embeddings) can then be used directly for text classification and clustering, or initializing the input layers of deep neural network models. This particular model has been shown to be performant when trained on a variety of corpora (Levy et al., 2015; Lai et al., 2015).

The word2vec model is a log-bilinear model, a neural network model with one hidden layer that has linear activation. The output layer of the network is a softmax layer. Given a word w_I and a set of words in its neighborhood (i.e., *skip-gram*), $w_{O_1}, w_{O_2}, \cdots, w_{O_C}$, the model predicts the probability of generating the words in the skip-gram given w_I. The probability is formulated as:

$$p(w_{O_1}, w_{O_2}, \cdots, w_{O_C}|w_I) = \prod_{c=1}^{C} \frac{\exp(\mathbf{v}_{w_I}^T \mathbf{v}'_{w_j})}{\sum_{j'=1}^{|V|} \exp(\mathbf{v}_{w_I}^T \mathbf{v}'_{w_{j'}})} \qquad (1)$$

where C is the size of the context; $\mathbf{v}_w$ is the vector of w taken from the weight matrix between the input and the hidden layers; $\mathbf{v}'_w$ is the vector of w taken from the weights between the hidden and the output layers; and $|V|$ is the size of the vocabulary.

The training of the model is done using backpropagation with stochastic gradient descent. The original form of the model requires updating every single element on the hidden→output weight matrix for each training instance, which makes training very inefficient. Mikolov et al. 2013 propose negative sampling, which randomly samples a few words from an empirical distribution per training instance, and only update the hidden→output weights associated with these words. The learning objective then becomes:

$$E = -\log\sigma({\mathbf{v}'_{w_O}}^T \mathbf{v}_I) - \sum_{w_j \in \mathcal{W}_{\text{neg}}} \log\sigma(-{\mathbf{v}'_{w_j}}^T \mathbf{v}_I) \qquad (2)$$

where $\mathcal{W}_{\text{neg}}$ is the set of words that are taken as negative samples.

Other optimization methods have also been shown to be very important (Levy et al., 2015). These include: downsampling frequent words; random shrinking of window size (implicitly taking care of contextual word proximity); the initialization of the word vectors, the learning rate, and the decay of learning rates are important as well, as we shall visually explore in the remainder of this paper.

4. LAMVI In Use

We briefly describe an example interaction to demonstrate LAMVI's expected use. Suppose Alice has trained a word2vec model on Jane Austen's *Pride and Prejudice* using the default model parameters. She inspects the nearest neighbors of "*wife*", expecting "*husband*" to be ranked the

[1] http://playground.tensorflow.org/

[2] http://bit.ly/wevi-online

highest. Instead, "*engagement*" and "*marriage*" are ranked higher than "*husband*,". She wants to find the reason for this behavior and to fix it. Using LAMVI, she visually inspects the vectors of the words in question, and identifies the most influential contexts that contribute to the false positive outputs. She then uses LAMVI's concordance view to examine the relative positions of these context words to the query word ("*wife*"), and finds that many contexts that contribute to the false positives are farther away from "*wife*" than those contributing to "*husband*". Knowing this, Alice reduces the context window size, and retrains the model. LAMVI automatically tracks the ranks of the words of interest across training iterations. By inspecting the ranking records, Alice confirms that "*husband*" stabilizes at the highest rank after a few iterations. This example illustrates one of the many debugging scenarios enabled by LAMVI.

5. The LAMVI Interface

Figure 1 shows a screenshot of LAMVI. The integrated interface supports many common debugging activities, including: configuring model parameters, overview of the training data, pausing training, and stepping in a training instance (Figure 1-A); specifying input queries and tracking the ranks of expected candidate outputs (Figure 1-C); viewing activation levels of hidden units (Figure 1-B) and vector interactions (Figure 1-D), as well as inspecting influential training instances (Figure 4) and checking 2D projections of vectors of interest (Figure 2).

