DNA Clustering

Shai Guendelman, Gal Matatyahou

March 2021

1 Introduction

In this work we explore the clustering problem in the DNA storage channel.

1.1 Background

DNA has a lot of potential for storage, as it can theoretically contain much more information than current technologies, by a few orders of magnitude. But it has a lot of technical issues - The process of synthesising DNA strands is very noisy, and is only able to produce DNA strands of a few hundreds symbols, and many copies of each. This requires us to split large chunks of data into many short strands. Those strands are not ordered as in standard storage system, but are all mixed together in a DNA pool. This makes the task more complex, because before we can use error correction algorithms to the strands, we first have to order them correctly. We also would like to take advantage of the fact that a single strand is not synthesized once, but many times, and use those multiple copies in our error correction algorithm. In order to use the noisy copies of the same original DNA strand, we first have to cluster them together, as the clusters are not given when reading. This is the **Clustering** problem in the DNA storage channel.

1.2 Special Requirements

Standard clustering algorithms are not suitable for DNA clustering for the following reasons:

1. **Efficiency** - reading a large file might require us to cluster billions of strands, with large number of clusters, with a reasonable amounts of time and compute. Current algorithms are not fast enough.

2. **Distance** - many clustering algorithms require a distance function between samples. It is not clear what is the best distance measure in this case, and the most reasonable one - the edit distance, is expensive to compute.

3. **Evaluation** - the quality of our clustering procedure should be related to efficient reconstruction of the stored data. Standard evaluation measures do not reflect that, even the one used by [8].

1.3 High Level Overview

We start by reviewing and implementing a related work [8] on this topic in 2. Then in 3 we define a more realistic error model for the DNA storage channel based on [5]. Then we describe and implement a data-driven approach to this problem in 4. We than present empirical results of our experiments in 5, and finish with a discussion of important notes and ideas in 6.

2 Clustering Billions of Reads [8]

DNA samples The main goal, as mentioned before, is to get extensive details about large amount of long strands. To sample those strands effectively we use 3 balanced probabilities (p/3) for the strand "noise" - insertion, deletion and substitution. Using noise model creates error strands used in the algorithm.

Accuracy To check the similarity of the strands clustering algorithm to the expected clustering of the original strands, we implanted the formula in the article. The accuracy tool is important for understanding how good our clustering was. The exact formula is shown in [8].

Distances Using two different distances makes the parameters choosing options more variant, that makes other clustering options. In the algorithm, the q-gram distance is bounded and the edit distance bound is optional.

1. **Q-gram distance:** For all q length binary strings, calculate the difference between appearances in two strands, then sum all together (absolute value). Choosing different q values clearly affect the q-gram distance, and affect the error strands difference from the original strand.

2. **Edit distance:** Edit distance is the minimum number of operations (insertion/deletion/substitution) required to reach the original strand from the error strand. The edit distance is optional because it is faster to check the q-gram distance, when the meanings of both distances are similar.

Main algorithm and parameters The algorithm in the [8] article is clustering the strands according to hash-partition function, which sets a representative to each cluster. Two cluster with the same representative that fits to distances bounds, are united to one shared cluster, and the process repeats itself. In the algorithm implementation we use w, l parameters to set the hash functions, q parameter for q-gram distance, $r, low, high$ bounding parameters, and $localsteps$

to repeat the algorithm several times. Choosing those parameters is very important part of the clustering, and we used some methods to choose them (see below).

3 Generative Models

Generative model is a model of how the original strands to be synthesized are selected, and how errors occur.

Simple Generative Model (SGM) In [8], the algorithms performance was evaluated under a simplified generative model for the DNA storage channel. Their model has 3 parameters - m, s, p. It starts by sampling uniformly a strand of length m, each strand s times. then p determines the probability of error occurring at each symbol in the sequence, $p/3$ for either substitution, deletion or insertion. Then we end up with a cluster of size s with $p - noisy$ copies of the original sampled strand.

Complex Generative Model (CGM) is inspired by [5]. In the paper, the process of synthesizing and sequencing (reading) DNA is described, and the errors that occur in each stage are characterized. We defined our CMG based on some of its findings.

