Grenoble INP - Ensimag

National School of Computer Science and Applied Mathematics

Academic year 2022 - 2023

Assistant Engineer Internship Report
Performed at CNRS

Antonin Point

First Year

Geometrical deep learning of proteins structure
June-July 2023

Laboratoire Jean Kuntzmann

700 Av. Centrale,

38400 Saint-Martin-d'Hères

Under the supervision of

Sergei Grudinin (CNRS)

JIMENEZ GARCES Sonia (ENSIMAG)

Abstract

Proteins are large chained molecules that perform various functions in different organisms. As they are essential to the development and the monitoring of our lives, we need to better understand their behaviour and all opportunities they offer us.

This project studies the spatial arrangement of proteins using geometrical deep learning. Combining biological knowledge and AI computing power helps us predict their specific configurations and has a lot of applications in the medical area such as cancer detection.

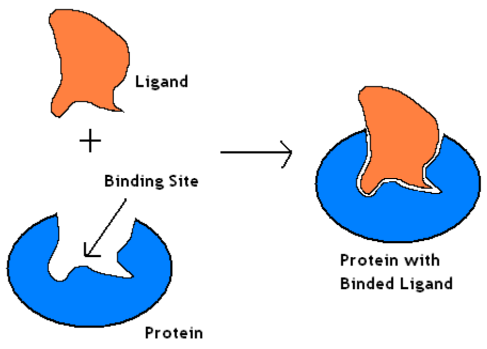

Figure 1: A schematic representation of a binding between a ligand and a protein. The protein shape, generally, changes upon the binding.

More precisely, the project focuses on finding protein-protein binding sites (PPBS); small pockets where ligands (molecules) can bind to (and therefore change the structure/functions of a protein). In that way, we tried to modify the architecture of an existing PPBS prediction model called ScanNet aiming at having a more precise model to predict the existence of those binding sites.

Acknowledgements

The CNRS is a research institution that covers many fields of study to face new challenges and promote science and innovation across societies. I would like to thank them for accepting me as a trainee for this internship.

The Jean Kuntzmann Laboratory (LJK) is a research institution focused on 2 disciplines : Computer Science and Applied Mathematics. With more than 250 researchers and 17 teams, this laboratory promotes discovery of new technologies and theories to face social and environmental challenges. I would like to thank them for accepting me inside their offices for this internship.

The DAO team (LJK) deals with mathematical optimization and its interactions with machine learning, statistics, and signal processing. I would like to thank Sergei Grudinin and all the members of the team for advising and supporting me during this experience as an engineer trainee.

The NANO-D team (INRIA) focuses on developing adaptive multi-scale algorithms for molecular systems. I would like to thank them for their advise and support to better understand the behaviour of those systems and how to manipulate Artificial Intelligence to deal with my project.

I would also like to thank Jerome Tubiana for helping me understand the functioning of ScanNet and for his support to expand the network's architecture.

Contents

Abstract 2

Acknowledgements 3

1 Introduction 1

1.1 Two main solutions to identify binding sites . . . 1

1.2 Neural network and deep learning . . . 1

1.3 ScanNet . . . 2

1.4 Positional encoding and SE3 operations . . . 3

1.5 Building a more precise model . . . 5

1.6 SignNet . . . 5

1.7 Objectives . . . 6

2 New version of ScanNet 8

2.1 Vocabulary . . . 8

2.2 PDB files preprocessing . . . 8

2.3 Spatial Alignement . . . 9

2.4 Sign invariance . . . 10

2.5 Extension of ScanNet architecture . . . 11

2.6 Tests & Training . . . 13

3 Results and Analysis 15

4 Personnal Overview 18

5 Conclusion 20

Chapter 1

Introduction

In living organisms, proteins monitor biological processes that are crucial to the functionning of our systems. The characteristics of a protein are defined by its shape and interactions with other molecules, we will call them ligands from now on. Ligands bind to a protein on specific areas known as binding sites. Predicting these binding sites is very useful achievement to better understand the functioning of proteins and has many applications such as drug design or disease detection.

1.1 Two main solutions to identify binding sites

To predict binding sites, two classes of methods currently prevail (Jérôme Tubiana and Wolfson 2022,introduction).

On the one hand, comparative modeling, which consists of recognizing patterns of behavior or spatial arrangement from a certain protein among other proteins. On the other hand, feature-based machine learning dealing with exploiting features (such as molecular surface curvature or polarity) of amino acids[1] to identify binding sites patterns. The second approach offers many advantages compared to the first one. Indeed, it can deal with larger sets of data, it offers the possibility to generalize the process to proteins without similarity with the dataset, it has fast inference speed and more.

1.2 Neural network and deep learning

Deep-learning models (or neural networks) are composed of layers of neurons. Each neuron of a specific layer is connected to neurons of other layers (the next one) trough

[1]molecules used to create proteins

weighted-edges. Each neuron performs some operations (linear or not) on its inputs (edges) and returns a certain result. This process goes from layer to layer until it reaches the output layer, where output data are identified with a label in most of machine-learning problems. The output can then be used in two ways. The first one deals with training the network. In this case, the network calculates an error between what was expected and what is obtained, and consequently diminishes this error (if needed) using *backpropagation*[2] (consisting of adjusting weights) over the layers. Repeating the process helps obtaining a more precise model. Once the error reaches an acceptable level, the output data can be used in a second way : making prediction on input data depending on the information stored in output data (recognize a certain pattern for example). Figure 1.1 schematically shows a typical fully-connected neural network.

