Learned Syntax Aware Embeddings Using Equivariant Graph Neural Networks

Pratham Soni
Department of Computer Science, Economics, Statistics
Stanford University
prathams@stanford.edu
Advised by Dr. Ron O. Dror

Abstract

In this paper, we aim to improve foundational model embeddings by explicitly leveraging syntactic information. Present foundational models do not explicitly model structural information, which can be problematic in certain domains such as biochemistry. We model this syntactic information as a graph as in [1] to auto-encode domain-specific information into the large model embeddings. Particularly, we implement a Equivariant GNN version [2] of the SIWR model to further improve downstream performance by considering a given sequence as a vector backbone. We pose a generalized learning framework for Syntactically Aware Embeddings (SAEs) that extend across learning domains. We pretrain SAE models using a small amount of data (~30,000 samples) and test downstream performance on a variety of learning domains and tasks. We test SAEs on the NLP (GLUE and CoNLL benchmarks) and biochemical (SMP) domains. Across the NLP domain, the SAEs outperform baseline embeddings on 10/11 tasks. In the biochemical domain, we observe improvements in 11/20 tasks. SAEs show particular promise in settings with limited fine-tuning when compared to baselines. [1]

Approved: [illegible]
Dr. Ron O. Dror
Advisor

Approved: [illegible]
Dr. Mykel J. Kochenderfer

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

1 Introduction

Embeddings have played a critical role in the advancement of deep learning application in many domains such as Natural Language Processing. These embeddings provide a dense vector representation of word-meaning in 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. Early methods such as Word2Vec [3] have often been static, with a singular vector representation. However, recent works, like BERT [4] have 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. Similar work leveraging large corpora of unsupervised data have been posed for other domains such as biomolecular chemistry with efforts such as ProteinBERT and MolBERT.

Recent 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-training data. Our work builds upon [1] by considering sentences as vector chains and applying a Equivariant Neural Network architecture (EGNN) [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 is an intuitive extension for word embeddings. If we consider the words in some latent vector space defined by their embedding features, some operation composed of translations, reflections, and/or rotations should not change the interpretation of the features overall.

In this paper, we present the Syntactically Aware Embeddings (SAE) as a generalized framework for incorporating domain-specific syntactic information into otherwise syntactically-devoid dense embeddings. Specifically, we utilize a light-weight EGNN as a pretraining model for existing foundational model encoders. We test the SAE framework in the NLP and biochemical domains, across a variety of tasks in each domain, to assess data and task generalizability. Across these settings, SAE performs strongly in generating embeddings for downstream tasks and serving as a pretrainer for the upstream FM.

2 Related Work

2.1 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. Further, 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].

2.2 AI in Proteomics

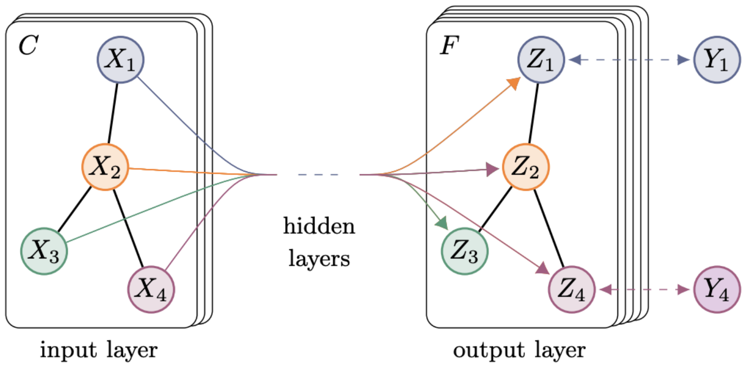


Figure 1: In GCN architectures messages from each of a node's neighboring nodes are pooled to get the new hidden state at each layer. Repeating across all hidden layers gives output features at each node. [15]

Graph Neural Networks (GNNs) have perform 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 relationally in terms of inter-atom or inter-amino acid connections rather than absolute physical positions [18]. As such, there is only an implicit understanding of the true physicochemical context. Equivariant Neural Networks and 3D CNNs represent the opposite end of the spectrum in the sense that they 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 Equivariant Graph Neural Networks (EGNN) comes into play as a way to combine the important features of both of these methodologies [2, 18].

