Learned Syntax Aware Word Embeddings Using Equivariant Graph Neural Networks

Stanford CS224N Custom Project

Pratham Soni
Department of Computer Science, Economics
Stanford University
prathams@stanford.edu

Abstract

In this paper, we aim to improve word embeddings (both static and contextual) by explicitly leveraging syntactic information such as part of speech (POS) tagging and dependency parsing (DP). We can model this information as a graph as in [1] to auto-encode the dependency information in the word embedding. Particularly, we implement a Equivariant GNN version [2] of the SIWR model to further improve downstream performance by considering a given sentence context as a vector backbone. This embedding model is pre-trained using a small amount of data (30,000 samples) to encode dependency and part of speech information. We test the model performance on three downstream Named Entity Recognition Tasks: Nested Entity Recognition, Binary Relationship Extraction, and N-Ary Relationship Extraction. The EGNN embeddings outperform the SIWR embeddings on all three of these tasks when using the same benchmark downstream models. [1]

1 Key Information to include

Mentor: Angelica Sun

2 Introduction

Word embeddings have played a critical role in the advancement of deep learning application in Natural Language Processing. These embeddings provide a dense vector representation of word-meaning in rela-valued numerical form, that has been empirically easier for models to understand. Much of the work in deep learning based natural language processing tasks is dependent on having high quality vector embeddings. These embeddings (such as Word2Vec [3]) have often been static, with a singular vector representation. However, recent work, like BERT [4] has shown the efficacy of contextual word embeddings that are dependent on the local context to which a word is embedded. Such models, dubbed Large Language Models (LLMs), do not incorporate syntactical information and, instead, rely on training a large number of parameters (100s of millions) over huge data corpora.

Further work has shown the propensity for using graph based architectures to improve these static/contextual word embeddings by leveraging syntactic structures like part of speech tagging and dependency trees [1], with a small amount of additional pre-trainig data. Our work builds upon [1] by considering sentences as vector chains and applying a Equivariant Neural Network architecture [2] in place of the standard GCN. This is motivated by existing work in the proteomics space where proteins are modelled in a vector space invariant to translation, reflection, and rotation operations. This allows the incorporation of vector features compared to just scalar ones, which seems like an intuitive extension for word embeddings. If we consider the words in some latent vector space defined

[1]Project code can be found at https://github.com/PrathamSoni/Equivariant-GNN-Word-Embeddings

by their embedding features, some operation composed of translations, reflections, and/or rotations should not change the interpretation of the features overall.

As in Tran et al., we test the syntactically aware embeddings produced by the model on hree downstream Named Entity Recognition Tasks: Nested Entity Recognition (NER), Binary Relationship Extraction (RE), and N-Ary Relationship Extraction (N-Ary). We measure performance using F1-scores for NER and RE and accuracy for N-Ary. Due to the scope of this paper, we only boost embeddings for Word2Vec initial embeddings. We then use these to train downstream reference models. The embeddings provided by the EGNN model outperform those provided by SIWR with the same initial embeddings for all three tasks.

3 Related Work

3.1 Word Embeddings

While there has been a desire to model language computationally, there has been a need to represent words in numerical form. Representation as continuous vectors rather than one-hot elements of a vocabulary have become dominant [3, 5, 6, 7, 8]. There has been a particular interest in using Neural Networks to model such representations. Recent efforts into improving embeddings follow the strategy of leveraging vast amounts of data against very large models to attempt to learn the breadth of embedding space [4, 9, 10, 11]. While the performance of these models has been excellent, most only present one embedding for a given word regardless of context. Furhter, they do not incorporate any kind of explicit syntactical information. This information has been shown to be potentially useful as obtained from dependency parsing tools [12, 13]. Given such dependency arcs, the natural representation for such information has been in the form of a graph [1, 14].