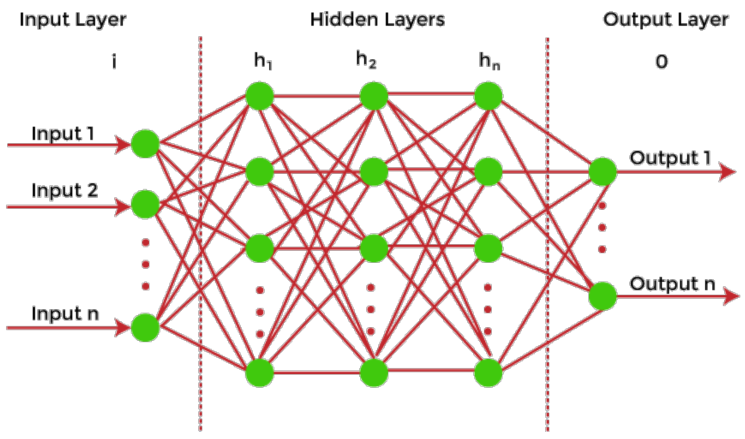

Figure 1.1: Schematic representation of a fully-connected deep neural network with three hidden layers.

1.3 ScanNet

In June 2022, Jerome Tubiana, Dina Schneidman-Duhovny and Haim J. Wolfson released an interpretable geometric deep learning model for structure-based protein binding site prediction called ScanNet (Jérôme Tubiana and Wolfson 2022). ScanNet relies on building representations of atoms[3] and amino-acids using spatio-chemical arrangement of their neighbors. Concretely, the network builds three-dimensional representations of each atom and amino acid (and a 3D graph) based on the spatio-chemical arrangement of a certain number of their neighbors (referred as K) and learns patterns (motifs) to predict the presence of binding sites via labels associated to each amino acid. Binding sites are associated with residues (amino acids) that have a high affinity

[2]for more details please see https://en.wikipedia.org/wiki/Backpropagation
[3]molecule components

PPIs (Protein-Protein Interactions).

Here is brief overview on how ScanNet is working : The network takes as input PDB[4] files (Jane S. Richardson 2021) and computes geometrical information at both amino-acid and atom levels (see blue blocks in Figure 1.2). The Point Cloud (spatial position of elements, cop), the Frame Index Triplet (frame assigned to a particular element) and the Sequence indexing blocks are used to create a *global frame*. This global frame represents a graph where nodes are atoms/amino acids and distances/angles between them constitute the edges. A local frame is also assigned to to each component so that the network can precisely encompass spatial relationships between nearest neighbors (discussed in the next section). Attributes (name of the amino acid for example) are discrete motifs that make each element unique and also help the network find contextual relationship among input data.

ScanNet uses these attributes to create a continuous representation (vectors) of information through a process called ”Embedding”. Starting at the atom level (upper right in Figure 1.2), the network combines embedded attributes with *local frames* to compute the continuous (vectors) dependencies between an atom and its K ($K = 16$ in our work) closest neighbors (process called ”Neighborhood Computation/Embedding”. Then it applies some hidden layers (attention pooling layers) to the resulting vectors and identifies relationships among this huge set of data. Especially attention pooling is an essential part of ScanNet as it helps to adjust weights in the neural network during the back propagation process. The result is then concatenated with attributes embedded at the amino-acid level (left part in Figure 1.2) and repeats the same process once more. In the end, the soft-max function enables ScanNet to transform the continuous representation into a probability distribution, i.e. the prediction of binding sites location in the input data.

1.4 Positional encoding and SE3 operations

As our architecture had to recognize actions between atoms/amino acids rather than components themselves, we used a technique called ”pose normalization” to make the network as invariant as possible to pose variation. More precisely, we have to make sure that it is equivariant with respect to $SE3$ operations (3D rotations and translations), which means that if, for example, the whole input 3D input data is rotated (or translated) the result should be the same as we would have considering its initial configuration. Such an equivariant representation ensures stable performance in the presence of nuisance transformations of the data input.

[4]Protein Data Bank files contain information ScanNet uses to train itself and predict the existence of bindingsites on a protein

(b)

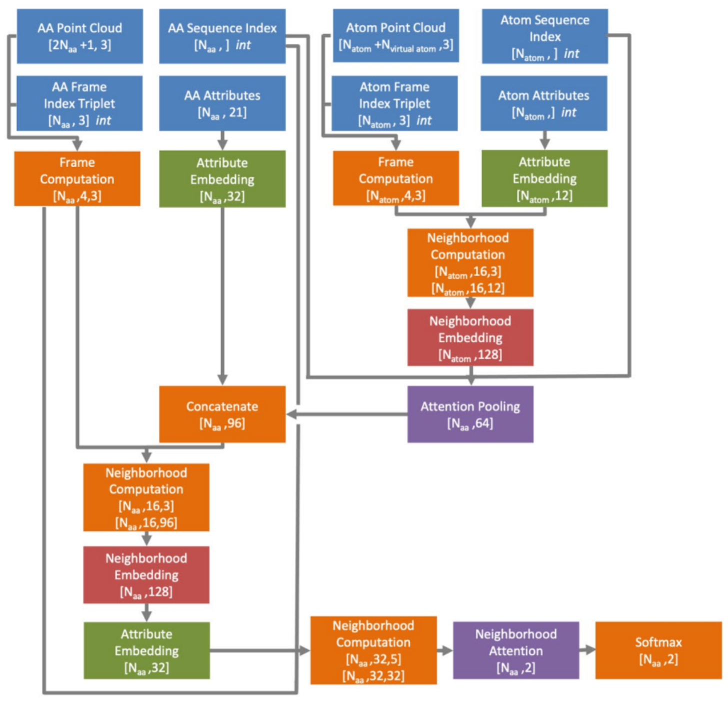
AA Point Cloud [2N$_{aa}$ +1, 3]
AA Sequence Index [N$_{aa}$,] int
Atom Point Cloud [N$_{atom}$ +N$_{virtual\ atom}$,3]
Atom Sequence Index [N$_{atom}$,] int
AA Frame Index Triplet [N$_{aa}$, 3] int
AA Attributes [N$_{aa}$, 21]
Atom Frame Index Triplet [N$_{atom}$, 3] int
Atom Attributes [N$_{atom}$,] int
Frame Computation [N$_{aa}$,4,3]
Attribute Embedding [N$_{aa}$,32]
Frame Computation [N$_{atom}$,4,3]
Attribute Embedding [N$_{atom}$,12]
Neighborhood Computation [N$_{atom}$,16,3] [N$_{atom}$,16,12]
Neighborhood Embedding [N$_{atom}$,128]
Concatenate [N$_{aa}$,96]
Attention Pooling [N$_{aa}$,64]
Neighborhood Computation [N$_{aa}$,16,3] [N$_{aa}$,16,96]
Neighborhood Embedding [N$_{aa}$,128]
Attribute Embedding [N$_{aa}$,32]
Neighborhood Computation [N$_{aa}$,32,5] [N$_{aa}$,32,32]
Neighborhood Attention [N$_{aa}$,2]
Softmax [N$_{aa}$,2]

Figure 1.2: Original architecture of ScanNet (Jérôme Tubiana and Wolfson 2022). Colors are used to differentiate different processes : blue for input data preprocessing, green and red for embedding, purple for attention step and orange for non trainable layers. Numbers in each box detail the dimension of current data. The arrows shows the continuity of the process.