Straying from the above methods, there has been a desire to adopt foundational model architectures to the field [21, 22]. These models come with the advantage of utilizing efficient unsupervised data for training. However, compared to their NLP counterparts, they do not achieve tangible improvements in performance within this domain. It stands to reason that in the proteomics space, structural information serves as a strong prior for latent embeddings. However, structural information is hard to generate (does not scale for large corpora) and may not be adequately treated by certain architectures (lack of equivariance).

To address these issues, we introduce the SAE framework for incorporating syntactic information into an existing FM.

3 Methods

In the SAE framework, we develop a mechanism that seamlessly encodes classical structural information and modern dense vector information in a unified loss function. To achieve this, we consider the following structure (Figure 2):

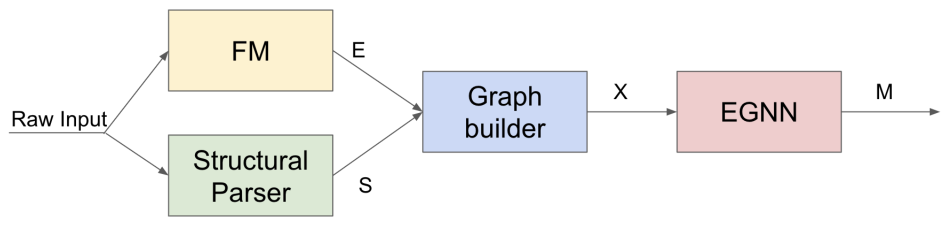


Figure 2: Framework for SAE model. We take dense representation E and structural information S from the FM and structural parser, respectively. We then construct graph embedding X to pass into the EGNN, which gives output features M.

For unit w in the sequence c, we wish to construct a vector embedding $v_{w|c}$. For some raw input (sentence or molecule), we generate two complementary initial embeddings: dense representation E from the FM that consists of a continuous vector representation for each unit in the sequence and structural information S that contains domain-specific syntactic information. We then are able to construct a graph embedding $g(w, c)$ from the original document. Again, the graph builder is domain-specific, which underlines the flexibility of the SAE framework as a generalized wrapper around FM models of choice. Particular definitions for these components are provided in sections 3.1 and 3.2. For the NLP domain, we use BERT [4] as the FM. For the biochemical domain, we use MolBERT [21]. In the MolBERT pipeline, we construct radius 1 Morgan fingerprints for each atom in the molecule and use the hash of the footprint as "words" in the "sentence."

These graph features are then passed to an EGNN model. As we wish to make the model as efficient as possible, we leverage a 2-layer architecture to give us a balance between expressivity and preventing overparameterization. We then add pretraining heads as described in 3.3.

To train this model, we adapt the methodology from [1] as described in Section 3.4. While the SIWR algorithm from [1] specifically trains on the edge-prediction task of the dependency arcs themselves, we extend the loss by allowing for the prediction of any subset of edges specified by the graph.

We can then use this pre-trained model to generate improved, syntactic context-aware embeddings as in Section 3.5 by stripping the model of the pretraining heads. These final embeddings differ from the EGNN output as we are pooling information between GVP layers. Further, we can choose to discard the EGNN head altogether as we propagate syntactic information back to the FM embeddings. Thus, we also test the use of the pre-trained FM embeddings on downstream tasks.

3.1 NLP

3.1.1 Structural Parser

To generate linguistics-specific syntactic information, we leverage the SpaCy package and utilize a variant of a non-monotonic arc-eager transition-system dependency parser [23] with pseudo-projective dependency transformation [24]. We also generate part-of-speech (POS) tags for each word in the sentence. This gives us information at both the node (POS) and edge (dependency arc) levels.

3.1.2 Graph Builder

To generate the improved embeddings, we start by first encoding the sentence as a graph. For node embeddings, we use the original scalar features (of dimension 768) 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 is true: $|j - i| \leq 1$ or there is a dependency arc from i to j or vice-versa. Thus, we have five types: the prior word, the subsequent word, self-loop, head of the dependency arc, and dependent of the 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 [25], 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. These 37 scalar features are concatenated together to give the edge scalar representation. This is summarized in Figure 3.