3.2 AI in Proteomics

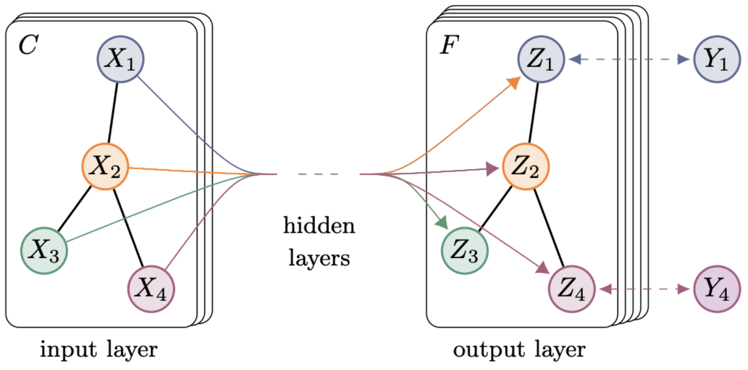

Figure 1: In a GCN architecture, at each layer, messages from each of a node's neighboring nodes are pooled to get the new hidden state. Repeating across all hidden layers gives output features at each node. [15]

Graph Neural Networks (GNNs) have performed very well on protein-like structures due to the natural sparsity of such structures lending to their representations as graph objects [16, 17, 15]. As a graph, the protein structure is represented as far as relations [18]. As such, there is only an implicit understanding of position. Equivariant Neural Networks and 3D CNNs present the opposite end of the spectrum with methods that operate on explicit point spaces but are susceptible to perturbation by geometric transformations [19, 20]. Combining these methods to represent a vector space in a graph model is where the Equivariant Graph Neural Network comes into play as a way to combine the important features of both of these methodologies [2, 18].

This study aims to combine these two lines of thought to use an EGNN architecture for word embeddings.

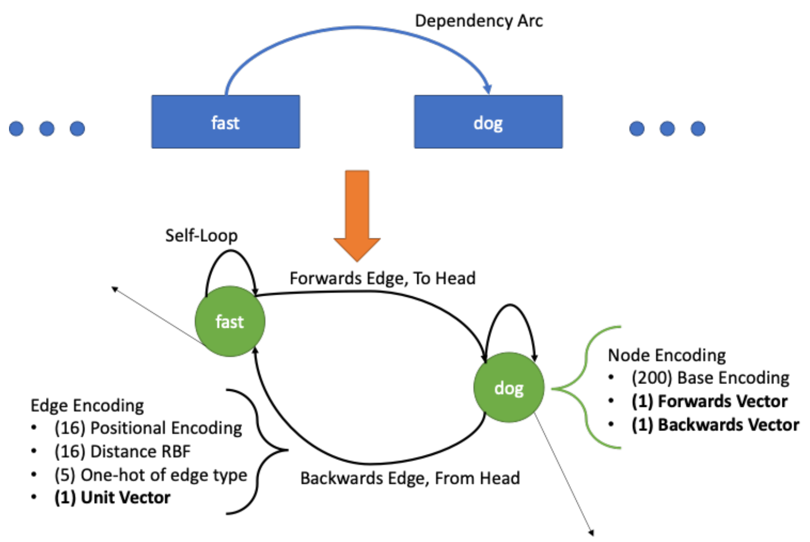

Figure 2: Summary of encoding process. Each node receives scalar embedding from the static Word2Vec model. The remaining scalar and vector channels are established from these. Edges are drawn between neighboring words, dependency arcs, and self-loops. Vector features are bolded for convenience.

4 Approach

For word w in sentence c, we wish to construct a vector embedding $v_{w|c}$. We first construct graph embedding $g(w, c)$ from the original document as described in Section 4.1. These graph features are then passed to a EGNN model as described in 4.2. To train this model, we adapt the methodology from [1] as described in Section 4.3. We can then use this pre-trained model to generate improved, syntactic context-aware embeddings as in Section 4.4.

4.1 Graph Embeddings with Vector Features

To generate the improved embeddings, we start by first encoding the sentence as a graph. For node embeddings, we use the original scalar features but also add forward and reverse unit vectors in the direction between words: $w_{i+1} - w_i$ and $w_i - w_{i-1}$.

For edges, we take the construction in [1] and augment the edge features. We draw an edge from vertex i to j if one of the following conditions are true: $|j - i| = 1, 0$ or there is a depndency arc from i to j or vice-versa. Thus, we have five types: the prior word, the subsequent word, self-loop, head of dependency arc, dependent of dependency arc. For each edge, $e^{i,j}$, we take the following features: the unit vector in the direction of $w_j - w_i$, an encoding of the distance between w_i and w_j in the form of Gaussian radial basis functions, and the sinusoidal positional encoding of $j - i$ as described by [21], consisting of the $\frac{d}{2}$ pairs $[\sin(w_k \cdot (i-j)), \cos(w_k \cdot (i-j))]$, where $w_k = \frac{1}{10000^{\frac{2k}{d}}}$, $0 \leq k < d = 16, 2|k$. We also encode a one hot representation of the five edge-types above. This is summarized in Figure 2.

4.2 Equivariant Graph Neural Network Model

If we wish for vector features to remain equivariant, we cannot use any off the shelf model, as in the GCN used by Tran et al. [1]. Instead, we replace the two standard graph convolution layers of the GCN with Graph Vector Perceptron convolution layers [2]. For each GVP layer, we make updates to the scalar and vector features as shown in Algorithm 1. With g representing a successive GVP layers for the message-passing layers and f representing b successive GVP layers for the point-wise feed-forward, we have the following update for each GVP convolution, where $h_v^(i)$ is the node-embedding at i and $h_e^{(j\to i)}$ is the edge-embedding between j and i [18]:

$$h_m^{(j\to i)} = g\left(\text{concat}(\,h_v^{(j)}, h_e^{(j\to i)}\,)\right)$$

$$h_v^{(i)} \leftarrow \text{LayerNorm}\left(h_v^{(i)} + \frac{1}{|j \text{ s.t. } e^{i,j}|}\text{Dropout}\left(\sum_{j \text{ s.t. } e^{i,j}} h_m^{(j\to i)}\right)\right)$$

We then do a feed-forward point-wise update at each node.

$$h_v^{(i)} \leftarrow \text{LayerNorm}\left(h_v^{(i)} + \text{Dropout}\left(f\left(h_v^{(i)}\right)\right)\right)$$

Algorithm 1 GVP update step. [2]

```
**Input:** $(S, V)$
**Output:** $(S', V')$
**GVP:**
  $h$                                                    ▷ Hidden dim size
  $V \leftarrow W_h V$
  $S_h \leftarrow ||V||$                                  ▷ Taken row-wise
  $S \leftarrow \text{concat}(s_h, S)$
  $S_m \leftarrow W_m S + b_m$
  $S' \leftarrow \text{ReLU}(S_m)$
  $V' \leftarrow \text{Sigmoid}(W_g \text{Sigmoid}(S_m) + b_g) \odot V$   ▷ Taken row-wise
```

