Computational Alignment of Single-Cell Multi-Omics Data

Dash Pickrell / CEng / CSSE: Proteins with similar functions can have different DNA sequences between species making direct matching difficult. We investigate whether we can use machine learning to match proteins between species based on network structure. PI: Dr. Borislav Hristov

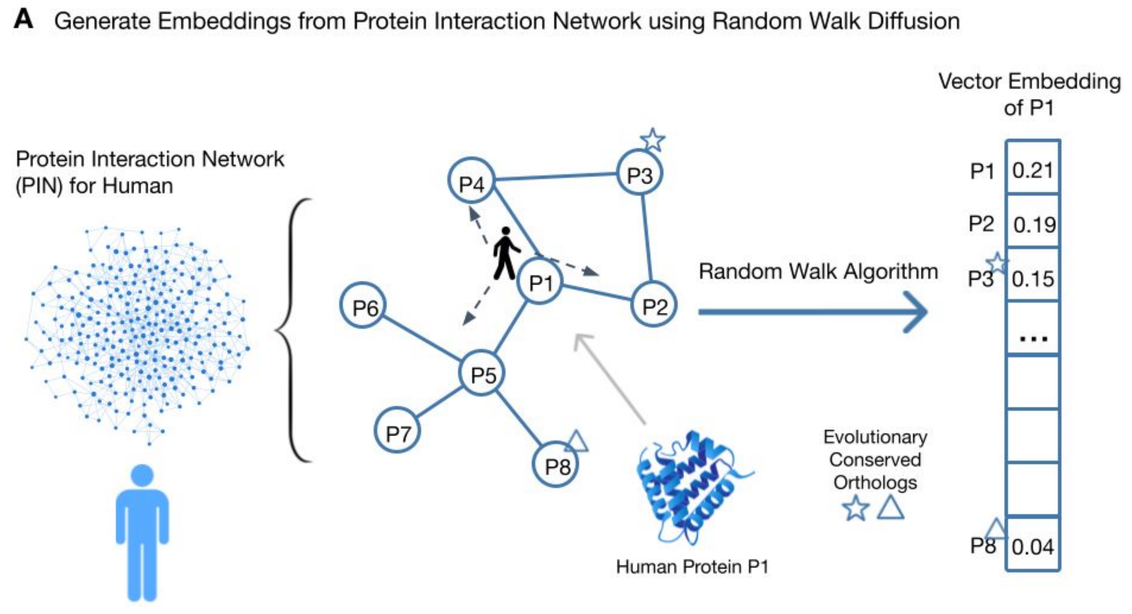


Figure A: The graph represents all the protein-protein interactions in the human. We use random walks to sample the local structure of the graph, and encode each node/protein as a vector representing its location.

Abstract

Matching proteins between species is important for understanding evolution and translating biological knowledge between species. We want to predict what a protein is in one species if we know a protein in another species. We model the proteins as a protein-protein interaction network, extract their location into a vector using random walks, and train aligned autoencoders to allow us to translate proteins from one species to another. For the median protein, we were able to predict the corresponding protein within 1% of all proteins, compared to 50% if we picked a protein at random.

Introduction

This work builds on machine learning and computational biology. Unsupervised ML models can be used to extract meaningful information from biological data. Autoencoders are commonly used for dimensionality reduction, but it is uncommon for them to be used to translate proteins between species. We predict that autoencoders along with meaningful features can be used to reduce two species to a shared dimension, and then with careful training align the shared space in order to translate proteins.

Methodology

Data Preprocessing and Feature Extraction

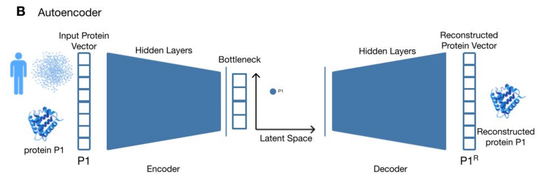
We construct a network for a given species where each node is a protein and each edge represents whether two proteins interact. Proteins with similar functions are more likely to have edges. We use a random walk algorithm to sample the local structure of the graph and encode the location of the node/protein as a vector (figure A).

Autoencoders

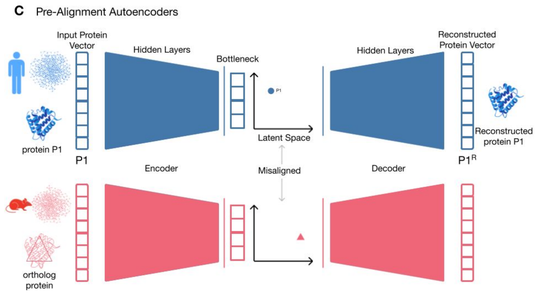
Using the vectors, we train an autoencoder on each species (figure B). Autoencoders learn to encode the protein into a compressed representation, and then decode it into a reconstructed version of the original vector. If we have two species with an autoencoder for each, we can compress proteins to a shared space but they are completely unrelated (figure C).

