UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 3028

Using Graph Neural Networks to Separate Haplotypes in Assembly Graphs

Filip Wolf

Zagreb, May 2022

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 3028

Using Graph Neural Networks to Separate Haplotypes in Assembly Graphs

Filip Wolf

Zagreb, May 2022

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 3028

Pristupnik:	**Filip Wolf (0036510053)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Mile Šikić

Zadatak: **Korištenje graf neuronskih mreža za odvajanje haplotipa u grafovima sastavljanja**

Opis zadatka:

Cilj diploidnog de novo sastavljanja genoma jest ne samo rekonstruirati genomsku sekvencu pojedinca, već i odvojiti dva haplotipa, po jedan naslijeđen od svakog roditelja. Čak i nakon godina istraživanja i brojnih pokušaja, pouzdan alat za ovu vrstu problema nije konstruiran. Međutim, s najnovijom HiFi tehnologijom sekvenciranja, jedan smo korak bliže rješenju. U ovom projektu, prvi korak je korištenje asemblera Raven za konstruiranje grafova sastavljanja iz diploidnih podataka, u kojima čvorovi predstavljaju sekvence koje pripadaju pojedinim haplotipovima, a bridovi predstavljaju preklapanju među tim očitanjima. Idući korak jest konstrukcija modela dubokog učenja za predikciju bridova koji povezuju čvorove iz različitih haplotipova. Micanje tih čvorova iz grafa sastavljanja bi problem diploidnog sastavljanja pojednostavilo na dva zasebna problema haploidnog sastavljanja genoma. Učenje treba biti napravljeno na sintetičkom skupu podataka simuliranom iz genoma bakterija i kvasaca. Evaluacija treba biti napravljena na stvarnim očitanjima bakterija i kvasaca sekvenciranih PacBio HiFi tehnologijom. Rješenje treba biti implementirano u Pythonu koristeći Pytorch ili sličnu biblioteku za duboko učenje. Kod treba biti dokumentirati koristeći komentare i razvijati prema Google Python Style Guide ako je moguće. Cijeli programski proizvod potrebno je postaviti na GitHub pod jednu od OSI odabranih licenci.

Rok za predaju rada: 27. lipnja 2022.

CONTENTS

1. Introduction

Traditionally, the focus of *de novo* genome assembly has always been on the reconstruction of an individual's genomes. We will here however focus on a different application of *de novo* genome assembly: haplotype separation. Every individuals genome is composed of both a mother's and a father's genome, each contributing about half of genetic material on average. This is called a haplotype. By separating these two haplotypes, we can distinguish which genes came from what parent, which has a wide range of applications, from ancestry tests to finding hereditary diseases.

Bioinformatics has long been dominated by algorithms that employ complex heuristics and expert knowledge to find solutions to the problems it faces. This is however slowly changing. More and more research is being done using *deep learning* to solve it's problems. First, it was employed only to find dense representations of the features of genomes, but it later started to completely replace the previously mentioned algorithms. Deep learning has contributed tremendously to the field in recent years and shows no signs of stopping, the most notable achievement being the solution to the protein folding problem which previously wasn't solved for 50 years (Jumper (2021)). Still, there is still a long way to go before it becomes completely standard within the field. Thus, this Thesis is concerned with applying recent deep learning techniques in order to solve the problem of separating the two haplotypes in an existing genome.

1.1. Deep Learning

1.1.1. Basics

In the last decade, deep learning has grown from a niche research area to one of the largest fields withing computer science. It is now actively employed in virtually every human endeavor, from medicine to astronomy. And it is still growing day by day on its mission to become the standard way of handling almost all data.

In deep learning, we use data processing structures called *artificial neural networks* (ANNs) to extract useful information from our data and learn to predict an outcome, such as some feature of the data or a target class. It does this by adjusting learnable *weights* defined for every neuron in our network. Due to these weights, neural networks are much denser structures when compared to previous machine learning methods and can be referred to as *universal function approximators* (Hornik et al. (1989)) due to their ability to, with large enough networks, approximate any function. This gives them unprecedented performance on previously unsolvable tasks, but due to their abstract structure, makes them somewhat difficult to interpret. Some networks, like *convolutional neural networks* (Lecun et al. (1998)), don't suffer from this problem as much and can produce some quite intuitive visualizations. On the other hand, some networks, like the ones we will use here, cannot be visually meaningfully interpreted.