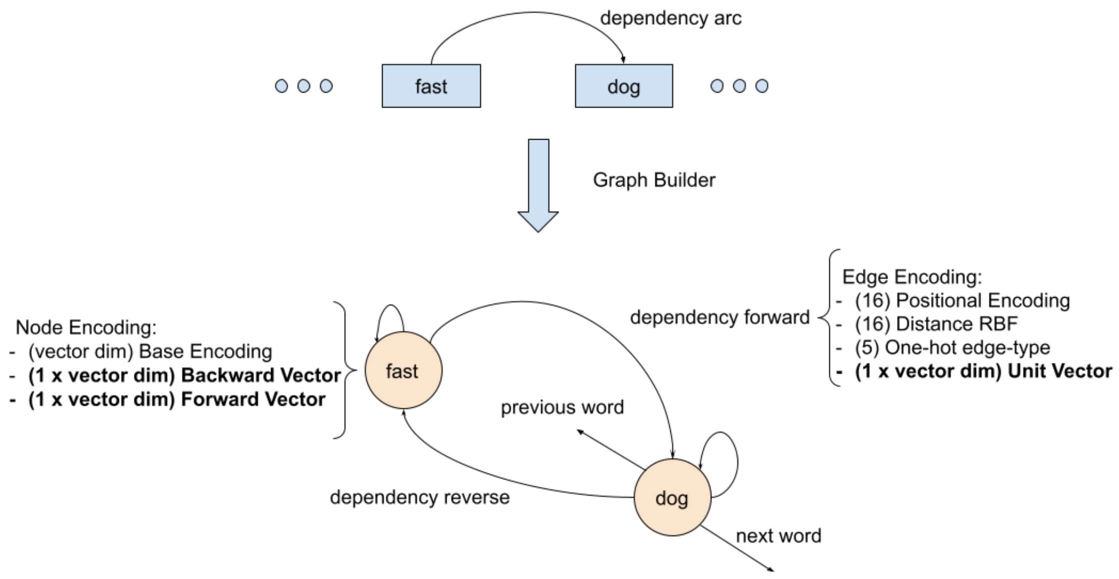


Figure 3: Summary of the NLP encoding process. Each node receives scalar embedding from the BERT model. The remaining scalar and vector channels are established from these. Edges are drawn between neighboring words, dependency arcs, and self-loops. Vector features are in bold for convenience.

3.2 Biochemistry

3.2.1 Structural Parser

To generate biophysical structure information, we leverage the RDKit package/existing pose information to generate a 3D representation of the molecule in space. For the pre-trianing task, we take information at both the node (atom identity) and edge (bond type) levels.

3.2.2 Graph Builder

To generate the improved embeddings, we start by first encoding the molecule as a graph. For node embeddings, we use the original scalar features (of dimension 300) but also add forward and reverse unit vectors in the direction between atoms: $a_{i+1} - a_i$ and $a_i - a_{i-1}$.

For edges, we take the construction in [18]. We draw an edge from vertex i to j if the distance between atoms a_i, a_j is less than some pre-set cutoff distance, representing the physical interactions of that atom up to a particular distance. For each edge, $e^{i,j}$, we take the following features: the unit vector in the direction of $a_j - a_i$, an encoding of the distance between a_i and a_j in the form of Gaussian radial basis functions, and the sinusoidal positional encoding of $j - i$ as described by [25], consisting of the $\frac{d}{2}$ pairs $[\sin(w_k \cdot (i-j)), \cos(w_k \cdot (i-j))]$, where $w_k = \frac{1}{10000^{\frac{2k}{d}}}$, $0 \le k < d = 16, 2|k$. These 32 scalar features are concatenated together to give the edge scalar representation. This is summarized in Figure 4.

