**Project: DNA storage simulation**

Team member: Longshen Ou

# 1 Scope of the project

In this project, a series of methods are adopted to build data synthesize systems to contaminate the clean DNA strands, to simulate the changes of DNA brought by the write and read operation of DNA-based storage systems. Specifically, a naïve rule-based method is first tested; then a series of neural network (NN)-based method are tested. In addition, a series of explorations are done to maximize the effectiveness of the NN-based method. Finally, some promising results are demonstrated: using the generated noisy strands from our system, a given trace reconstruction algorithm behaves very similarly as when giving the real noisy data as input.

# 2 Datasets

## 2.1 Selection of dataset

Among the four given datasets, the Microsoft Nanopore dataset is adopted during all the experiments. There are several reasons for this choice.

(1) Compared to Illumina data, the data obtained from Nanopore devices is more error-prone, where different types of error happens more frequently than Illumina data. Hence by using Nanopore data, the error curve of the trace reconstruction algorithm will be sharper, hence the differences of the algorithm performance can be more clearly distinguished when the characteristics of the input data is different.

(2) Compared to Illumina data, the given Nanopore dataset has quite even distribution of cluster size, which might reduce the chance of our system to biased towards strands that are in large clusters.

(3) The sizes of the Illumina datasets are huge because certain strands have very large read coverage as the result of PCR. In this case, many components when building the system, including the clustering, trace reconstruction, pairing cluster centroids with reference strands, and system training, will have huge amount of time overhead and is not acceptable.

(4) The strand length of Stanford Nanopore dataset varies greatly, and there is no document about dataset processing, so it’s hard to pair the noisy strands with the corresponding reference strands.

## 2.2 Split of dataset

The original Microsoft Nanopore dataset are split into three subsets for the system development. When splitting, it is ensured that different split of data does not contain noisy data from a same original DNA strand, to make the testing data be able to more accurately reflect the system’s generalization ability. Moreover, it is ensured that every clean strand in validation and test split has more than 5 noisy counterparts, to enable trace reconstruction algorithm to operate with at least coverage=5. Here are the details of the dataset after splitting:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Cluster num** | **Strand num** | **Cluster size avg:min:max** | **Clean strand num Train : Valid : Test** |
| MS Nanopore | 10,000 | About 270k | 26.97 : 0 : 164 | 7988 : 998 : 998 |

Table 1 Dataset details after splitting.

## 2.3 Data Preprocessing

Datasets are converted from txt file to a structured format (JSON) to facilitate further handling:

A picture containing text

Description automatically generated

Figure 1 Structured data format.

# 3 Other Preparation work

## 3.1 Tokenizer

To facilitate the algorithms to handle data, a simple tokenizer is constructed to convert nucleotides sequence between the format of a string of characters and a list of integer numbers. We define the vocabulary in the experiments as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Token** | <pad> | <bos> | <eos> | <unk> | A | T | C | G |
| **Id** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Table 2 Tokenizer dictionary.

Where the <pad> is padding token, <bos> is “begin of sequence”, <eos> is “end of sequence”, <unk> represent unknown characters.

During tokenizing, <bos> and <eos> are first inserted to the beginning and ending of a sequence. Then, all sequences in a same batch will be then padded with <pad> token to the same length before they are converted to integer lists in batch. Finally, characters are converted into integers according to the dictionary, while the unknown characters will be represented by <unk>.

In the reverse process, integers are converted to character sequences according to the dictionary, and then special tokens are removed and only nucleotides characters are left.

## 3.2 Performance measure

To measure the degree of similarity of the generated noisy strands with the real noisy strands from DNA-based storage system, trace reconstruction is performed on both generated and real noisy data and the error pattern of the reconstruction algorithm of different runs are compared. Generally, the more similar the algorithm performs, the generated data is more similar to real noisy data. Specifically, below metrics are computed for comparison.

1. Error rate in different positions of the reconstructed strands. The result will be visualized in figures. The more similar the two figures look like, the better the result is.
2. Average value of (1) on real and generated data. The closer they are, the better the result is.
3. Number of perfect reconstructed strands. Closer is better.
4. Average value of absolute difference of (1) along different position, on real and generated data. Smaller is better. This single-value measure is more representative and straightforward. It is used when selecting the best NN-based models.

