Meta-Embedding Generation for Unsupervised Protein Models

Tolga Dimlioglu
New York University
Brooklyn, NY 11201
td2249@nyu.edu

Serhat Bakirtas
New York University
Brooklyn, NY 11201
sb7082@nyu.edu

Brian McMinn
New York University
Brooklyn, NY 11201
bwm7543@nyu.edu

1 Introduction

The success of traditional supervised learning models for classification tasks relies heavily on feature extraction methods. This is abundantly clear in recent literature on protein modeling, wherein features are extracted either by computationally complex pre-processing of a protein's composite amino-acid sequences (*e.g.* BLAST [1]) or by cumbersome, experimental verification of the features and labels (*e.g.* Swiss-Prot [2]). In the past decade, with the exponential growth of datasets containing unlabeled amino acid sequences [3], these computationally complex and manual methods are no longer feasible. In turn, there has been an uptick in research on unsupervised learning of protein representations, inspired by algorithms in the Natural Language Processing (NLP) domain. Consequently, several unsupervised models [4–7], trained on various protein datasets such as Uniref50 [4] and Pfam [8], have emerged. As the corresponding *potentially-universal* protein embeddings are pretrained with different models and/or datasets, their evaluation has become a topic of interest. In [7], Rao *et al.* listed a set of downstream tasks and demonstrated that there was not a single embedding which out-performed all others across all of these tasks, implying that each embedding may have captured complementary biological features of the proteins. Our experiments so far have focused on the *remote homology classification* task, one of the five tasks put forward by Rao *et al.* in [7], to standardize the evaluation of protein embeddings.

2 Task, Dataset and Embeddings

2.1 Remote Homology Classification Task

A pair of proteins is called *homologous* if they share a common evolutionary ancestor, which often implies a certain level of similarity in biochemical structure and functionality. Therefore grouping homologous proteins into homology classes becomes an important task since homology classification is directly related to practical problems such as antibiotic-resistant gene detection [9] and enzyme classification [10]. In our setting, the remote homology problem is formalized as a low-similarity sequence classification problem, where a protein, being an amino acid sequence, is assigned to one of the 1195 protein fold classes. These classes map to 3D protein folds which are essential to the function of the protein.

2.2 Dataset

In our dataset (which we obtain from from [11]), each protein is represented by a sequence of amino acids and assigned to one of the 1195 remote homology classes, depending on its fold structure, according to Structural Classification of Proteins (SCOP) [12]. The dataset is split into a training set of 12312 proteins, a validation set of 736 proteins and three test sets consisting of 1272, 1254 and 718 proteins. These test sets are called *family-level*, *superfamily-level*, and *fold-level*, respectively. The three test sets are chosen in the following fashion: All of the superfamilies present in the fold-level test set are absent in the training set. Similarly, all of the families present in the superfamily-level test set are absent in the training set. Finally, there are family-level overlaps between the training and the family-level test sets. Therefore as we move from the family-level to fold-level test sets, the classification task becomes harder. This allows us to use the fold-level test set to investigate a model's ability to generalize to unseen superfamily and family distributions for any given fold.

2.3 Embeddings

We make use of three pretrained embedding models, namely the Elmo [6]-based embedding, as described in [13], Unirep [5] and the Transformer-based embedding given in [7]. For the sake of brevity, throughout this work we will call these embeddings Elmo, Unirep and Transformer, respectively. All three of these embedding models are pretrained over tens of millions of proteins and we apply the pretrained models to our dataset to generate the corresponding embeddings. For a given protein, each model generates multiple embeddings, one for each amino acid (residue) in the said protein. In other words, after applying each of these pretrained models, a protein is represented by a matrix, each row corresponding to the embedding of a single residue. As in [13], we obtain a single vector representation for each protein by taking an average over the residual embeddings.

