Monash University

Department of Electrical and Computer Systems

Engineering

Final Year Project

Digital Data Storage on Synthetic DNA with Machine Learning

Ryan Wee Qhai-Yhee (28327519)

(Supervisor: Prof. Emanuele Viterbo)

May 28, 2022

MONASH University
Engineering

Department of Electrical and Computer Systems Engineering

ECE4095 Final Year Project 2022

Digital Data Storage on Synthetic DNA with Machine Learning

Ryan Wee
Supervisor: Prof. Emanuele Viterbo

What is Synthetic DNA

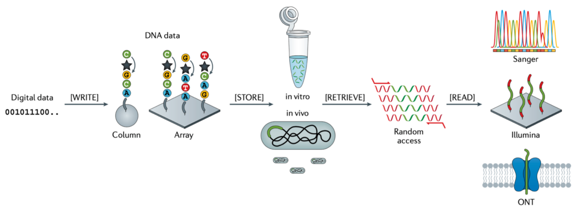
DNA is a molecule that carries genetic instructions in all living things. Information is stored in DNA as a code of four bases - A, T, C and G.

Synthetic DNA is DNA artificially created in a laboratory. Digital data is coded into four bases, then written to DNA upon synthesis.

Why Store Data on Synthetic DNA

Data storage needs are ever increasing, existing mediums (flash memory, hard disks) lack storage longevity and density.

DNA has high information density (215 petabytes per gram) and can be preserved for over 2 million years in right conditions.



Reading Stored Data off DNA

DNA is passed through a nano-sized pore. Each different base changes the electrical current of the pore. The nanopore current is decoded to recover the base sequence.

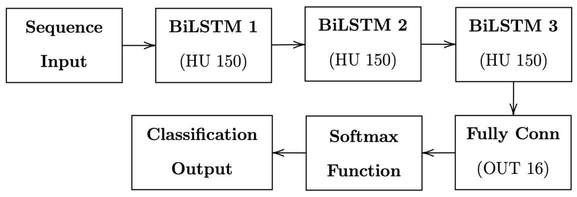
Challenges with Decoding

The nanopore current measurement is noisy and stochastic. Individual bases pass through the nanopore at different speeds, causing time stretch inconsistencies. This introduces challenges in signal processing.

Decoding with Machine Learning

Oxford Nanopore developed a technology demonstrator (Scrappie) that predicts nanopore current measurements from the encoded sequence of bases.

A recurrent neural network (RNN) was developed natively in MATLAB to decode current measurements. The RNN is made up of a stack of 3 Bi-LSTM layers and trained on a dataset was generated with Scrappie.



94 % **Success Rate**

Network Training Time
(for 16 alphabets) 8 mins

Significant contributions

Implemented a neural network based basecaller in MATLAB with improved training time at comparable success rates to existing neural network basecallers.

Executive Summary

Deoxyribunucleic acid (DNA) is a molecule that contains all genetic information required to build and maintain an organism. Information is stored in DNA as a code of four bases - A, T, C and G. DNA has a very high information density (215 petabytes per gram), hence there are compelling reasons to store digital data on artificially synthesised DNA. Information stored on DNA is read by passing the DNA through a nano-sized pore and measuring the electrical resistance of each base. However, this is a noisy and stochastic process that reduces reading accuracy. A machine learning neural network was developed in MATLAB to classify the encoded data from the noisy electrical signal from the nanopore. For 16 encoded alphabets, the network was trained in 8 minutes with a reading accuracy of 94%. This is more efficient than the existing network developed by the research team at Monash University, yet comparable in accuracy.

Acknowledgements

This endeavour would not have been possible without my supervisor Professor Emanuele Viterbo and PhD student Viduranga Wijekoon who graciously offered for their expertise, guidance and support throughout this project.

Special thanks to Richard Nugroho and Yashdeep Dahiya who offered second technical opinions and suggestions whenever I needed them.

I extend my sincere gratitude to my parents, without whom I would not have enjoyed so many opportunities and the privilege to further my higher education in a prestigious university abroad.

Lastly, I would like to thank my incredible friends who have offered me their unconditional love and support at all times. In alphabetical order:
Amy, Anya, Arlie, Brilliana, Doris, Elvin, Gina, Jared, Jasmine, Jay, Jesh, Marcellia, Mason, Paige, Sophie, Stef, Sunny, Yasmin, Zeeva and many many more.

Abbreviations

Application-specific Integrated Circuit (ASIC)

Artificial Neural Network (ANN)

Connectionist Temporal Classification (CTC)

Convolutional Neural Network (CNN)

Correlation Optimised Warping (COW)

Constraint randomised non-negative factor analysis (CRNNFA)

Deoxyribonucleic Acid (DNA)

Dynamic Time Warping (DTW)

Forward error correction (FEC)

Hidden Markov Model (HMM)

Long Short Term Memory Network (LSTM)

Oligonucleotides (Oligo)

Oxford Nanopore Technologies (ONT)

Polymerase Chain Reaction (PCR)

Recurrent Neural Network (RNN)

DNA neither cares nor knows. DNA just is.
And we dance to its music.