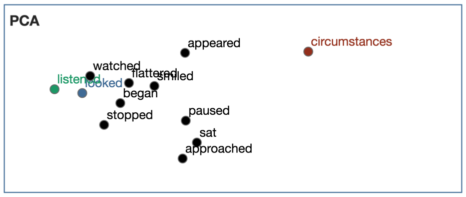


Figure 2. 2D visualization of word embedding vectors of user-specified input query, watched candidate outputs, as well as nearest neighbors of the input query. PCA is used for dimensionality reduction. The user may label certain candidate outputs as *good* or *bad*, and such candidates will be colored differently.

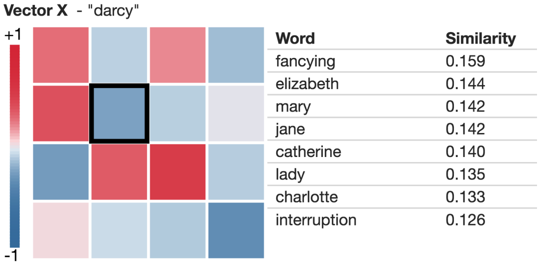
5.1. Tracking Ranking of Specific Candidates

As shown in Panel C of Figure 1, the user can monitor the change of ranks of specific candidate words given an input query she selects. We find that monitoring the rank trend over iterations can be informative in a number of scenarios. The *ideal* situation is that an expected "good" candidate starts off at a random position and, as the training proceeds, gradually moves to the top among all candidates where it stabilizes (e.g., "listened" in Figure 1, Panel C, given the input term "looked"). If the rank of an expected output stabilizes at a low rank position, it can be because of lack of relevant training instances; whereas the rank of an unexpected output stabilizing at a high rank position may indicate too much noise in the training data. If the ranks of multiple watched candidates remain unstable and oscillate, it can be because the learning rate or the learning rate decay strategy may be set inappropriately, or the training data contains too much noise.

While the visualization may not identify a single cause, it narrows the possibilities down and provides avenues for additional exploration. Rank monitoring provides a high-level sense of the training process. However, one can dig deeper into the low-level details of word representations learned by the model in order to identify the true causes of certain model behavior.

5.2. Inspecting Vector Representations

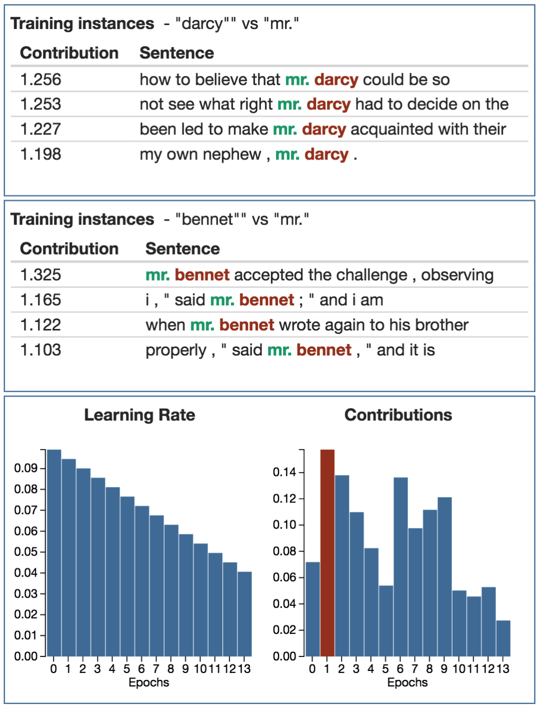
We provide three different ways to let the user explore the vector representations learned by the model.



Word	Similarity
fancying	0.159
elizabeth	0.144
mary	0.142
jane	0.142
catherine	0.140
lady	0.135
charlotte	0.133
interruption	0.126

Figure 3. Inspection of the topic(s) associated with a single vector component, highlighted by a black rectangle. The query word is "*darcy*".

First, a heat map (Figure 1, Panel B) can directly illustrate the values of different components of a word vector. Given a word w, suppose the hidden layer size is K, then its vector can be denoted as $\mathbf{v}_w = \{v_{w,1}, \cdots, v_{w,K}\}$. Each cell is a color-encoded representation of a vector component, $v_{w,j}$, $j \in \{1, \cdots, K\}$. While the positions of the cells do not have actual meanings, rendering them as a matrix instead of an array not only makes the layout "tighter," but also makes it easier for human eyes to spot patterns. However, we do recognize the inherent risk that gestalt heuristics will lead the end-user to spot a pattern that isn't there (see Chapter 6 of (Ware, 2012)). If the training configuration is set properly, the user can typically observe that the cells start off with random colors, and, as the training proceeds, a few cells turn darker colors (meaning $v_{w,j}$ is getting close to either -1 or 1) and stabilize, while a majority of cells stabilize at lighter colors ($v_{w,j}$ close to 0). A deviation from this pattern may indicate improper learning rate selection (e.g., causing vector components explode to infinity), error in model implementation, and improper initial value selection.