2.3.1 Elmo Model

The Embedding from Language Models (Elmo) model from [13] was one of our three chosen embedding models. It consists of a character-level Convolutional Neural Network (CNN) which is followed by two layers of bidirectional Long Short-Term Memory (LSTM) architectures. While the CNN embeds each amino acid into a latent space, the LSTMs use that embedding to model the context of the surrounding residues (amino acids). The model is pretrained with approximately 31 million unlabeled proteins taken from the Pfam [8] protein database, and outputs protein embeddings of dimension $d_1 = 1024$.

2.3.2 Unirep Model

The Unified Representation (Unirep) model, presented in [5], is an mLSTM with 1900 hidden units. The model is pretrained with approximately 24 million unlabeled proteins taken from the UniRef50 [4] protein database, using next residue prediction. The model outputs protein embeddings of dimension $d_2 = 1900$.

2.3.3 Transformer Model

As presented by [7], the Transformer model we use has 12 layers with 512 hidden units and 8 attention heads. Similar to Elmo, the model is pretrained with approximately 31 million unlabeled proteins taken from the Pfam [8] protein database, and outputs protein embeddings of dimension $d_3 = 768$.

3 Ensemble Methods

3.1 Concatenation (CONCAT)

As proposed in [14], the concatenation method refers to a simple concatenation of the source embeddings after the l_2-normalization of each embedding. Given r source embeddings with the respective dimensions $d_1, \ldots, d_r$, this method yields a meta-embedding with $d_{\text{CONCAT}} = \sum_{j=1}^{r} d_j$ dimensions. As seen in Section 5, this simple method boosts the accuracy as it increases the Euclidian distance in the new representation space at the expense of increased dimensionality.

3.2 Averaging (AVG)

Another simple ensemble method, proposed in [15], is taking the average of the source vectors, after l_2 normalization, in order to generate the meta-embedding. As described in [15], in the presence of dimension mismatch between the source embeddings, the embedding vectors with the lower dimensions are zero padded from the end.

As explained in our midterm paper, we investigated the empirical distribution between different protein embedding pairs and argued that since θ follows a normal distribution around $\mu = \frac{\pi}{2}$, AVG approximates CONCAT without suffering from the increased dimensionality.

3.3 Interpolated Average

While taking averages over the source embeddings with dimensions $d_1, \ldots, d_r$, interpolation arises as a natural alternative to zero-padding. In this work, we applied a simple linear interpolation to the "shorter" embeddings to match the dimensionality $d_{\text{AVG}} = \max_{i \in [r]} d_i$, before averaging.

3.4 SVD-based Meta-Embedding

Although the embeddings capture complementary features and each embedding can potentially contribute to the generation of the meta-embedding, the potential performance loss due to increased dimensionality presents a challenge. In our prior work, the concatenation of all three embeddings (E+U+T) underperforms the concatenation of the Elmo and Transformer embeddings (E+T) in two of the three test datasets. To break the curse of dimensionality, we use the SVD-based meta-embedding generation method proposed in [14]. We first concatenate all three embeddings. Here given training dataset $\mathbf{E}^{training}_{n_{tr}\times\tilde{d}}$ for a given source embedding with dimension $\tilde{d} < n_{tr}$, we first take the truncated SVD decomposition $\mathbf{E}^{training}_{n_{tr}\times\tilde{d}} = \mathbf{U}^{training}_{n_{tr}\times\tilde{d}}\mathbf{\Sigma}^{training}_{\tilde{d}\times\tilde{d}}(\mathbf{V}^{training}_{\tilde{d}\times\tilde{d}})^T$. Then, the SVD-based meta-embedding is given by the first $d < \tilde{d}$ columns of $\mathbf{U}^{training}_{n_{tr}\times\tilde{d}}$. Here the dimension d of the meta-embedding is a hyperparameter.

