ADVANCED DEEP LEARNING

Lab 9: Applications of Deep Learning in Biomedicine

Authored by: Giannis Nikolentzos & Johannes Lutzeyer
Taught with: Guillaume Lachaud & Robin Courant

06 March 2025

We will use Python, and the following three libraries: (1) PyTorch (`https://pytorch.org/`), (2) NetworkX (`http://networkx.github.io/`), and (3) Graphein (`https://graphein.ai/`).

1 Learning with Proteins

Proteins are large, complex molecules that are present in all living organisms. They are typically made up of several organic compounds, known as amino acids, that bond with each other to form long chains. There are 20 different amino acids that commonly occur in the proteins of living organisms. Small proteins may contain just a few hundreds of amino acids, while large proteins may contain thousands of amino acids. Amino acids are connected together by a series of peptide bonds to form a polypeptide, while the elements of water are removed, and what remains of each amino acid is called an amino acid residue. Proteins consist of such polypeptide chains. Notably, proteins are associated with specific functions. For instance, enzymes catalyze chemical reactions. They enable an organism to create the chemical substances necessary for life, to convert them into other substances, and to degrade them.

There are four levels of protein structure: (1) primary, (2) secondary, (3) tertiary, and (4) quaternary structure. The lowest level, a protein's primary structure, is its sequence of amino acids. The four levels of protein structure are illustrated in Figure 1. The function of a protein is highly dependent on its three-dimensional (i.e., tertiary) structure. However, protein folding, i.e., the physical process by which a protein chain is translated to its native three-dimensional structure, is still not fully understood. Experimentally resolving the structure of a protein remains a difficult, time- and cost-intensive task. Thus, a lot of efforts have been devoted to the development of computational approaches that would allow us to predict the three-dimensional structure faster and more cheaply. Recently, DeepMind has released AlphaFold2, a deep learning model that can predict the structure of proteins in a few minutes with high accuracy [2].

The resulting structure is typically stored as a Protein Data Bank (PDB) file. The PDB file format is a textual file format describing the three-dimensional structures of molecules held in the Protein Data Bank. More specifically, the PDB format provides a description and annotation of protein and nucleic acid structures including atomic coordinates, secondary structure assignments, as well as atomic connectivity. Figure 2 shows an example of a PDB file.

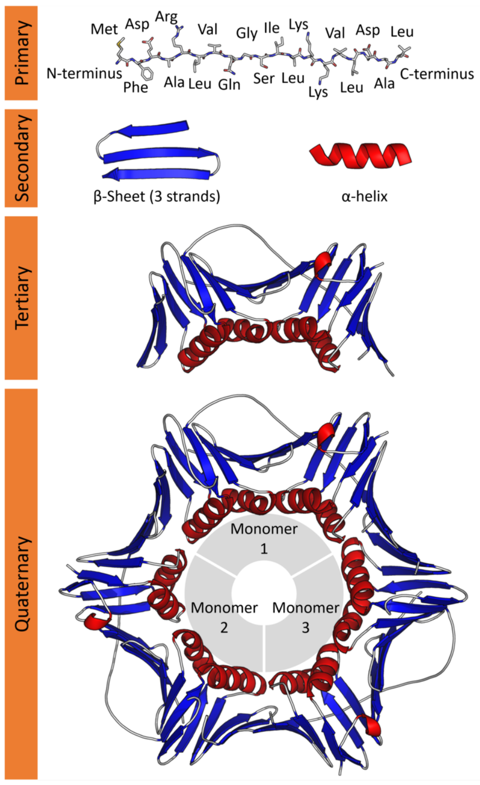
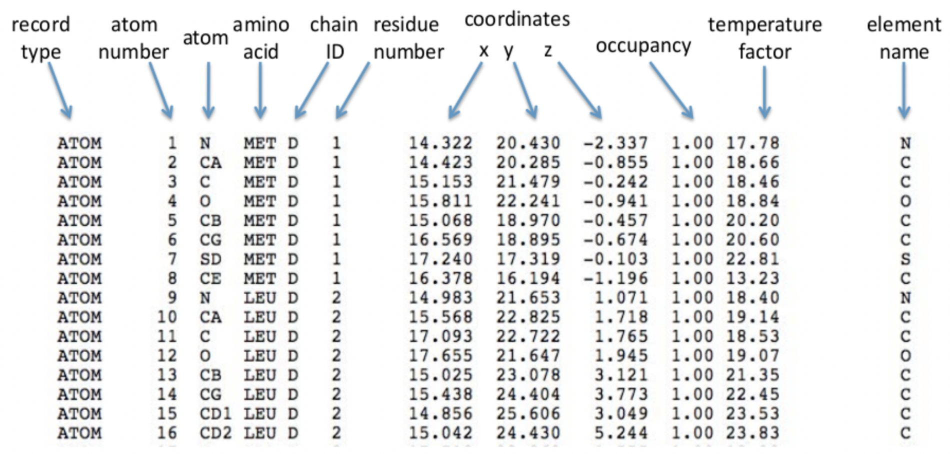