Training instances - "darcy"" vs "mr."

Contribution	Sentence
1.256	how to believe that **mr. darcy** could be so
1.253	not see what right **mr. darcy** had to decide on the
1.227	been led to make **mr. darcy** acquainted with their
1.198	my own nephew , **mr. darcy** .

Training instances - "bennet"" vs "mr."

Contribution	Sentence
1.325	**mr. bennet** accepted the challenge , observing
1.165	i , " said **mr. bennet** ; " and i am
1.122	when **mr. bennet** wrote again to his brother
1.103	properly , " said **mr. bennet** , " and it is

Figure 4. Inspecting training instances. The top panels show context words in sentences. The bottom panel compares the learning rates and contributions of a single sentence by training epoch.

Second, we provide a list of nearest neighbors based on a specific dimension j^*. When the user selects a specific cell, e.g. j, the corresponding dimension is then used to find a ranked list of words $\{w'\}$ that are both (1) similar to the current query word w, where similarity is measured by cosine similarity,

$$\cos(\mathbf{v}_w, \mathbf{v}_{w'}) = \frac{\mathbf{v}_w \cdot \mathbf{v}_{w'}}{\|\mathbf{v}_w\| \|\mathbf{v}_{w'}\|}; \tag{3}$$

and (2) share similar activity as w on dimension j^*, i.e., $v_{w,j^*} v_{w',j^*}$ is high. Using this view (see Figure 3), the end-user can gain an understanding of the "meaning" of a dimension by observing which words are activated.

Third, LAMVI offers a 2D plot showing the nearest neighbors of the query as well as all the watched candidates (see Figure 2). We use principal component analysis (PCA) for dimensionality reduction.[3] As the training proceeds, the user can monitor the change of the positions of vectors, and even potentially spot interesting clusters.

While these visual tools allow the user to quickly gain insights into what is learned by the model, they do not directly answer *why* certain candidates are ranked higher than others.

5.3. Inspecting Interactions of Vectors

To understand how a pair of vectors $(\mathbf{v}_w, \mathbf{v}_{w'})$ become "close neighbors" we would like to inspect the training instances we have encountered. We can calculate, among all the training instances we have encountered while learning ($\mathbf{v}_w$ and $\mathbf{v}_{w'}$), which ones are the most influential. Figure 1, Panel D) illustrates two ways of inspecting such results.

First, we can show the element-wise product of the two vectors, i.e., $\{v_{w,1}v_{w',1}, \cdots, v_{w,K}v_{w',K}\}$ as a heatmap. As the user focuses on individual cells, we show which words are most activated by that corresponding vector component. Through this view the user can get an idea of what topics are *shared* by the two words.

Second, we show a list of training instances that have made the most contributions. Each instance is a context word w_c encountered for the word of interest w during training, and the contribution of the instance is the L2 norm of the gradient on $\mathbf{v}_w$ when that instance is encountered, i.e., $\|\frac{\partial E}{\partial \mathbf{v}_w}\|$, where E is the learning objective.

For example, if the vectors of two characters' names are close (e.g., "darcy" and "bennet"), we may observe that they share most influential features related to human beings, such as "mr." and "mrs." When some expected features do not show up, the user may consider actions such as improving the corpus preprocessing routine or adjusting the window size.

5.4. Inspecting Training Instances

Since an influential feature may occur in multiple sentences throughout the corpus, we may want to go further and inspect which specific training instances (i.e., sentences) contribute the most to certain associations learned by the model. Figure 4 illustrates two of our solutions.