–Richard Dawkins

River Out of Eden: A Darwinian Life

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Brief Background

Deoxyribonucleic acid (DNA) is a double helix molecule responsible for carrying genetic instructions in all living things. Information is stored in DNA as a code made up of one of four bases - A, T, C and G [1][2].

Today's ever increasing data storage needs are causing traditional storage mediums, such as flash memory and hard disks, with their limited storage density and longevity to become obsolete. The evolution of DNA over billions of years through natural selection gives DNA high information density and longevity, thereby making synthetic DNA derived from the design of biological DNA a compelling method of digital data storage.

However, DNA storage in its current form is impractical for large-scale use [7]. This project aims to explore the potential of this emerging technology, while finding new methods to overcome its existing downsides and limitations.

1.2 Process

The major steps in DNA digital data storage are:

1. Encoding and writing the digital data to the synthetic DNA strand
2. Storing the synthetic DNA strand in a suitable environment
3. Retrieving the strands of synthetic DNA containing the relevant data to be accessed
4. Reading and decoding the information off the synthetic DNA strand

Due to the novelty of the technology, conducting research and development on most of the process is cost prohibitive. Instead, this project aims to establish a new technique to improve on the reading and decoding phase of the process. This implementation would build upon one of the well-established industry standards for the synthetic DNA storage process.

1.3 Objectives

Noise introduced during the reading process introduces errors and efforts to handle these errors reduce the information rate. This reduces the efficiency and viability of synthetic DNA as a data storage medium. To effectively implement data storage on synthetic DNA, the implemented decoding process needs to correctly decode the synthetic DNA strands despite the presence of noise.

There are existing solutions designed by research fellows and students within Monash University. This project aims to contribute to them by improving upon an existing approach or designing a novel approach with better performance or the promising potential for better performance.

A well designed encoding and decoding scheme improves accuracy and reduces loss, thereby increasing information rate, reducing cost and pushing digital data storage on synthetic DNA one step closer to practical implementation.

1.4 Project Scope

The scope of this project is limited to development of new encoding/decoding schemes to improve the performance of existing nanopore sequencing technology implemented by Oxford Nanopore Technologies (ONT).

Potential improvements in the sequencing process and other hardware implementations are beyond the scope of this project.

Chapter 2

Literature Review

2.1 Artificial DNA Synthesis

DNA synthesis refers to the joining process of deoxynucleic acids, adenine (A), thymine (T), cytosine (C) and guanine (G) to form DNA [23]. The double stranded, twisted ladder helix structure of DNA is constructed by joining these chemicals to form base pairs. Adenine is joined to thymine by two hydrogen bonds and cytosine is joined to guanine by three hydrogen bonds. The complementarity in size (small pyrimidines A and G are paired with large purines T and C) and hydrogen bonding is crucial for the stability and functionality of biological DNA [24].

Unlike biological DNA synthesis, artificial DNA synthesis need not follow a template DNA, allowing for the synthesis of virtually any DNA sequence, including nucleotides that are unavailable in nature, also known as unnatural base pairs (UBP) [24].

Oligonucleotide synthesis through solid-phase chemical synthesis remains the most widely used form of artificial (synthetic) DNA synthesis. Relatively short fragments of DNA (oligonucleotides) are chemically synthesised (created) with ribonucleosides (A, T, C and G) [25]. The current practical limit in oligonucleotide synthesis is around 200 base pairs and needs to be taken into account when designing a digital data storage scheme.

2.2 Viability of Data Storage on Synthetic DNA

2.2.1 Advantages

Information density is the biggest advantage of DNA storage. By encoding data on a molecular level, DNA strands average just 2 nanometers in width [3]. A method published in 2017, DNA fountain, successfully stored 215 petabytes of data in one gram of DNA [4][7]. This is 85% of the theoretical limit in the information transfer rate of DNA, otherwise known as its channel (Shannon) capacity [7].

DNA storage can be implemented as a three-dimensional volume unlike traditional storage media such as disks and tape that store data on a two-dimensional surface. This ability to compact individual DNA strands in three-dimensional space further increases the storage density of synthetic DNA [13].

Existing storage mediums such as magnetic hard disks and flash memory have limited data retention times of around 10 years [5]. Accelerated aging experiments published in 2015 showed the perfect recovery of 83 kilobytes of information after an equivalent storage period of 2000 years with the help of error-correcting codes and an inorganic storage matrix. Results from the experiment predict that data encapsulated within synthetic DNA can be stored successfully for over 2 million years in the Global Seed Vault [6]. This makes DNA storage a viable solution for long term storage applications (>50 years), such as archiving.