1. Sample the original strands uniformly out of all the strands of a given length.
2. Substitutions, insertions, deletions might happen in the **synthesis** stage, each having a different probability of occurring (substitutions are more common).
3. Termination - some strands might be cut off during synthesis, this is the main reason why we use many short strands instead of one long strand in the first place. We give higher termination probabilities to strands with long runs (i.e. same symbol repeats many times in a row).
4. **Amplification** stage, where the synthesised DNA strands are amplified with several **PCR** rounds. PCR is a chemical process where most of the strands are duplicated (i.e. the number of strands grows exponentially with the number of PCR rounds). But the chance for duplication is not uniform for all the strands, for example, high CG rates in the strand (large number of G,C nucleotide) reduces the probability for duplication. We add this to our CGM.
5. **Storage Decay** some symbols decay over time, some are more likely then others. Decay might result in incorrect reading of the symbol.
6. **Sampling** - We don't sequence the entire DNA pool, but first we sample from it. This might result in clusters in different sizes, and maybe also some not present in the sample at all.

7. **Sequencing** process might result in some substitution error (and a very low probability for insertions and deletions), but errors are more common on the ends of the strands.

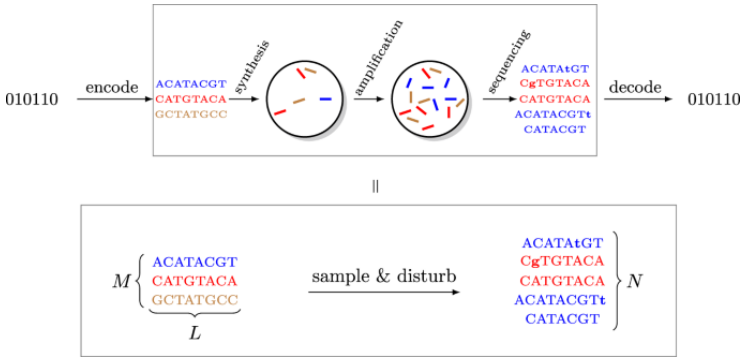

Figure 1: Complex generative model

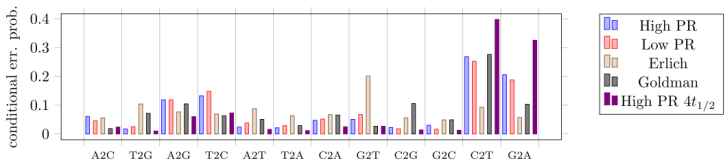

Figure 2: Substitution errors rate - we can see that some errors are far more likely then others. The A2G is the rate where A symbol is substituted for G

4 Deep Clustering

The idea we wanted to try is, if the errors are indeed systematic as suggested by [5], might we use a learning algorithm to take advantage of those patterns for improving the clustering accuracy? We choose to try Deep Clustering [1, 7], for a few reasons:

- **Quality** - It showed good result on different clustering tasks, on images, texts, audio and even biological data [6].
- **Efficiency** - A lot of work has gone to making neural networks more computationally efficient. Using a small network + optimization techniques like quantization [4], can result in a very efficient system.
- **Flexibility** - [5] suggests that the types of errors highly depend on the storage conditions and technology used. The system can be adjusted by re-training the network, instead of manual adjusting.

- **Data** - It is easy to get large amounts of labeled data in this case, since we can just synthesis some samples, and cluster them using a slow but accurate algorithm. Lack of labeled data is usually the bottleneck for deep learning.

Method Description Different methods for deep clustering exist, we choose this specific one for it's simplicity and compatibility with the problem. The method is based on a type of network called **Auto-Encoder**. We denote the learnable parameters of the model with θ. In this network has two parts - an *Encoder* E_θ, and a *Decoder* D_θ:

$$E_\theta : \mathbb{R}^n \to \mathbb{R}^m; D_\theta : \mathbb{R}^m \to \mathbb{R}^n \tag{1}$$