Alignment Function

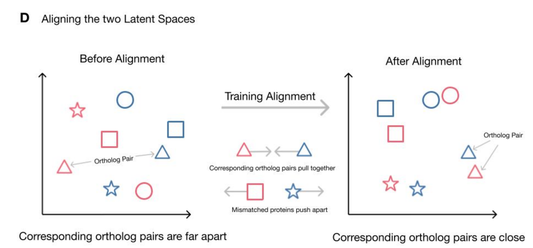
Different species have many proteins that perform the same function. We call these pairs orthologs. We want a way to predict the ortholog in one species, say human, if the matching protein is known in mouse. We can align the spaces with a loss function pulling known pairs together and pushing mismatched proteins apart. This allows us to convert the proteins between species.



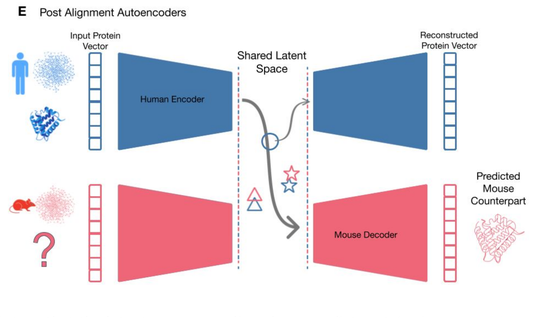
B: An autoencoder is a type of neural network trained to copy its input to its output. The autoencoder must first encode the data to a bottleneck layer smaller than the input size, and then decode it to the original. An autoencoder learns to compress data while minimizing reconstruction error (www.tensorflow.org/tutorials/generative/autoencoder). For our project, the autoencoder is useful because we can compress high dimensional data (~18k) into a lower dimension (256) fixed size latent space (the area represented in the bottleneck). We use the human vectors generated in figure A as input, and train the autoencoder to reconstruct these vectors. Since the autoencoder is trained on human proteins, it will generalize to reconstruct human proteins it hasn't seen before.



C: We train an autoencoder for different species: human and mouse. Human and mouse have different amounts of training data available, meaning they have different network sizes. The input vector dimension is the number of nodes in the network, 18k for human and 6k for mouse. By training two autoencoders with the same bottleneck size, we represent proteins in both species in a shared space. At this point we could put an encoded human protein into the mouse decoder, but the output would be random since there is no relation between the meaning in the two latent spaces.

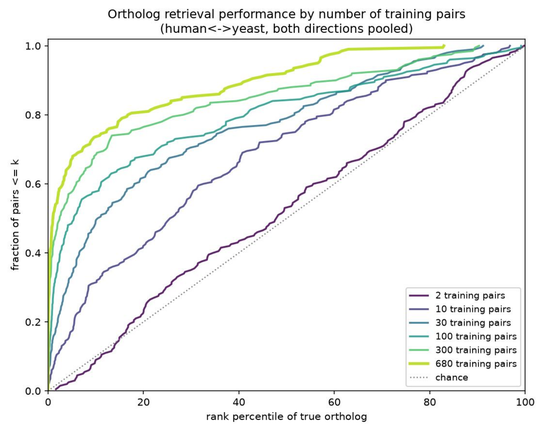


D: Different species have many proteins that have the same function. These pairs are called orthologs. For example, both humans and mice have a hemoglobin protein. If we know the function of a protein in one species, say mouse, and which proteins it interacts with, we want to be able to predict what the equivalent protein in humans is. One way to do this is by putting the known mouse protein through the mouse encoder and out through the human decoder instead of the mouse decoder. This will only output near the equivalent human protein if the latent spaces are aligned, meaning related proteins between species are nearby. To do this, we train alignment on the two autoencoders with a custom loss function. Using training ortholog pairs, we add a loss term pulling matching pairs closer, pushing mismatched pairs further, and keeping the original reconstruction term for the autoencoders.



E: Now that the latent spaces are aligned, we predict proteins in mouse using known proteins in human. We take a known human protein and encode it. In the latent space, the area the vector is in corresponds to both the human and mouse version of the same protein. We put it through the mouse decoder, yielding the predicted mouse counterpart.

MAIN RESULTS



Performance: We measure the performance of the model by ranking how close the predicted protein was to the true protein, compared to all the other proteins. The graph reads the fraction of test pairs that ranked better than the percentile on the X axis. Our best model, trained on 680 pairs, had half the test pairs rank in the top 1% of closest proteins. In other words, the model was able to predict much closer to the true protein compared to a random baseline. More training pairs results in better performance.

HYPERPARAMETER SEARCH

Parameter	Values Tried
alpha (loss)	0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 0.99
beta (loss)	0.001, 0.02, 0.05, 0.1, 0.2, 0.99
margin (loss)	0.001, 0.01, 0.2, 0.4, 0.6, 0.8
hidden layers	1, 2
bottleneck	64, 128, 256, 512, 1024
learning rate	1e-6, 1e-5, 1e-4, 1e-3
weight decay	0.0, 1e-5

Tuning: We conduct a hyperparameter search to find the best performance for our model, using the San Diego Supercomputer (2).

REFERENCES

1 "Intro to Autoencoders." TensorFlow, www.tensorflow.org/tutorials/generative/autoencoder. Accessed 23 Sept. 2026.

2 "San Diego Supercomputer Center." University of California San Diego, www.sdsc.edu. Accessed 23 Sept. 2026.