To successfully explain how ANNs work, we need to introduce two concepts: a loss function and backpropagation. When use our deep learning model to learn from data, we pass it through our network and compare the output to a previously defined true value by using a *loss function*. A loss function abstracts the error of our network prediction to a single number which is then used to calculate gradients in respect to our data. These gradients are then propagated back through the network using *backpropagation*. In essence, the backpropagation algorithm tells every weight in our network how to change in order to better predict our data. If we imagine our data as a 2-dimensional function on a plane, a gradient is the information of how steeps the function is at any specified point. This steepness value tells the network weights how much they should change, while its sign specifies the direction of change. By correctly propagating these gradient values back through the network, which backpropagation does, we an successfully make our network learn from data.

1.1.2. Graph Neural Networks

While standard ANNs are great at predicting simple data with no underlying structure (or at least one that isn't known), to successfully make our network learn from assembly graphs, we will need something more refined. Yes, it is true that we can simply represent our graph in the form of a 1-dimensional vector, but we then lose precious structural information about the contig overlaps. By using networks more tailor-made for data representation on graphs, we can take advantage of the graph's underlying structure. The networks in question are called *Graph Neural Networks* (GNNs) (Scarselli et al. (2004)). Most modern graph neural networks work on the principle of *message passing* (Gilmer et al. (2017)). A node accumulates information from adjacent nodes and the edges connecting them and uses it to update its own weights. By repeating this process enough times, we can converge to a stable solution. This can be represented wit the following equation.

Let $x_v \in \mathbf{R}^{d_1}$ be the feature for node v, and $w \in \mathbf{R}^{d_2}$ be the feature for edge (u, v). The message passing paradigm defines the following node-wise and edge-wise computation at step $t + 1$:

Edge-wise: $m_e^{(t+1)} = \phi(x_v^{(t)}, x_u^{(t)}, w_e^{(t)}), (u, v, e) \in \mathcal{E}$.

Node-wise: $x_v^{(t+1)} = \psi(x_v^{(t)}, \rho(\{m_e^{(t+1)} : (u, v, e) \in \mathcal{E}\}))$.

In the above equations, ϕ is a message function defined on each edge to generate a message by combining the edge feature with the features of its incident nodes; ψ is an update function defined on each node to update the node feature by aggregating its incoming messages using the reduce function ρ.

The GNN can be though of as an extension of CNNs. A CNN takes an image's local neighborhood and extracts information from it. It does this using convolution *filters*, which take a certain amount of pixels in a neighborhood and multiply them with weights. Now, the size of this filter is predefined and cannot be changed. For instance, it can have a size of 3 x 3 or 5 x 5. If we were to create such a filter for use on graphs, we would simply designate the central weight of the filter to be the node we are currently looking at, and the surrounding weights would be its neighboring nodes. As we can see, this would limit us to graphs where nodes had a constant number of neighbors, or graphs where we could look only at a limited number of neighbors. GNNs do not have this limitation. The ψ function takes all nodes in a central node's neighborhood into account equally.

2. Conclusion

Bibliography

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/gilmer17a.html`.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL `https://www.sciencedirect.com/science/article/pii/0893608089900208`.

E. R. P. A. e. a. Jumper, J. Highly accurate protein structure prediction with alphafold. *Nature*, 2021. doi: 10.1038/s41586-021-03819-2. URL `https://doi.org/10.1038/s41586-021-03819-2`.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

F. Scarselli, A. C. Tsoi, M. Gori, and M. Hagenbuchner. Graphical-based learning environments for pattern recognition. In A. Fred, T. M. Caelli, R. P. W. Duin, A. C. Campilho, and D. de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 42–56, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-27868-9.

R

Abstract

Keywords:

Sažetak

Ključne riječi: