DNA sequencing based on Graphical Neural Networks

January 2020

Contents

List of Figures

Introduction

Chapter 1

An Overview Of The DNA Sequencing Problem And GNN

1.1 The DNA Sequencing Problem

The aim of the DNA sequencing problem is the reconstruction of the ordered DNA based on some observed fragments. Thus, the idea is to determine the order of the four bases: A adenine, G guanine, C cytosine, and T thymine. It was and it's still a challenging combinatorial and optimization problem. In the literature it was modeled as a salesman problem. And a variety of heursitics and metaheuristics were applied to it. The knowledge of the DNA sequencing problem have been applied to numerous areas and fields such as Medical Diagnosis, Virology, Biotechnology, Evolutionary Biology, etc.

The first DNA sequence was obtained in the late 1970's. And by using a DNA sequencer based on fluorescence-based sequencing, the applications of DNA sequencing became everywhere.

It is hard to overstate the importance of DNA sequencing to biological research; at the most fundamental level it is how we measure one of the major properties by which terrestrial life forms can be defined and differentiated from each other. Therefore over the last half century many researchers from around the globe have invested a great deal of time and resources to developing and improving the technologies that underpin DNA sequencing.

The development of sequencing methods, both in the chemical layer and computational layer, have contributed to minimize the cost of DNA sequencing and to its democratization in the scientific community. The figure below by the NHGRI (National Human Genome Research Institute) illustrates the evolution of the cost of DNA sequencing over time.

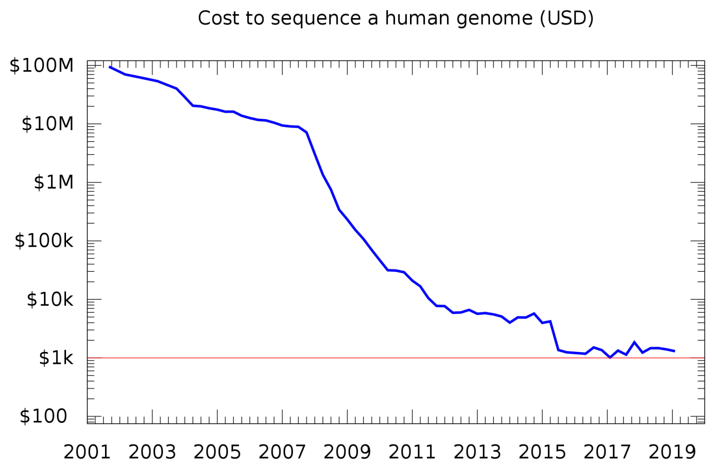


Figure 1.1: Cost to sequence a human genome (USD)

1.2 Graphical Neural Networks

Graphical Neural Networks are a combination of Graph Theory and Neural Networks. These neural networks provide an easy way to do lodge-level, edge-level, and graph-level prediction tasks. The most fundamental part of GNN is a Graph. In computer science, a graph is a data structure (a couple of two sets) of two main components: nodes and edges. So a graph G can be defined mathematically as $G = (V, E)$, where V is the set of nodes (or vertices), and E is the set of edges. If the set E is constituted of sets of two nodes than the the links aren't oriented and the graph is said to be undirected. Otherwise if they are couple of nodes, than it is a directed graph. A graph can represent things like social media networks, or molecules. Think of nodes as users, and edges as connections. A social media graph might look like this:

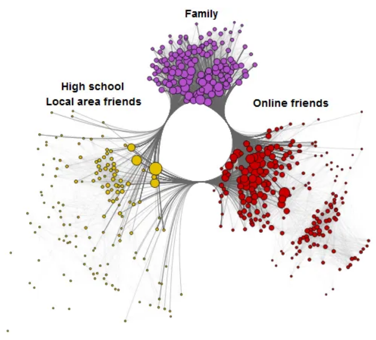


Figure 1.2: Social Media Graph

A graph is often represented by A, an adjacency matrix. If a graph has n nodes, A has a dimension of (n $\times$ n). Sometimes the nodes have a set of features (for example, a user profile). If the node has f numbers of features, then the node feature matrix X has a dimension of (n $\times$ f).

In graphical neural networks, we implement the concept of Node Embedding. It means mapping nodes to a d- dimensional embedding space (low dimensional space rather than the actual dimension of the graph), so that similar nodes in the graph are embedded close to each other.

