bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

Evaluating Representations for Gene Ontology Terms

Dat Duong, Ankith Uppunda, Chelsea Ju, James Zhang, Muhao Chen, Eleazar Eskin, Jingyi Jessica Li, Kai-Wei Chang.

University of California Los Angeles, California, USA.

datdb@cs.ucla.edu, eeskin@cs.ucla.edu, jli@stat.ucla.edu, kwchang@cs.ucla.edu

Abstract

Recently, as the number of new protein sequences being collected is rising at a faster pace than the number being annotated, there have been efforts in developing better methods for predicting protein functions. Because protein functions are annotated by Gene Ontology (GO) terms, one key auxiliary resource is the GO data itself. GO terms have definitions consisted of a few sentences describing some biological events, and are arranged in a tree structure with specific terms being child nodes of generic terms. The definitions and positions on the GO tree of the GO terms can be used to construct their vector representations. These vectors can then be integrated into existing prediction models to improve the classification accuracy. In this paper, we adapt two neural network architectures, Embeddings from Language Models and Bidirectional Encoder Representations from Transformers, to encode GO definitions into vectors. We evaluate these new encoders against the previous definition and position encoders in three tasks: (1) measuring similarity between GO terms (2) asserting relationship for orthologs and interacting proteins based on their GO annotations and (3) predicting GO terms for protein sequences. Results in task 1 and 2 find that BERT-based encoders are often better than the other kinds of encoders. Result in task 3 shows that using GO vectors as additional prediction features increases the accuracy, primarily for GO terms with low occurrences in the dataset. In task 3, we also observe that having GO vectors as features definitely helps, but the choice of encoders does not greatly affect the outcome.

1 Introduction

The Gene Ontology (GO) provides descriptions for functions of genes and proteins [9]. This database [1] contains terms referred to as GO terms, each term has a definition describing some biological events. To clearly annotate the locations and functions of proteins, this database is further divided into three smaller ontologies: cellular components (CC), molecular functions (MF) and biological processes (BP). In each smaller ontology, the GO terms are arranged into a directed tree with one single root (GO tree), where terms describing more specific biology functions are child nodes of more generic terms.

In late 2017, [33] reported that only about 1% of the proteins in the GO database have manually verified annotations. With the advancement of sequencing technology, this fraction is expected to drop in the incoming years; hence, there have been great efforts in developing methods to automatically assign GO terms for unknown sequences [11, 19, 27, 33, 34]. The manually annotated data, which is often used as training and evaluation sets, have many GO terms annotating only a few proteins; for example, [27]

[1] https://www.uniprot.org/

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

estimate that about half of the GO terms annotate about 10 proteins in Human and Yeast database. To increase the prediction accuracy, automatic annotation methods have been using two additional data resources.

The first data resource is the sequence-to-sequence relationship. For example, protein-protein interaction network and structural homology are often used to constrain the fact that closely related proteins should have similar GO labels [4, 33]. The second data resource is in fact the Gene Ontology itself. For example, distance metric for GO terms can be inferred from the GO tree or their definitions, and then be used as the intuitive constraint that forces similar terms to have equivalent prediction probabilities for a given protein sequence [27]. More importantly, for rare label prediction problems, works in other research domains have shown that using vector representations of labels as one of the features can boost the classification accuracy [2, 21, 30].

This paper will focus on the second data resource, the Gene Ontology itself. On this end, there have been efforts in developing distance metric for GO terms [8, 13, 15, 17, 20, 24, 32]. Most traditional methods for computing semantic similarity of GO terms rely on the Information Content (IC) and the GO tree. For two GO terms, the key idea is to first retrieve the shared common ancestors and then weigh these nodes with their IC values. For example, the most basic method [20] takes the maximum IC of the shared ancestors as the similarity score for two given GO terms. Methods based on shared ancestors and IC values have two drawbacks. First, they do not consider the definitions of the GO terms which have been shown to yield better semantic similarity scores in many cases [8, 15]. Second, they are unable to create vector representations of GO terms which then can be integrated into other annotation models to predict functions for protein sequences.

In recent years, with the advancement of computing power, neural network (NN) encoders have been introduced to map GO terms into vectors based on the principle that the vectors of related terms should have similar values [8, 24]. Once the GO vectors are created, then their distance metric naturally follows; for example, cosine similarity or Euclidean distance can be applied. Thus, NN encoders solve the same problem as IC models, and also provide GO vectors which later can be integrated into known annotation methods. Typically, NN encoders are divided into two classes; they either transform GO definitions or the GO entities (e.g. GO names) into vectors. For example, consider two recent methods [8] and [24]. [8] apply Long-short Term Memory on the GO definitions; whereas, Onto2vec in [24] apply Word2vec on axioms, for example “GO:0060611 is_subclass GO:0060612", to capture relatedness of GO entities by using the vectors representing their GO names.

In principle, both types of encoders solve the same problem; however, one key question is: in practice, which type of encoder tends to be better? Unfortunately, there have not been any extensive works comparing these encoders. Moreover, despite these methods providing the GO vectors, there have not been works assessing how do these vectors affect the prediction of GO labels for protein sequences; for example, do these GO vectors indeed increase the annotation accuracy, and what types of GO labels benefit the most from having GO vectors as extra features?

In this paper, we address these questions and introduce a few more encoders based on two recent neural network architectures which have attain state-of-the-art results in many language tasks like textual entailment, name entity recognition, sentimental analysis, and language translation. These two architectures are Embeddings from Language Models

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

(ELMo) and Bidirectional Encoder Representations from Transformers (BERT) [7, 18].

The core of ELMo is the Bidirectional Long Short Term Memory (BiLSTM) encoder. Encoding GO definitions via a single layer BiLSTM encoder has been studied before in [8]; in this paper, we compare how their method performs against ELMo which has two sequential layers of BiLSTM encoders where the output of the first layer is the input of the second layer. To transform a GO definition into a vector, we take the weighted average of the output of the two BiLSTM layers, and then compute the simple mean of the final word vectors to represent a GO definition.

BERT is entirely different from the other previous neural networks in that it does not use any LSTM or convolution layer. Rather, BERT uses a 12-layer attention mechanism based on the Transformer encoder [7, 26]. Loosely speaking, for one input sentence having L words, at the layer i and the word j, the word vector w_{ij} is a weighted average of the vectors $w_{i-1,k}$ for all $k \in [1, L]$. BERT outputs a matrix embedding where column j corresponds to j^{th} word in the input. BERT original implementation does not return a vector representation for each sentence in its input; in fact, encoding a sentence was not a key objective in the original paper [7]. At the time of writing this paper, there has not been any consensus of how to represent a sentence from BERT.

In this paper, we introduce and evaluate five sentence encoders based on BERT to transform a GO definition into a vector. First, conditioned on two given GO terms, we train BERT to (1) predict missing words in the two definitions and (2) test if the second definition follows the first one (e.g. if the two GOs are child-parent terms). To extract the vector for one GO definition, we average the token embedding in layer 11 of BERT (to be explained later). Second, we average the word embedding in layer 12 of BERT to represent one GO definition. Third, following ELMo we take the weighted average of the output in layer 11 and 12. Fourth, we use the header token of the GO definition to represent the entire definition. Fifth, we reuse the original BERT framework, and simply replace the GO definitions with the GO names so that we convert GO names into vectors; this idea is similar to Onto2vec [24].

We evaluate the ELMo and BERT encoders along with BiLSTM [8], Graph Convolution Network [10], and Onto2vec [24] in three downstream tasks: (1) measuring semantic similarity between GO terms, (2) asserting relationship for orthologs and interacting proteins based on their GO annotations and (3) predicting GO terms for protein sequences. For tasks 1 and 2, we include two IC methods: Resnik and Aggregated Information Content (AIC) [20, 25]. IC methods do not return vector representations for GO terms, and so are not used in task 3. In task 1 and 2, neural network encoders can outperform IC methods only when the data are well annotated with GO terms having high IC values. In task 3, using GO vectors as prediction features increases the annotation accuracy, primarily for rare GO terms. In all the tasks, definition encoders are often better than entity encoders. Within definition encoders, BERT-based encoders are usually better than LSTM-based encoders. Our code and data are at `https://github.com/datduong/EncodeGeneOntology`.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

2 Methods

In this paper, we use the word *encoders* to refer to NN methods that transform GO terms into vectors. Typically, there are two types of encoders. *Sentence encoders* convert the definitions of GO terms into vectors; by default, terms describing related biology events will have similar vectors. *Entity encoders* treat a GO term as a single entity and encode it into a vector based on its position in the GO tree without using its definition. Here, terms within the same neighborhood in the GO tree will have similar vector values. We will first describe the sentence encoders, and then the entities encoders.

