Advanced Learning for Text and Graph Data

Lab session 7: Learning on Sets / Learning with Proteins

Lecture: Prof. Michalis Vazirgiannis
Lab: Giannis Nikolentzos, Johannes Lutzeyer

Wednesday, December 14, 2022

This handout includes theoretical introductions, coding tasks and questions. Before the deadline, you should submit on moodle **or** here a **.zip** file named `Lab<x>_lastname_firstname.pdf` containing a `/code/` folder (itself containing your scripts with the gaps filled) and an answer sheet, following the template available here, and containing your answers to the questions. Your answers should be well constructed and well justified. They should not repeat the question or generalities in the handout. When relevant, you are welcome to include figures, equations and tables derived from your own computations, theoretical proofs or qualitative explanations. **One submission is required for each student. The deadline for this lab is December 21, 2022 11:59 PM**. No extension will be granted. Late policy is as follows: $]0, 24]$ hours late $\rightarrow$ -5 pts; $]24, 48]$ hours late $\rightarrow$ -10 pts; > 48 hours late $\rightarrow$ not graded (zero).

---

1 Learning objective

The goal of this lab is to introduce you to machine learning models for data represented as sets. Furthermore, you will deal with a problem that arises in the field of bio-informatics and which can be solved using machine learning techniques. Specifically, in the first part of the lab, we will implement the DeepSets model. We will evaluate the model in the task of computing the sum of sets of digits and compare it against traditional models such as LSTMs. In the second part of the lab, you will learn how proteins can be represented as graphs, and you will implement a graph neural networks (GNN) to deal with the antibiotic resistance classification task. 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/`).

2 DeepSets

Typical machine learning algorithms, such as the Logistic Regression classifier or Multi-layer Perceptrons, are designed for fixed dimensional data samples. Thus, these models cannot handle input data that takes the form of sets. The cardinalities of the sets are not fixed, but they are allowed to vary. Therefore, some sets are potentially larger in terms of the number of elements than others. Furthermore, a model designed for data represented as sets needs to be invariant to the permutations of the elements of the input sets. Formally, it is well-known that a function f transforms its domain $\mathcal{X}$ into its range $\mathcal{Y}$. If the input is a set $X = \{x_1, \ldots, x_M\}$, $x_m \in \mathfrak{X}$, i.e., the input domain is the power set $\mathcal{X} = 2^{\mathfrak{X}}$, and a function $f : 2^{\mathfrak{X}} \to Y$ acting on sets must be permutation invariant to the order of objects in the set, i.e., for any permutation $\pi : f(\pi\{x_1, \dots, x_M\}) = f(\{x_1, \dots, x_M\})$. Learning on sets emerges in several real-world applications, and has attracted considerable attention in the past years.

2.1 Dataset Generation

For the purposes of this lab, we consider the task of finding the sum of a given set of digits, and we will create a synthetic dataset as follows: Each sample is a set of digits and its target is the sum of its elements. For instance, the target of the sample $X_i = \{8, 3, 5, 1\}$ is $y_i = 17$. We will generate $100,000$ training samples by randomly sampling between 1 and 10 digits ($1 \leq M \leq 10$) from $\{1, 2, \dots, 10\}$. With regards to the test set, we will generate $200,000$ test samples of cardinalities from 5 to 100 containing again digits from $\{1, 2, \dots, 10\}$. Specifically, we will create $10,000$ samples with cardinalities exactly 5, $10,000$ samples with cardinalities exactly 10, and so on.

Task 1
Fill in the body of the `create_train_dataset()` function in the `utils.py` file to generate the training set (consisting of $100,000$ samples) as discussed above. Each set contains between 1 and 10 digits where each digit is drawn from $\{1, 2, \dots, 10\}$. To train the models, it is necessary that all training samples have identical cardinalities. Therefore, we pad sets with cardinalities smaller than 10 with zeros. For instance, the set $\{4, 5, 1, 7\}$ is represented as $\{0, 0, 0, 0, 0, 0, 4, 5, 1, 7\}$ (Hint: use the `randint()` function of NumPy to generate random integers from $\{1, 2, \dots, 10\}$).

Task 2
Fill in the body of the `create_test_dataset()` function in the `utils.py` file to generate the test set (consisting of $200,000$ samples) as discussed above. Each set contains from 5 to 100 digits again drawn from $\{1, 2, \dots, 10\}$. Specifically, the first $10,000$ samples will consist of exactly 5 digits, the next $10,000$ samples will consist of exactly 10 digits, and so on.