4.3 Pretraining Algorithm

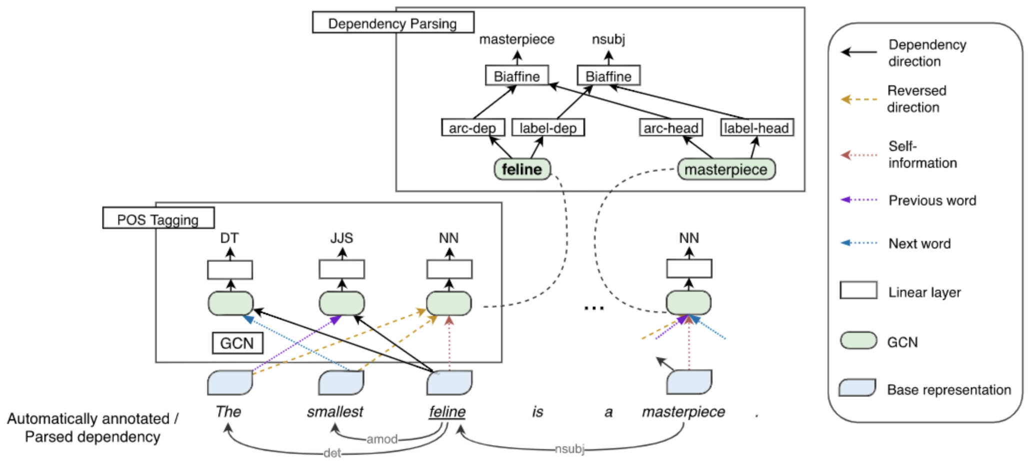

Figure 3: Pretrain process for SIWR model from [1]. For a given sentence, the base embeddings are used as node features for a word graph. These are then fed into a GCN to provide a new word embedding for each word. For the POS task, these are then fed into a softmax classifier to predict the POS for each word. For the dependency task, we use a biaffine attention mechanism to predict the dual probability of every (head, dependency label) pair for each word. This work replaces the graph features to include vector information and the GCN architecture with an Equivariant Graph Neural Net.

As mentioned in Figure 3, we take the two subtasks of POS prediction and dependency arc prediction. The following equations are adapted from [1]. For both subtasks, we take the output from l_2 and project using a GVP layer [2] to get latent vectors $\mathbf{h^x}$, where $x \in \{\text{POS}, \text{arc\_head}, \text{arc\_dependent}, \text{label\_head}, \text{label\_dependent}\}$. $\mathbf{h^{POS}} \in \mathbb{R}^{d_p}$ and the rest $\in \mathbb{R}^{d_w}$, where $d_p = 16$ and $d_w = 100$. We then get $p(y^{\text{POS\_pred}}) = \text{softmax}(\mathbf{W}h^{\text{POS}} + \mathbf{b})$. Now to get the probability of the dependency arc, we first take $p(y^{\text{arc\_pred}}) = \text{softmax}(\mathbf{H}^{\text{arc\_head}}\mathbf{W}^{\text{arc}}h^{\text{arc\_dep}})$. Then for each arclabel l, we take $s^l = \mathbf{H}^{\text{label\_head}}\mathbf{W}^l h^{\text{label\_dep}}$. Concatenating, we get $p(y^{\text{label\_pred}}) = \text{softmax}(\text{Concat}_l s^l)$. Thus, we have probability distributions for the arc and the label for the arc. We take the element-wise product to get the combined distribution: $p(y^{\text{arc\_label\_pred}}) = p(y^{\text{label\_pred}}) \odot p(y^{\text{arc\_pred}})$.

We utilize a L2 regularized loss combining the Negative Log Likelihoods of the POS prediction and the DP prediction. Thus we take:

$$J = J_{POS} + J_{DP} + \lambda||\mathbf{W}||_2 = -\sum_{v \in c} \log(p(y_v^{\text{POS\_pred}})_{y^{\text{POS\_true}}}) + \log(p(y_v^{\text{arc\_label\_pred}})_{y^{\text{arc\_label\_true}}}) + \lambda||\mathbf{W}||_2$$

Thus, we not only encode the part of speech information, but we also auto-encode the dependency information as that signal is incorporated into the original graph encoding.

4.4 Constructing Embeddings

Let the two convolution layers be l_1, l_2 and the initial embedding be v_o. Then we have $v_{w|c} = v_o + v_1 + v_2$, where v_2, v_2 represent the scalar portion of the output from l_1, l_2 respectively. We have $v_1 = \text{scalar}(l_1(g(w,c)))$ and $v_2 = \text{scalar}(l_2(l_1(g(w,c))))$. In this way, the output is a combination of the inital base embedding and two layers of pooled embeddings that carry the signal of the syntactic features.

5 Experiments

5.1 Data

5.1.1 Pretraining

In the pre-training step, we take sentences and then aim to predict the part of speech, dependency arc, and dependency label for each word. Thus, we simply need corpora with sentences to construct the semi-supervised task given the labels are generated through the sentence parser. We train two models over two different datasets.