2.1 Sentence encoders

2.1.1 BiLSTM

We first describe the Bidirectional Long Short Term Memory (BiLSTM) model to encode sentences into vectors. BiLSTM provides contextualized vector for each word in a sentence, so that the same word will have different vectors depending on its position in the sentence and the surrounding words. We begin with the input of BiLSTM which is usually the Word2vec encoder. Word2vec assigns similar vectors to words with related meanings or are likely to occur in the same sentence [14, 22]. We train our own Word2vec using open accessed papers on Pubmed following the setting in [8] where the word dimension is 300. For this process, we keep stop-words (e.g. but, and not) and symbols like $+$ and $-$ because they may have important biological meanings such as positive and negative charged molecules.

Given one sentence, when using Word2vec, we would convert the sentence into a matrix M where each column M_j is vector for the word at position j in the sentence. Regardless of the sentence, the same word is always assigned to the same vector. To capture the fact that the same word can have different meanings depending on its position in the sentence, we apply $\tilde{M} = \text{BiLSTM}(M)$ where the same word in $\tilde{M}$ will have different vectors. For example, consider the word vector M_j at position j in a sentence of length L. BiLSTM computes the forward and backward LSTM model to produce the output vectors $\overrightarrow{h_j} = \text{LSTM}(\overrightarrow{h}_{j-1}, M_j)$ and $\overleftarrow{h_j} = \text{LSTM}(\overleftarrow{h}_{j+1}, M_j)$ and then returns $\tilde{M}_j = [\overrightarrow{h_j}; \overleftarrow{h_j}]$ where $[\overrightarrow{h_j}; \overleftarrow{h_j}]$ indicates the concatenation of the two vectors into one single vector.

To encode a matrix of words into a vector of a sentence, we take max-pooling across the columns of $\tilde{M}$, $\text{maxpool}(\tilde{M})$ [5]. Next, we apply one linear transformation to this *aggregated* vector to produce a final representation of the GO definition. We set the BiLSTM hidden layer at 1024, and apply one final linear layer of size 768. During training, we freeze the input M and update only BiLSTM parameters.

2.1.2 ELMo

Embeddings from Language Models (ELMo) improves the BiLSTM encoder in two key steps [18]. First, instead of representing a whole word as a vector, ELMo represents each character in the alphabet as a vector and then uses convolution filters of varying sizes to transform the alphabet vectors into a word vector. Second, ELMo trains a 2-layer

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

BiLSTM. The first BiLSTM input are the word vectors from the character layer, and the second BiLSTM input are the output of the first BiLSTM. The final vector for one word is a weighted average of the word vector from the character layer, and the output of the first and second BiLSTM, where the weights are computed for a given specific task and thus are jointly trained with the other parameters. In this paper, we load the ELMo pretrained on Pubmed [2], freeze the character convolution layer, and train only the two biLSTMs. Borrowing notations from the previous section, let $\tilde{M}_j^l$ be the BiLSTM output of layer l. Our final vector for a word in a GO definition is $a_j\tilde{M}_j^1 + (1 - a_j)\tilde{M}_j^2$ where $a_j \in [0, 1]$ is specific to position j and is jointly trained with the two biLSTMs. To encode a sentence (e.g. GO definition), we take mean-pooling over ELMo final token embedding matrix. Default ELMo output embedding is size xxx; thus, to match pre-trained BERT, we pass this *aggregated* vector through a linear layer sized 768 to produce a final vector representation of the GO definition.

2.1.3 BERT

We provide a high-level explanation for BERT. Like BiLSTM, BERT converts words in an input sequence (which can be more than one sentence) into a contextualized embedding where the same word has different vector representations depending on its position in the sequence and the surrounding words. Unlike BiLSTM, BERT's key internal structure is the Transformer framework which relies on attention mechanism and will be described below.

We will use this BERT architecture to capture the relationship of the GO definitions. Consider the example in Figure 1, where the input sequence is the child-parent description *perforation plate* (GO:0005618) and *cellular anatomical entity* (GO:0110165). We are using the short descriptions in this example, but in the experiment we will use the complete descriptions. To capture the relationship that *perforation plate* is a *cellular anatomical entity*, we input both sentences into BERT (Fig. 1).

In the first step, BERT splits each word into smaller segments called tokens; for example the word *perforation* is segmented into 3 tokens *per ##fo ##ration*. We use the same segmentation rule as in the original paper [7]. The symbol ## is only a naming convention and bares no significant meaning. For our example, BERT processes the GO terms into the format *[CLS] per ##fo ##ration plate [SEP] cellular an ##ato ##mic ##al entity [SEP]*, where the special token [CLS] denotes the start of the whole input and [SEP] denotes end of each sentence [7]. BERT internal structure is the Transformer encoder which is described in detail in [26]. Here, we briefly describe the key idea in Transformer. Transformer has several independent heads, each using its own attention mechanism. Loosely speaking, for each head h in the layer i, the output vector $o_{i,j}^h$ for the word at position j is computed

[2] https://allennlp.org/elmo

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

as a weighted average

$$o^h_{i,j} = \sum_{k \in \{1:L\}} a^h_{ik} V^h(w_{ik}) \tag{1}$$

$$a^h_{ik} = \text{softmax}(e^h_{ik}) \tag{2}$$

$$e^h_{ik} = Q^h(w_{ij})^\intercal K^h(w_{ik}) \tag{3}$$

where L is input length, and V^h, Q^h, K^h are transformation functions. To merge all the heads at the layer i, Transformer concatenates the output $o^h_{i,j}$ at the position j of all the heads, and then applies a linear transformation on this concatenated vector. The output of this linear transformation o_{ij} at position j is then passed onto the next layer $i+1$.

The input of the first layer denoted as w_{0j} is a summation of the token, position and type embedding. Token embedding is analogous to the Word2vec embedding where every token is assigned to exactly one vector. Position embedding assigns a vector to each location in the sentence; in our example, we would add the position vector p_1 to the token at position 1 which is *per*. Type embedding assigns the same vector v_1 to tokens in the first sentence, and the same vector v_2 to tokens in the second sentence. In our example, we would add the same vector v_1 to each token vector in the first sentence which are *[CLS] per ##fo ##ration plate [SEP]*.

We use the same hyper-parameters as the original BERT in [7], where the Transformer encoder has 12 heads, 12 layers, and the linear transformation matrix produces a vector size 768. The final results are 12 layers of embedding, each with size $768 \times L$. Based on the framework of Transformer, the final output vector of the token [CLS] is a function of all the other words in both GO definitions, and therefore can be viewed as an aggregated representation of both GO definitions. For this reason, in the original implementation [7], the embedding of [CLS] in layer 12 is passed through a full connected layer to predict if *perforation plate* is a *cellular anatomical entity*. To ensure that BERT returns high probability for this example case, we will need to train the BERT parameters with respect to the context of the Gene Ontology.

We use the Pytorch BERT code [3] and initialize the parameters with a BERT pretrained on Pubmed [12]. Following [7], our BERT model is trained with two tasks: masked language model and next-sentence prediction. Masked language task randomly removes words in a sentence, and then uses the remaining words to predict the missing words. Next-sentence task estimates if two sentences are sequential or chosen randomly from the corpus. Next-sentence prediction uses the [CLS] embedding in layer 12 to make the final decision as described above. In our example, the next-sentence task should confirm that the two sentences are sequential. To train, we create our own data with respect to the context of the Gene Ontology. To create one *document*, we concatenate the definitions of all GO terms in one single branch of the GO tree, starting from the leaf node to the root. We consider only is-a relation, and randomly select only one parent if the given node has many parents. Our fine-tune BERT will capture the relationships of words within a sentence, and also the relationships of GO definitions that are on the same path to the root nodes.