2.2 Implementation of DeepSets

It can be shown that if $\mathfrak{X}$ is a countable set and $\mathcal{Y} = \mathbb{R}$, then a function $f(X)$ operating on a set X having elements from $\mathfrak{X}$ is a valid set function, i.e., invariant to the permutation of instances in X, if and only if it can be decomposed in the form $\rho(\sum_{x \in X} \phi(x))$, for suitable transformations ϕ and ρ.

DeepSets achieves permutation invariance by replacing ϕ and ρ with multi-layer perceptrons (universal approximators). Specifically, DeepSets consists of the following two steps:

- Each element x_i of each set is transformed (possibly by several layers) into some representation $\phi(x_i)$.
- The representations $\phi(x_i)$ are added up and the output is processed using the ρ network in the same manner as in any deep network (e.g., fully connected layers, nonlinearities, etc.).

An illustration of the DeepSets model is given in Figure 1.

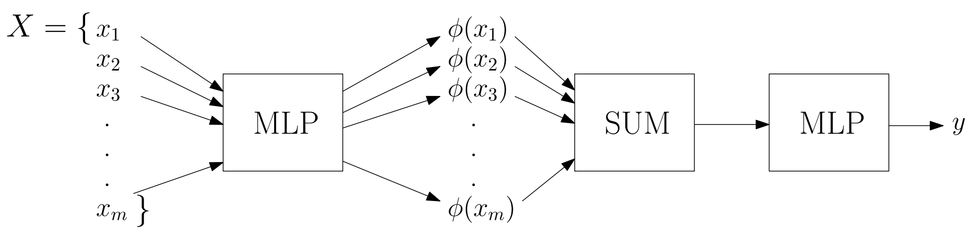


Figure 1: The DeepSets model.

Task 3
Implement the DeepSets architecture in the `models.py` file. More specifically, add the following layers:
– an embedding layer which projects each digit to a latent space of dimensionality h_1
– a fully-connected layer with h_2 hidden units followed by a tanh activation function
– a sum aggregator which computes the sum of the elements of each set
– a fully-connected layer with 1 unit since the output of the model needs to be a scalar (i.e., the prediction of the sum of the digits contained in the set)

Question 1 (5 points)
For the task that we perform here, i.e., determining the sum of a set of integers, what are the optimal MLP parameters (weights and bias) that we expect the DeepSets architecture learn?

We have now defined the DeepSets model. We will compare the DeepSets model against an LSTM, an instance of the family of recurrent neural networks. The next step is thus to define the LSTM model. Given an input set, we will use the LSTM hidden state output for the last time step as the representation of entire set.

Task 4
Implement the LSTM architecture in the `models.py` file. More specifically, add the following layers:
– an embedding layer which projects each digit to a latent space of dimensionality h_1
– an LSTM layer with h_2 hidden units
– a fully-connected layer which takes the LSTM hidden state output for the last time step as input and outputs a scalar

2.3 Model Training

Next, we will train the two models (i.e., DeepSets and LSTM) on the dataset that we have constructed. We will store the parameters of the trained models in the disk and retrieve them later on to make predictions.

Task 5
Fill in the missing code in the `train.py` file, and then execute the script to train the two models. Specifically, you need to generate all the necessary tensors for each batch.

Question 2 (5 points)
Let X_1 and X_2 denote two sets that consist of 2 vectors and which are defined as follows:

$$X_1 = \left\{[1.2,\ -0.7]^\top, [-0.8,\ 0.5]^\top\right\}$$
$$X_2 = \left\{[0.2,\ -0.3]^\top, [0.2,\ 0.1]^\top\right\}$$

Clearly, the two sets are not identical to each other. Show that there exists a DeepSets model that can embed the two sets into different vectors, i.e., there exist weight matrices and bias vectors such that the two sets are mapped to different vectors.

2.4 Predicting the Sum of a Set of Digits

We will now evaluate the two models on the test set that we have generated. We will compute the accuracy and mean absolute error of the two models on each subset of the test set separately. We will store the obtained accuracies and mean absolute errors in a dictionary.

Task 6
In the `eval.py` file, for each batch of the test set, generate the necessary tensors for making predictions. Compute the output of two models. Compute the accuracy and mean absolute error achieved by the two models and append the emerging values to the corresponding lists of the `results` dictionary (Hint: use the `accuracy_score()` and `mean_absolute_error()` functions of scikit-learn).

