Hierarchical Structural Learning for Language Generation

Benjamin Striner [*] [1] **Siddharth Dalmia** [*] [1]

Abstract

We develop novel generative methods for language modeling that leverage graphical structure instead of traditional, strict left-to-right generation. Language generation is viewed as a top-down process through a parse tree. Our models are applicable to language modeling and representation as well as neural language parsing. We also propose a novel distribution for language modeling that enables these models. We both experiment with methods using supervised parse trees and methods that infer the most useful parse tree in an unsupervised manner.

1. Introduction

A language model is a probability distribution over a sequence of words. A language model is different than a Gaussian, for example, because it is a distribution over a discrete sequence of unknown length. Language models provide a way of representing the distribution of language, and can be used for language generation or any other task where language is an input or output.

In this project, we explore language modeling and generation using probabilistic graphical models. We treat the generation of language as a recurrent process, whereby the content is generated through a hierarchy of phrases. We treat each phrase as independent of other phrases, given its parent phrase and its children. We model language as a structured tree, where the leaves are observed words, and internal nodes are higher-level organization such as phrases and clauses. For a graphical representation of this generative story, see Figure 3.

As an idealized example, we decide the topic of a sentence. From the topic, we draw representations of the noun, verb and object phrases. A concrete noun is drawn from the noun phrase representation, and some adjectives may be drawn based on the noun.

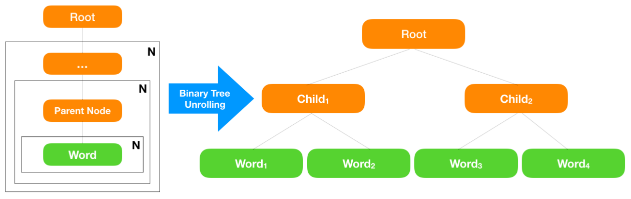

Figure 1. **Generative Story :** Top-level sentence topic is drawn and recurrently expands into a complete sentence, e.g., noun phrase expands into some number of adjectives and a noun.

Our model contrasts with traditional language modeling performed in a left-to-right manner using a recurrent formulation. Each word is drawn based on the previous words using a network such as an LSTM. Our project attempts to break this mold by utilizing graph-based methods that do not require strict left-to-right generation and are instead top-down. Our models are recursive rather than simply recurrent.

We enable our model by proposing a novel distribution for language modeling. Our distribution allows a greater freedom of structure than traditional left-to-right language modeling, which we use to explore incorporating graphical model structure into language modeling.

Our contributions:

- We introduce CTC mixture models to approximate distributions over a sequence of tokens of unknown but bounded length
- We introduce a tree-based graphical model for language modeling
- We show how CTC mixture models can be used to derive unknown graphical structures in languages.
- We provide results applying our tree-structured language model to language generation

[*]Equal contribution [1]Carnegie Mellon University. Correspondence to: Benjamin Striner <bstriner@andrew.cmu.edu>, Siddharth Dalmia <sdalmia@andrew.cmu.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

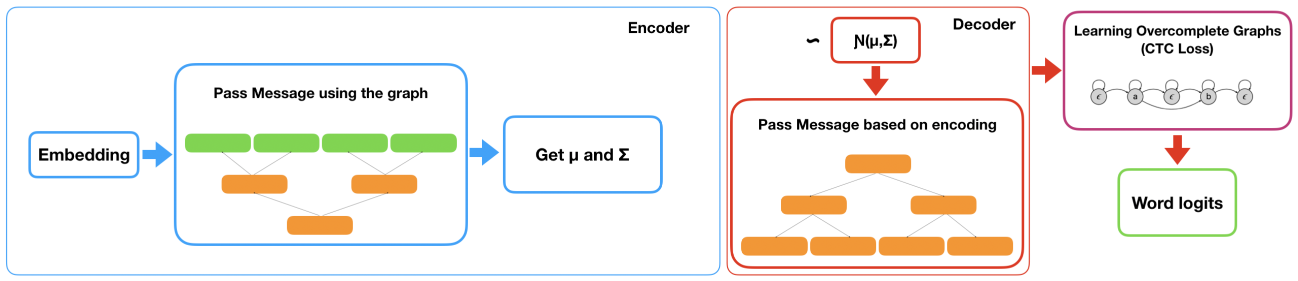
Encoder
Embedding
Pass Message using the graph
Get μ and Σ
Decoder
N(μ,Σ)
Pass Message based on encoding
Learning Overcomplete Graphs (CTC Loss)
Word logits