Figure 1: Protein structure (from Wikipedia).

1.1 Protein Graph Model

We design our graph models to contain information about the structure and chemical properties of a protein. Specifically, we model proteins as attributed and undirected graphs. Each graph $G = (V, E)$ represents exactly one protein and is constructed as follows: Each node of the graph $v \in V$ represents one amino acid. Each node is also annotated with a feature vector $\mathbf{x}_v$. This vector stores the one hot representation of the amino acid type (there are 20 different amino acids) along with some features related to the amino acid such as its molecular weight, its polarity, and others. The edges of the graph connect amino acids to each other. In our setting, multiple types of edges can occur. More specifically, two amino acids v, u are connected by an edge if at least one of the following holds:
(1) v and u are adjacent in the primary sequence and a peptide bond occurs between them.
(2) v and u are spatially close to each other. For this type of edges, we consider pairs of amino acids whose distance is less than a pre-defined threshold ϵ usually chosen between $6.5\,\text{Å}$ and $9.5\,\text{Å}$ in the relevant literature [1]. Here, we will set the value of the threshold to $8\,\text{Å}$.
(3) v is one of the k nearest neighbors of u or u is one of the k nearest neighbors of v (based on their Euclidean distance).
(4) hydrogen bonds exist between residues (i.e., amino acids that bond with each other).

We will next use the Graphein package to download the predicted structure of a protein and to convert the protein into a graph.



```
ATOM      1  N   MET D   1      14.322  20.430  -2.337  1.00 17.78           N
ATOM      2  CA  MET D   1      14.423  20.285  -0.855  1.00 18.66           C
ATOM      3  C   MET D   1      15.153  21.479  -0.242  1.00 18.46           C
ATOM      4  O   MET D   1      15.811  22.241  -0.941  1.00 18.84           O
ATOM      5  CB  MET D   1      15.068  18.970  -0.457  1.00 20.20           C
ATOM      6  CG  MET D   1      16.569  18.895  -0.674  1.00 20.60           C
ATOM      7  SD  MET D   1      17.240  17.319  -0.103  1.00 22.81           S
ATOM      8  CE  MET D   1      16.378  16.194  -1.196  1.00 13.23           C
ATOM      9  N   LEU D   2      14.983  21.653   1.071  1.00 18.40           N
ATOM     10  CA  LEU D   2      15.568  22.825   1.718  1.00 19.14           C
ATOM     11  C   LEU D   2      17.093  22.722   1.765  1.00 18.53           C
ATOM     12  O   LEU D   2      17.655  21.647   1.945  1.00 19.07           O
ATOM     13  CB  LEU D   2      15.025  23.078   3.121  1.00 21.35           C
ATOM     14  CG  LEU D   2      15.438  24.404   3.773  1.00 22.45           C
ATOM     15  CD1 LEU D   2      14.856  25.606   3.049  1.00 23.53           C
ATOM     16  CD2 LEU D   2      15.042  24.430   5.244  1.00 23.83           C
```

Figure 2: PDB File description.

Task 1
Fill in the missing code in the `pdb_to_graph.py` file to download the structure of protein `Q5VSL9` as predicted by AlphaFold2. Then, construct a NetworkX graph that represents the `Q5VSL9` protein using the graph model described above. (Hint: to download the structure of a protein and to construct its graph representation, you can use the `download_alphafold_structure()` and `construct_graph()` functions of Graphein, respectively).

Subsequently, we will study the properties of the generated graph.

Task 2
Compute the mean, median, maximum and minimum degree of the nodes of the graph. Then, plot the distribution of node degrees by residue type (Hint: use the `plot_degree_by_residue_type()` function of Graphein to obtain a Plotly figure and then the `write_image()` function to store the plot in the disk). Plot the distribution of edge types in the graph (Hint: use the `plot_edge_type_distribution()` function of Graphein followed by the `write_image()` function). Plot the residue composition of the graph (Hint: use the `plot_residue_composition()` function of Graphein followed by the `write_image()` function). Finally, plot the protein structure graph (Hint: use the `plot_protein_structure_graph()` function of Graphein).