ScanNet treats proteins (ordered sequence of amino acids) and relies on a positional encoding (using global and local frames) of geometrical information so that each element (amino acid or atom) of a particular sequence is assigned to an unique representation. Positional encoding is used in deep learning to create a network that is permutation-sensitive : it helps to deal with set of data where the order of elements is important, such as words of a sentence in Natural Language Processing, for example. In our case, positional encoding ensured that if the neighborhood of an amino acid changes (in the 3D graph) then the model would consequently capture different actions and probabilities.

ScanNet is using attention layers (White 2022) to construct a representation of atoms/amino acids easier to manipulate (by reducing the dimension of the vectors). This encoding allows the model to simplify the utilization of filters mechanisms, the capture of relative position of elements from the dataset and generalize across input data with different shapes.

1.5 Building a more precise model

Despite ScanNet being very geometrically expressive, it does not consider motion of proteins which can result in the opening or the closing of a pockets (binding sites) as interaction with other ligands (proteins) may change. In that way, proteins can be imagined as 3D molecular graphs where each point (amino acid) is assigned one or numerous motions. Using this extra information may help the neural network identify more precisely real-time patterns that lead to the creation of a biding site.

This was our motivation to add motion as an extra geometrical information to help ScanNet predict the binding sites with a better accuracy. To match the constraints observed in the previous chapter, motion eigenvectors had to be positionaly encoded[5] and equivariant with respect to $SE3$ operations. To deal with these motion spatial information, we used a technique called "Laplacian positional encoding", which helped us capture relationships between nodes of our 3D graph. This positional encoding relies on building the Laplacian matrix to reflect proximity between nodes of our 3D graph (Schumacher 2023). However this method is sensible to the sign flip of Laplacian eigenvectors, that's why we needed to make the network invariant to the sign of the motion. More precisely, for each eigenvector v, both v and $-v$ had to have the same influence on predictions of the model.

1.6 SignNet

SignNet (Lim et al. 2022) is a neural network that is invariant to symmetric operations on eigenvectors. To deal with the sign ambiguity of Laplacian eigenvectors, we used this architecture to compute the sign invariance of motion vectors (SignNet is therefore a sign invariant neural network).

This network takes as input a set of vectors v_k and applies to them an *injective* function f that computes the sign-invariant representation through the following process:

$$f(v_1, ..., v_k) = \rho(\phi(v_1) + \phi(-v_1), ..., \phi(v_k) + \phi(-v_k)), \tag{1.1}$$

where ρ is a Multi Layer Perceptron (MLP,Swarnimrai 2021) and ϕ is GIN(Keyulu Xu and Jegelka 2019), a convolution operator for graph isomorphism network (which helps to keep the same relationships between nodes in the graph). A MLP is a neural network constituted of multiple (more than three) dense layers of neurons, where each neuron is connected to every neuron of the previous layer, as shown in Figure . GIN performs an aggregation on each node of the graph (in our case amino acid and more especially

[5]which means extra geometrical information must be ordered with the existing ones

their motion eigenvectors) and updates its value using the K closest neighbors at each step. This operator can be described by the following equation:

$$v_k^{i+1} = MLP\left((1+\epsilon)v_k^i + \sum_{j\in N(v_k)} (v_j^i)\right). \qquad (1.2)$$

For each vector v_k, at step $i+1$ of the learning process, v_k is updated using $N(v_k)$, a set containing its K closest neighbors and an extra parameter ϵ that gives an extra weight importance to the current vector in the process.

1.7 Objectives

To start with, this internship was about being introduced to a real scientific problem, study some state-of-the art techniques and literature, be able to independently develop some soft and hard skills to solve a problem. It was also an experience to discover the work in a team, to formulate questions and find answers, write algorithm prototypes, unit tests, analyse data and results, explore different approaches, discuss with experts and write scientific reports.

One of the main scientific objectives of this internship was to discover a new architecture of a deep neural network that is able to *expressively* and *robustly* use additional information about motion manifolds of input 3D data. Modifying the architecture of ScanNet developed by our collaborators that helped us with the implementation was the best way to do so.

When it comes to the project (encode the new version of ScanNet), we first needed to create eigenvectors related to motions. For that, we used a specific tool developed by Alexandre Hoffmann and Sergei Grudinin : NOLB (Grudinin 2017). This module takes as input parsed PDB files containing specific amino acids we want to assign a motion vector to (among a whole protein structure) and compute *normal modes* associated to this motion. The computation is performed using the eigenvectors of the associated Hessian matrix on a coarse-grained amino-acid level representation. As a results, principal (eigen) motions are described using 6-vectors per amino-acid.

The first step of our resolution was to parse input PDB files (input of ScanNet) to create personalized files that will be given as input to NOLB. Once normal modes were computed, we needed to treat and make them $SE3$-equivariant and sign-invariant (second and third steps).

The fourth step of our work would be to insert these motion vectors as extra attributes in the modified architecture of ScanNet and retrain the parameters in the *end-to-end* fashion. In the end, we trained the model on the whole dataset used to train ScanNet originaly and observed the performance of the new implementation.