Figure 2. **Model Overview: [We describe all the components of our proposed model –SD]**

2. Background and Related Work

2.1. Graph Structure Inference

Research has shown that tree structures are powerful tools for language, especially in the realm of sentiment analysis (Kokkinos & Potamianos, 2017). Although simple left-to-right models work well enough for many tasks, tree structured models appear to handle some linguistic issues such as negation better. Intuitively, negation takes a phrase and inverts its meaning, which is easily modeled if phrases and sub-phrases are represented in a tree.

2.2. Language Syntax Trees

2.2.1. Dependency Parse Trees

A dependency parse represents the relationships in a sentence by words linked to "head" words that they modify. An exemplary dependency parse is shown in Figure 3.

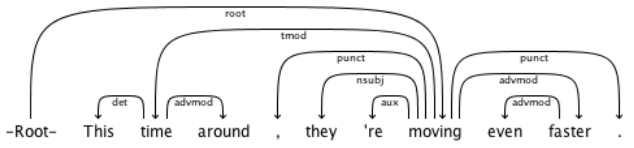

Figure 3. Example dependency parse https://nlp.stanford.edu/software/nndep.html

A common method for producing dependency parses is to control a transition-based parser using a neural network (Chen & Manning, 2014; Yang et al., 2017; Kiperwasser & Goldberg, 2016). A transition-based parser incrementally builds a dependency parse by making decisions about pushing to a stack, making an arc, and other operations until it reaches a goal state. The specifics of those operations are what makes each transition-based parser unique. A greedy approach to selecting operations is common but a beam search is also possible.

In our work, we use Stanford NLP as a reference dependency parser, for providing supervision or for comparison to our methods (Manning et al., 2014).

2.2.2. Constituency Parse Trees

A constituency parse tree breaks the sentence into it's constituents or sub-phrases. The tree structure has a root node which has sub-branches corresponding to different components constituting the syntax of the sentence and the leaf nodes are the words of the sentences. An exemplary constituency parse tree is shown in Figure 4.

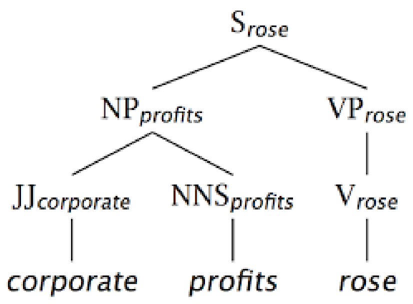

Figure 4. Example Constituency Parse Trees (Charniak)

A typical way to dependency parsing is to serialize the build the tree using recurrent neural network grammars (Kuncoro et al., 2016) where instead of predicting the tree structure the model learns to predict serialized stack operations that will lead to the tree. This is typically how a FST grammer is decoded by a computer. There have also been a lot of recent work on parse trees which consider the tree as a lexicalized context free grammar. Which means that every node of the tree is associated with one of the words in the leaf node, allowing us to build an attention type hirerichial encoder. (Charniak et al., 2016; Kitaev & Klein, 2018; Fried et al., 2017)

2.3. Graph Structure Inference

2.3.1. Learning Bayesian Networks

A unique method of parameterizing DAGs is presented in (Zheng et al., 2018). This provides a differentiable method of learning and parameterizing a DAG for a single problem. As dependency parses can be represented as DAGs, it would be valuable to combine the NOTEARS method with language modeling to infer hidden structures.