1.2 Antibiotic Resistance Classification

Once the graph representations of proteins are generated, we can feed them to graph learning algorithms (e.g., graph kernels or GNNs) to deal with different problems that emerge in the field of bioinformatics. Several of these problems have attracted a lot of attention recently, and in the context of this lab, we will focus on the task of antibiotic resistance classification. Due to the overuse and misuse of antibiotics, bacteria can develop immunity to drugs, rendering them ineffective and thus posing a serious threat to global health. Identifying and classifying the genes responsible for this resistance is critical for the prevention, diagnosis, and treatment of infections as well as for understanding the underlying mechanisms.

In this part of the lab, we will formulate antibiotic resistance classification as a graph classification task. We will then train a GNN to predict the antibiotic resistance class associated with each protein.

The proteins have already been converted into graphs using the protein graph model described above. Note that AlphaFold2 was employed to compute the structure of all proteins. The dataset consists of $3,002$ proteins which are divided into two antibiotic resistance classes: (1) BETA-LACTAM class, and (2) FOLATE-SYNTHESIS-INHABITOR class.

You will next implement a GNN model that consists of two message passing layers followed by a sum readout function and then, by two fully-connected layers. The first layer of the model is a message passing layer:

$$\mathbf{Z}^{(1)} = \text{ReLU}(\tilde{\mathbf{A}}\,\mathbf{X}\,\mathbf{W}^{(1)})$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\mathbf{W}^{(1)}$ is a trainable matrix (we omit bias for clarity). The second layer of the model is again a message passing layer:

$$\mathbf{Z}^{(2)} = \tilde{\mathbf{A}}\,\mathbf{Z}^{(1)}\,\mathbf{W}^{(2)}$$

where $\mathbf{W}^{(2)}$ is another trainable matrix (once again, we omit bias for clarity). The two message passing layers are followed by a readout layer which uses the sum operator to produce a vector representation of the entire graph:

$$\mathbf{z}_G = \text{READOUT}(\mathbf{Z}^{(2)})$$

where READOUT is the readout function (i.e., the sum function). The readout layer is followed by two fully-connected layers which produce the output (i.e., a vector with one element per class):

$$\mathbf{y} = \sigma\big(\text{ReLU}(\mathbf{z}_G\,\mathbf{W}^{(3)})\,\mathbf{W}^{(4)}\big)$$

where $\mathbf{W}^{(3)}, \mathbf{W}^{(4)}$ are matrices of trainable weights (biases are omitted for clarity) and $\sigma(\cdot)$ denotes the softmax function.

Task 3
Implement the architecture presented above in the `models.py` file. More specifically, add the following layers:
– a message passing layer with h_1 hidden units (i.e., $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h_1}$) followed by a ReLU activation function
– a dropout layer with with p_d ratio of dropped outputs
– a message passing layer with h_2 hidden units (i.e., $\mathbf{W}^{(2)} \in \mathbb{R}^{h_1 \times h_2}$)
– a readout function that computes the sum of the node representations
– a batch normalization layer
– a fully-connected layer with h_3 hidden units (i.e., $\mathbf{W}^{(3)} \in \mathbb{R}^{h_2 \times h_3}$) followed by a ReLU activation function
– a dropout layer with with p_d ratio of dropped outputs
– a fully-connected layer with n_{class} hidden units (i.e., $\mathbf{W}^{(4)} \in \mathbb{R}^{h_3 \times n_{class}}$) where n_{class} is the number of different classes
(Hint: use the `Linear()` module of PyTorch to define a fully-connected layer. You can perform a matrix-matrix multiplication using the `torch.mm()` function).

We will next iterate over the different batches to train the model. Once the model is trained, we will classify the proteins of the test set into the two anitibiotic classes.

Task 4
Execute the `arg_classification.py` script to train and evaluate the model.

References

[1] Jun Huan, Deepak Bandyopadhyay, Wei Wang, Jack Snoeyink, Jan Prins, and Alexander Tropsha. Comparing Graph Representations of Protein Structure for Mining Family-Specific Residue-Based Packing Motifs. *Journal of Computational Biology*, 12(6):657–671, 2005.

[2] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.