Chapter 2

New version of ScanNet

This chapter explores in detail how the new version of ScanNet was built. The main logic was to assign extra motion attributes to each component of a protein to find spatial relationships between its components with a higher accuracy (more expressively). These new motion vectors represent possible movement (fluctuations) of a particular element through six axis of freedom (three translations and three rotations along each axis x,y,z of a certain local frame).

2.1 Vocabulary

To explain our methodology, we need to introduce some technical notations: As ScanNet treats huge amount of information, input data is organized in *batches* to help accelerate calculus when applying filters(thanks to parallel calculus and vectorised operations that are performed through all the vectors in the batch simultaneously). These batches are filled with default values when no more data can be stored. B denotes the batch dimension (number of batches) in every tensor (using tensors helps easily apply operations on the data set). L_{max} represents the number of amino acids selected per chain. K_{max} is the number of nearest neighbors we relate any amino acid to. This is a preselected constant in our network. m is the number of motions per amino acid.

2.2 PDB files preprocessing

To start with, we wanted to create a specific PDB file format for NOLB to compute motion eigenvectors. We decided to do it at the amino acid level (and not the atom level) to accelerate calculus when training the new model (an amino acid encompasses multiple atoms at one time).

Starting with a PDB file and a list of specific chains (of amino-acids), we write a new PDB file containing selected chains and basic information NOLB needs (type of each component, their index in the current sequence ...) from the PDB format[1]. We therefore added a couple of new functions to the module PDB_processing.py :

- supress_H that takes as input a PDB file and create a new PDB file where HETAMTM[2] group are removed.
- reduced_model that takes the output of supress_H and concatenates the different models from the PDB file into a single one (NOLB only treat single model files). Its output is a new PDB file.

Once file were fitting the NOLB requirement, we computed eigenvectors by applying NOLB to the single model files and stored in a structure folder to use them whenever we wanted (again to accelerate the calculus). NOLB generated a file with 10 motions (6 coordinates each) for each amino_acid. Then we selected for each amino acid the correct number of motion we wanted (called m in the model) using the function NOLB_to_motion_vectors. To simplify the implementation, we also created a function called apply_NOLB that resumes all those operation, suppresses intermediate files once motion vectors are computed and concatenates motions vectors into a single array with shape $[N_{aa}, m, 6]$ where N_{aa} is the number of amino_acid in the chain.

2.3 Spatial Alignement

The second step of our implementation was to align each neighbors (K closest) in the local frame of the corresponding amino acid, as ScanNet does for each geometric attribute to ensure the SE3 equivariance. To do so, we extended the original module computing the local frame alignment (network/neighborhood.py) into a new module neighborhood_modif.py, adding a bloc of codes inspired from the one computing euclidean alignment in the LocalNeigbhorhood class. Indeed as the original version of LocalNeighborhood already extracts the local frame and the K closest neighbors of each amino acid, we just needed to compute the corresponding rotation matrix and align new motion information to this local frame. In that way, we added an extra parameter called "vectors" telling us how many vectors we were working with (initialized to False if no vector is used).

More precisely, we aligned these eigenvectors using the following process :

[1]You can find more about this format on https://www.biostat.jhsph.edu/~iruczins/teaching/260.655/links/pdbformat.pdf

[2]HETMATM are not connected to other residues and must not be considered as part of the namewise residue

Let us call each motion set $[v, w]$ (representing the translation and rotation motion vectors assigned to each amino acid). To start with, we computed the set of rotation matrices M of the corresponding amino acid (let us call it "AA"), given by the frame attribute of AA. M has shape $B \times L_{max} \times m \times 3 \times 3$ and both v and w have shape $B \times L_{max} \times K \times 3$. Using the tensor element by element multiplication, we multiplied each vector related to AA (axis with shape $[K \times m \times 3]$) by the corresponding rotation matrix stored in M (axis with dimension $[m \times 3 \times 3]$), making sure that the K closest motion representations of AA were multiplied by the same rotation matrix.

In the end, we obtained a graph representing amino acids that were "correctly" bounded to their K closest neighbors (the motion vectors of which were aligned to the local frame of the amino acid).

2.4 Sign invariance

Once we obtained our SE3 equivariant model, we still needed to compute its invariance with respect to the sign of motion eigenvectors. We discussed in chapter 1 about SignNet (Lim et al. 2022), a sign invariant neural network adapted to our representation of information (graph and positional encoding). As this network is using a operator called GIN that is not implemented in our version of TensorFlow (1.1.14), it must be done manually. To do so, we created a new module called "signnet.py" in the "network" folder.

The main logic was to create the neural network as we did for ScanNet and then feed it with our eigenvectors (a more optimal version would have created a network that is fed with all the amino acid attributes).

We wanted to compute the following process :

$$f(v_1, ..., v_k) = \rho(\phi(v_1) + \phi(-v_1), ..., \phi(v_k) + \phi(-v_k)), \tag{2.1}$$

To do so, a main function called GINDeepSigns took as input the previous graph (containing each amino acid motion vectors from its K closest neighbors), the dimension of each vector (hidden_channels), ϵ and other parameter related to neural network training (batch normalization, activation fonction and dropout rate). It then initiated GIN (because we wanted to apply the same GIN to all vectors) by creating the series of layers (input,hidden and output layers) we would apply to our eigenvectors : this is what the function init_GIN is about.

As a reminder this function applies the same method to each node of the graph (vectors)

:

$$v_k^{i+1} = MLP\left((1+\epsilon)v_k^i + \sum_{j \in N(v_k)} (v_j^i)\right) \tag{2.2}$$

More precisely, for each depth of layers we appended 2 or 3 layers to SignNet architecture (in that order) : an update_graph layer, a MLP layer and a Dropout layer (the third one was used only if there were more than 2 depths of layers, which correspond to the deep learning case).

The first layer updated the graph. More especially it computed the in-brackets aggregation part of equation 2.2 for each node of the graph using a function called update_graph. The second layer (MLP) was computed using the init_MLP function. The Dropout layer, which correspond to interrupt certain connections between neurons to prevent over-fitting (meaning the model is learning too much and gives false results) already existed in the keras package.