Let's define u and v as two nodes in a graph. x_u and x_v are two feature vectors. Now we'll define the encoder function $Enc(u)$ and $Enc(v)$, which convert the feature vectors to z_u and z_v. The similarity function could be Euclidean distance.

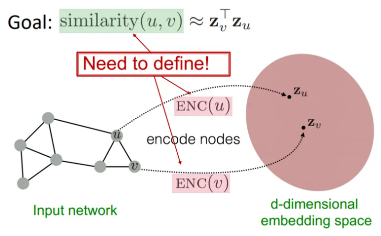


Figure 1.3: Encoding in a Graph

The encoder function will be able to perform locality, aggregate information, and stacking multiple layers.

1.3 The DNA Sequencing Problem As A Graphical Problem

We will now model the DNA sequencing problem as a link prediction problem in a graph. We will have the vertices as small sequences of DNA of a small fixed size. And the vertices will be the connections. So the aim isn't to have a full ordering of the DNA but to see if two small sequences come in a consecutive order. For instance the sequence 'ATTA' may occur 10 times in a sequence of 1000 length of the DNA and it will be connected to 20 other sequences. so the graph will be have 'ATTA' as a vertice and it will be connected to 20 other nodes.

Let's take a small simple DNA sequence of length 14. And we will have a shift of two when reading small fragments of length 2. The DNA sequence is 'ATAGGCAGGGACAA'. The associated graph is:

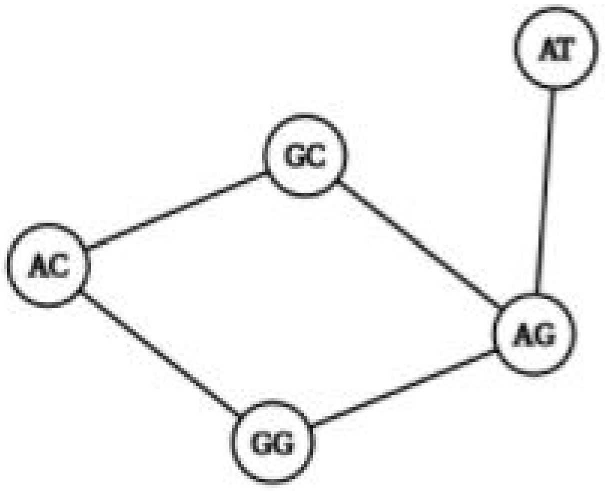


Figure 1.4: From Sequence to a Graph

This is the model of the problem. The inference will help us to just estimates the existence of a link between two given nodes. And, some information is lost in order to have the exact ordering of the DNA which can be added in extension to this project by using directed graphs.

Chapter 2

GraphSAGE

In this project, we will use GraphSAGE as our Graph Algorithm. The goal of GraphSAGE is to learn a representation for every node based on some combination of its neighbouring nodes, parametrized by h.

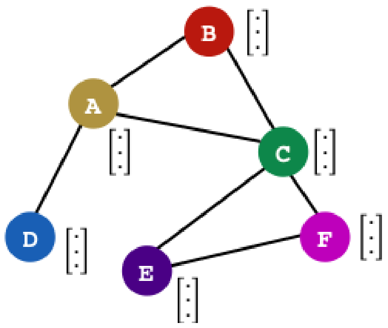


Figure 2.1: Node Features

Recall, every node can have their own feature vector which is parameterized by X. Let's assume for now that all the feature vectors for every node are of the same size. One layer of GraphSAGE can be run for k iterations — therefore, there is a node representation h for every node, at every k iteration. Observe the following notation:

- X_V = Node features of a random node V
- h_V^0 = Initial node embedding representation for a random node V
- h_V^k =Node embedding representation for a random node V at k^{th} iteration
- z_V = Final node embedding representation for a random node V after a round of GraphSAGE