2.3.2. Learning Overcomplete Graphs

One fantastic work presented in (Graves et al., 2006) gave a unique way of learning alignments between two spaces using the connectionist temporal loss (CTC). A model trained with CTC loss is a sequence based model which automatically learns alignment between input and output by introducing an additional label called the blank symbol ($\emptyset$), which corresponds to 'no output' prediction. Given a sequence of acoustic features $\mathbf{X} = (\mathbf{x_1}, \ldots, \mathbf{x_n})$ with the label sequence $\mathbf{z} = (\mathbf{z_1}, \ldots, \mathbf{z_u})$, the model tries to maximize the likelihood of all possible CTC paths $\mathbf{p} = (\mathbf{p_1}, \ldots, \mathbf{p_n})$ which lead to the correct label sequence $\mathbf{z}$ after reduction. A reduced CTC path is obtained by grouping the duplicates and removing the $\emptyset$ (e.g. $\mathcal{B}(AA\emptyset AABBC) = AABC$).

$$P(\mathbf{z}|\mathbf{X}) = \sum_{\mathbf{p} \in \texttt{CTC\_Path}(\mathbf{z})} P(\mathbf{p}|\mathbf{X})$$

This is a non-autoregressive top-down approach with the power of handling "non-important" input frames. We would like to use this in our task by creating an overcomplete flat representation and using the CTC loss function to do a top-down language modeling. We believe this would have some robust features like handling noises by learning a higher order dependency between word sequences than just sequential dependency learned by LSTMs.

A similar idea was also done in a recent paper that attempts to train a speech recognition system without any labeled data (Yeh et al., 2018). The insight is that the distribution of phonemes output by the speech recognizer should match the language model. An ASR system can be trained on unlabeled speech by maximizing the likelihood of the ASR output under a language model. This model is however limited by requiring phoneme segmentations for the ASR component. A separate model estimates phoneme boundaries. A method similar to EM is used to alternate between training the two models.

Overcomplete representation learning is well studied in the cases where the input size is much larger than output, for example sound waveforms, spectrograms etc. Techniques like dictionary learning and sparse coding have been used to learn representations for speech (Sivaram et al., 2010; Vinyals & Deng, 2012; Sainath et al., 2011) and audio signals (Plumbley et al., 2010).

2.4. Generative Modeling

We use two types of generative models in our experiments, Variational Autoencoders (VAEs) and Adversarial Autoencoders (AAEs). Both models have two main components:

- The encoder produces a latent representation from inputs
- The decoder predicts inputs from a latent representation

The latent representations are regularized such that their marginal distribution is a known prior, such as a Gaussian. We are then able to pass Gaussian samples to the decoder to produce samples from the same distribution as our data.

We experimented with four types of models, summarized in Figure 2. These models are listed in order of both expressive power and feasibility to evaluate.

- Autoencoders use a deterministic encoder and decoder. These models do not provide the ability to generate samples but are fast to train and useful for finding initial hyperparameters.
- Variational Autoencoders use a stochastic encoder using the reparameterization trick. These models can generate samples and easily calculate likelihood. However, the lower bound that is optimized causes "blurry" results.
- Adversarial Autoencoders using a stochastic encoder using the reparameterization trick. These models can generate samples. Results are sharped due to optimizing an approximation instead of a bound. Conditional distributions are Gaussian. It is somewhat reasonable to calculate likelihood.
- Adversarial Autoencoders using stochastic functions. The conditional distribution for each encoding is not constrained to be Gaussian, so it is more difficult to evaluate likelihood.

2.4.1. Variational Autoencoders (VAEs)

Autoencoding variational Bayes utilizes a variational lower bound on the divergence between the latent representations and the prior to encourage the latent representations to match the prior (Kingma & Welling, 2013). Using the reparameterization trick, this model can be trained easily with gradient descent. However, due to utilizing a variational lower bound, results from this model are known to be "blurry".

2.4.2. Adversarial Autoencoders (AAEs)

Adversarial Autoencoders utilize an adversary to ensure that the marginal distribution of encodings matches a prior (Makhzani et al., 2015). A discriminator evaluates samples from a prior and compares them to samples from the latent representation and tries to tell the difference. This model is similar in many respects to a Generative Adversarial Network (GAN) (Goodfellow et al., 2014) but differs in key aspects:

- AAEs train encoders and decoders but a GAN trains just a decoder

- AAE discriminator works on latent space but GAN discriminator works on input space. If input space is discrete but latent space is continuous, GAN discriminstator is not differentiable but AAE discriminator is differentiable.

We utilize the Wasserstein GAN objective to stabilize adversarial training (Arjovsky et al., 2017). We also utilize spectral normalization to constrain the Lipschitz constant of the discriminator (Miyato et al., 2018).