Once GIN was ready, the main function applied it to the original eigenvectors and then iterated over the m (k goes from 1 to m in the formula) possible motions assigned to each amino acid. For each loop, the network computed g_minus, a graph were the kth motion vector is multiplied by -1 (for each amino acid). To ensure the gradient back propagates during the training process, we had to make sure that this g_minus was not a simple copy of the original graph. To meet this constraint, the function minus used strided slices to keep the connection between both graphs (this method is specific to Tensorflow 1). GINDeepSigns then computed the new motion kth vector in that way : it applied GIN to each eigenvector from g_minus, added the result to the corresponding vector in computed with the original graph and extracted the kth motion using strided slices once more(the function apply_GIN_permuts deals with this part).

In the end, we concatenated the different tensors representing the different motions (Concatenate layer) and applied a last MLP layer to complete the network.

2.5 Extension of ScanNet architecture

Whe,n that every tools needed to construct our new model were computed, we could implement our new version of ScanNet. We created a new module "scannet_modif.py" inspired from the "scannet.py" module located in the "network" folder. The function "ScanNet" is used to initialize the model (series of layers) and then to feed it. In our new version, we added an extra parameter ("motion_vectors") that refers to the number of different motions we assign to each amino acid so that the model can work with or without motion vectors (False if no motion is used).

As shown in 2.1 (details of the new architecture), we computed the input of motion vectors by adding extra Input and Masking layers as done for other input information (blue boxes). We did not change the atom level part of ScanNet.

For the amino acid level, we modified the Neighborhood Computation and Embedding so that eigenvectors were considered in both processes. More precisely, we appended the list of attributes the model received (motion vectors were considered as ”attributes”) and we added an extra attribute to each function that should treat amino acid attributes (this parameter details if motion vectors are used or not).

We also extracted aligned motion vectors from the Neighborhood Computation to treat them separately from others attributes (output of the Neighborhood Embedding part) and fed SignNet with them (by applying the series of layers that constitute this network). The output was then embedded (as every green boxes in the scheme) to capture relationships between nodes of the graph (amino acids).

In the end, we concatenated the result with the original attributes of ScanNet (Last Concatenate boxe) and continued the original process as we did for ScanNet at the beginning.

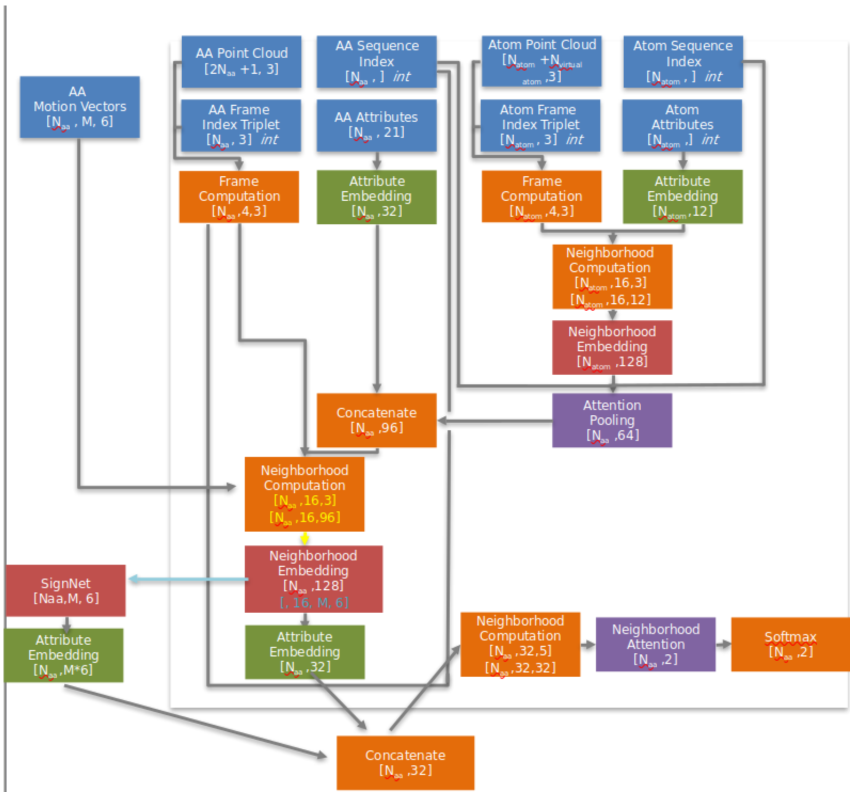
AA Motion Vectors [N_aa , M, 6]
AA Point Cloud [2N_aa +1, 3]
AA Sequence Index [N_aa ,] int
Atom Point Cloud [N_atom +N_virtual atom ,3]
Atom Sequence Index [N_atom ,] int
AA Frame Index Triplet [N_aa , 3] int
AA Attributes [N_aa , 21]
Atom Frame Index Triplet [N_atom , 3] int
Atom Attributes [N_atom ,] int
Frame Computation [N_aa ,4,3]
Attribute Embedding [N_aa ,32]
Frame Computation [N_atom ,4,3]
Attribute Embedding [N_atom ,12]
Neighborhood Computation [N_atom ,16,3] [N_atom ,16,12]
Neighborhood Embedding [N_atom ,128]
Attention Pooling [N_aa ,64]
Concatenate [N_aa ,96]
Neighborhood Computation [N_aa ,16,3] [N_aa ,16,96]
Neighborhood Embedding [N_aa ,128] [, 16, M, 6]
SignNet [Naa,M, 6]
Attribute Embedding [N_aa ,M*6]
Attribute Embedding [N_aa ,32]
Neighborhood Computation [N_aa ,32,5] [N_aa ,32,32]
Neighborhood Attention [N_aa ,2]
Softmax [N_aa ,2]
Concatenate [N_aa ,32]

Figure 2.1: New architecture of ScanNet.

2.6 Tests & Training

To verify our implementation, different test modules were created : each new function implemented was tested among those files to point out dilemma and validate their functioning observing the output information by hand.