[3] `https://github.com/huggingface/pytorch-pretrained-BERT`

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

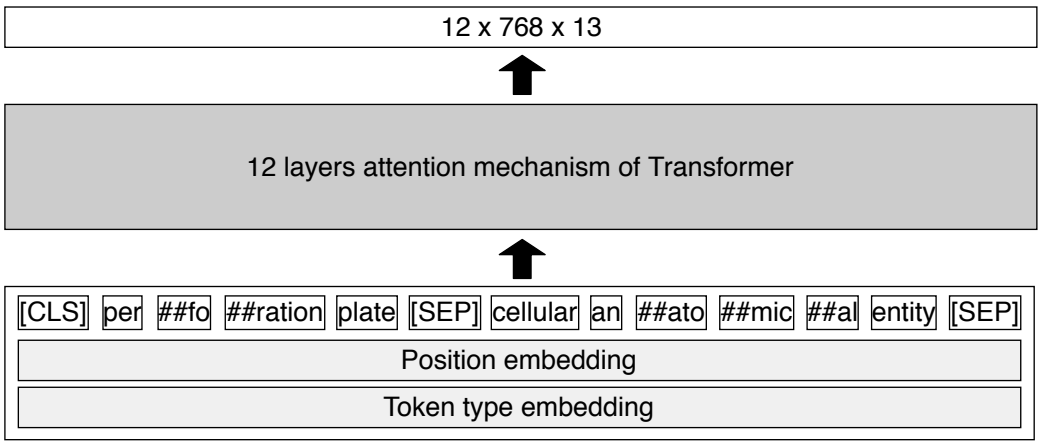

Figure 1: Consider child-parent terms GO:0005618 (perforation plate) and GO:0110165 (cellular anatomical entity). We input into BERT the tokenized words of *perforation plate* and *cellular anatomical entity*. This illustration uses the short descriptions, but in the experiment we will use the complete descriptions. The [CLS] specifies the start of the whole input, and [SEP] specifies the end of each sentence. At the lowest level, BERT has three embedding layers: word, position, and token type. For example, [SEP] appears twice but will have different position vector representations. Each token in the first segment *[CLS] per ##fo ##ration plate [SEP]* will have toke type 1 embedding, whereas the rest of the input will have toke type 2 embedding. In general, the type 1 embedding is assigned to words in the first sentence, and type 2 embedding is meant for words in the second sentence. For each token, BERT adds the embedding of the three layers and then send this summation into the Transformer encoder. BERT outputs 12 layers of embedding size $12 \times 768 \times 13$ for this example, where 13 is the total length of both GO terms including the [CLS] and [SEP] tokens.

We emphasize that by default, BERT does not provide a vector representation for a given GO definition. BERT only provides the matrix embedding for the words in a GO definition. After tuning BERT, we test two methods to retrieve the vector representations for the GO definitions from the word embedding matrix.

For our first method (BERTas), we follow Bert-as-service [29] and do not further update any the model parameters. To transform the GO description *perforation plate* into a vector, we input it into BERT as *[CLS] per ##fo ##ration plate [SEP]* without any second sentence which would be *cellular anatomical entity* in the example in Fig. 1. Then we average the vectors of all the word tokens with the [CLS] and [SEP] token in layer 11. [29] recommends this layer because layer 12 may be too affected by the masked language model and next-sentence prediction task instead of our key objective which is to extract the sentence representation. Because we use the same hyper-parameters as the original BERT, by default, BERTas returns a GO vector of length 768.

For the second method, we continue training the BERT parameters; however, we will apply a new objective loss function and train on the dataset described in section 2.3. We remove masked language model so that our only task is next-sentence prediction. We emphasize that this task will be trained differently from the original paper by [7]. In this new task, unlike [7] we do not append two GO definitions into one single input, and then

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

use the [CLS] token to make the classification which relies on cross-entropy loss. Instead, we first use BERT to encode two GO definitions into two vectors, and then measure their cosine distance. Intuitively, two similar GO definitions will have high cosine distance, and vice versa. Our new objective loss function is to maximize cosine distance of child-parent GO definitions, and vice versa. Unlike [7] our approach first requires an explicit method to encode the GO definitions into vectors, and then applies the cosine loss function to these vectors. We will evaluate three different ways to encode the GO definitions: BERT12, BERT11+12, and BERTCLS.

BERT12 follows similar idea in BERTas. For a GO term, we send only its definition through BERT (not appending definitions of parent terms), and then average the token embedding in the layer 12. We add one extra linear layer to transform this *mean* vector to retrieve the final representation of the GO definition.

For BERT11+12, we follow the idea of ELMo. We take the weighted average of the output in layer 11 and 12 so that the final vector of the token at position j is $o_j = a_j o_{11,j} + (1 - a_j) o_{12,j}$ where $a_j \in [0, 1]$ is jointly trained with the other parameters. To encode a GO definition, we take the mean of o_j over j and then linear-transform this vector.

In BERTCLS, for each GO term, we again send only its definition through BERT. Next, we use the *pooled output* of layer 12 which is simply the [CLS] token in layer 12 transformed by a linear layer with Tanh activation. We pass this *pooled output* through one more linear layer to produce the final vector representation of the GO definition.

In BERT12, BERT11+12 and BERTCLS, the final linear transformation returns an output of size 768 to match BERTas output which is 768. To review, for BERT12, BERT11+12 and BERTCLS, given two GO terms, we independently transform each of their definitions into a vector (by individually sending each definition through BERT, and not by concatenating them first as one long sentence). Then for training, our loss function is to maximize or minimize the cosine distance of these two vectors depending on whether the terms are child-parent or randomly chosen.

2.2 Entity encoders

Because GO terms are arranged in a directed tree, we can treat a GO term as a single entity and encode it into a vector without using its definition. In this paper, we test Graph Convolution Network (GCN) and Onto2vec. There are other node embedding methods, but GCN has shown to work well in practice for prediction tasks when labels have low occurrence frequencies [10, 21, 30].

2.2.1 GCN

Graph Convolution Network encodes each GO term in the tree into a vector [10]. Let A be the adjacency matrix, where $A_{ij} = 1$ if GO_i is the parent of GO_j. Compute $\tilde{A} = A + I$, where I is identity matrix. Compute the degree matrix $\tilde{D}$, where $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Next scale A into $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$. Let W_1 and W_2 be two transformation matrices. Define X to be the initial vector embedding for the GO terms, where a column in X corresponds to a GO vector. Before training, X is initialized with random numbers. During training X is

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

transformed into a new matrix $E = W_2\hat{A}\,\text{relu}(W_1\hat{A}X)$. Loosely speaking, one column i in $\hat{A}X$ is some summation of all its neighbor nodes and itself. $W_1\hat{A}X$ then transforms this summation into a new vector. We repeat this transformation twice as in [10, 21]. At the end, column i in E is the vector for GO_i and is a function of its child nodes. We train GCN to minimize the cosine distance loss of the column vectors in E using the data in section 2.3. We set GCN to produce the final vector representation of size 768, same as BERTas.

2.2.2 Onto2vec

Onto2vec encodes GO terms into vectors by transforming their relationships on the GO tree into sentences, which are referred to as axioms in the original paper [24]. For example, the child-parent GO terms GO:0060611 and GO:0060612 are rewritten into the following sentence "GO:0060611 is_subclass GO:0060612". Onto2vec then applies word2vec on these sentences, so that GO names occurring in the same sentence are encoded into similar vectors. Because the training *sentences* are constructed from the GO trees without GO definitions, Onto2vec can conceptually be viewed as method that encodes nodes on graph into vectors like GCN. Because word2vec objective function is based on cosine similarity, for Onto2vec, GO terms in close proximity will have high cosine similarity score. We set Onto2vec to produce the final vector representation of size 768, same as BERTas.

2.2.3 BERT as entity encoder

Following Onto2vec, we apply BERT as an entity encoder (our BERTname) where the key objective is to encode the GO names into vectors. We create training data as follows. For each GO term, we select one path from that term to the root node via only is_a relation. For each path, we split the set of GO terms into half so that they represent the *first* and *second* sentence. BERTas and BERTname have similar idea. In BERTas, the training step requires GO definitions, whereas this phase in BERTname uses only the GO names. For example, consider the path GO:0000023, GO:0005984 GO:0044262, GO:0044237, and GO:0008152. In BERTname, we format it into the input *[CLS] GO:0000023 GO:0005984 [SEP] GO:0044262 GO:0044237 GO:0008152 [SEP]*. Next, we set the words in the BERT vocabulary as the GO names. Then, we train masked language model and next-sentence prediction on this data, so that we can capture relatedness among the GO names like Onto2vec. We use the same hyper-parameters as original BERT, where the final token embedding is size 768. The final vector representation for GO terms is the BERT's initial token embedding. We do not take the last layer output because we do not want the contextualized vectors of the GO names which will vary depending on their locations in the input sequence and the surrounding words.

2.3 Training data

We train BERTas and BERTname using the data and fine-tune procedure described in section 2.1.3 and 2.2.3. We train all the other encoders using the data described here. Our objective is to first use the encoders to transform GO terms into vectors, and then to maximize and minimize cosine distance for child-parent and unrelated GO pairs sampled

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

from the GO database For our training data, we treat the BP, MF and CC ontology as one connected network; this approach has shown to increase accuracy for downstream tasks [8]. We randomly pair a GO with one of its parents, treating the follow one-directional relationship "is a", "part of", "regulates", "negatively regulates", and "positively regulates" as the same edge. To ensure that these child-parent terms are very similar, we compute their Aggregated Information Content (AIC) scores and retain pairs with scores above the median [25].