The N-Ary dataset for the downstream task is not general (Drug Gene Mutation [14], biomedical focused). As such we train over the PubMed dataset [3] to get a better representation of the vocabulary/syntax of the downstream dataset. For the PubMed dataset, we download a data dump provided by PubMed and then extract 30000 samples, consisting of article titles and abstracts, for each of the train and validation sets.

The NER and RE tasks both use a general dataset (ACE2005 [22]). As such, we use a general dataset to train. As above, we take 30000 samples from the Billion Word Benchmark dataset for each of the train and validation sets [23].

5.1.2 NER

As stated above, for the NER task, we use the ACE2005 dataset [22]. The dataset consists of sentences with associated entity tags. For NER, we wish to predict entities from inside to out. For example, we can have a span such as [the [U.S.] government], where the inside span is U.S., tagged as a country entity, and the outside span is the U.S. government, tagged as a government entity. For this study, we only utilize the English corpus of the dataset and take train/validation/test splits of 8:1:1.

5.1.3 RE

For the RE task, we use the ACE2005 dataset [22]. For this task, we take a sentence and wish to predict the relationship between pairs of entities. For example, we have the entities "Iraqi" and "Iraq", where the relationship is the people of a country. For this study, we only utilize the English corpus of the dataset and take train/validation/test splits of 8:1:1.

5.1.4 N-Ary

For the N-Ary task, we use the Drug Gene Mutation dataset [14], where we take documents and attempt to predict relationships across sentences. Particularly, we consider triples of drugs, genes, and mutations, and the pairwise permutations thereof.

5.2 Evaluation method

We measure performance using F1-scores for NER and RE and accuracy for N-Ary. This is based off previous work as in [1].

5.3 Experimental details

5.3.1 Pretraining

For the model configuration, we set each of the GVP Convolution Layers to take in and outputs node features with size $([200], [2, 200])$ and edge features with size $([37], [1, 200])$, where the pair represents the scalar and vector dimensions, respectively. Each of the GVP convolution layers uses $a = 3$ GVP layers for message passing and $b = 2$ GVP layers for point-wise feed-forwarding.

As stated previously, the N-ary task uses a scientific dataset, so we us a scientific sentence parser as provided by SciSpaCy [13] and Word2Vec embeddings trained on the PubMed dataset [3]. For the general pre-training, we use the Stanford Stanza parser [12] with Word2Vec embeddings trained on the Wikipedia corpus [24]. We take hyperparameters as shown in Table 1. Both Word2Vec models have dimensionality 200, which gives an overall model size of just over 1.5 million parameters.

Hyperparameter	Value
Batch Size	1000 nodes
Dropout Rate	0.1
Learning Rate	0.001

Table 1: Hyperparameters for pretraining

5.3.2 NER

For the NER task, we use a Layered Bi-LSTM [25]. The hyperparemeters used are default as shown in Table 2.

Hyperparameter	Value
Character Embedding	25
Batch Size	10
Learning Rate	0.0001
Weight Decay	0.0001

Table 2: Hyperparameters for NER

5.3.3 RE

For the RE task, we use the Walk-based RE model [26]. The model uses a fully-connected graph across all the entities in a sentence, and the edges are position-aware. Instead of the default embeddings, we utilize the EGNN ones. We use the default hyperparameters provided as shown in 3.

Hyperparameter	Value
Batch Size	10
Learning Rate	0.002
Weight Decay	0.000057

Table 3: Hyperparameters for RE

5.3.4 N-Ary

For the N-Ary task, we use a BiLSTM as in [14]. We use the hyperparameters as shown in 4.

Hyperparameter	Value
Batch Size	8
Learning Rate	0.0001

Table 4: Hyperparameters for N-Ary

5.4 Results

5.4.1 Main Results

	NER	RE	N-Ary
	F1-Score	F1-Score	Accuracy
Word2Vec Embedding	71.26	64.45	77.1
ELMo	75.93	65.74	75.8
SIWR	77.54	67	79.3
EGNN	79.44	69.56	80.3

Table 5: Experimental results

As we can see in Table 5, the embeddings for the EGNN model outperform the embeddings for the SIWR model/other embeddings. This is consistent with expectations given that the model leverages additional feature representations in the form of the vector channels that SIWR simply does not have access to.