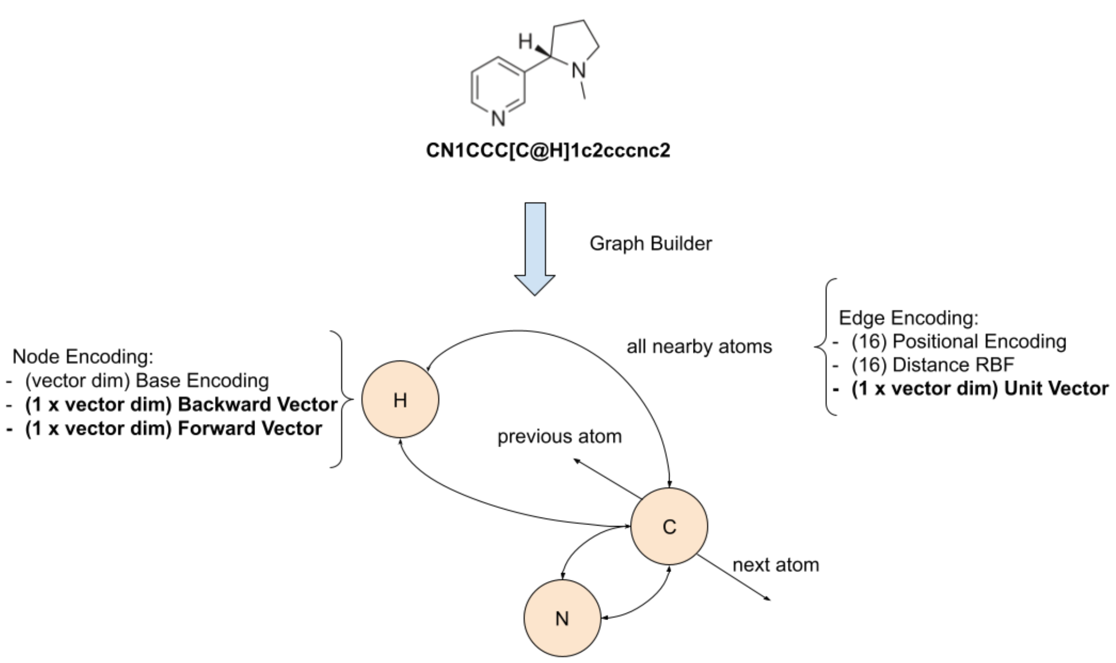

CN1CCC[C@H]1c2cccnc2
Graph Builder
Node Encoding:
- (vector dim) Base Encoding
- **(1 x vector dim) Backward Vector**
- **(1 x vector dim) Forward Vector**
Edge Encoding:
- (16) Positional Encoding
- (16) Distance RBF
- **(1 x vector dim) Unit Vector**
all nearby atoms
previous atom
next atom
H
C
N

Figure 4: Summary of the biochemical encoding process. Each node receives scalar embedding from the MolBERT model. The remaining scalar and vector channels are established from these. Edges are drawn between all close atoms. Vector features are in bold for convenience.