After obtaining the training meta-embedding, we generate the test meta-embedding given the test dataset $\mathbf{E}^{test}_{n_{test}\times\tilde{d}}$ for the source embedding as follows $\mathbf{U}^{test}_{n_{test}\times\tilde{d}} = \mathbf{E}^{test}_{n_{test}\times\tilde{d}}\mathbf{V}^{training}_{\tilde{d}\times\tilde{d}}(\mathbf{\Sigma}^{training}_{\tilde{d}\times\tilde{d}})^{-1}$ where we use the first d columns of $\mathbf{U}^{test}_{n_{test}\times\tilde{d}}$ as the meta-embedding for the test dataset.

We stress that the performance of this method strongly depends on the datasets having correlated features. Correlated features allow compression without a considerable loss. Indeed, when we compute the correlation coefficients among the features and the corresponding p-values, we observe a high correlation among the features within each embedding. On average, a feature is found to be strongly-correlated with 3388 other features, with a standard deviation of 87. Thus, as can be seen in Table 2 and Table 3, SVD-based meta-embeddings perform well even for modest dimensionalities due to the aforementioned high correlation. In this project we experimented with $d = 1000 : 250 : 3500$, where $d = 1000$ is the approximate median dimension of the source embeddings and $d = 3500$ is the approximate dimension of concatenated meta-embedding.

3.5 1-to-N

Proposed in [14], 1-to-N is a single learning-based ensemble method. The 1-to-N method is based on the assumption that individual embeddings are linear projections of a single meta-embedding of dimension d, into different dimensions. Formally, given training datasets for source embeddings $\mathbf{E}^{(i)}_{n\times d_i}$, $i = 1, 2, 3$, we learn the projection matrices $\mathbf{P^{(i)}}_{d_i\times d}$ and the meta-embedding $\mathbf{W}_{n\times d}$ such that $\mathbf{E}^{(i)}_{n\times d_i} = \mathbf{W}_{n\times d}(\mathbf{P^{(i)}}_{d_i\times d})^T$. Thus, we train a simple network with the following regularized loss function

$$J = \sum_i \|\mathbf{E}^{(i)}_{n\times d_i} - \mathbf{W}_{n\times d}(\mathbf{P^{(i)}}_{d_i\times d})^T\|_2^2 + \lambda\sum_i \|\mathbf{P^{(i)}}_{d_i\times d}\|_F^2 \tag{1}$$

Here the regularization parameter λ is a hyperparameter. However due to limited time and computational power, throughout our experiments we stuck with $\lambda = 1$, based on initial observations. For each d, we trained the network for 1000 "useful" epochs. Since the loss function is strictly convex, we adapted the following schedule: If the loss $J^{(t+1)}$ does not improve upon the loss $J^{(t)}$, we reran the $t + 1^{\text{st}}$ epoch with a reduced learning rate. After training, the test datasets are generated by applying projections $\mathbf{P^{(i)}}_{d_i\times d}$ to the source embeddings.

Similar to SVD, we experimented with $d = 1000 : 250 : 3500$. We observed that unlike SVD, which yields its best performance for small values of d, 1-to-N performs best for large d.

3.6 Dynamic Meta Embeddings

In this meta-embedding scheme, the supervised learning algorithm has access to several different embeddings for the same word and the model learns to prefer which embedding is more important for a certain word by learning the weights for them. This idea is proposed and implemented in [16]. First, all the available embeddings are mapped to a common d dimensional space with a learnable linear function. Then, the projected embeddings are combined by taking weighted sum using the weights obtained from attention mechanism. The authors also proposed the contextualized version of the same method. As for the encoder, they used Bi-directional LSTM with Max Pooling

(BiLSTM-Max) that computes two sets of hidden states from left to right and right to left [16]. We integrated our protein dataset with 3 different embeddings, namely Transformer, Unirep and Elmo, and attempted to train our model with the proposed method using their implementation. Even though authors achieved state-of-the-art performance in Natural Language Inference and Sentiment Analysis tasks [16], we observed that the training was not successful in our case. In our experiments, training takes much more time than theirs and even the training accuracy barely reaches to 80%.