For the PDB files computation, the module test_PDB_preprocessing.py (ScanNet/preprocessing) helped verify if new PDB files have the right format and if NOLB is working on it.

As tensors we manipulated in our network were empty at first, numerous modules were used to work on filled tensors and check if the process was working. It offered an alternative way to the exhausting debugging method when we wanted to check if a specific function using tensors was working or not.

The test_motion module (ScanNet/motion) dealt with checking the implementation of each operations used to treat eigenvectors : SE3 spatial alignment,Tensorflow operations, matrices product ...

For the creation of SignNet, we implemented a separate iterative version of the network (ScanNet/layers) where MLP and GIN were computed from scratch in a different way : a series of functions (instead of layers) to check step by step how the network was treating eigenvectors. A module test_layers.py enabled us to check if the SignNet gave us expected results. For example, we created a set of vectors s1, a copy s2 were the sign of some vectors were flipped, fed them to the same network and checked if results were the same (the difference between both results should be filled with zeros). We also tried to compare the Pytorch and our Tensorflow versions of SignNet but found that initial parameters of networks (gaussian kernels) were not initialized in the same way (hence no comparison was possible).

The final step was to train the whole network and observe results. To verify if the model was working, the original version of ScanNet offered a parameter called "check" that enabled us to train the network on a very small part of the input data set, to check if the network returned something.

We also used a platform called TensorBoard to observe if the model was training (meaning trainable parameters in the network were effectively changing). We could display the graph that sum up how the network was working and the time series to check if weights were updated among edges in the network.

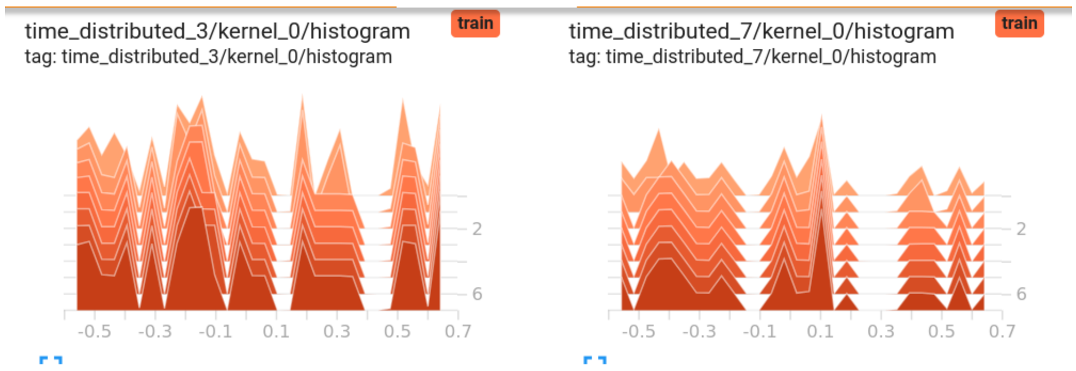

Figure 2.2: Histogram of two layers (SignNet) that are trained (TensorBoard visualization). Each curve represents the weight distribution on a particular epoch (period of training)

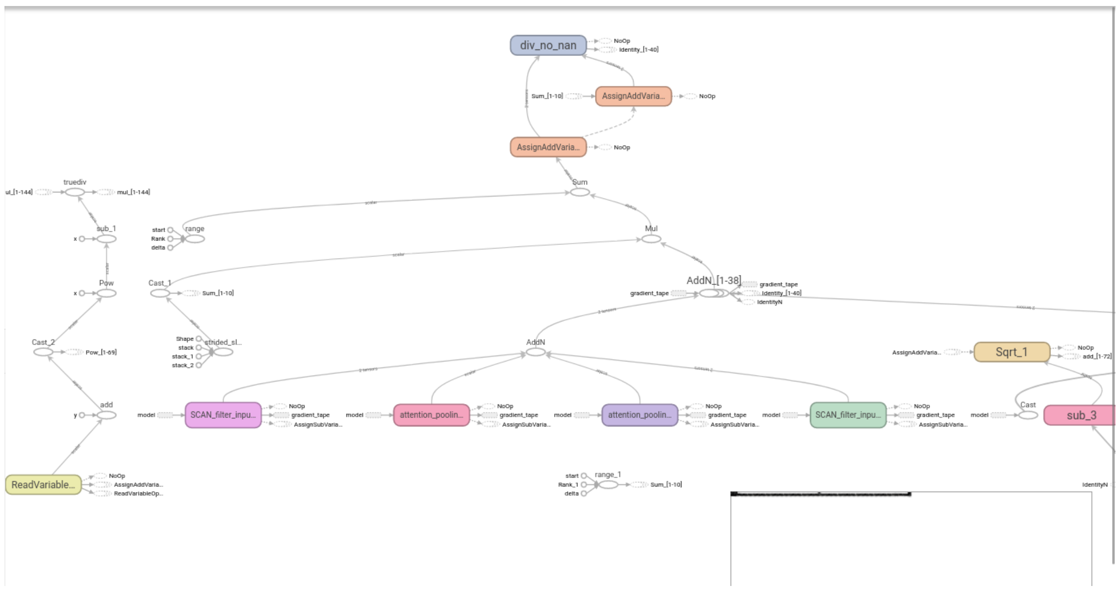

Figure 2.3: Tensorboard visualization of the network (part of it) functioning. Colored boxes conrrespond to specific layers (of the model) and node between them to a training process (gradient backpropagation for example)

Chapter 3

Results and Analysis

To start with, we managed to construct a functioning model considering motion vectors as extra attributes (running on a small data set). The next step was to train our new version of ScanNet on the whole pdb files we have (in the data set).

As ScanNet relied on a very large dataset to train itself (more than 100 Gb of data), we launched the training process using Gricad, a platform dedicated to intense calculus (accessed via the web portal PERSEUS NG). As processes required a lot of time to train (approximately 20 hours each), we could only launch 2 training sessions on the whole data set. Since we needed to see if the new model is more efficient than the previous one, we decided to launch a first experiment without motion (motion_vectors and vectors are initialized to False) and another one with 10 motions, which is the maximal number of different motions NOLB can compute.