We will next compare the performance of DeepSets against that of the LSTM. Specifically, we will visualize the accuracies of the two models with respect to the maximum cardinality of the input sets.

Task 7
Visualize the accuracies of the two models with respect to the maximum cardinality of the input sets (Hint: you can use the `plot()` function of Matplotlib).

Question 3 (5 points)
The elements of a set can be mathematical objects of any kind. Suppose you are given a classification dataset where each sample is a set of graphs, e.g., $X = \{G_1, \dots, G_m\}$ where G_i is a graph. Describe an end-to-end learning approach you could employ to classify those sets.

3 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

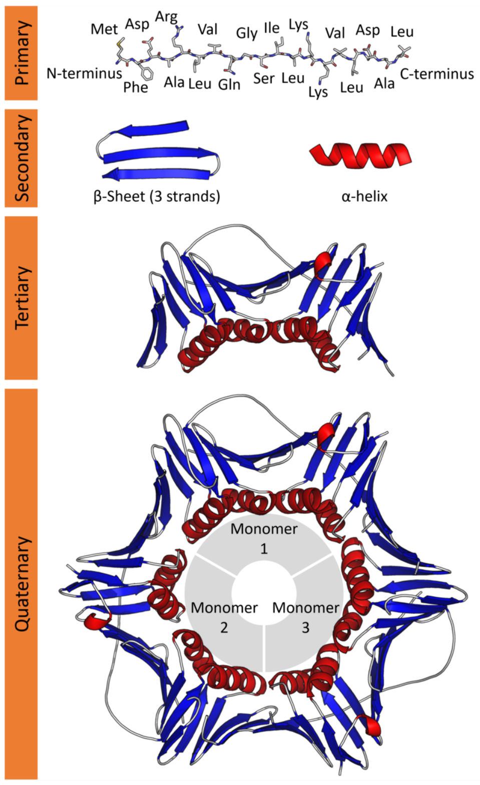


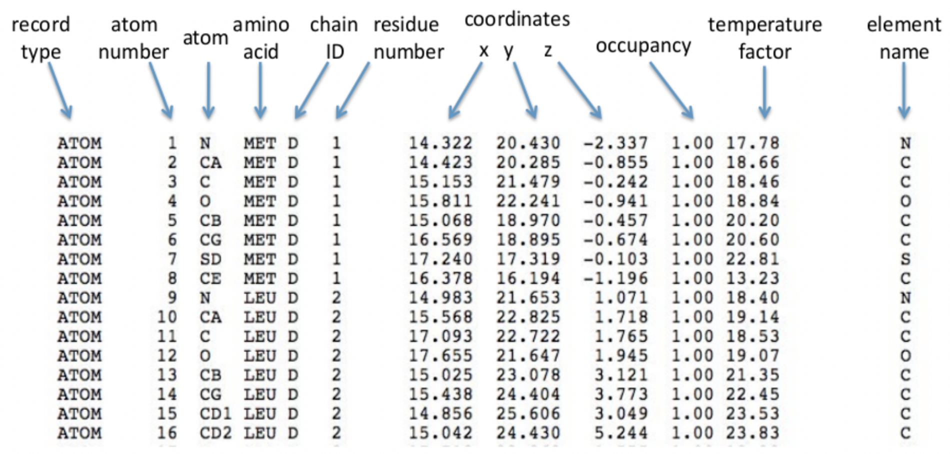
Figure 2: Protein structure (from Wikipedia).

levels of protein structure are illustrated in Figure 2. 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 3 shows an example of a PDB file.

3.1 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:



```
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 3: PDB File description.

(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.

Task 8
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 9
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).

3.2 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 10
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 11
Execute the `arg_classification.py` script to train and evaluate the model.

Question 4 (5 points)
Given that proteins arise as sequences of amin oacids, comment on the appropriateness of the permutation equivariant property of our message passing layers and permutation-invariance of the readout function in our proposed architecture. (Note that permutation-invariance was defined in the introduction of Section 2 and that a permutation-equivariant function f acting on sets satisfies $f(\pi(x_1, \ldots, x_M)) = \pi f(x_1, \ldots, x_M)$ for any permutation of sets π). Please discuss which modifications could be made in our model to take the protein sequence ordering into account in our 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.

[3] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3744–3753, 2019.

[4] Konstantinos Skianis, Giannis Nikolentzos, Stratis Limnios, and Michalis Vazirgiannis. Rep the Set: Neural Networks for Learning Set Representations. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, pages 1410–1420, 2020.

[5] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.