3.3 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
```

Notably, [18] and [2] provide proofs for rotation invariance in three dimensions. We can easily extend these proofs for n-dimensional vector embeddings by replacing the requisite rotation matrices with their n-dimensional equivalents.

3.4 Pretraining Algorithm

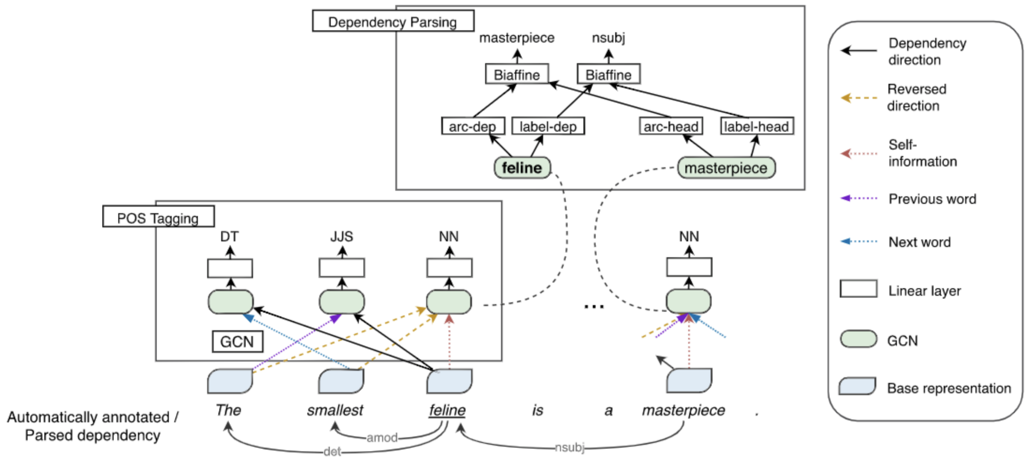


Figure 5: Pretrain process for SIWR model from [1]. For a given sentence, the base embeddings are used as node features for a word graph. These are then fed into a GCN to provide a new word embedding for each word. For the POS task, these are then fed into a softmax classifier to predict the POS for 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.

We take the two subtasks of node prediction and edge prediction to generalize the tasks in Figure 5. 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 $x \in \{\text{node}, \text{edge\_start}, \text{edge\_tail}, \text{label\_start}, \text{label\_tail}\}$. $\mathbf{h^{node}} \in \mathbb{R}^{d_v}$ and the rest $\in \mathbb{R}^{d_e}$. We then get $p(y^{\text{node\_pred}}) = \text{softmax}(\mathbf{W}h^{\text{node}} + \mathbf{b})$. Now to get the probability of all edges, we first take $p(y^{\text{edge\_pred}}) = \text{softmax}(\mathbf{H}^{\text{edge\_start}}\mathbf{W}^{\text{edge}}h^{\text{edge\_tail}})$. Then for each edge label l, we take $s^l = \mathbf{H}^{\text{label\_start}}\mathbf{W}^l h^{\text{label\_tail}}$. 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: $p(y^{\text{edge\_label\_pred}}) = p(y^{\text{label\_pred}}) \odot p(y^{\text{edge\_pred}})$.

We utilize a L2 regularized loss combining the Negative Log Likelihoods of the node prediction and the edge prediction across all required nodes ($\mathbb{V}$) and edges ($\mathbb{E}$). Thus we take:

$$J_{SAE} = J_{node} + J_{edge} + \lambda||\mathbf{W}||_2 = -\sum_{v \in \mathbb{V}} \log(p(y_v^{\text{node\_pred}})_{y^{\text{node\_true}}}) - \sum_{e \in \mathbb{E}} \log(p(y_e^{\text{edge\_label\_pred}})) + \lambda||\mathbf{W}||_2$$

Thus, we not only encode the FM dense representation at nodes, but we also auto-encode the structural edge features as that signal is incorporated into the original graph encoding. Accordingly, this loss will backpropagate that structural signal back to the FM embeddings.

3.5 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 initial base embedding and two layers of pooled embeddings that carry the signal of the syntactic features.

4 Experiments

We construct experiments for both domains in a similar fashion. We first choose a pre-trained FM model to provide base embeddings. We then pre-train the SAE model. Here we may choose to freeze the FM model and only train the GVP layer of the SAE model (SAE only). After pre-training, we may choose to use either the SAE embeddings (SAE) or the base embeddings (BERT). Finally, we apply the models to downstream tasks by appending a final linear layer to the network. For these final tasks we may fine-tune the whole framework (All), just the SAE network (SAE only), just the FM (BERT) or just the final linear layer (None), giving us all the combinations in Table 1.

Table 1: Table of possible experimental combinations

Pretrain	Embedding	Finetune
All	SAE	All
All	SAE	SAE Only
All	SAE	None
All	BERT	BERT
All	BERT	None
SAE Only	SAE	All
SAE Only	SAE	SAE Only
SAE Only	SAE	None
None	SAE	All
None	SAE	SAE Only
None	BERT	BERT

4.1 Data

4.1.1 Pretraining