Here are the curves showing the results for different tests (ScanNet classify each amino acid in a binding site or not, for each protein). The graphs shows Precision based on Recall (Reilly 2022). Precision measures the percentage of predictions made by the model that are correct (which means considered amino acids are indeed subjects to creation of binding sites). Recall measures the percentage of relevant data points (amino acid with high affinity PPIs) that were correctly identified by the model. The specification for the different test (Jérôme Tubiana and Wolfson 2022) is the following :

Test 70% : at least 70% sequence identity with at least one train set example.

Test homology: at most 70% sequence identity with any train set example, at least one train set example belonging to same protein superfamily (H level of CATH classification [Sillitoe et al. 2021]).

Test topology: at least one train set example with similar protein topology (T level of CATH classification), none with similar protein superfamily.

Test none: none of the above.

The summary metrics used is AUC-PR (Steen 2020), that represent the area under the Precision/Recall curve. The higher the AUC-PR score, the better our model is efficient to identify binding sites.

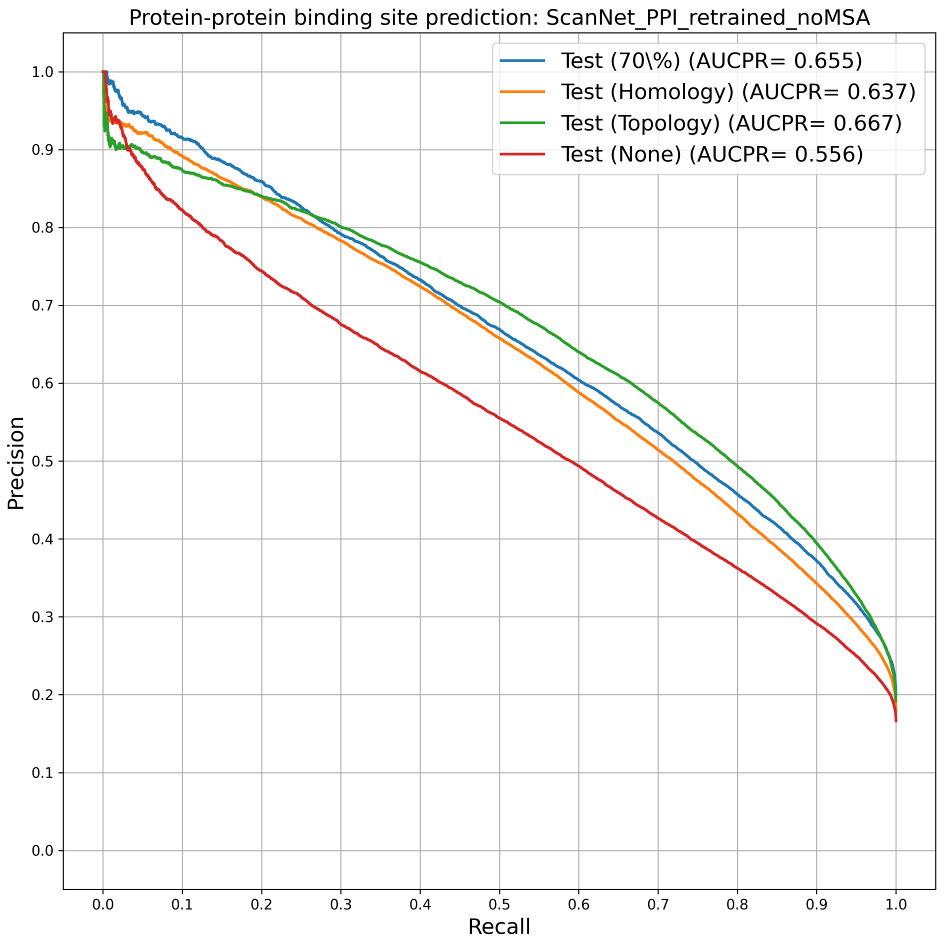

Figure 3.1: Prediction model curves (ScanNet new version) without using motion vectors.

On the whole, our new implementation of ScanNet does not seem to improve the predictions. Even though it is a little bit more efficient on the Homology and 70% tests, it shows less encouraging results on the None one.

This lack of improvement can be due to many reasons : Indeed we made some simplification in our model to facilitate its implementation : We applied the extra motion