The double-sided Bitwise Majority Alignment (BMA) algorithm provided in the minilab is adopt as the trace reconstruction algorithm.

Trace reconstruction is only performed on the test split of data, which is data that is unseen during the building of the model. When running the trace reconstruction algorithm, we limit the coverage from both real and synthesized data to 5, to make the algorithm’s performance not too perfect.

# 4 A rule-based algorithm as baseline

According to the project guideline, the noisy data contains some complex features that cannot to be easily captured. To verify this claim, a simple rule-based algorithm is first tested to generate noisy strands.

## 4.1 Error matrix

The idea of this algorithm is quite straightforward. On the training split of data, the frequency of different errors in different positions, between every pair of clean strand and noisy strand, are first computed. Such frequencies are used as an estimation of probability of error distribution. In this way, an “error matrix” is constructed:

A picture containing Excel

Description automatically generated

Figure 2 Visualization of error matrix from different splits. From top to bottom: (1) train, (2) validation, (3) test. In each matrix, X axis represent position, Y axis represent different error types. There are 12 types of errors in total, they are: [A->T, A->C, … G->T, G->C].

In the error matrix of training split (Figure 1 (1)), we can get some interesting observations. First, the overall error distribution shows that error is more likely to happen in the middle of one strand compared to the side area. Second, the distribution is not symmetric along the position axis. Third, some error types are overall more likely to happen than other types in nearly every position, e.g., type 2,4,7,9.

If we compare visualization from different splits, we can find that although the error distribution from the validation and test split is not as “smooth” as that of training split, the overall shape of distribution is very similar across different splits. This shows that the error-distribution on training set can be used to predict unseen data (e.g., on test set).

## 4.2 Decoding: random sampling

Based on these observations, we conclude that it is reasonable to predict noisy strands according to the error distribution on the test data. Here is the prediction procedure.

For each nucleotide in each position on a clean strand, there are 4 types of results that might happen in total: 3 types of substitution error that might happen, each associated with a relatively small possibility, and can also remain the same (error free) with a relatively large possibility. A random sampling is performed among the 4 possible results according to the error distribution that has been obtained in previous section. After finish sampling for every nucleotide in a clean strands, we obtain a noisy counterpart of a clean strand.

## 4.3 Generation results

Here is one pair of generated noisy strands with the clean strand:

A picture containing graphical user interface

Description automatically generated

Figure 3 Generation result example of rule-based algorithm.

We can observe that the generated strands are very similar to the clean strands, with some occasional substitution error.

## 4.4 Drawbacks

There are two major drawbacks of this method

1. Characters in different positions of a same strands is synthesized independently. However, this assumption does not hold true.
2. Second, This method cannot model the insertion and deletion error, and the generation results are all of the same length as the reference results.

As the consequence, although the error distribution remains the same as the real data, the difficulty of trace reconstruction is lower than on the real data. As we can see from below figure and table, the trace reconstruction algorithm performs quite differently on real and noisy datasets:

|  |  |  |
| --- | --- | --- |
|  | Average error rate | Perfect strands |
| Real | 0.1181 | 332 |
| Generated | 0.0372 | 300 |

Table 3 Trace reconstruction performance comparison on real data and data synthesized by rule-based algorithm.

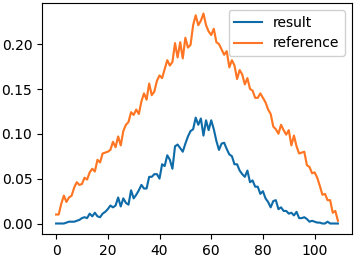


Figure 4 Error rate per position comparison.

# 5 Multi-layer Perceptron

## 5.1 The Model

When building neural-network models for certain application scenarios, it is a good choice to start from a simple model and then improve it from possible directions.