To create the negative dataset where each sample is a pair of two randomly chosen GO terms, we sample two types of unrelated pairs. For the first type, we randomly pick about half the GOs seen in the positive dataset. We pair each term c in this set with a randomly chosen term d. For the second type, we pair the same term d with another randomly chosen term e. This strategy helps the encoders by letting the same GO terms to be seen under different scenarios. Next, to ensure that these random pairs are very dissimilar, we retain pairs with AIC scores below the median. This training data is available at `https://github.com/datduong/EncodeGeneOntology`.

3 Evaluation

We evaluate the GO encoders in three tasks. First, we measure the semantic similarity for two types of GO pairs: child-parent and unrelated terms. The objective is to determine which encoders can best distinguish the two kinds of GO pairs. Here, we also observe how the number of neighbors (degrees) and ICs of the GO terms affect their similarity scores. Second, we assert the relationship for orthologs and interacting proteins based on their GO annotations. Here, we consider only manual annotations data. This task is similar the first task; that is, if an encoder does well in task 1 then it is likely to do well in task 2. However, task 2 provides a more holistic picture because in practice, genes and proteins are not often manually annotated by uninformative GO terms which have high degrees and low IC values. Hence, a method can possibly perform well in task 2 even if it does not do well in task 1. Third, we edit the DeepGO model so that it takes the GO vectors as an extra input. We test if the GO vectors indeed boost the accuracy for predicting GO labels of protein sequences. Moreover, suppose the GO vectors improve the accuracy, then we want to know if the increment occurs for rare GO labels. In some essence, being able to well predict rare GO labels is important because these terms are often located lower in the GO tree, describe more detail biology events, and are closer to the true properties of the proteins. For example, predicting GO:0005618 *perforation plate* is more precise to a protein's location than predicting its parent term GO:0110165 *cellular anatomical entity* or its ancestor GO:0005575 *cellular component*.

3.1 Semantic similarity task

Theoretically speaking, a good GO encoder will clearly separate child-parent GO terms from unrelated pairs regardless of the degrees and ICs for these GO terms. We shall see in practice, such proposition does not hold true; however, the GO encoders which align most closely with the theoretical expectation will be considered best. In general, a GO term's

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

Information Content is negatively correlated with its number of neighbors (or degree) in the GO tree. We estimate this correlation to be -0.445 for 20,283 Human terms. For this experiment, we observe how the degrees and ICs of the terms affect the similarity scores for child-parent and unrelated GO pairs, by seeing how well the inter-quantile ranges (IQRs) of the scores for these two groups stay separated at different degree and IC values. We randomly select Human GO pairs A, B with $\max(\text{deg}_A, \text{deg}_B) \leq 100$ and $\min(\text{IC}_A, \text{IC}_B) \geq 1$; the final set contains 3069 child-parent pairs and 3069 unrelated pairs. For each GO pair A, B, we compare its $\max(\text{deg}_A, \text{deg}_B)$ and $\min(\text{IC}_A, \text{IC}_B)$ against its similarity score. We include AIC method which does not encode GO terms into vectors; it is however informative to compare AIC against the GO encoders.

In Figure 2, the performances of all the methods are inversely correlated to the degrees of GO terms. When $\max(\text{deg}_A, \text{deg}_B)$ is small, that is when terms are near the leaves or have few neighbors, then all methods except for BERTname perform well, where the IQRs for child-parent and unrelated GOs do not intersect. For AIC, as $\max(\text{deg}_A, \text{deg}_B)$ increases, the two IQRs remain well separated, despite their trend lines are getting closer. For neural network encoders, as $\max(\text{deg}_A, \text{deg}_B)$ increases, the scores of child-parent pairs decrease, so that the IQRs overlap, making it harder to distinguish related GOs from unrelated ones. BERTname is the only exception, where the two trend lines diverge. However, BERTname has its IQRs for the two labels intersected at almost all degree values, making it the least desirable metric. Onto2vec has its IQRs first intersect at $\max(\text{deg}_A, \text{deg}_B) > 2.5$ GCN and ELMo have their IQRs first intersect at $\max(\text{deg}_A, \text{deg}_B) > 12.5$, respectively. For BiLSTM and BERTCLS, this number is $\max(\text{deg}_A, \text{deg}_B) > 17.5$. In some sense, BiLSTM and BERTCLS are better than Onto2vec, GCN and ELMo because they can adequately classify GO pairs containing terms with more neighbors. BERTas, BERT12, and BERT11+12 are best; these methods have their IQRs first intersect at $\max(\text{deg}_A, \text{deg}_B) \geq 22.5$.

In Figure 2, the performances of all the metrics are positively associated to the IC values of GO terms. As the $\min(\text{IC}_A, \text{IC}_B)$ increases, that is when the GO terms annotate very few proteins, then the IQRs of scores for child-parent and unrelated pairs do not intersect, so that the methods can better identify the two labels. BERTname and Onto2vec underperform; despite their two trend lines diverging as IC increases, the two IQRs overlap even for large IC values. For GCN, BiLSTM, ELMo, BERTas, BERT12, BERT11+12 and BERTCLS, the two IQRs are entirely separated when $\min(\text{IC}_A, \text{IC}_B)$ is strictly over 6.25, 4.25, 7.25, 4.75, 4.25, 4.25, and 5.75, respectively. Here, BiLSTM, BERT12 and BERT11+12 are the best metrics because they can adequately classify GO pairs containing more generic terms which are annotating more proteins.

Figure 2 indicates four points. First, encoding a GO term via its definition appears to be better than encoding a GO term based on its position on the GO tree. Second, we observe that ELMo does not outperform the single layer BiLSTM. Works in other research areas have showed that adding more LSTM layer does not guarantee better result [6, 23] (cite more). Third, within the BERT architecture, BERTas, BERT12, and BERT11+12 are better than BERTCLS and BERTname. Compared to BERTas and BERTname, BERT12, BERT11+12 and BERTCLS have scores ranging from -1 to 1, most likely because we explicitly train the GO vectors using the cosine distance loss. Fourth, neural network encoders would perform well only for specialized GO terms with low degrees and/or high ICs. To achieve the best result for all GOs, we must integrate the newer methods

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

with IC based models as noted in earlier works [8, 32].

3.2 Set comparison task

Because genes and proteins are annotated by GO terms, good GO metrics should well differentiate similar genes and proteins from unrelated ones. To compare the GO encoders, we conduct two experiments (1) classifying orthologs in Human, Mouse, and Fly and (2) classifying true protein-protein interactions in Human and Yeast.

3.2.1 Orthologs

We download the orthologs datasets and GO annotations in [8]. This data retains orthologs annotated by at least one GO term in each GO category and removed GOs with tag IEA, NAS, NA, and NR [16]. We test the following species: Human-Mouse (HM), Human-Fly (HF), and Mouse-Fly (MF). For each dataset, the positive set contains orthologs from the two species; whereas, the negative set contains randomly-matched genes. We set the sizes of the positive set and negative set to be equal. The HM, HF, and MF data has 10235, 4880, and 5091 pairs for each set, respectively. Here, comparing two genes is equivalent to comparing their two sets of GO annotations [16]. We use the best max average distance for the GOs in the two annotation sets [8, 16]. For this experiment, we use the entire GO annotations and compare terms across different ontology as in [8, 24].

Table 1 shows the summary statistics for the annotation in each ortholog dataset. Table 2 shows the Area Under the Curve (AUC) for each method. When compared to Resnik and AIC, performances of all the encoders drop for orthologs data having less informative GO terms. The AUCs for every GO position encoders decrease the most as compared to GO definition encoders. Within the definition encoders, ELMo's AUCs decrease the most. This outcome agrees with Fig. 2, where the position encoders and ELMo do not perform well for GO terms with low ICs and/or high degrees. Here, encoding GO definitions often yields better accuracy than encoding GO positions on the GO tree. For the three datasets, BERTas ranks first twice among the definition encoders; whereas BERT12 ranks first once. For the position encoders, there is not one method that is consistently better than the others.

3.2.2 Protein interaction network

Following [8], we download the Human data in [13] and Yeast data in [15]. These data have 6031 and 3938 positive Human and Yeast protein-protein interactions, respectively. For the negative set, we follow the same sampling procedure in [13]. We randomly assign edges between proteins that do not interact in the real PPI network. The real and random PPI network have the same proteins; we only require that they have different interacting partners. Table 2 shows the AUC for each method. Here, the AUC is computed using the exact process in section 3.2.1. For this experiment, we also include SimDef which uses Term Frequency – Inverse Document Frequency to compare the GO definitions [15]. Among the definition encoders, BERTas and BERT12 rank best for Yeast and Human data, respectively. For position encoders, GCN is the best method for the two datasets.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

Table 1: GO annotation summary statistics in each ortholog dataset. GO frequency counts the number of times the GO terms appear (including duplication); for example, if GO A and B both annotate protein C and D, then the total GO frequency is 4. Median degree (Deg) and Information Content (IC) for GO terms (including duplication) are proxies for how well annotated the datasets are.