We think that the main problem lies in the nature of the datasets and tasks. We inspected the datasets and embeddings used in the original experiments. First, their embeddings (GloVE, word2vec) are of length 300 whereas our embeddings have dimensions 768, 1024 and 1900. Second, their model inputs consist of few sentences and small number of words. Conversely the shortest protein in our set has 17 residues whereas longer ones may contain hundreds. Note that we used each protein as a body of text and each protein residue is treated as a word. Another major difference is in the number of output classes. The results in the paper are obtained on the datasets with 3 to 5 classes while ours have 1195 different classes. All in all, we did not achieve promising results with this approach in our task and we conclude that this approach might not be applicable to protein embeddings at all.

4 Experiment Settings

In this section, we provide details about the models that we have used and the settings of our experiments. The code for our experiments can be found in our Github repository.

4.1 Base Models

For the MLP, we have picked a model with 2 hidden layers and both of the layers have $2^{14} = 16384$ neurons. It is noteworthy that the layers contain a large number of neurons since there are many features that the model must learn, necessitating such a large capacity. Our initial experiments with smaller hidden layer sizes showed that, after convergence, the model does not achieve an accuracy score close to 100% even on the training set. The input size depends on the embedding scheme and the output size is 1195. The hidden layers are followed by ReLU activation functions and we have added a dropout layer after the first hidden layer. For all experiments, we initialized the models from the same seed number.

For the logistic regression, we set the maximum number of iterations to 5000, meaning the classifier stops training after 5000 iterations regardless of the convergence. For all experiments, we used the same random state for the logistic regression.

4.2 Attention Based Models

We have also implemented 2 different attention based models which are inspired from classical attention mechanism [17] and the self-attention [18].

4.2.1 Classical Attention Based Model

In this model, we tried to imitate the role of the classical attention mechanism in each task. In the machine translation task, attention mechanism plays important role. It is used to form a context vector by calculating the weighted average of the encoder hidden states based on the similarity score of each of them with the current decoder input. Note that we do not have sequential data, since protein embeddings are obtained by averaging along their residue axes. As a result, we will be creating our hidden states with sequential linear layers. The illustration of this architecture is provided in the following figure.

As seen in Figure 1, input embedding of size $B \times E$, where B is the batch size and E is the embedding dimension, is first mapped to M dimensional space with a linear layer followed by a ReLU activation. This creates the first nonlinear representation that is obtained by the weighted combination from the original embedding. As we add more layers, with linear weights of size $M \times M$ followed by ReLU activation functions, we get higher level representations of the same embedding by extracting different and deeper relations from the original embedding. This yields a learnable attention weights denoted with $W^A_{M\times 1}$. By carrying out matrix multiplication with these weights and the representations, we obtain similarity score with each of the representations. After normalizing the sum of similarity scores by applying softmax function, we get the refined context by taking weighted average among the representations as is the case in the classical attention. Finally, this context representation is forwarded to an MLP classifier that has 2 hidden layers, each of which has

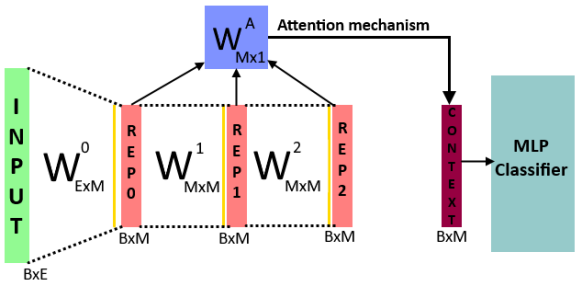

Figure 1: Our model architecture inspired from classical attention mechanism

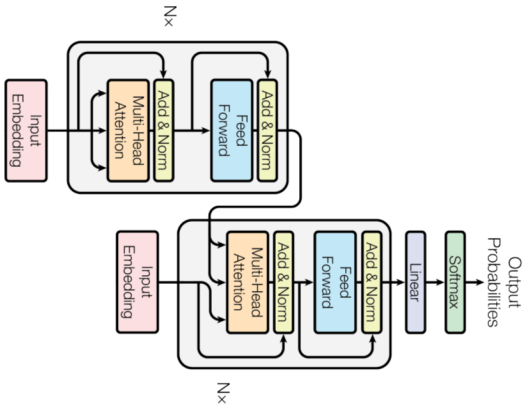

Figure 2: Modified transformer architecture with self-attention

4096 neurons followed by ReLU activation. For the hyperparameters, we used 32 representations and we varied M from 1000 to 3500 with a step size of 250.