2.5. Language Modeling

There has been a lot of work on neural language models, these is particularly useful in natural language generation tasks and auxiliary tasks such as speech recognition, machine translation etc. Most of the language modeling tasks assume the language generation story to be left to right, which they model it using an LSTM (Hochreiter & Schmidhuber, 1997; Mikolov & Zweig, 2012; Joulin et al., 2016). Though this works reasonably well in terms of the perplexity. It has a few drawbacks -

- Humans don't really think of sentence generation in the same fashion.

- It is slow as the first word needs to be generated to produce the next.

Recently with transformer based models (Vaswani et al., 2017) there has been a growing interest of flat models which are non-autoregressive. This has led to research into "flat language models". OpenAI GPT (Radford et al., 2019) uses a flat encoder but uses a set of previous attentions to generate the words.

Google recently solved the problem of flat generation by introducing a new technique called the masked language model approach (Devlin et al., 2018), which basically recognizes the mask given the rest of the sentence. Though this is good for classification tasks in terms of generation this doesn't really work. To do the generation task in this model we would have to first start with a sentence with all masks and then do some kind of variational inference or greedy sampling to produce words. This makes the approach extremely slow during inference.

There has also been some work on making tree based LSTMs (Dyer et al., 2015) which try to model the hierarchical structure of the sentence using LSTMs. But these are quite slow and have not been shown for language generation tasks.

Our model attempts to solve both these issues by learning a DAG and using that encoding and decode this over-complete representation using CTC loss function.

3. Models and Methods

Figure 5 summarizes our two proposed architechture along with the standard unidirectional LSTM based language model.

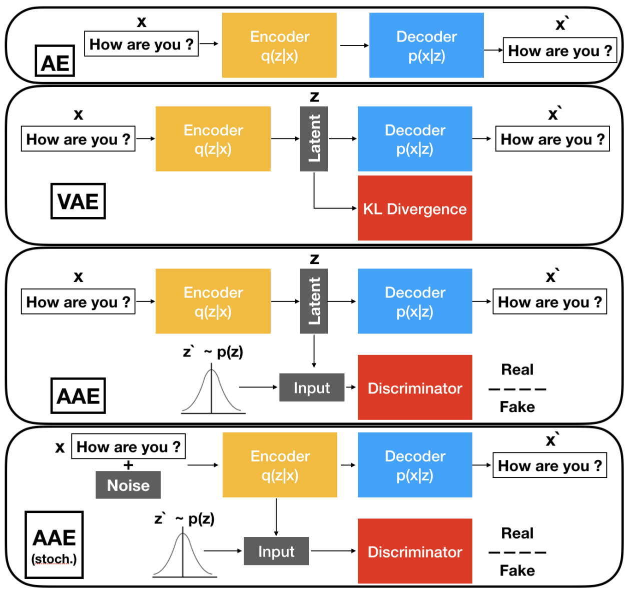

Figure 6. **Various Generative Models:** We utilized several models with differing levels of complexity and expressive power

3.1. Standard LSTM based LM

The most standard way to model language and generate language is by using a unidirectional LSTM which takes words as input from left to right and at each step predicts the next word given the already seen words in the past.

Standard LSTM LM model has a lot of disadvantages that we try to fix in our model -

- The model doesn't fully exploit the bi-directional capabilities of the LSTMs as using the bidirectional information will make the model prediction trivial and the model won't learn any useful information.

- The model generates words left to right which tends to be less robust towards errors. As the error would simply propagate till the end of the sentence if the model receives a word which it has not seen before.

- These models tend to prefer shorter sentences because of vanishing gradients problem.