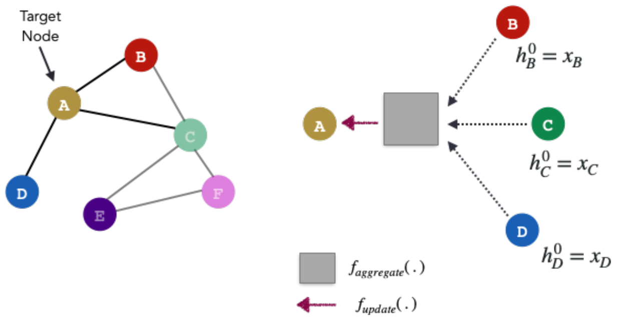


Figure 2.2: First Iteration of GraphSAGE

The GraphSAGE algorithm follows a two step process. Since it is iterative, there is an initialization step that sets all the initial node embedding vectors to their feature vectors. (k would start iterating from 1...K)

$h_V^{k-1} = h_V^0 = x_v$

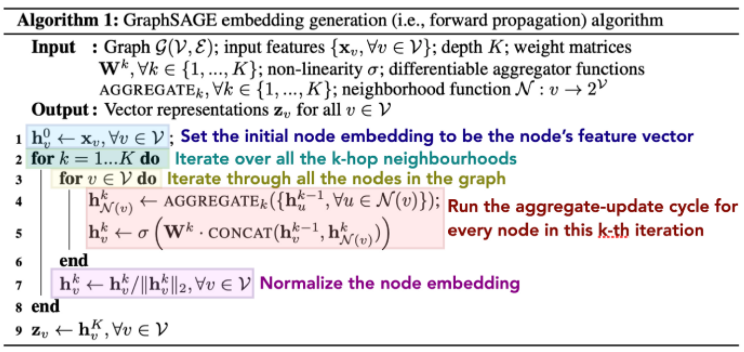
The first step is to aggregate neighbouring node representations for our target node. The f aggregate function is a placeholder for any differentiable function. This could be as simple as an averaging function or as complex as a neural network.

$a_v = f_{aggregate}(\mathrm{h}_u \mid \mathrm{u} \in N(v))$

After obtaining an aggregated representation for node v based on its neighbours, update the current node v using a combination of its previous representation and the aggregated representation. The f update function is a placeholder for any differentiable function which, can once again, be as simple as an averaging function, or as complex as a neural network.

$h_V^k = f_{update}(\mathrm{a}_v, h_V^{k-1})$

This is the algorithm as written in the original paper.



Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \to 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

```
$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$; Set the initial node embedding to be the node's feature vector
for $k = 1...K$ do   Iterate over all the k-hop neighbourhoods
    for $v \in \mathcal{V}$ do   Iterate through all the nodes in the graph
        $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;   Run the aggregate-update cycle for
        $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$   every node in this k-th iteration
    end
    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$   Normalize the node embedding
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

Figure 2.3: The algorithm of GraphSAGE

Chapter 3

Implementation and Results

3.1 Importing the data and modeling the graph

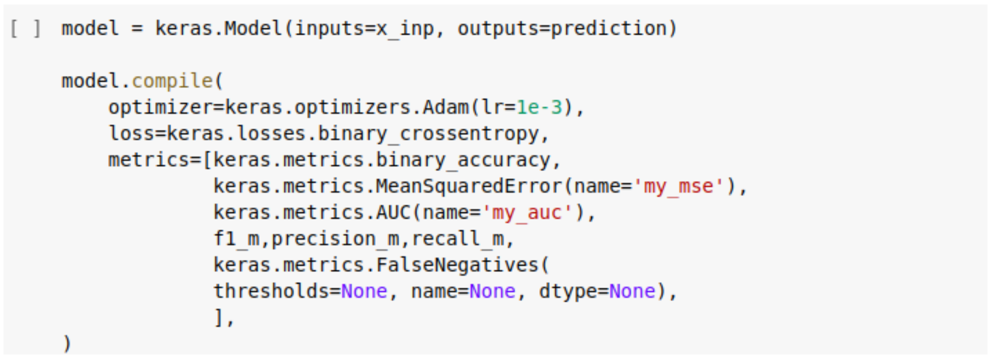
The algorithm will have as input a graph associated to given data, i.e a sequence of DNA. The length of the small fragments was chosen to be 4. And the shifting while reading the string of DNA was chosen to be 1 to generate more sequences.

3.2 Test, train, and validation sets

The train data will have the same number of nodes with 0.6 the number of links. The information will be learned from the existing links as well as a vector associated to each node. The feature associated to each node is chosen to be the occurrence of each letter in the fragment. Thus, in this case it is an integer between 0 and 5.

3.3 Defining and training the model

Now, we define the neural network model that is going to be trained on our graph.