4.2.2 Transformer Based Model

We have implemented a transformer-based model from scratch. For inputting this model, we treated every feature in the protein embedding as a word embedding with size 1, in other words, a protein embedding with dimension E is treated as a sentence with E words. We have implemented the encoder exactly as it is in [18]. As explained in the paper, the motivation behind using self-attention in the encoder is to extract the importance map between the features [18]. For the decoder, we dropped the self-attention part since we will be using the same input embedding that is used in the encoder. In the decoder's attention part, we are using the *Key* and *Query* obtained from the encoder and the *Value* is the input embedding itself. In this way, we are imitating the conventional encoder-decoder mechanism by using the importance relations obtained in the encoder and using the original value of the embedding. Aside from these, we have two more significant difference from the original architecture. First, we do not use positional encodings since there is no sequential nature within the features of the protein embeddings. Second, since, again, the protien features are independent of sequential order, we incorporate all the relations within the embedding regardless of their order.

For the hyperparameter selection, we needed to stick to $h = 1$ and $N = 1$ in [18] even though we used a GPU with 11GB of VRAM. This is mainly because the model size increases drastically when the the input text, or the embedding dimension size in our case, increases. Hence, we were only able to experiment with an input embedding size of at most 1800. We were only able to do this with a batch size of 1.

Embeddding Method	Combination	MLP			Logistic Regression		
		Fold	Superfamily	Family	Fold	Superfamily	Family
Individual Embeddings	E	23.96	43.62	93.16	24.37	44.02	93.32
	U	21.45	33.65	84.28	22.42	33.49	86.16
	T	21.17	39.63	90.80	23.12	39.00	90.88
Concatenation	E + U	23.68	41.71	92.30	25.63	43.46	93.47
	E + T	25.07	44.90	94.65	26.60	**46.73**	95.28
	U + T	22.98	40.91	91.67	26.04	42.50	93.08
	E + U + T	25.07	44.74	93.95	27.30	46.57	**95.44**
Padded Average	E + U	24.09	42.11	92.06	25.21	43.86	93.40
	E + T	24.79	44.50	**94.73**	26.32	45.93	94.73
	U + T	23.40	40.11	91.82	24.65	42.19	92.85
	E + U + T	**25.35**	43.38	93.79	**27.43**	46.17	94.81
Interpolated Average	E + U	24.23	42.82	92.14	24.79	43.94	93.39
	E + T	24.79	44.82	94.26	27.02	45.22	94.89
	U + T	22.14	42.03	92.53	24.37	42.19	93.55
	E + U + T	25.07	**45.53**	94.18	26.18	45.93	94.97

Table 1: Results obtained with MLP and Logistic Regression Models

4.3 Data Processing

Let N be the number of embeddings or protein samples of a given protein embedding set S. Since different proteins have different numbers of residues – because they can vary in length – each protein embedding has a different number of row dimensions. Let L be the embedding size, so each of the samples S_i in S for $i = 1 : N$ is of size $R_i \times L$ where R_i is the number of residues for the i^{th} protein. For MLP and Logistic Regression Models, we take mean of each protein embedding with respect to its residues so that its size becomes $1 \times L$ or simply L. After processing the embeddings this way, the result is a training set of size $N \times L$ that will be used for training and inference.

4.4 Training

We have trained the MLP and Classical Attention based model for 100 epochs using momentum and Nesterov accelerated SGD and weight penalties of $1e-5$ and $1e-4$. This showed better performance compared to AdaGrad and Adam optimizers in our tests. The initial learning rate is $1e-4$ with a 10-fold learning rate drop applied every 30 epochs.