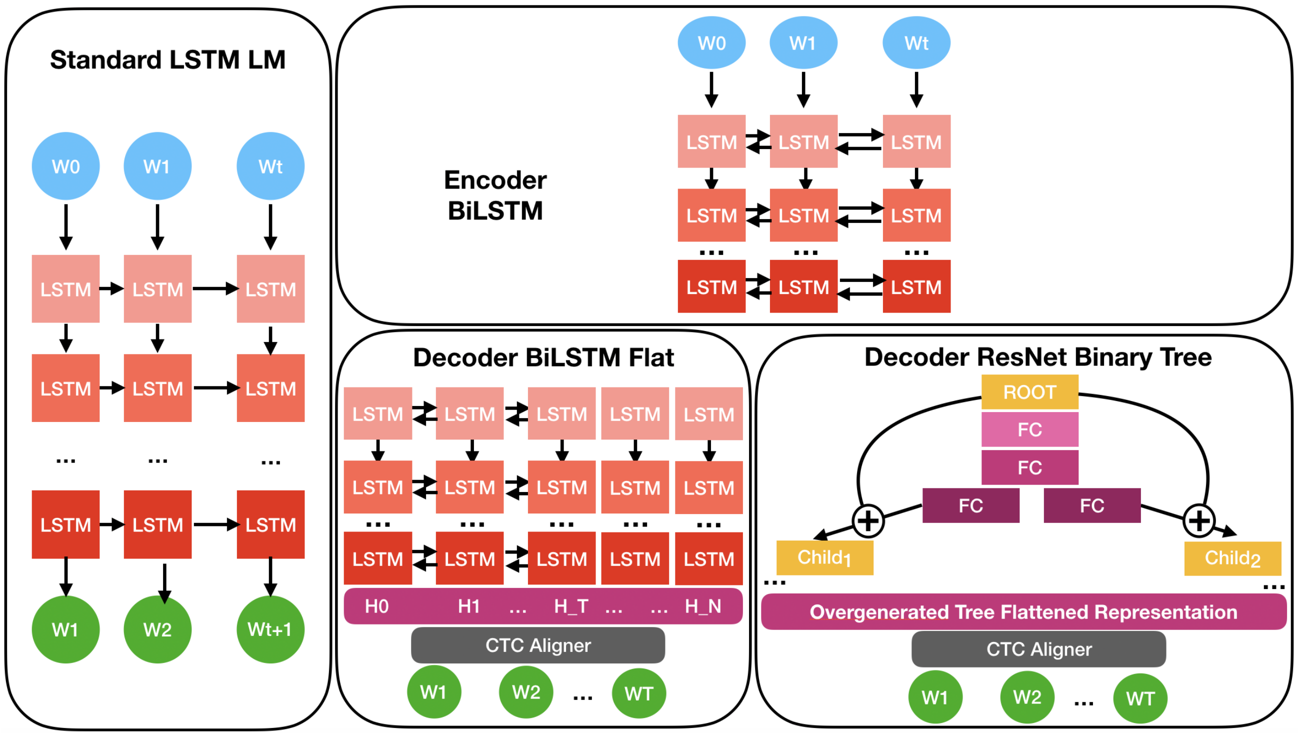

Figure 5. **Various Language Modeling Architectures:** Traditional language model at left. Our encoder uses a BiLSTM at top. We experimented with both flat and tree-structured decoders.

3.2. Representation of Sentence Length

One of the most important differences between CTC mixture models and traditional models is in regards to the representation of sentence length.

- Traditional models emit tokens until a special “end” token is reached
- CTC models emit a fixed number of tokens, and remove some number of “blank” tokens

3.3. CTC mixture model based LM

The distribution over language is traditionally modeled as a chain of multinomial distributions, each conditioned on the previous words. Special words for start and end of sentence are typically incorporated.

$$P(Y_t \mid Y_{0:t-1})$$

We propose to model language as a mixture of CTC distributions. Each CTC distribution is a joint distributions over tokens and alignments.

An individual CTC distribution makes independence assumptions that make it incapable of modeling the complexity of language. However, just as a Gaussian mixture can approximate any distribution, we propose that a CTC mixture can approximate any language model. As explained in the background CTC loss presents a unique way of learning alignments between two sequence representations. It can approximate distributions over a sequence of tokens of unknown but bounded length.

We exploit this fact in an auto-encoder based training by first learning a sentence representation using some birectional LSTM which can then be used to predict the mean and variance of our latent representation. We then sample from this latent distribution and which is then passed to a decoder composed few bidirectional LSTMs which generates a fixed sequence length encoding of the sentence. This is then used by the CTC aligner to give the best sequence of words.

By generating a fixed length sequence encoding rather than generating words from a single vector helps the model better focus on learning the structure of the language and allow the CTC to find the best alignment of words that matches the structure, thereby increasing the model’s capability. This way the model encoder models the structure (grammar) of the language and the CTC loss models the words that are best suited for a particular structure.