For NLP pre-training, 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 that the labels are generated through the sentence parser. We take 30000 samples from the Billion Word Benchmark dataset for each of the train and validation sets. [26].

For the biochemical pre-training, we source 30000 from the Zinc20 dataset for each of the train and validation sets. The Zinc20 [27] represents a large screening dataset of commercial chemicals.

4.2 Evaluation method

For the NLP domain, we utilize the GLUE [28] and CoNLL [29] benchmarks for downstream tasks. The GLUE datasets test a variety of language-dependent sentence classification tasks (9 in total) that are highly dependent on embedding quality. The CoNLL benchmark tests token-level Named Entity Recognition (NER), which is also particularly dependent on embedding quality.

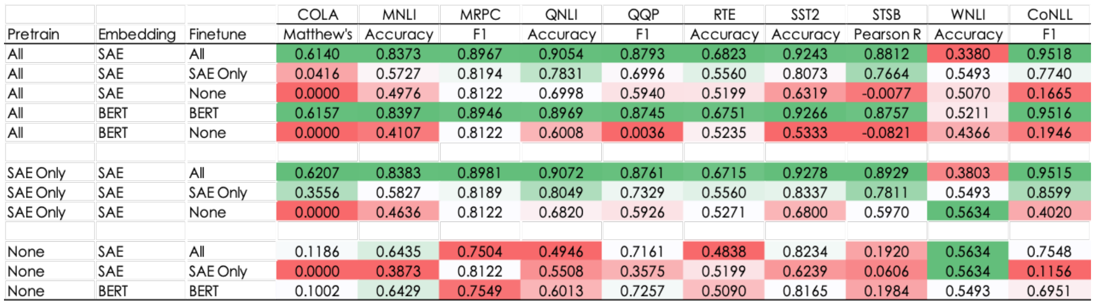
We now enumerate each dataset: The COLA task provides a binary classification of whether or not a particular sentence is grammatically correct. Thus, this task tests a specific grammatical construction. We measure performance using Matthew's Correlation. The MNLI task is a complex prediction of whether or not a premise hypothesis pair consists of entailment, contradiction, or is neutral. We measure performance using accuracy. The MRPC task measures the semantic similarity between two sentences. We measure performance using F1. The QNLI task consists of asking whether or not a given context sentence contains the answer to a given question. We measure performance with accuracy. The QQP task measures the semantic similarity between two questions. Performance is measured using F1. The RTE task test a binary classification of sentence entailment. In the SST2 task, the model must classify the binary sentiment of a movie-review. Performance is measured using accuracy. The STSB task categorizes the relationship between pairs of text into one of 5 categories. We report the Pearson R for this task as is standard. The WNLI task is a reading comprehension task that tests the model to provide the referent for a given pronoun. We measure performance using accuracy. As stated previously, the CoNLL tasks tests NER at a token level. We measure performance using an F1 score taken across all tokens.

For the biochemical domain, we test on the SMP dataset from Atom3D [30]. The SMP task requires predicting the physiochemical properties of small molecules, a common real-world task. The SMP dataset provides labels for 20 tasks, and we provide comparisons across all tasks. Performance is reported in terms of the cumulative mean absolute error (MAE) across the test set.

5 Results

5.1 NLP

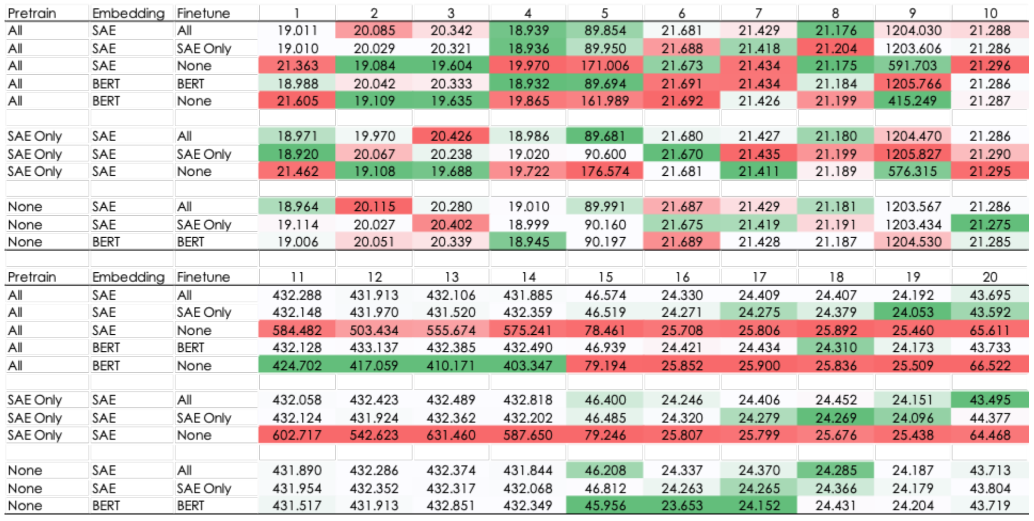
Table 2: Table of results for NLP tasks (GLUE and CoNLL).



			COLA	MNLI	MRPC	QNLI	QQP	RTE	SST2	STSB	WNLI	CoNLL
Pretrain	Embedding	Finetune	Matthew's	Accuracy	F1	Accuracy	F1	Accuracy	Accuracy	Pearson R	Accuracy	F1
All	SAE	All	0.6140	0.8373	0.8967	0.9054	0.8793	0.6823	0.9243	0.8812	0.3380	0.9518
All	SAE	SAE Only	0.0416	0.5727	0.8194	0.7831	0.6996	0.5560	0.8073	0.7664	0.5493	0.7740
All	SAE	None	0.0000	0.4976	0.8122	0.6998	0.5940	0.5199	0.6319	-0.0077	0.5070	0.1665
All	BERT	BERT	0.6157	0.8397	0.8946	0.8969	0.8745	0.6751	0.9266	0.8757	0.5211	0.9516
All	BERT	None	0.0000	0.4107	0.8122	0.6008	0.0036	0.5235	0.5333	-0.0821	0.4366	0.1946
SAE Only	SAE	All	0.6207	0.8383	0.8981	0.9072	0.8761	0.6715	0.9278	0.8929	0.3803	0.9515
SAE Only	SAE	SAE Only	0.3556	0.5827	0.8189	0.8049	0.7329	0.5560	0.8337	0.7811	0.5493	0.8599