	Negative set			Positive set		
	GO freq.	Deg	IC	GO freq.	Deg	IC
Dataset 1						
Human	123400	8	6.063	128666	8	6.142
Mouse	102775	8	5.907	147012	7	6.314
Dataset 2						
Human	57678	8	6.221	61708	8	6.158
Fly	28169	10	5.437	41481	8	6.290
Dataset 3						
Mouse	48520	9	6.012	67449	8	6.366
Fly	29026	10	5.404	39558	8	6.248

Table 2: AUC for classifying true ortholog pairs in Human, Mouse and Fly, and interacting proteins in Human and Yeast.

	Orthologs data			PPI data	
	Human-Mouse	Human-Fly	Mouse-Fly	Human	Yeast
Info Content					
Resnik	95.27700	94.19600	89.74800	86.97300	90.67000
AIC	95.79100	93.90600	89.84100	87.88900	87.77500
TF-IDF					
SimDef	NA	NA	NA	87.03400	88.22300
GO definition					
BiLSTM	95.19232	91.49026	80.28224	86.60700	88.23649
ELMo	92.99974	85.28422	76.84590	86.58324	81.42202
BERTas	96.72200	92.94441	79.61700	88.15400	88.99700
BERT12	95.94186	92.49684	81.64635	88.33334	89.95763
BERT11+12	95.51213	91.53214	80.85883	87.17318	88.44685
BERTCLS	96.05200	90.80000	78.99000	86.94500	89.26698
GO position					
BERTname	96.27481	85.39845	70.48564	83.92643	82.66531
GCN	94.98700	85.53500	72.99300	85.45300	86.74929
Onto2vec	91.79953	82.83256	74.41180	79.71700	83.98171

3.3 Annotation task

3.3.1 DeepGO

For this task, we do not aim to design a completely new model that is better than existing baselines for prediction GO annotations. Rather, our purpose is to determine how much can the GO vectors affect the prediction results. For this purpose, we use the data and existing framework of DeepGO [11]. DeepGO data consists of 3 sets BP, MF, and CC. BP,

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

MF and CC terms in this data (combining train, development, and test set) annotate at least 250, 50, 50 proteins. For this data, the parents of all the GOs annotating one protein are also added into the label set predicted. In total, the number of GO terms and proteins for each BP, MF and CC training dataset are 932 | 36375, 589 | 25199, and 436 | 35546. During training and testing, we use the whole label set size 932, 589 and 436.

Next, we briefly describe the neural network in DeepGO. Given an amino acid sequence, for example $p = \text{MARS} \cdots$, DeepGO converts p into a list of 3-mer as MAR ARS $\cdots$. Each 3-mer is assigned a vector of dimension 128, so that if p has length 1002 amino acids, then the matrix representing p is $E_p \in \mathbb{R}^{128\times 1000}$. A 1D-convolution layer and 1D-maxpooling are then applied to E_p. Flatten layer is applied to get a vector v_p representing p; loosely speaking, we have $v_p = \text{flatten}(\text{maxpool}\ (\text{conv1d}(E_p))$. DeepGO includes information from a protein-protein interaction network by concatenating $c_p = [n_p\ v_p]$, where n_p is a vector for protein p in the interaction network produced in [1] . To predict if GO_i is assigned to p, DeepGO fits a logistic regression layer $\text{sigmoid}(B_i^\intercal c_p + b_i)$, where B_i and b_i are parameter specific to GO_i. The loss function is binary cross entropy. DeepGO can be applied with only the protein sequences and without the additional protein network in [1]; in Table 3, we use the name *DeepGOSeq* to refer to this simple implementation, and *Baseline1* as the version of DeepGO having the protein network data.

To add GO encoders into DeepGO, we make one minor change to avoid significantly altering the original DeepGO model. Let g_i be the vector of GO_i, for example $g_i = \text{BERTCLS}(\text{definition of GO}_i)$. We concatenate $\hat{c}_{pi} = [c_p\ g_i]$, and apply one linear transformation $\tilde{c}_{pi} = \text{relu}\ W\hat{c}_{pi}$. $\tilde{c}_{pi}$ captures the interaction of the protein and GO vectors. To predict if GO_i is assigned to p, we fit $\text{sigmoid}(B_i^\intercal[\tilde{c}_{pi}\ c_p] + b_i)$ where $[\tilde{c}_{pi}\ c_p]$ is the concatenation of the two vectors. For this experiment, we freeze the GO vectors and train only the DeepGO parameters. In this paper, our intention is to determine which GO encoders can work best out-of-the-box for predicting functions of unknown sequences. In future research, we will consider jointly training both GO-to-GO and GO-to-protein relationships.

The transformation $\tilde{c}_{pi} = \text{relu}\ W[c_p\ g_i]$ may capture only interactions of values from the protein vector; in other words, the values of W corresponding to any values in g_i can be all zeros. Thus, we create one more baseline (Baseline2) for this experiment where we remove g_i and let $\tilde{c}_{pi} = \text{relu}\ Wc_p$. The rest of the layers follows exactly as in the previous paragraph. In Baseline2, $\tilde{c}_{pi}$ represents the interaction of the protein vector from [1] and the encoded amino sequence v_p without any GO vectors.

We compute three metrics Fmax score, macro and micro-AUC which do not require the prediction probabilities to be rounded at a specific threshold. Fmax and micro-AUC put more weights on frequently occurring GO terms, so that mislabeling infrequent GO terms do not greatly affect the outcome; whereas macro-AUC treats all the GO labels equally so that mislabeling infrequent GO terms can significantly affect the outcome [3, 28].

Table 3 shows that our interaction layer $\text{relu}\ Wc_p$ alone (Baseline2) improves upon original model (Baseline1) in [11] for all metrics in the three ontology. This suggests that there is much more information from the sequences alone which can further be extracted with a more complex neural network encoder for protein sequences. We reserve this topic for further research work.

All the GO encoders increase the evaluation metrics with respect to Baseline2 in all

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

three ontology. Because the frequencies of GO terms in training data affect their prediction accuracy [11, 33], we partition the GO terms based on their frequencies to further understand how each GO encoder performs. The 25%, 50%, and 75% quantile frequency for GO terms in the BP, MF, and CC training dataset are 233 | 365 | 860, 49 | 88 | 227, and 59 | 111 | 293 respectively. The number of GO terms in the $<25\%$ and $>75\%$ quantile groups are 232 | 232, 143 | 147, and 110 | 110 for BP, MF and CC respectively. We compute recall-at-k (R@k) and precision-at-k (P@k) for each group (Figure 3). For discovering unknown functions of protein sequences, having high recall rate is important so that we do not miss any annotations. However, by also observing precision rate, we can determine which GO encoders are the most well-balanced.

In Figure 3, we use Baseline2 because it is the most competitive against the GO encoders. When evaluating the GO labels altogether, Figure 3 shows that having the GO label vectors as extra features increases the recall for BP, MF, and CC ontology, and the precision for only BP and MF (for CC, our precision is at least the same as Baseline2). We next discuss the GO terms with lower occurrence frequencies. Loosely speaking, these terms are closer to the true protein functions; for example, the label GO:0005618 *perforation plate* is more precise to a protein's location than its parent term GO:0110165 *cellular anatomical entity* or its ancestor GO:0005575 *cellular component*. We first focus on recall, because it is important to not miss true protein function labels. Having GO label vectors from any encoder increases the recall for rare MF and CC terms. Surprisingly, only GO vectors from BERTas and BERT12 fail to obtain better recall for rare BP terms. Figure 3 shows that we have not sacrifice precision to attain higher recall on rare labels. For rare BP, MF, and CC terms, the precisions of GO encoders remain at least the same as Baseline2 (and in fact they are better than Baseline2 for MF).

When we evaluate more frequently occurring GO labels, having GO vectors as extra features does not guarantee better recall and precision; for example, these two metrics on very common CC terms are about the same as Baseline2. This finding agrees with the general concept in machine learning, that is, by having more observations with certain labels, we will often attain good prediction outcome for these labels.