We trained the Transformer based model for 60 epoch using the Adam optimizer initial learning rate of $1e-4$. Then, we applied a 10-fold learning rate drop every 20 epochs. We also used the following Adam optimizer parameters: $\beta_1 = 0.99$, $\beta_2 = 0.98$ and $\epsilon = 1e-9$ similar to [18]. Also, we added a weight penalty of $1e-5$.

5 Results

In this section, we provide the results obtained with different embedding methods and different classifiers. In table 1, we refer to embeddings obtained with Elmo, Unirep and Transformer models as E, U and T respectively. In tables 2, 3, first column corresponds to the dimension that is obtained with SVD and 1-to-N methods. In table 4, for classical attention model, dimension represents the value of M in figure 1. As shown in the midterm report, incorporation of the residue information with the recurrent models underperform compared to our other classifiers. Hence, we decided to stop experimenting with them.

We observed that, among all the ensembling methods, SVD proved to be the best. Without loss of information SVD easily compresses the dimensionality. This result is also anticipated since we

	1 to N			SVD		
Dimension	Fold	Superfamily	Family	Fold	Superfamily	Family
1000	23.96	44.02	93.00	27.44	**48.64**	96.07
1250	24.51	42.58	92.77	27.16	48.41	**96.15**
1500	24.51	43.14	93.00	**28.13**	48.09	95.91
1750	**24.93**	43.06	92.45	27.58	47.37	95.68
2000	24.37	43.86	93.32	27.72	47.05	95.52
2250	23.96	43.78	93.55	26.88	46.97	95.83
2500	23.40	44.10	93.47	25.77	45.77	95.83
2750	24.51	44.02	**93.55**	25.49	45.69	95.05
3000	23.82	**44.26**	93.16	25.21	44.74	95.28
3250	23.40	**44.26**	93.47	24.51	43.30	94.81
3500	23.26	43.30	93.08	24.37	43.86	94.5

Table 2: Results obtained with MLP model on the embeddings obtained with 1-to-N and SVD.

observed that embedding features are highly correlated with each other and this property makes way for an effective compression. The details are provided in section 3.4. Although the Classical Attention based and transformer based models obtained inferior results compared to SVD, transformer based model outperformed classical attention, which is expected due to the more sophisticated and successful relation-extracting mechanism of the transformer model.

	1 to N			SVD		
Dimension	Fold	Superfamily	Family	Fold	Superfamily	Family
1000	25.49	45.61	95.05	25.21	48.09	95.52
1250	25.35	45.77	95.05	**27.16**	**48.41**	96.15
1500	25.21	45.85	95.05	25.35	48.17	95.99
1750	25.77	45.85	95.05	24.79	48.09	95.99
2000	25.63	46.25	95.60	24.65	47.53	96.07
2250	25.63	46.01	95.68	24.93	47.39	**96.23**
2500	25.77	46.09	95.68	24.65	46.89	**96.23**
2750	26.18	46.01	95.91	24.23	47.21	95.83
3000	**26.46**	46.17	95.91	24.51	46.09	95.99
3250	25.49	**46.57**	**96.15**	23.12	46.17	95.36
3500	25.63	46.41	96.07	22.84	45.93	95.20

Table 3: Results obtained with logistic regression on the embeddings obtained with 1-to-N and SVD.

6 Information-Theoretic Interpretation of the Results

As can be seen from Table 1, when we pairwise ensemble the embeddings, we observe that the Elmo-Transformer (E+T) pair outperforms the Elmo-Unirep (E+U) and Unirep-Transformer (U+T) embedding pairs. This implies that the embedding pair E+T should capture a higher number of complementary features, compared to U+T and E+U. Hence, we expect a lower dependence between E+T, compared to U+T and E+U. To test this hypothesis, we utilize Mutual Information as a metric of dependence. Mutual Information $I(X;Y)$ measures the dependence between two random variables X and Y drawn from a joint distribution with PDF $f(X,Y)$, where

$$I(X;Y) = \int f(x,y) \log_2 \frac{f(x,y)}{f(x)f(y)} dxdy \tag{2}$$

and a higher dependence means higher mutual information [19].