With $m < n$. The traditional way that this method is trained is with the reconstruction loss:

$$L_{rec}(x, \theta) = ||x - D_\theta(E_\theta(x))||_2^2 \tag{2}$$

it is called the "reconstruction loss" since the network learns to represent it's inputs of n dimensions with m dimensions, because $m < n$ it learns to compress the important features of it's inputs so it will be able to reconstruct it as close as possible to the original input. This loss can be improved during training only if the original data distribution $x \sim D_x$ has some underlying structure in it. Instead of clustering in the original n dimensional space (input space), we can perform the clustering on the lower dimensional encoding space, $\mathbb{R}^m$, i.e. the on the outputs of the *Encoder*, using the $L2$ norm as a distance measure. It was shown to be useful to include another loss function [7], the "K-Means loss", that forces the encoding of the sample x to be close to the encoding of it's cluster center c_x:

$$L_{kmean}(x, c_x, \theta) = ||E_\theta(x) - E_\theta(c_x)||_2^2 \tag{3}$$

Notice that using L_{kmean} alone is a bad idea, as it has a trivial minimum point by encoding all the samples to zero. We also include a third loss function that is commonly used as a regularization term to prevent over-fitting, that is just the $L2$ norm on the model parameters. To combine those loss functions we take a weighted sum of them:

$$L(x, c_x, \theta) = \lambda_1 \cdot L_{kmean}(x, c_x, \theta) + (1 - \lambda_1) \cdot L_{rec}(x, \theta) + \lambda_2 \cdot L_2(\theta) \tag{4}$$

where $\lambda_1, \lambda_2 \in [0, 1]$ are hyper-parameters of the model.

Technical details We choose to use a network with 2 Convolution layers both for the encoder and the decoder. Full details can be found in the code. The primary reasons for using convolution layers are: (1) they are more computationally efficient then recurrent layers and fully connected layers, (2) they were shown to be efficient for approximating edit distance [3], a task that is similar to ours. To cluster the outputs we use the simple *K-means* algorithm. We use 3 different datasets, (1) a large train set to optimize θ on, (2) a small dev

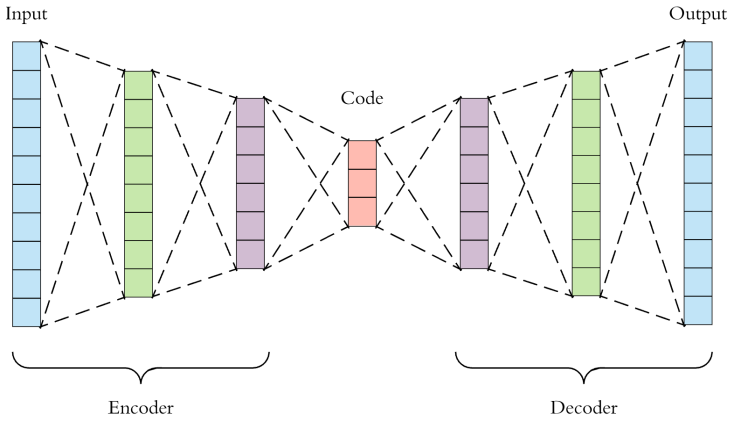

Figure 3: Auto-Encoder

(development) set to select the hyper-parameters of the model, (3) a small test set to evaluate the accuracy of the model. Those datasets are generated using CGM and SGM accordingly. The reason for using small datasets for dev and test is that K-Means does not scale well with large number of clusters, and is very slow. We tried other clustering algorithms available in scikit-learn package, all of them were very slow. In future work it might be possible to combine the algorithm from [8] for efficient clustering. The size of the test/dev sets is large enough to examine whether this method is useful for DNA clustering or not.

Pre-processing steps Convolution layers can operate on inputs of different lengths, but the dimensions of their outputs depends on the dimensions of the inputs. Because we want to use the L2 norm over the encoding space we need to have the same encoding space dimensions m, thus the same input space dimensions n. In order to do this we pad the shorter input sequences with a spacial pad symbol $\perp$, to make all the sequences of the same length. Before feeding the sequences to the model we first One-Hot-Encode them, i.e. each symbol is mapped to a 4 dimensional vector, with a single 1 entry and 0 in all the other.

$$A \mapsto (1,0,0,0); C \mapsto (0,1,0,0); G \mapsto (0,0,1,0); T \mapsto (0,0,0,1); \perp \mapsto (0,0,0,0)$$

5 Results

Algorithm results of [8] article algorithm - The results depends on the variant choosing options of the algorithm parameters. The algorithm produces different size clusters (both in SGM and CGM) that do not fit by size to the true expected clustering. There is similarity between most of the two clustering sets. The accuracy tool is not useful in this algorithm, because we used parameters in different scale than the expected scale in the article. The algorithm

used many random aspects, yet the result is still rational in our case. Editing all parameters affect mostly on the number of clusters and their size, can be viewed in code.