First, we borrow the idea of concordances and show a ranked list of sentence snippets from which the influential features are extracted. These snippets are ranked, again, by their contributions to the position of $\mathbf{v}_w$. By inspecting the actual sentences one may spot errors or potential improvements to be made in the corpus preprocessing routines (e.g., "mr. darcy" may be concatenated as a phrase to distinguish the term from "mrs. darcy"), or make better judgment about the size of the context window.

Second, since a single sentence may be encountered multiple times in different training epochs we need access to the contribution each context has made. In the interface we

[3] One can also use many other techniques, such as *t*-SNE (Van der Maaten & Hinton, 2008).

show what learning rate is applied to the contexts in the sentence (in each epoch), and what contributions (again, in the epoch) that the context has made to $\mathbf{v}_w$. By inspecting such information, the user may compare the effects of different relevant hyperparameters, including learning rate, speed of learning rate decay, downsampling, and negative sampling. For example, if the user spots that some "good" features are downsampled too heavily, she may consider adjusting the "sampling" hyperparameter of the model to avoid losing important signals in the corpus.

Note that there are hazards of overfitting the specific instances inspected here. The user should always check multiple instances and have a benchmark to keep track of the model's overall performance. Ideally, future instances of the tool can help guard against this by better supporting this kind of tracking.

6. Discussions

There are several aspects of the system that we are working to improve.

Scaling up: The current implementation of LAMVI runs fully in the browser and can only support a small corpus and vocabulary. However, the framework and visual tools are designed to be extensible to support full-sized models with millions of words in the vocabulary using a server-client model. Since most interactive visualizations are focused on a watch-list of just a few words, the overhead of logging additional information per training instance is small. Also note that training efficiency is not usually the primary concern for those who are debugging the model for its quality.

Scaling to other embedding models: The proposed framework can be extended to support a full range of embedding models, including GloVe (Pennington et al., 2014), DeepWalk (Perozzi et al., 2014), and LINE (Tang et al., 2015), because they all share the same underlying neural network architecture. Our framework can also be adapted to embedding models with (slightly) more complex structures, such as Doc2Vec (Le & Mikolov, 2014) and bimodal embedding models (Allamanis et al., 2015). Adapting to these would require making model-dependent modification to the visualization interface, such as adding a new input channel (e.g. document identity, or input of a different modality). However, the nature of inspecting vector similarity, vector interaction, and tracking the ranks of watched candidate items will remain the same.

Scaling to sequential contexts: It is also possible to extend LAMVI to support neural language models that make predictions using sequential contexts. For example, memory networks can "generate" sentences given a few cue words or a piece of computer source code given a few characters (Karpathy et al., 2015). To debug such models, The user may specify the inputs as a sequence of words or characters, and observe, as the model consumes training data, how different candidate words or characters are reranked among the model's predicted probabilistic distribution. One may also look "further into the future", making the model generate N words or characters in a row, and inspecting how the likelihood of generating a given expected output evolves as the training proceeds. However, it can be challenging to locate specific influential training instances in a meaningful way given the complex nature of sequential contexts.

Explaining model behavior: There are many limitations to our current way of defining most influential training instances or features. An important part of our future work is to develop meaningful metrics that distinguish which set of training instances or which aspect of the model configurations is most responsible for a given candidate being ranked higher than another.

Supporting exploratory data analysis: In our system, as the model consumes training instances, a wide variety of information is logged. For example: the ranks of watched vectors, their gradients, and learning rates. When using LAMVI to debug a model, the user may have her own information need. Therefore, providing an exploratory data analysis environment provides the end-user with greater flexibility in terms of generating different visualizations and getting insights from the model's training footprint. For example, the user may define customized grouping of the contexts (e.g. by part-of-speech, or rarity of words), and inspect the influences of these training instances category by category.

Linguistic regularity: Our current implementation also supports inspecting the emergence of linguistic regularity captured by the model. The user may enter queries like "king –queen woman" and observe how the desired candidate, "man", evolves. The user may also inspect the activation levels of the hidden units given all three words as context.

Model diff: Our current version does not support direct comparison between two model versions trained with different configurations. Such comparison can be potentially very useful, as the user may directly see the effects of changing one hyperparameter. It would also be interesting to enable the user to adjust the model configurations and see what potential impact that configuration may have on the contributions of specific features on-the-fly, which can, nonetheless, be far more challenging than doing diffs on trained models.