We experimented with a method for performing joint modeling of language and structure on an overcomplete graph. We built a large, over-complete binary tree, where every node emits a word or a blank. We used CTC loss to marginalize over all possible alignments between the sentence and the tree (Graves et al., 2006).

The outputs of the graph are read in infix order (left, self, right recursively) to support our generative model. Each node may emit a word and may emit more nodes to its left and right, enabling the recurrent generation of language.

The tree itself we model as a simple Bayesian network, where the children of a node given a parent node are parameterized by an artificial neural network. The initial representation of the root phrase is actually the latent encoding provided by a VAE or AAE architecture.

We found that training on our original network was difficult. We added batch normalization and used residual calculations to make training more efficient.

4. Dataset

We did our experiments on the english penn treebank dataset (LDC2015T13) and the english web treebank (LDC2012T13) as these datasets are reasonably sized for us to run some learning algorithm on. These datasets also come with the default dependency parses which enables us to perform further analysis and help us validate individual components of our model.

4.1. Penn Treebank Dataset

English Penn Treebank is set of annotated wall street journal stories. There are a total of 2,312 articles comprising of roughly 49k sentences. These sentences are annotated at various levels including tokenization, part-of-speech, dependency parse, etc.

This is particularly useful as this dataset has also been well studied for the task of language modeling and generation tasks which would complete our entire generative story.

5. Experiments and Analysis

Since our model is a generative model it is typically quite hard to get a sense of how good our model is doing. We use some of the standard quantitative measures and also come up with some interesting experiments that explains the power of our model.

5.1. Conventional LM

We began by training a conventional left-to-right factored language model. This model consisted of an embedding from vocabulary to D dimensions, a L layer unidirectional LSTM, followed by a projection layer with weights tied to the embedding layer. Our best hyperparameter tuning was $D = 256$, $L = 3$, resulting in a validation NLL of 83.36.

We used this model as an initial estimate of the perplexity of the dataset, as well as a tool to evaluate the performance of other models.

5.2. Quality Estimate using External LM

Since our model is a tree based top-down language generator we first want to verify if we can preserve the left-to-right dependencies needed to understand the sentence.

To do that we take an external language model which is trained on PTB and test it on 10,000 randomly generated samples generated by our model which was trained on the penn tree bank dataset. We can see from the table 1 that our model has decent perplexities when compared to language models trained on that dataset.

Model	Validation	Test
Zaremba et al. (2014) - LSTM	82.2	78.4
Merity et al. (2016) - Sentinel-LSTM	72.4	70.9
Inan et al. (2016) - Variational LSTM	71.1	68.5
Shen et al. (2017) - Joint-Syntax-LM	–	62.0
Our best VAE Model	88.61	
Our best AAE Model	79.90	

Table 1. Word perplexity on validation and test sets for the Penn Treebank language modeling task.

5.3. Tree Generation

We generated trees from our model and visualized how those sentences were generated, shown in Figure 7.

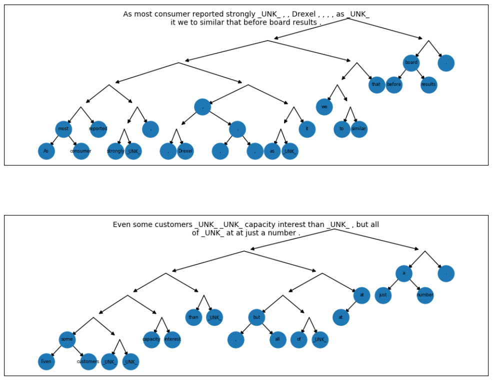
As most consumer reported strongly _UNK_ , , Drexel , , , , as _UNK_ it we to similar that before board results .
board
.
that
before
results
we
most
reported
,
,
,
it
to
similar
As
consumer
strongly
UNK
,
Drexel
,
,
as
UNK
Even some customers _UNK_ _UNK_ capacity interest than _UNK_ , but all of _UNK_ at at just a number .
a
.
at
just
number
than
UNK
but
at
some
capacity
interest
,
all
of
UNK
Even
customers
UNK
UNK

Figure 7. Sentence trees generated by our model

6. Conclusions

We built models around a top-down generative story of language production. Our models are applicable to language understanding, representation and parsing.