A multi-layer perceptron network, with an embedding layer and two linear transform layers as components, is built to test the synthesize performance:

Text, letter

Description automatically generated

Figure 5 MLP structure and number of parameters

It receives a clean strand as input as a whole. The embedding layer first convert the integer input to vectorized representation, then the linear transform layers output the probability of each character in each position in the output sequence. Because it generates the output according to the whole clean sequence, it does not have the problematic independent assumption of method in section 4.

## 5.2 Alignment

MLP usually does not perform well when the input and output sequence are not well aligned, and this is also the case for other similar “encoding-only” NN-based method. However, due to the common and variable insertion and deletion errors in real noisy strands, there is no simple and universal way to perfect align a noisy strand with its clean counterpart.

To minimize the misalign issue, this experiment adopts the strategy to “align from double side”:

For each pair of clean and noisy strands, they are first cut from the middle point to form two groups of sub-strands. Then, the second half of both strands are reversed. Finally, all sub-strands are aligned from the beginning.

## 5.3 Training

The model is trained with AdamW [1] optimizer, with learning rate 1e-3 and weight decay 1e-3. Experiment is performed on RTX A5000 GPU with batch size 512. The model is trained for 200 epochs with early stopping (patience=20).

## 5.4 Decoding: random sampling

The MLP outputs 4 values at each position, representing the possibility of different nucleotides to appear at this position. Random sampling is performed according to the output distribution to determine what is the character in the noisy strands at each position.

## 5.5 Results

Here are some generated noisy strands from MLP method:

A picture containing table

Description automatically generated

Figure 6 Generation example from MLP

Compared to the rule-based method, more complicated error patterns are shown in the generated strands. In addition, we can see that MLP frequently generate substrings with a same characters, which makes the generated strands quite different from the original strands.

Here is the trace reconstruction results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Average error rate | Perfect strands | Average positional error rate difference |
| Real | 0.1181 | 332 | - |
| Generated | 0.4274 | 0 | 0.3095 |

Table 4 Trace reconstruction performance comparison on real data and data synthesized by MLP.

Chart, line chart

Description automatically generated

Figure 7 Result comparison of MLP.

The difficulty of reconstruction on the MLP synthesized data is much higher than the real noisy data, so that not a single strand is perfectly recovered. From the figure, we can also see that the reconstruction algorithm’s behavior diverges greatly between the generated and real data.

## 5.6 Drawbacks

There are several major drawbacks of the MLP method:

1. The network models the token distribution of every position of a whole sequence directly. This task might be too difficult without specifically designed model structure and training strategy.
2. The alignment method is problematic. The double-direction alignment cannot guarantee every nucleotide in the noisy strand is perfectly aligned with the clean strand.
3. This method treats the first half and the second half in the same way, which brings the assumption that the error distribution is symmetric. However, this assumption is questionable as conflicts with observations in section 4. Moreover, the nanopore device is designed to perform the sequencing in a single-directional manner, hence we cannot expect the error pattern is symmetric.

# 6 Sequence-to-sequence Model

## 6.1 The Model

The alignment is important for the noise synthesis because the majority operation of the synthesis process is simply copying. When generating noisy strands for a certain region, only when the model knows the location of the corresponding region in the clean strand, it has higher chance to perform the copying more accurately. However, from previous experiment, we conclude that it is not a trivial problem to find a proper alignment between the target sequence and input sequence.

We can borrow some experience from neural machine translation (NMT) because of the similar nature of the two tasks. Both tasks aim to generate a new sequence according to a given sequence; both tasks require to find a proper alignment between the target sequence and input sequence. Hence, we adopt the widely used attention-based encoder-decoder structure in NMT to solve our task.

Specifically, the design of this model adapted from the NMT model in [1], as shown below:

Diagram, engineering drawing

Description automatically generated

Figure 8 The sequence to sequence RNN model in [1]. Red box: encoder. Blue box: decoder.

Before the encoder or decoder handle the sequence input, the sequence is converted to a vectorized representation by a shared embedding layer.

The encoder network (in the orange box) is a bi-directional recurrent neural network (RNN) that aims to convert the raw input sequence into a sequence of “annotations”. The annotation contain information about the whole input sequence with a strong focus on the parts surrounding the -th token in the input sequence.