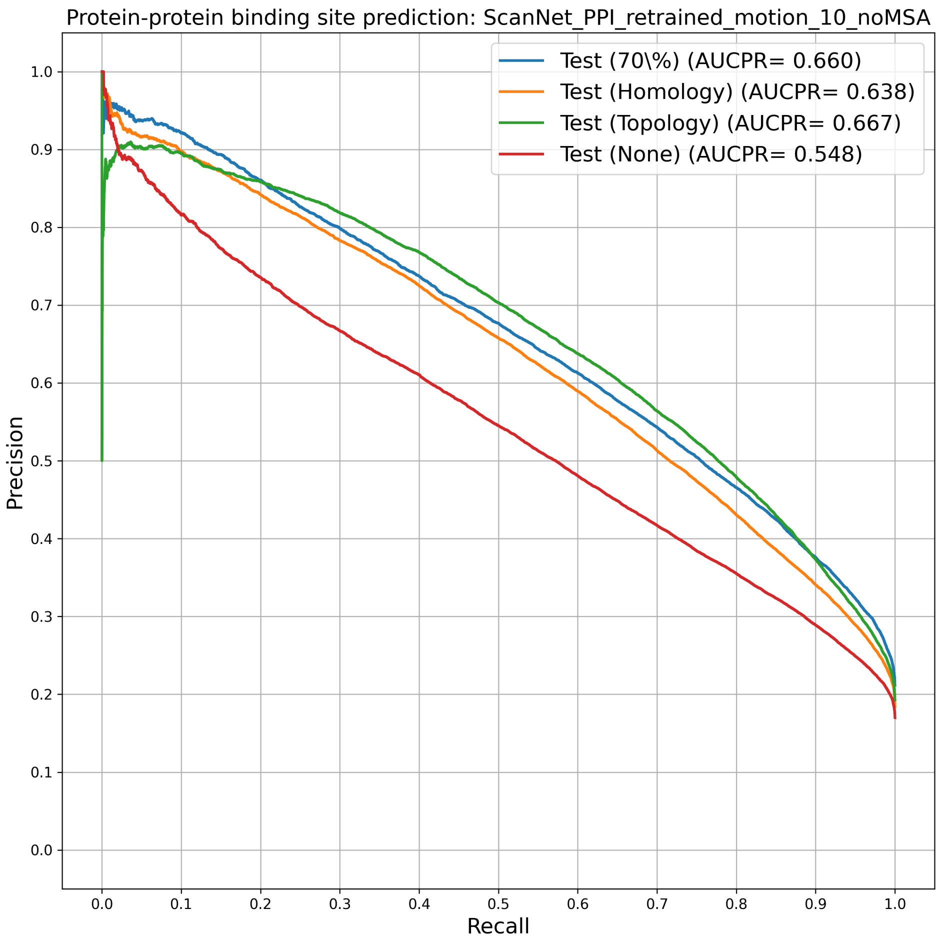

Figure 3.2: Prediction model curves (ScanNet new version) using motion vectors (10 normal modes)

strategy at the amino acid level only, we treated motion vectors instead of every geometric attribute with SignNet, we just considered using the K closest neighbors to compute spatial dependencies. Therefore extending the motion strategy at the atom level, treating every attribute in SignNet and relying on much more neighbors to capture relationships may help improve the performances of the model.

Chapter 4

Personnal Overview

This internship introduced me to many concepts and was a highly valuable experience for me : it was my first time working in a research laboratory, on a concrete project dealing with artificial intelligence (AI) and inside a team of engineers.

To start with, I started this internship without any (or just a few) knowledge about neural networks or AI at all. Thanks to a very involved supervisor (Sergei Grudinin) and all the members of the DAO team (plus many Phd students from my office) I have discovered and understood numerous aspects of neural networks. Thanks to the help of Jerome Tubiana (creator of ScanNet), I have learnt in depth how these networks work (especially ScanNet). This internship was also a great opportunity for me to manipulate various Python libraries such as Tensorflow, even though it was difficult at first since I was starting from scratch. I spent a lot of time reading documentation and asking questions to understand every element/detail and also most of my time debugging the huge amount of errors I received each time I launched tests on my algorithms; but it was worth it (now I am way more comfortable with them). I am a little bit disapointed that the new model does not significantly improve performances of the original one but I stay optimistic considering all the possibilities it offers.

My work left aside, I want to expand on the atmosphere in the laboratory. My work colleagues were very kind and inclusive. Indeed there were a lot of events for trainees to discover what it is like to work in a research environment : I assisted to many conferences about new technologies and there were weekly meetings to discuss our advances and learn more about work of other people from the team. Outside the laboratory, I was invited many times to play Volley-Ball or to play cards in a bar with members of the laboratory. In a word there was a very pleasant and stimulating atmosphere. I really enjoyed this internship and thank one more time every one I met during this experience that have brought me a lot.

Chapter 5

Conclusion

To conclude, this internship was a great insight in the field of Artificial Intelligence and the research environment.

On the one side, we succeed to construct a new version of ScanNet that is able to deal with motion vectors as extra geometrical information. We also managed to make this new implementation run and train on the same dataset as ScanNet. Even if results are not very convicing, there are still many uncharted strategies to discuss and explore aiming at building a more efficient version of ScanNet.

On the other side, this internship helped me improve many skills I will need as a future engineer : my oral fluency (trough technical discussion and meetings), my project managing skills (by settin deadlines and scheduling meetings) and my integrity (I encountered people from different background and different culture).

I am now confident about my choice of career and will continue to explore my opportunities in the field of Artificial Intelligence.

Thanks for reading,
Antonin Point

Bibliography

Grudinin, Alexandre Hoffmann Sergei (2017). “NOLB: Nonlinear Rigid Block Normal-Mode Analysis Method”. In: *J. Chem. Theory Comput.* URL: https://doi.org/10.1021/acs.jctc.7b00197.

Jane S. Richardson David C. Richardson, David S. Goodsell (2021). “Seeing the PDB”. In: *Journal of Biological Chemistry* Volume 296. URL: https://doi.org/10.1016/j.jbc.2021.100742.

Jérôme Tubiana, Dina Schneidman-Duhovny and Haim J. Wolfson (2022). “ScanNet: an interpretable geometric deep learning model for structure-based protein binding site prediction”. In: *Nat Methods* 19, pp. 730–739. URL: https://doi.org/10.1038/s41592-022-01490-7.

Keyulu Xu Weihua Hu, Jure Leskovec and Stefanie Jegelka (2019). “How Powerful are Graph Neural Networks?” In: *ICLR*. URL: https://arxiv.org/abs/1810.00826.

Lim, Derek et al. (2022). “Sign and Basis Invariant Networks for Spectral Graph Representation Learning”. In: *ICLR*. URL: https://openreview.net/forum?id=BlM64by6gc.

Reilly, Jon (2022). *Precision vs Recall: How to Use Precision and Recall in Machine Learning - Complete Guide*. URL: https://www.akkio.com/post/precision-vs-recall-how-to-use-precision-and-recall-in-machine-learning-complete-guide.

Schumacher, Devin (2023). *Laplacian Positional Encodings*. URL: https://serp.ai/laplacian-positional-encodings/.

Sillitoe, Ian et al. (2021). “CATH: increased structural coverage of functional space”. In: *Nucleic Acids Research, Volume 49, Issue D1*. URL: https://doi.org/10.1093/nar/gkaa1079.

Steen, Doug (2020). *Precision-Recall Curves*. URL: https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248.

Swarnimrai (2021). *Multi-Layer Perceptron Learning in Tensorflow*. URL: https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow.

White, Andrew D. (2022). *Attention Layers*. URL: https://dmol.pub/dl/attention.html.