SAE Only	SAE	None	0.0000	0.4636	0.8122	0.6820	0.5926	0.5271	0.6800	0.5970	0.5634	0.4020
None	SAE	All	0.1186	0.6435	0.7504	0.4946	0.7161	0.4838	0.8234	0.1920	0.5634	0.7548
None	SAE	SAE Only	0.0000	0.3873	0.8122	0.5508	0.3575	0.5199	0.6239	0.0606	0.5634	0.1156
None	BERT	BERT	0.1002	0.6429	0.7549	0.6013	0.7257	0.5090	0.8165	0.1984	0.5493	0.6951

5.2 Biochemical

Table 3: Table of results for SMP dataset tasks.



Pretrain	Embedding	Finetune	1	2	3	4	5	6	7	8	9	10
All	SAE	All	19.011	20.085	20.342	18.939	89.854	21.681	21.429	21.176	1204.030	21.288
All	SAE	SAE Only	19.010	20.029	20.321	18.936	89.950	21.688	21.418	21.204	1203.606	21.286
All	SAE	None	21.363	19.084	19.604	19.970	171.006	21.673	21.434	21.175	591.703	21.296
All	BERT	BERT	18.988	20.042	20.333	18.932	89.694	21.691	21.434	21.184	1205.766	21.286
All	BERT	None	21.605	19.109	19.635	19.865	161.989	21.692	21.426	21.199	415.249	21.287
SAE Only	SAE	All	18.971	19.970	20.426	18.986	89.681	21.680	21.427	21.180	1204.470	21.286
SAE Only	SAE	SAE Only	18.920	20.067	20.238	19.020	90.600	21.670	21.435	21.199	1205.827	21.290
SAE Only	SAE	None	21.462	19.108	19.688	19.722	176.574	21.681	21.411	21.189	576.315	21.295
None	SAE	All	18.964	20.115	20.280	19.010	89.991	21.687	21.429	21.181	1203.567	21.286
None	SAE	SAE Only	19.114	20.027	20.402	18.999	90.160	21.675	21.419	21.191	1203.434	21.275
None	BERT	BERT	19.006	20.051	20.339	18.945	90.197	21.689	21.428	21.187	1204.530	21.285

Pretrain	Embedding	Finetune	11	12	13	14	15	16	17	18	19	20
All	SAE	All	432.288	431.913	432.106	431.885	46.574	24.330	24.409	24.407	24.192	43.695
All	SAE	SAE Only	432.148	431.970	431.520	432.359	46.519	24.271	24.275	24.379	24.053	43.592
All	SAE	None	584.482	503.434	555.674	575.241	78.461	25.708	25.806	25.892	25.460	65.611
All	BERT	BERT	432.128	433.137	432.385	432.490	46.939	24.421	24.434	24.310	24.173	43.733
All	BERT	None	424.702	417.059	410.171	403.347	79.194	25.852	25.900	25.836	25.509	66.522
SAE Only	SAE	All	432.058	432.423	432.489	432.818	46.400	24.246	24.406	24.452	24.151	43.495
SAE Only	SAE	SAE Only	432.124	431.924	432.362	432.202	46.485	24.320	24.279	24.269	24.096	44.377
SAE Only	SAE	None	602.717	542.623	631.460	587.650	79.246	25.807	25.799	25.676	25.438	64.468
None	SAE	All	431.890	432.286	432.374	431.844	46.208	24.337	24.370	24.285	24.187	43.713
None	SAE	SAE Only	431.954	432.352	432.317	432.068	46.812	24.263	24.265	24.366	24.179	43.804
None	BERT	BERT	431.517	431.913	432.851	432.349	45.956	23.653	24.152	24.431	24.204	43.719

6 Discussion

6.1 NLP

In the COLA dataset, we note that pre-training dramatically improves performance and that embeddings are required to be task-specific. This is evidenced by the fact that the None fine-tuning setting performs poorly across the board.

Further, the SAE methods with all layers fine-tuned perform the best (scores of 0.6207 and 0.6140). This trend is generally consistent across the tasks (MNLI: 0.8373 and 0.8383 versus 0.6429, MRPC: 0.8967 and 0.8981 versus 0.7549, etc.). The only notable exception is WNLI. In this task, the model does quite poorly over the pre-train setting, indicating that the pre-train task is somewhat orthogonal to the task.

In STSB, notably, the two SSE Only fine-tuned models also perform admirably, showing there are gains to be had in fine-tuning the SAE GVP layers (0.7664 and 0.7811). This effect is similar in the CoNLL dataset showing there are gains to be had in fine-tuning the SAE GVP layers (0.7740 and 0.8599).