```
[ ] model = keras.Model(inputs=x_inp, outputs=prediction)

    model.compile(
        optimizer=keras.optimizers.Adam(lr=1e-3),
        loss=keras.losses.binary_crossentropy,
        metrics=[keras.metrics.binary_accuracy,
                 keras.metrics.MeanSquaredError(name='my_mse'),
                 keras.metrics.AUC(name='my_auc'),
                 f1_m,precision_m,recall_m,
                 keras.metrics.FalseNegatives(
                 thresholds=None, name=None, dtype=None),
                 ],
    )
```

Figure 3.1: The model

3.4 Approaches

What makes difference between the following approaches is the node features.

- **Auto Encoder with Occurrence**: In this approach, we will use an Auto Encoder to reduce the size of the Input. And the feature of each node in the Graph will be a 4-length vector containing the occurrence of each character 'A','T','C','G' in the sequence.
  So our initial feature will be the vector containing the Occurrence of each character, then this vector will be the input of an Auto Encoder to reduce its size from 4 to 3.
- **Auto Encoder with ASCII Code**: In this approach, we will use an Auto Encoder to reduce the size of the Input. And the feature of each node in the Graph will be a 4-length vector containing the ASCII Code of each character 'A','T','C','G'.
  So our initial feature will be the vector containing the ASCII Code, then this vector will be the input of an Auto Encoder to reduce its size from 4 to 3.
- **Not Auto Encoder with Occurrence**: The feature of each node in the Graph will be a 4-length vector containing the occurrence of each character 'A','T','C','G' in the sequence.
- **Not Auto Encoder with ASCII Code**: The feature of each node in the Graph will be a 4-length vector containing the ASCII Code of each character 'A','T','C','G'.

3.5 Results

Auto Encoder with Occurence			
Metrics	Train	Validation	Test
Accuracy	0.69	0.68	0.66
F1$_score$	0.73	0.45	0.46
Recall	0.84	0.42	0.43
Precision	0.65	0.48	0.49

Auto Encoder with ASCII Code			
Metrics	Train	Validation	Test
Accuracy	0.51	0.51	0.53
F1$_score$	0.62	0.46	0.48
Recall	0.82	0.45	0.47
Precision	0.51	0.48	0.49

Not Auto Encoder with Occurence			
Metrics	Train	Validation	Test
Accuracy	0.78	0.69	0.73
F1$_score$	0.80	0.47	0.48
Recall	0.94	0.46	0.46
Precision	0.70	0.47	0.47

Not Auto Encoder with ASCII Code			
Metrics	Train	Validation	Test
Accuracy	0.50	0.5G	0.5
F1$_score$	0.65	0.47	0.5
Recall	0.49	0.5	0.5
Precision	0.99	0.45	0.5

By analyzing these results, we see that our model performs better without an Auto Encoder and it's totally normal because there is no remarkable reduction in the dimension of the input (from 4 to 3).
Maybe we can see the importance of an Auto Encoder when we use sequences with 10 or more

as length. But to train a model on a thousands of 10 length sequences requires a highly powerful Machine.

3.6 Comparison between different species

Now, it's the time to apply our trained model on three species : Human, Bacteria:Ferroglobus placidus and Chimpanzee. As we get the best results with the **Not Auto Encoder with Occurence** approach, we will use it for the comparison.

Metrics	Human	Bacteria:Ferroglobus placidus	Chimpanzee
Accuracy	0.78	0.73	0.71
$F1_score$	0.80	0.77	0.76
Recall	0.94	0.94	0.91
Precision	0.70	0.67	0.66

We see by applying the trained model to a DNA sequencing of a bacteria (so far genetically from humans) and to a chimpanzee (so close genetically), we see that the performance of the algorithm doesn't change. Thus, it is learning chemical properties and not able to extract biological distinctive features of different species.

Conclusion

In brief, Graphical Neural Networks are a promising field of research with important applications in Biology and Social Sciences. The small application the we presented here seems to be performant when it comes to predicting links between small sequences, when having the accuracy as an encoded feature. But, it is just learning chemical features and the performance doesn't vary if trained on a DNA of some species and tested on others.