Deep Clustering Experiments We trained the network with different hyper-parameters, λ_1 this is the ratio between the reconstruction loss and the k-means loss, in the code it is named 'loss_lambda', and λ_2 is the coefficient on the regularization term for the L2 norm of the model's weights. In the code it is named 'weight_decay'. For each combination we trained a model on the train set, and evaluated it's performance on the dev set. The dev set accuracy's are described in tables 1, 2. Then the model with the best dev accuracy was used to cluster the test set. Those results are seen in table 3. We repeated this with the SGM and the CGM.

	$\lambda_2 = 0$	$\lambda_2 = 10^{-4}$
$\lambda_1 = 0$	0.075	0.015
$\lambda_1 = 0.1$	0	0.105
$\lambda_1 = 0.5$	0	0.03
$\lambda_1 = 0.9$	0.015	0.115
$\lambda_1 = 0.99$	0	0

Table 1: SGM best dev accuracy with different hyper-parameters

	$\lambda_2 = 0$	$\lambda_2 = 10^{-4}$
$\lambda_1 = 0$	0.195	0.245
$\lambda_1 = 0.1$	0.3	0.1
$\lambda_1 = 0.5$	0.02	0.155
$\lambda_1 = 0.9$	0.1	0.22
$\lambda_1 = 0.99$	0.055	0

Table 2: CGM best dev accuracy with different hyper-parameters

	SGM	CGM
Deep Clustering	0.11	0.3
Agg. Clustering [8]	0.0	0.0

Table 3: Test accuracy of our method

6 Discussion

Although we did not get good results using the deep clustering method, we note that there are many decisions that we made in the process that are not

	SGM	CGM	Original
Deep Clustering	0.6404	0.7107	0.5576
Agg. Clustering [8]	0.0258	0.0025	0.0093

Table 4: FMI score: FMI method

optimal, due to lack of time. We did show that when the errors are not uniformly distributed, a learning algorithm can use the underlying structure to perform better clustering. This can be beneficial in 2 different ways: (1) by incorporating a learning mechanism into the clustering algorithm, like that of [8], we might improve it's performance on real data, (2) by using *interpretability* methods [2] on the learned model, we might draw insights about systematic errors, and use those to design better algorithms and coding schemes for DNA.
The [8] article algorithm of clustering is hard to analyze. Some of the clusters include an expected cluster, some of the expected cluster include the algorithm cluster. The size is not uniform yet the numbers are close to expected clustering. The accuracy tool is based on cluster sizes, so it was not useful. We used other informative accuracy tool, the Fowlkes-Mallows index, that helped us understand how many pairs of points are in the same cluster. The conclusion is that the algorithm is not accurate, due to difficult parameter choosing and not enough DNA strands. The conclusion is that checking the accurate clustering is depends on other parameters, and on big numbers we might use it carefully.

References

[1] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, Maximilian Strobel, and Daniel Cremers. Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*, 2018.

[2] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M Rao, et al. Interpretability of deep learning models: a survey of results. In *2017 IEEE smartworld, ubiquitous intelligence & computing, advanced & trusted computed, scalable computing & communications, cloud & big data computing, Internet of people and smart city innovation (smartworld/SCALCOM/UIC/ATC/CBDcom/IOP/SCI)*, pages 1–6. IEEE, 2017.

[3] Xinyan Dai, Xiao Yan, Kaiwen Zhou, Yuxuan Wang, Han Yang, and James Cheng. Convolutional embedding for edit distance. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 599–608, 2020.

[4] Yunhui Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018.

[5] Reinhard Heckel, Gediminas Mikutis, and Robert N Grass. A characterization of the dna data storage channel. *Scientific reports*, 9(1):1–12, 2019.

[6] Md Rezaul Karim, Oya Beyan, Achille Zappa, Ivan G Costa, Dietrich Rebholz-Schuhmann, Michael Cochez, and Stefan Decker. Deep learning-based clustering approaches for bioinformatics. *Briefings in bioinformatics*, 22(1):393–415, 2021.

[7] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514, 2018.

[8] Cyrus Rashtchian, Konstantin Makarychev, Miklós Z Rácz, Siena Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. Clustering billions of reads for dna data storage. In *NIPS*, volume 2017, pages 3360–3371, 2017.