The magnitude of performance differences among the GO encoders are often minimal. For example, the Fmax of all BERT encoders are similar (Table 3), although BERTname cannot well separate child-parent GOs from unrelated pairs (Figure 2) and is worst than the other BERT encoders at validating protein-protein interactions (Table 2). We suspect that the other parameters in the prediction model can be well trained to compensate for the imperfect GO vectors as the input.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

Table 3: Evaluating how much can GO vectors boost DeepGO result. DeepGOSeq uses only protein sequences to predict GO annotation. Baseline1 adds protein network data from [1] into DeepGOSeq. Baseline2 improves Baseline1 by adding one linear layer to convolve vector from protein network and the representation of the amino acid sequence without any GO vectors.

	BP			MF			CC		
		AUC			AUC			AUC	
Method	Fmax	Macro	Micro	Fmax	Macro	Micro	Fmax	Macro	Micro
Baseline									
DeepGOSeq	34.05	62.67	81.89	38.63	72.42	86.73	57.20	67.01	92.39
+Baseline1	41.80	81.87	89.70	46.77	83.73	90.60	62.70	87.80	96.47
+Baseline2	42.51	82.97	90.27	48.62	87.25	93.14	63.77	89.79	97.10
GO definition									
BiLSTM	43.48	83.49	90.73	50.20	88.02	93.84	65.47	90.48	97.25
ELMo	43.94	83.47	90.52	50.23	87.41	93.27	66.45	90.78	97.29
BERTas	43.45	83.11	90.55	49.24	87.43	93.52	65.39	90.27	97.27
BERT12	43.92	83.49	90.77	50.30	87.68	93.66	65.83	90.51	97.30
BERT11+12	43.76	82.84	90.28	50.67	88.05	93.94	65.68	90.62	97.29
BERTCLS	43.15	83.41	90.59	49.93	87.29	93.39	65.24	90.57	97.26
GO entity									
BERTname	43.35	83.35	90.26	49.13	87.97	93.07	65.66	90.27	97.24
GCN	43.65	83.46	90.63	49.74	87.78	93.50	65.20	90.39	97.20
Onto2vec	42.91	83.48	90.57	49.35	87.92	93.83	64.82	90.50	97.21

3.3.2 Expand DeepGO dataset

We repeat the experiment above, but extend the number of GO terms to be predicted to have more rare terms. Using the same annotation file in [11], we expand the GO sets in their original data; we include BP, MF, and CC terms that have at least 50, 10, and 10 annotations (5$\times$ less than the original criteria). We ensure that all the GO terms in the original dataset are included into this larger dataset; hence, most of these original terms will have a much larger occurrence frequencies. The BP training dataset now has 2980 terms; the 25%, 50% and 75% quantile occurrence frequency are 62 | 113 | 276. Here, our new BP dataset is harder to predict, because 75% of the GO terms occur less than 276 times (these GO terms barely pass the original cutoff criteria at 250); whereas, the original data has about 75% of terms occurring more than 276 times.

For MF and CC, the new number of GO terms to be predicted are 1677 and 979. The 25%, 50% and 75% quantile frequency are 13 | 22 | 66 and 15 | 34 | 117.5 respectively. Here, the new MF and CC data contain about 75% and 50% of the terms that would barely make the frequency cutoff at 50 in DeepGO original data [11]. CC data becomes harder to predict but not as much as BP and MF data. The number of GO terms in the <25% and >75% quantile groups are 736 | 742, 417 | 417, and 232 | 245 for BP, MF and CC respectively.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

Table 4 and Figure 4 show the prediction outcome. Here, we compare the best BERT architectures in section 3.3.1, BERT12 and BERT11+12, against BiLSTM, ELMo, and GCN which have showed success in other datasets with many rare labels [21, 31]. In Table 4, the encoders increase the Fmax but not always macro and micro AUC. As in the previous section, regardless of the encoders, having GO vectors as extra features raises the recall and precision for rare labels (count frequency below 25% quantile) (Figure 4). We discuss three key observations. First, except for rare CC terms (25% of the training labels), recall and precision of GCN are worst than Baseline2; hence, when considering every training label, these two metrics are below Baseline2. Our finding agrees with other works which have indicated that GCN might increase the accuracy of only rare labels [21]. Second, when evaluating the CC ontology, BERT12 although better than Baseline2 for rare labels, fails to increase recall and precision for labels with medium counts, and is about equivalent to Baseline2 for more common labels. Third, ELMo and BERT11+12 are consistently better than Baseline2 and the other encoders.

Table 4: Evaluating how much can GO vectors improve Baseline2 for the expanded DeegGO dataset, where we lower the inclusion criteria to have GO terms with occurrence frequency from 250, 50 and 50 to 50, 10, and 10 for BP MF and CC.

	BP			**MF**			**CC**		
		AUC			AUC			AUC	
Method	Fmax	Macro	Micro	Fmax	Macro	Micro	Fmax	Macro	Micro
Baseline2	40.03	86.06	93.94	46.19	87.01	95.01	63.87	90.52	98.02
BiLSTM	40.58	85.68	93.85	47.46	85.68	95.01	64.17	90.99	98.12
ELMo	41.67	85.98	93.99	48.87	86.91	95.50	65.10	90.35	98.12
BERT12	40.34	86.08	93.90	48.17	88.09	95.50	64.02	90.20	98.05
BERT11+12	41.62	85.70	93.75	48.78	86.40	95.17	65.35	90.39	98.14
GCN	41.10	86.07	93.86	48.02	86.49	95.31	64.64	90.24	97.99

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

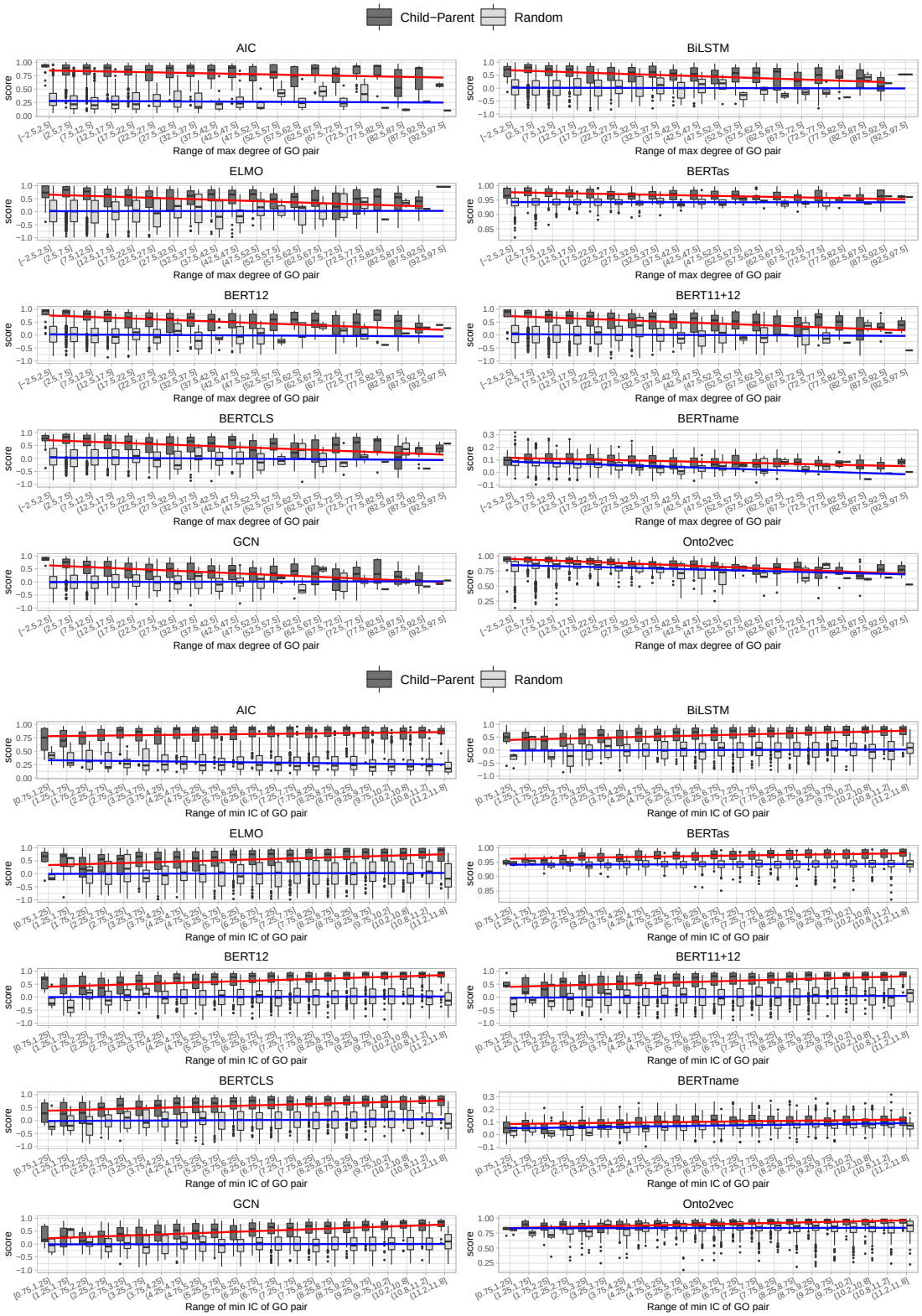

Figure 2: Encoder's ability to classify GO pairs is inversely correlated to the degrees of GO terms and is positively correlated to the ICs of GO terms.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