We note that the computation of $I(X;Y)$ requires the availability of the joint distribution $f(X,Y)$. To that end, after experimenting with the embeddings, we observed that the features and fea-

Classical Attention Model			
M	Fold	Superfamily	Family
1000	21.87	42.66	92.61
1250	20.89	42.66	92.85
1500	21.31	**43.46**	92.85
1750	21.59	43.06	92.77
2000	**22.56**	42.19	92.92
2250	22.42	**43.46**	92.53
2500	22.7	42.74	**93.08**
2750	21.31	41.55	92.85
3000	22.01	42.42	92.77
3250	22.14	42.26	92.69
3500	21.17	42.5	93.0

Transformer Based Model			
Method	Fold	Superfamily	Family
E	24.79	40.83	91.43
T	23.96	38.84	89.86
E + T Inter. Avg.	26.18	43.22	93.71
E + T Pad. Avg.	26.18	43.46	94.34
E + T Concat.	25.63	43.62	94.34
1 to N - 1000	25.77	42.11	91.75
1 to N - 1250	26.32	41.31	91.9
1 to N - 1500	**26.74**	41.87	91.98
1 to N - 1750	26.18	41.95	91.75
SVD - 1000	26.04	45.85	95.44
SVD - 1250	22.7	40.83	92.69
SVD - 1000	24.79	**45.93**	**95.68**
SVD - 1000	22.98	43.62	94.5

Table 4: Results obtained with Classical Attention and Transformer based models

Embedding Pair	E+U	E+T	U+T
Mutual Information (bits)	406.09	222.77	259.35

Table 5: Mutual information between the embedding pairs, computed over $n = 16292$ samples.

ture pairs demonstrate approximately-normal and approximately-jointly-normal behaviours, respectively. Thus, we made a joint-normality assumption on the embedding pairs and computed their mutual information. One can show that under the joint-normality assumption, the mutual information between embeddings $\mathbf{Emb}_1$ and $\mathbf{Emb}_2$ becomes

$$I(\mathbf{Emb}_1; \mathbf{Emb}_2) = \frac{1}{2} \log_2 \left(\frac{\det(\mathbf{\Sigma}_1) \det(\mathbf{\Sigma}_2)}{\det(\mathbf{\Sigma}_{1+2})} \right) \tag{3}$$

where $\mathbf{\Sigma}_1$, $\mathbf{\Sigma}_2$ and $\mathbf{\Sigma}_{1+2}$ denote the covariance matrices of individual embeddings $\mathbf{Emb}_1$ and $\mathbf{Emb}_2$ and the covariance matrix of associated with the joint distribution. The computed mutual information of the embedding pairs are given in Table 5. We see that the E+T pair has the lowest mutual information, verifying our hypothesis. Furthermore, note that the E+U pair has the maximum mutual information by a considerable margin. We hypothesize that this is because both Elmo and Unirep embeddings are pretrained with LSTM varieties and thus they mostly capture overlapping sets of features. We believe this overlap alongside with the curse of dimensionality is the reason the pair E+U mostly underperforms when compared with Elmo. Although these hypotheses require more comprehensive experiments, the preliminary results look promising.

7 Conclusions & Future Work

In this project, via experiments in Remote Homology Classification task, we demonstrated that the protein embeddings output by Elmo [6], UniRep [20] and Transformer [7] models have complementary sets of features. Further, this complementarity can be exploited by combining these embeddings through different ensemble methods. We showed that both traditional and learning-based ensemble methods yield meta-embeddings which, in turn, outperform individual embeddings.