For COLA, we also note that pre-training the FM and then fine-tuning provides significant gains compared to fine-tuning the SAE only on pre-trained SAE embeddings (0.6157 versus 0.3556). This indicates that the raw base FM embeddings may not be sufficiently adaptable to this task. For MRPC, however, it appears much of the benefit is achieved from fine-tuning the BERT model itself, with the non-pre-trained model achieving a score of 0.6429.

The SAE framework has strong properties as a pretraining module. For QNLI, the pre-trained SAE embeddings outperform the baseline even without fine-tuning (0.6998, 0.6820). In the QQP and SST2 tasks, for the SAE model, we see that pre-training and fine-tuning is nearly identical, comparing rows 3 and 10.

6.2 Biochemical

For the SMP dataset, we observe far more erratic behavior compared to the NLP case. This shows that perhaps the SMP data is not as readily data-scalable compared to the general NLP tasks above. Interestingly, some tasks do not have any performance impact from training (tasks 10, 16-18 for example). For other tasks, such as tasks 4,5,8, and 20, the SAE method provides similar gains when fine-tuning all. We also remark on the overfitting issue of the datasets, where, for example, tasks 2, 11, and 12 display the best results when just finetuning the final projection head rather than all layers.

7 Conclusion

Overall, this study contributes a novel EGNN model architecture for boosting FM embeddings by incorporating syntactical information. The model outperforms baseline networks in a wide-variety of tasks and domains. As the applications of deep learning in many AI domains grow, the importance of good embeddings grows 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 foundational models. For future work, there is still space to leverage the strong pre-training abilities of the model at greater scale. Further, introducing mechanisms to weigh the balance between originally encoded information and new syntactic information is important to understand in order 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] Juncai Li and Xiaofei Jiang. Mol-bert: an effective molecular representation with bert for molecular property prediction. *Wireless Communications and Mobile Computing*, 2021:e7181815, September 2021.

[22] Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, and Michal Linial. ProteinBERT: a universal deep-learning model of protein sequence and function. *Bioinformatics*, 38(8):2102–2110, April 2022.

[23] Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[24] Joakim Nivre and Jens Nilsson. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[25] 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.

[26] 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.

[27] John J. Irwin and Brian K. Shoichet. Zinc – a free database of commercially available compounds for virtual screening. *Journal of chemical information and modeling*, 45(1):177–182, 2005.

[28] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: a multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, 2018. Association for Computational Linguistics.

[29] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.

[30] Raphael J. L. Townshend, Martin Vögele, Patricia Suriana, Alexander Derry, Alexander Powers, Yianni Laloudakis, Sidhika Balachandar, Bowen Jing, Brandon Anderson, Stephan Eismann, Risi Kondor, Russ B. Altman, and Ron O. Dror. Atom3d: tasks on molecules in three dimensions, January 2022. arXiv:2012.04035 [physics, q-bio].