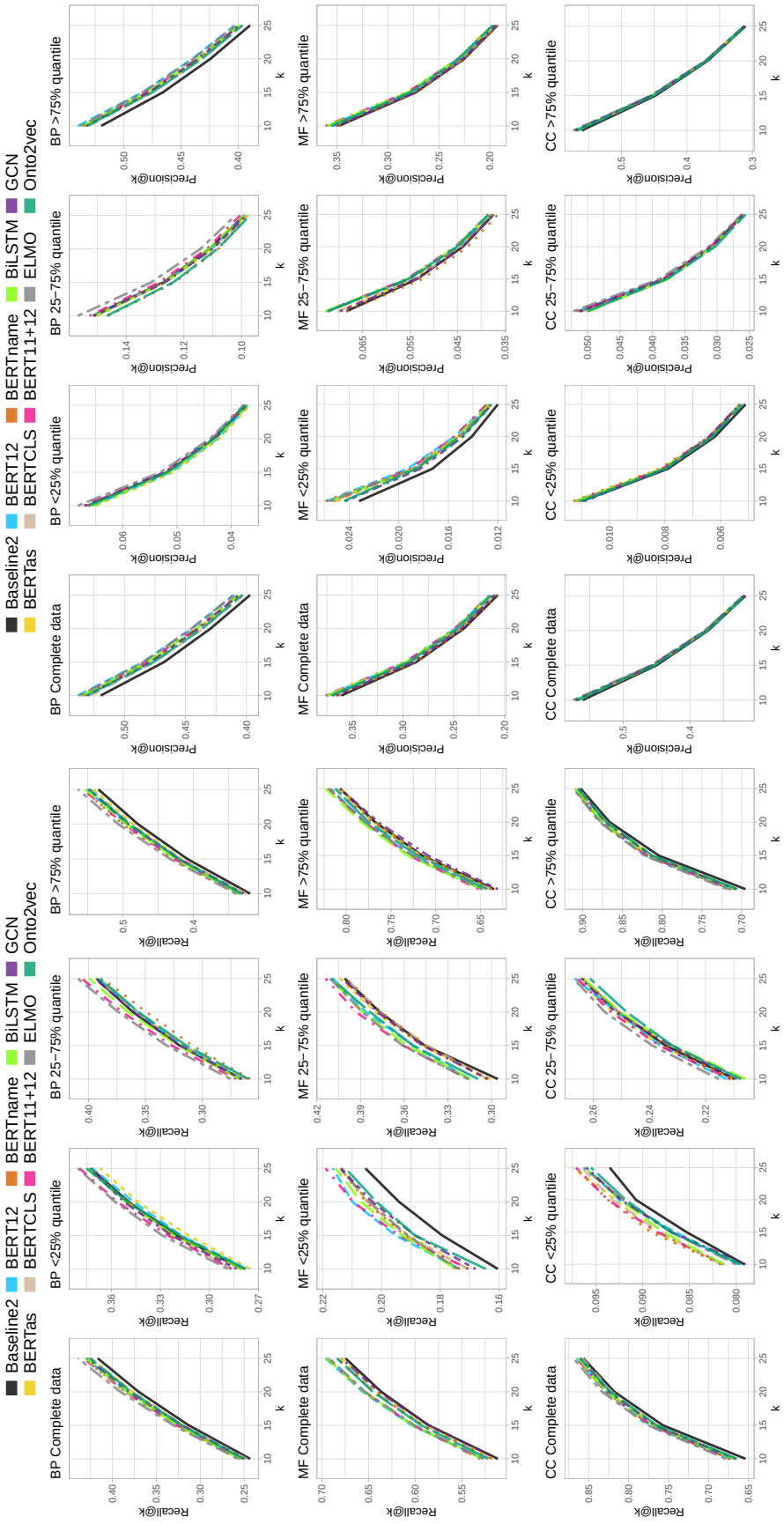

Figure 3: Recall and precision-at-k for annotations of protein sequences. For each ontology, we split the GO terms into 3 sets based on their frequencies in the training data.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

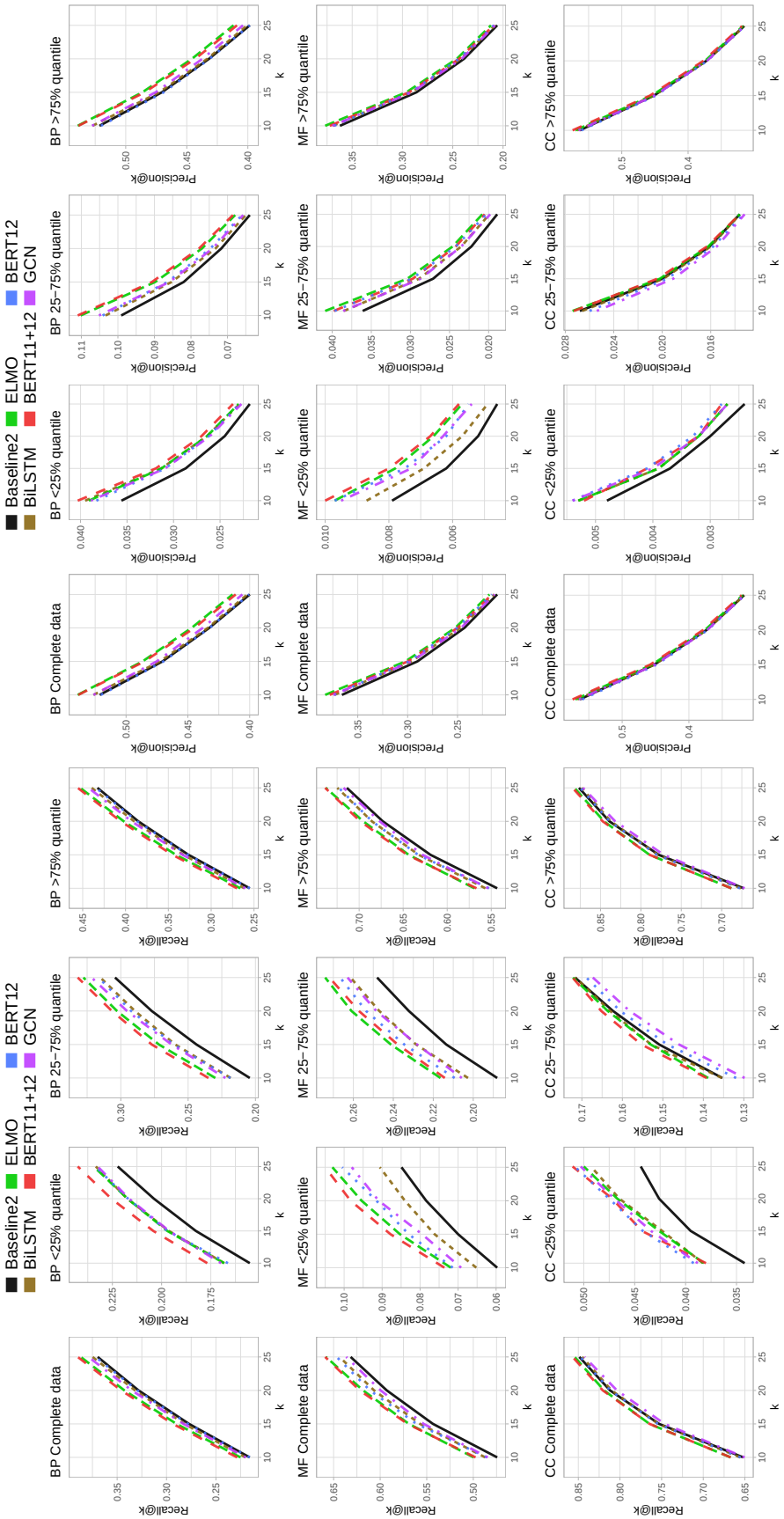

Figure 4: Methods are tested on the expanded DeepGO dataset, where we lower the inclusion criteria to have GO terms with occurrence frequency above 50, 10, and 10 for BP MF and CC. Recall and precision-at-k for annotations of protein sequences. For each ontology, we split the GO terms into 3 sets based on their frequencies in the training data.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

4 Conclusion

In this paper, we evaluate how well GO terms can be encoded by neural network models developed from the architecture of Word2vec (e.g. Onto2vec), LSTM (e.g. BiLSTM and ELMo), and Transformer (e.g. BERTas, BERT12, BERT11+12, BERTCLS, BERTname).