Avoiding overfitting: A potential hazard of the presented debugging pattern is that the user may possibly overfit the specific cases that she selects to focus on, and fail to make the model work well on the overall dataset. Therefore, it is important that the user combine such kind of case-specific debugging routines with benchmark-based testing mechanisms (train/validate/test routines) to avoid overfitting. It would also be interesting to develop a recommended workflow/debugging strategy that combines low-level and high-level debugging routines.

7. Conclusion

In this paper we have described LAMVI, a work-in-progress interactive tool that supports the debugging of neural language models. The visual components of LAMVI are designed to support end-user inspection of model inputs and outputs of the model as well as hidden layers. At a high level, our objective is to support a developer's process of model training. Specific "bugs" can be detected, and decisions on which pieces to tune are supported through the GUI. Through the interface, for example, the developer can understand what training instances lead to the learned associations of the representations, and how the model hyperparameters may impact the results. We provided a preliminary analysis on the usefulness of the model via specific examples and listed a number of directions for future development.

[Code and data will be available at https://github.com/ronxin/lamvi/].

References

Allamanis, Miltos, Tarlow, Daniel, Gordon, Andrew, and Wei, Yi. Bimodal modelling of source code and natural language. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2123–2132, 2015.

Anthony, Laurence. Antconc (version 3.2. 2)[computer software]. *Tokyo, Japan: Waseda University*, 2011.

Chuang, Jason, Manning, Christopher D, and Heer, Jeffrey. Termite: Visualization techniques for assessing textual topic models. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pp. 74–77. ACM, 2012.

Chuang, Jason, Gupta, Sonal, Manning, Christopher, and Heer, Jeffrey. Topic model diagnostics: Assessing domain relevance via topical alignment. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 612–620, 2013.

Cui, Weiwei, Liu, Shixia, Wu, Zhuofeng, and Wei, Hao. How hierarchical topics evolve in large text corpora. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2281–2290, 2014.

Jolliffe, Ian. *Principal component analysis*. Wiley Online Library, 2002.

Karpathy, Andrej, Johnson, Justin, and Li, Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Kruskal, Joseph B. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

Kulesza, Todd, Stumpf, Simone, Wong, Weng-Keen, Burnett, Margaret M, Perona, Stephen, Ko, Andrew, and Oberst, Ian. Why-oriented end-user debugging of naive bayes text classification. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 1(1):2, 2011.

Lai, Siwei, Liu, Kang, Xu, Liheng, and Zhao, Jun. How to generate a good word embedding? *arXiv preprint arXiv:1507.05523*, 2015.

Le, Quoc V and Mikolov, Tomas. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

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

Liu, Shixia, Zhou, Michelle X, Pan, Shimei, Qian, Weihong, Cai, Weijia, and Lian, Xiaoxiao. Interactive, topic-based visual text summarization and analysis. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pp. 543–552. ACM, 2009.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pp. 1532–1543, 2014.

Perozzi, Bryan, Al-Rfou, Rami, and Skiena, Steven. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Rong, Xin. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

Rong, Xin, Chen, Zhe, Mei, Qiaozhu, and Adar, Eytan. Egoset: Exploiting word ego-networks and user-generated ontology for multifaceted set expansion, 2016.

Samek, Wojciech, Binder, Alexander, Montavon, Grégoire, Bach, Sebastian, and Müller, Klaus-Robert. Evaluating the visualization of what a deep neural network has learned. *arXiv preprint arXiv:1509.06321*, 2015.

Sievert, Carson and Shirley, Kenneth E. Ldavis: A method for visualizing and interpreting topics. In *Proceedings of the workshop on interactive language learning, visualization, and interfaces*, pp. 63–70, 2014.

Tang, Jian, Qu, Meng, Wang, Mingzhe, Zhang, Ming, Yan, Jun, and Mei, Qiaozhu. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

Van der Maaten, Laurens and Hinton, Geoffrey. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

Ware, Colin. *Information visualization: perception for design*. Elsevier, 2012.

Yosinski, Jason, Clune, Jeff, Nguyen, Anh, Fuchs, Thomas, and Lipson, Hod. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.