5.4.2 Parameter Scaling

Of particular note is how the parameters of the EGNN model scale. Including the vector features actually does not introduce a lot of overhead as far as number of parameters. In fact, removing the vector features altogether only leads to a reduction of about 20000 parameters of a total of over 1.5 million. However, we do note that there is a quadratic scaling of the model parameters to the size of the base embedding dimensionality as shown in Figure 4. This is notable due to the assessment of model training time. Graph Neural Networks are notoriously hard to train as far as compute time, and increasing the base dimensionality to something like 750, which is approximately that of BERT [4], would be expected to take over 10 times longer than for the 200 dimensional Word2Vec embeddings used in this study.

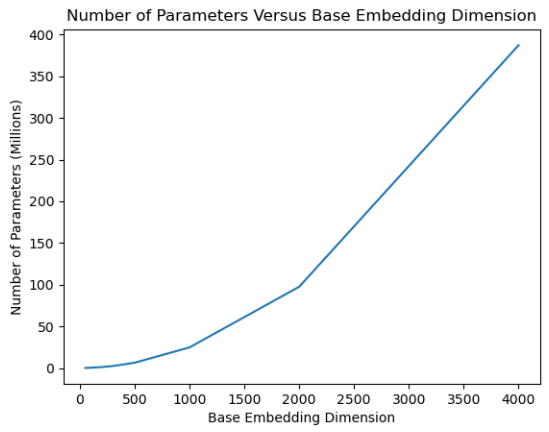

Figure 4: Comparison of model parameter count versus the base embedding dimensionality. Note the quadratic scaling.

6 Analysis

To perform a qualitative analysis, we look at a toy sentence with variations to determine how well the model is able to differentiate localized contexts and the degree to which it does so. To do this, we first take a sentence, and pass it through the embedder. We then select the word embeddings of interest and perform a PCA dimensionality reduction to visualize the pattern of the resultant embeddings. We also compare to the embeddings produced by the baseline model. We perform this analysis in lieu of something more typical such as looking at attention maps or specific dataset examples to assess some general examples of model patterns outside of the original training dataset. The four sentences we test are "he leaves the pile of leaves in disarray", "he leaves the pile of sticks in disarray", "he finds the pile of leaves in disarray", "he finds the pile of sticks in disarray." We have three goals here: are the meanings of the two forms of leaves embedded differently, how does the embedding change for one word if we swap another word, and how does the embedding compare for words occuppying the same role. The PCA plot is shown in Figure 5.

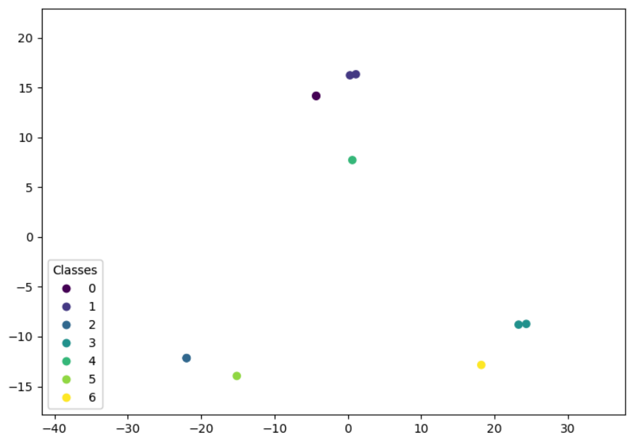

Figure 5: Two dimensional PCA plot of word embeddings. Class 0 is the first "leaves", class 1 is the second "leaves", class 2 is "finds", class 3 is "sticks", class 4 is "leaves" from the base embeddings, class 5 is "finds" from the base embeddings, and class 6 is "sticks" from the base embeddings.