We also proposed a new type of language model and a method for aligning language to an arbitrary structure.

Future work should include more investigation into what graphical structures or neural networks are best for generating CTC mixture models. We only explored word-level CTC models but character-level CTC models may be even more powerful.

The main limitations of this work are fully tuning and evaluating each of the models against baselines. There are many types of models and we did a lot of engineering to get the resnet-based trees we ended up with. We also hope to show that the representations learned by our models are beneficial in downstream tasks such as sentiment classification.

References

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein GAN. *arXiv e-prints*, art. arXiv:1701.07875, Jan 2017.

Charniak, E. Statistical parsing with a context-free grammar and word statistics.

Charniak, E. et al. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2331–2336, 2016.

Chen, D. and Manning, C. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 740–750. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1082. URL http://aclweb.org/anthology/D14-1082.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

Fried, D., Stern, M., and Klein, D. Improving neural parsing by disentangling model combination and reranking effects. *CoRR*, abs/1707.03058, 2017. URL http://arxiv.org/abs/1707.03058.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pp. 369–376, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143891. URL http://doi.acm.org/10.1145/1143844.1143891.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Inan, H., Khosravi, K., and Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. *arXiv e-prints*, art. arXiv:1312.6114, Dec 2013.

Kiperwasser, E. and Goldberg, Y. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *CoRR*, abs/1603.04351, 2016. URL http://arxiv.org/abs/1603.04351.

Kitaev, N. and Klein, D. Constituency parsing with a self-attentive encoder. *arXiv preprint arXiv:1805.01052*, 2018.

Kokkinos, F. and Potamianos, A. Structural attention neural networks for improved sentiment analysis. *CoRR*, abs/1701.01811, 2017. URL http://arxiv.org/abs/1701.01811.

Kuncoro, A., Ballesteros, M., Kong, L., Dyer, C., Neubig, G., and Smith, N. A. What do recurrent neural network grammars learn about syntax? *CoRR*, abs/1611.05774, 2016. URL http://arxiv.org/abs/1611.05774.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. Adversarial Autoencoders. *arXiv e-prints*, art. arXiv:1511.05644, Nov 2015.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60, 2014. URL http://www.aclweb.org/anthology/P/P14/P14-5010.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mikolov, T. and Zweig, G. Context dependent recurrent neural network language model. *SLT*, 12:234–239, 2012.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral Normalization for Generative Adversarial Networks. *arXiv e-prints*, art. arXiv:1802.05957, Feb 2018.

Plumbley, M. D., Blumensath, T., Daudet, L., Gribonval, R., and Davies, M. E. Sparse representations in audio and music: from coding to source separation. *Proceedings of the IEEE*, 98(6):995–1005, 2010.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Sainath, T. N., Nahamoo, D., Kanevsky, D., Ramabhadran, B., and Shah, P. A convex hull approach to sparse representations for exemplar-based speech recognition. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, pp. 59–64. IEEE, 2011.

Shen, Y., Lin, Z., Huang, C.-W., and Courville, A. Neural language modeling by jointly learning syntax and lexicon. *arXiv preprint arXiv:1711.02013*, 2017.

Sivaram, G. S., Nemala, S. K., Elhilali, M., Tran, T. D., and Hermansky, H. Sparse coding for speech recognition. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4346–4349. IEEE, 2010.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Vinyals, O. and Deng, L. Are sparse representations rich enough for acoustic modeling? In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

Yang, L., Zhang, M., Liu, Y., Yu, N., Sun, M., and Fu, G. Joint POS tagging and dependency parsing with transition-based neural networks. *CoRR*, abs/1704.07616, 2017. URL http://arxiv.org/abs/1704.07616.

Yeh, C.-K., Chen, J., Yu, C., and Yu, D. Unsupervised Speech Recognition via Segmental Empirical Output Distribution Matching. *arXiv e-prints*, art. arXiv:1812.09323, Dec 2018.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Zheng, X., Aragam, B., Ravikumar, P., and Xing, E. P. DAGs with NO TEARS: Continuous Optimization for Structure Learning. *arXiv e-prints*, art. arXiv:1803.01422, Mar 2018.