The decoder network (in the blue box) is a single-directional RNN that receives the annotations and previous generated tokens as input, and predict the probabilities of different tokens as the next token to be generated. Inside the decoder, before generating the probability for the next token, it uses attention mechanism, which is also a group of learnable parameters, to determine the part of annotations from encoder that need to be pay special attention to because it might contain more helpful information for this step of generation.

With the help of the attention model, the information from encoder’s annotations is not passed to the decoder as a whole; instead, what is passed to the decoder is a weighted average of different annotations, where annotations that deserve more attention has higher weight value.

When implementation, the Gated Recurrent Unit (GRU) cells are adopted in both RNNs in encoder and decoder because it is not less easy to overfit than the original Long Short-Term Memory (LSTM) cells. For the best model, the hidden size is set to 128, the number of GRU layers is 1 for both encoder and decoder.

## 6.2 Training

The model is trained with AdamW [1] optimizer, with learning rate 1e-3 and weight decay 1e-3. Experiment is performed on RTX A5000 GPU with batch size 512. The model is trained for 20 epochs with early stopping (patience = 5).

## 6.3 Decoding: greedy sampling

The greedy sampling is adopted for decoding process. The “greedy” here means when computing probabilities of different tokens at a certain position, we perform the sampling right after the probability distribution of this position is computed. Another alternative is beam sampling, which consider the top-k subsequence with highest probability before doing the sampling. We choose the greedy algorithm due to the simplicity of implementation.

## 6.4 Results

Here are some generated noisy strands from the sequence-to-sequence model:

A picture containing table

Description automatically generated

Figure 9 Generated data example from Sequence-to-sequence model.

We can observe that no matter at which position in the noisy strands, we can find the counterpart subsequence in the clean strand. In addition, different types of errors, including insertion, deletion, substitution, are successfully reflected in the generation results.

Here is the trace reconstruction results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Average error rate | Perfect strands | Average positional error rate difference |
| Real | 0.1181 | 332 | - |
| Rule-based | 0.0372 | 300 | 0.0909 |
| MLP | 0.4274 | 0 | 0.3095 |
| **Seq-to-seq** | **0.1235** | **333** | **0.0078** |

Table 5 Result comparison of different methods.

Chart, line chart

Description automatically generated

Figure 10 Trace reconstruction algorithm behavior comparison.

Trace reconstruction has very similar behavior on the real data and on the synthesized data by the sequence-to-sequence model.

## 6.5 More explorations

The best performing model is selected among a series of explorative experiments. Their results are shown in the below table:

|  |  |  |
| --- | --- | --- |
| **Models** | **Average positional error rate difference** | **Comments** |
| Hidden=64 | 0.0214 | Model too small. |
| **Hidden=128** | **0.0078** | **Best model.** |
| Hidden=256 | 0.0114 | Wider model does not bring performance gain. |
| Hidden=512 | Not converge | Model too large. |
| Single directional encoder | 0.0175 | The encoder need to be bi-directional. |
| 2-layer encoder | 0.0121 | Deeper model does not bring performance gain. |
| 2-layer decoder | 0.0104 | Deeper model does not bring performance gain. |
| Add 1-d convolution to encoder | 0.0153 | It does not help to add more locality focus. |
| Add 1-d convolution to encoder with residue | 0.0187 | Same as above. |
| Add self-attention to encoder | 0.0631 | Compared to RNN, self-attention might not be compatible with this task. |
| Add Transformer layers to encoder | 0.0232 | Model too large. Transformers might not be suitable. |

Table 6 Result of all exploration experiments.

# 7 Conclusion

In this report, several different designs and implementations are demonstrated to build a system to synthesize the behavior of a DNA-based storage system. Among all methods that have been tested, the sequence-to-sequence NN-based method shows the best performance where the trace reconstruction algorithm gives very similar results on the synthesized data compared to real noisy data.

# Reference

1. Loshchilov, Ilya, and Frank Hutter. "Decoupled weight decay regularization." *arXiv preprint arXiv:1711.05101* (2017).
2. Bahdanau, Dzmitry, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *3rd International Conference on Learning Representations, ICLR 2015*. 2015.