In task 1, we encode the GO terms into vectors and observe how the ICs and degrees can affect their cosine distances. We focus on two key types of relation: child-parent versus random. In principle, the scores of child-parent pairs must be higher than those of unrelated ones regardless of the ICs and degrees of the terms found in the pairs. In practice, the IC-based method AIC is least affected by ICs and degrees of the terms; whereas, the GO encoders can well differentiate the two groups only if the terms in the pairs have high ICs and/or low degrees. In task 2, when asserting relationship between genes and/or proteins based on their annotations, the encoders outperform Resnik and AIC for datasets annotated with more specific GO terms (e.g. terms with lower degree and higher ICs), but their accuracies drop compared to Resnik and AIC for less well annotated datasets. In task 1 and 2, encoding the GO definition is often better than encoding GO position in the ontology. Here, definition encoders based on BERT framework perform the best. We emphasize that our BERT encoders are trained on data created from the ontology tree; for example, we sample path from a leaf GO label to the root node, and train BERT parameters to recognize that these GO definitions are related. Loosely speaking, our BERT encoders use both GO definitions and GO positions on the ontology. Lastly, results in task 1 and 2 indicate that, to attain the best performance, future research direction must focus on integrating the ICs and degrees of GO terms as explicit features for the neural network encoders.

In task 3, we edit the DeepGO architecture so that our new predictor takes as extra features the GO vectors computed from the encoders. This new model is evaluated on the original DeepGO dataset and our own expanded DeepGO dataset. This new dataset is created by lowering the exclusion criteria where we only remove GO terms with frequencies below 50, 10, and 10 (from the original criteria of 250, 50, and 50) for BP, MF, and CC, respectively. On both datasets, we find that recall and precision of rare labels, but not of common labels, benefit the most from having GO vectors as extra features. Interestingly, we observe that parameters in the DeepGO architecture can be trained to compensate for the imperfect GO vectors as the input, so that GO encoders not performing well in task 1 and 2 can still produce high accuracy scores. This fact indicates that having GO vectors as features definitely helps, but the choice of encoders does not greatly affect the outcome.

We hope that our encoders for GO terms can provide the basis for more advanced encoding techniques. In the future work, we will continue developing better encoders for GO terms, and integrate them with other predictors besides DeepGO to better estimate protein functions.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

References

[1] Alshahrani, M., Khan, M.A., Maddouri, O., Kinjo, A.R., Queralt-Rosinach, N. and Hoehndorf, R. (2017). Neuro-symbolic representation learning on biological knowledge graphs. Bioinformatics, **33**(17), 2723–2730.

[2] Belanger, D. and McCallum, A. (2016). Structured prediction energy networks. In International Conference on Machine Learning, pages 983–992.

[3] Chase Lipton, Z., Elkan, C. and Narayanaswamy, B. (2014). Thresholding classifiers to maximize f1 score. arXiv preprint arXiv:1402.1892.

[4] Chen, M., Ju, C.J.T., Zhou, G., Chen, X., Zhang, T., Chang, K.W. et al (2019). Multifaceted protein–protein interaction prediction based on siamese residual rcnn. Bioinformatics, **35**(14), i305–i314.

[5] Conneau, A., Kiela, D., Schwenk, H., Barrault, L. and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364.

[6] Cui, Z., Ke, R. and Wang, Y. (2018). Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction. arXiv preprint arXiv:1801.02143.

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

[8] Duong, D., Ahmad, W.U., Eskin, E., Chang, K.W. and Li, J.J. (2018). Word and sentence embedding tools to measure semantic similarity of gene ontology terms by their definitions. Journal of Computational Biology, **26**(1), 38–52.

[9] Gene Ontology Consortium (2017). Expansion of the gene ontology knowledgebase and resources. Nucleic acids research, **45**(D1), D331–D338.

[10] Kipf, T.N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[11] Kulmanov, M., Khan, M.A. and Hoehndorf, R. (2017). Deepgo: predicting protein functions from sequence and interactions using a deep ontology-aware classifier. Bioinformatics, **34**(4), 660–668.

[12] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C.H. et al (2019). Biobert: pre-trained biomedical language representation model for biomedical text mining. arXiv preprint arXiv:1901.08746.

[13] Mazandu, G.K. and Mulder, N.J. (2014). Information content-based gene ontology functional similarity measures: Which one to use for a given biological data type? PLoS ONE, **9**(12), e113859.

[14] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119.

[15] Pesaranghader, A., Matwin, S., Sokolova, M. and Beiko, R.G. (2015). simdef: definition-based semantic similarity measure of gene ontology terms for functional similarity analysis of genes. Bioinformatics, **32**(9), 1380–1387.

[16] Pesquita, C., Faria, D., Bastos, H., Ferreira, A.E., Falcão, A.O. and Couto, F.M. (2008). Metrics for go based protein semantic similarity: a systematic evaluation. BMC bioinformatics, **9**(5), S4.

[17] Pesquita, C., Pessoa, D., Faria, D. and Couto, F. (2009). Cessm: Collaborative evaluation of semantic similarity measures. JB2009: Challenges in Bioinformatics, **157**, 190.

bioRxiv preprint doi: https://doi.org/10.1101/765644; this version posted December 8, 2019. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC 4.0 International license.

[18] Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. et al (2018). Deep contextualized word representations. In Proc. of NAACL.

[19] Profiti, G., Martelli, P.L. and Casadio, R. (2017). The bologna annotation resource (bar 3.0): improving protein functional annotation. Nucleic acids research, **45**(W1), W285–W290.

[20] Resnik, P. (1999). Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. J. Artif. Intell. Res.(JAIR), **11**, 95–130.

[21] Rios, A. and Kavuluru, R. (2018). Few-shot and zero-shot multi-label learning for structured label spaces. In Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing, volume 2018, page 3132. NIH Public Access.

[22] Rong, X. (2014). word2vec parameter learning explained. arXiv preprint arXiv:1411.2738.

[23] Sak, H., Senior, A. and Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In Fifteenth annual conference of the international speech communication association.

[24] Smaili, F.Z., Gao, X. and Hoehndorf, R. (2018). Onto2vec: joint vector-based representation of biological entities and their ontology-based annotations. Bioinformatics, **34**(13), i52–i60.

[25] Song, X., Li, L., Srimani, P.K., Yu, P.S. and Wang, J.Z. (2014). Measure the semantic similarity of GO terms using aggregate information content. IEEE/ACM Transactions on Computational Biology and Bioinformatics, **11**(3), 468–476.

[26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N. et al (2017). Attention is all you need. In Advances in neural information processing systems, pages 5998–6008.

[27] Wang, S., Cho, H., Zhai, C., Berger, B. and Peng, J. (2015). Exploiting ontology graph for predicting sparsely annotated gene function. Bioinformatics, **31**(12), i357–i364.

[28] Wu, X.Z. and Zhou, Z.H. (2017). A unified view of multi-label performance measures. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 3780–3788. JMLR. org.

[29] Xiao, H. (2018). bert-as-service. github.com/hanxiao/bert-as-service.

[30] Xiong, W., Yu, M., Chang, S., Guo, X. and Wang, W.Y. (2018a). One-shot relational learning for knowledge graphs. CoRR, **abs/1808.09040**.

[31] Xiong, W., Yu, M., Chang, S., Guo, X. and Wang, W.Y. (2018b). One-shot relational learning for knowledge graphs. arXiv preprint arXiv:1808.09040.

[32] Yang, H., Nepusz, T. and Paccanaro, A. (2012). Improving go semantic similarity measures by exploring the ontology beneath the terms and modelling uncertainty. Bioinformatics, **28**(10), 1383–1389.

[33] Zhang, C., Zheng, W., Freddolino, P.L. and Zhang, Y. (2018). Metago: Predicting gene ontology of non-homologous proteins through low-resolution protein structure prediction and protein–protein network mapping. Journal of molecular biology, **430**(15), 2256–2265.

[34] Zhang, Z., Zhang, J., Fan, C., Tang, Y. and Deng, L. (2017). Katzlgo: large-scale prediction of lncrna functions by using the katz measure based on multiple networks. IEEE/ACM transactions on computational biology and bioinformatics, **16**(2), 407–416.