In all, we can see that the base embeddings, (classes 4,5,6), are already differentiated. But for something like "leaves", there is only one vector representation. However, we see that classes 0 and 1, the two different EGNN "leaves" representations are ever so slightly off from the parent distribution. Further, we see the vector differences between classes 0, 2, and 3 are very similar to those between classes 4, 5, and 6, respectively. This indicates that the embeddings are primarily a result of the original base embedding, but can be modified slightly by the context as seen by class 1. Further, as seen by class 1, changing the context slightly does in fact cause a slight change in the embedding as evidenced by the small separation between the points. However, given a desire to model not just what a word is but also how it functions within a sentence, it can be expected that the embeddings for "leaves" as a verb is closer to that of "finds" and "leaves" as a noun is closer to "sticks" than what is observed.

7 Conclusion

Overall, this study contributes a novel EGNN model architecture for boosting word embeddings by incorporating syntactical information. The model outperforms similar previous work [1] using symmetrical downstream tasks of Nested Entity Recognition, Binary Relationship Extraction, and N-Ary Relationship Extraction. As the applications of deep learning in NLP grow, the importance of good word embeddings grow as well. The proposed method enables learning from syntactic information in the form of vector features as a possible avenue for improving embeddings in a relatively efficient manner compared to Large language models. For future work, there is still space to apply the methodology to base-embeddings for models that already give contextual embeddings such as BERT [4] and other large models [9]. Further, introducing mechanisms to weigh the balance between originally encoded information and new syntactic information is important to understanding how to optimally construct embeddings for downstream tasks after pre-training.

References

[1] Thy Thy Tran, Makoto Miwa, and Sophia Ananiadou. Syntactically-informed word representations from graph neural network. *Neurocomputing*, 413:431–443, 2020.

[2] Bowen Jing, Stephan Eismann, Pratham N. Soni, and Ron O. Dror. Equivariant graph neural networks for 3d macromolecular structure. *ArXiv*, abs/2106.03843, 2021.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

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

[9] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. 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 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[10] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[11] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020.

[12] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020.

[13] Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 319–327, Florence, Italy, August 2019. Association for Computational Linguistics.

[14] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-Sentence N-ary Relation Extraction with Graph LSTMs. *Transactions of the Association for Computational Linguistics*, 5:101–115, April 2017. _eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00049/1567450/tacl_a_00049.pdf.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[16] John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. Generative models for graph-based protein design. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[17] F. Baldassarre, D. Menéndez Hurtado, A. Elofsson, and H. Azizpour. GraphQA: protein model quality assessment using graph convolutional networks. *Bioinformatics*, 37(3):360–366, 04 2021.

[18] Bowen Jing, Stephan Eismann, Patricia Suriana, Raphael J. L. Townshend, and Ron Dror. Learning from protein structure with geometric vector perceptrons. *ArXiv*, abs/2009.01411, 2021.

[19] Stephan Eismann, Raphael J.L. Townshend, Nathaniel Thomas, Milind Jagota, Bowen Jing, and Ron O. Dror. Hierarchical, rotation-equivariant neural networks to select structural models of protein complexes. *Proteins: Structure, Function, and Bioinformatics*, 89(5):493–501, Dec 2020.

[20] Nathaniel Thomas, Tess E. Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick F. Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *ArXiv*, abs/1802.08219, 2018.

[21] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[22] Ace 2005 multilingual training corpus.

[23] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling, 2014.

[24] Makoto Miwa and Mohit Bansal. End-to-end relation extraction using lstms on sequences and tree structures. *arXiv preprint arXiv:1601.00770*, 2016.

[25] Meizhi Ju, Makoto Miwa, and Sophia Ananiadou. A neural layered model for nested named entity recognition. 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 1446–1459, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[26] Fenia Christopoulou, Makoto Miwa, and Sophia Ananiadou. A walk-based model on entity graphs for relation extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 81–88. Association for Computational Linguistics, 2018.