We also proposed two novel hypotheses regarding the information-theoretic interpretation of the comparative performances of the embedding pairs. In these hypotheses, we suggested that pairwise mutual information between the embeddings is closely related to the observed performance boosts regarding each embedding pair. Beyond this project, as future work, we plan to test these hypotheses experimentally in different tasks with several word embeddings and larger dataset sizes.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[2] A. Bairoch and B. Boeckmann, "The swiss-prot protein sequence data bank," *Nucleic acids research*, vol. 19, no. Suppl, p. 2247, 1991.

[3] T. U. Consortium, "UniProt: a worldwide hub of protein knowledge," *Nucleic Acids Research*, vol. 47, pp. D506–D515, 11 2018.

[4] B. E. Suzek, Y. Wang, H. Huang, P. B. McGarvey, C. H. Wu, and U. Consortium, "Uniref clusters: a comprehensive and scalable alternative for improving sequence similarity searches," *Bioinformatics*, vol. 31, no. 6, pp. 926–932, 2015.

[5] E. C. Alley, G. Khimulya, S. Biswas, M. AlQuraishi, and G. M. Church, "Unified rational protein engineering with sequence-only deep representation learning," *bioRxiv*, 2019.

[6] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[7] R. Rao, N. Bhattacharya, N. Thomas, Y. Duan, P. Chen, J. Canny, P. Abbeel, and Y. Song, "Evaluating protein transfer learning with tape," in *Advances in Neural Information Processing Systems*, pp. 9689–9701, 2019.

[8] J. Mistry, S. Chuguransky, L. Williams, M. Qureshi, G. A. Salazar, E. L. Sonnhammer, S. C. Tosatto, L. Paladin, S. Raj, L. J. Richardson, *et al.*, "Pfam: The protein families database in 2021," *Nucleic acids research*, vol. 49, no. D1, pp. D412–D419, 2021.

[9] L. S. Tavares, C. d. S. F. d. Silva, V. C. Souza, V. L. d. Silva, C. G. Diniz, and M. D. O. Santos, "Strategies and molecular tools to fight antimicrobial resistance: resistome, transcriptome, and antimicrobial peptides," *Frontiers in microbiology*, vol. 4, p. 412, 2013.

[10] A. Ben-Hur and D. Brutlag, "Remote homology detection: a motif based approach," *Bioinformatics*, vol. 19, no. suppl_1, pp. i26–i33, 2003.

[11] J. Hou, B. Adhikari, and J. Cheng, "Deepsf: deep convolutional neural network for mapping protein sequences to folds," *Bioinformatics*, vol. 34, no. 8, pp. 1295–1303, 2018.

[12] A. Andreeva, E. Kulesha, J. Gough, and A. G. Murzin, "The SCOP database in 2020: expanded classification of representative family and superfamily domains of known protein structures," *Nucleic Acids Research*, vol. 48, pp. D376–D382, 11 2019.

[13] A. Villegas-Morcillo, S. Makrodimitris, R. van Ham, A. M. Gomez, V. Sanchez, and M. Reinders, "Unsupervised protein embeddings outperform hand-crafted sequence and structure features at predicting molecular function," *bioRxiv*, 2020.

[14] W. Yin and H. Schütze, "Learning meta-embeddings by using ensembles of embedding sets," *arXiv preprint arXiv:1508.04257*, 2015.

[15] J. Coates and D. Bollegala, "Frustratingly easy meta-embedding–computing meta-embeddings by averaging source word embeddings," *arXiv preprint arXiv:1804.05262*, 2018.

[16] D. Kiela, C. Wang, and K. Cho, "Context-attentive embeddings for improved sentence representations," *CoRR*, vol. abs/1804.07983, 2018.

[17] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[19] T. M. Cover, *Elements of Information Theory*. John Wiley & Sons, 2006.

[20] E. C. Alley, G. Khimulya, S. Biswas, M. AlQuraishi, and G. M. Church, "Unified rational protein engineering with sequence-only deep representation learning," *bioRxiv*, p. 589333, 2019.