The 4-base (A, T, C, G) encoding method used in DNA is comparable to the sequential use of 1's and 0's found in digital data [5]. This simplifies the conversion process between DNA and digital information, unlike if the storage medium were analog, which would require conversion between digital and analog forms of the data.

2.2.2 Limitations

DNA storage in its current form is impractical for large scale use. Even though the aforementioned DNA Fountain scheme is relatively cost effective, 2 megabytes of data cost US$7000 to synthesise and US$2000 to read [7].

Uncertainties in DNA encoding caused by oligo synthesis, polymerase chain reaction (PCR) amplification and storage decay processes require robust error detection and correction codes for practical application of DNA storage [7][9]. Developing new error correction approaches is an area with significant potential for improvement of overall efficiency with the lowest cost and is the sole focus of this project.

The existing limit in sequence length of DNA that can be reliably synthesised is 200 nucleotides, equivalent to a theoretical maximum of 50 bytes of data per strand per the entropy limit of 2 bits per nucleotide [10]. In practice, this information limit cannot be reached as redundancy in the form of error and detection codes are needed. A 50 byte limit is impractical for most data storage uses, hence there is also a need to break data down into chunks and include an indexing scheme to infer their order. Error correction and indexing combined decrease the information rate and viability of digital data storage on synthetic DNA.

Direct access (random access) refers to the ability to read a selected portion of stored data without the need to read all stored data. Most storage applications are designed for direct access to maximise efficiency and most storage devices enable this by storing data in discrete locations with a unique address [17]. However, implementing direct access in synthetic DNA with traditional synthesis techniques is inefficient when DNA strands are stored, free-floating in a matrix and indexing is only as efficient as the random success rate of drawing a DNA strand that is relevant to the set of data to be decoded. Research into algorithms that maximise sequential read coverage and Nanopore technology discussed later improves on this [12].

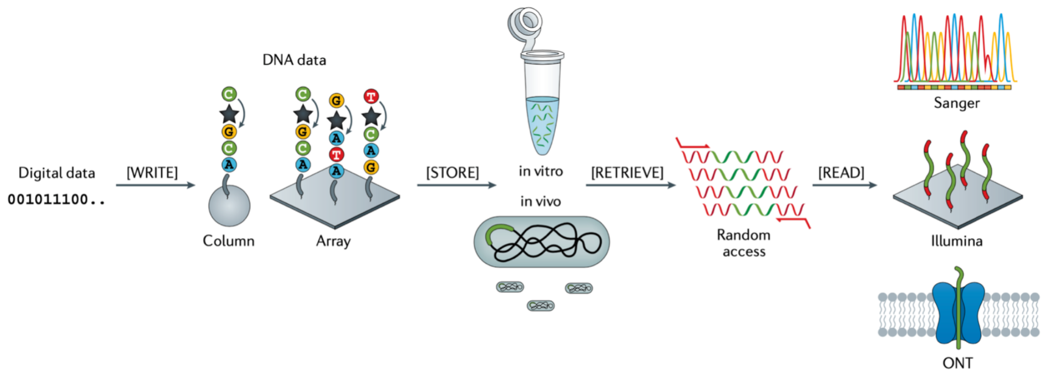


Figure 2.1: Overview of major steps in data storage on synthetic DNA [5]

2.3 DNA Sequencing Methods

DNA sequencing refers to the process of determining the order of nucleotides in DNA, that is the order of the four bases, adenine (A), thymine (T), cytosine (C) and guanine (G) [19]. In the context of data storage on synthetic DNA, DNA sequencing is used in the reading process to read the bases on the synthesised DNA strands that encode the digital information. There are currently 3 major DNA sequencing methods:

2.3.1 Sanger Sequencing

Sanger sequencing was developed in 1977 by Frederick Sanger and colleagues and was the first widely adopted DNA sequencing method. Despite replacement by next-generation sequencing methods (Illumina and nanopore, discussed later) for large volume sequencing, Sanger remains widely used for smaller-scale projects. Its main advantage over Illumina and nanopore sequencing is its ability to read long-length DNA sequences of over 500 nucleotides at a 'gold standard' accuracy rate of around 99.99% [20][21][22].

However, long-length DNA sequencing is not beneficial for this project as DNA synthesis is only practical for producing short DNA strands of up to 200 base pairs. Recently, Sanger sequencing was used to sequence the spike protein of the SARS-CoV-2 virus to be used in COVID-19 vaccine research and development [21].

2.3.2 Illumina Sequencing

Illumina popularised what is currently the most common DNA sequencing technology through synthesis. DNA strands are immobilised on a solid support as the sequence of base pairs within the DNA are read through a 3-step process known as Illumina dye sequencing [8][11]:

1. **Amplify**

   Purified DNA is fragmented for processing. Adapters are added to the fragmented DNA (oligo) to provide reference points. The DNA is loaded onto a flow cell containing nanowalls that the adapters latch on to. Once the fragments of DNA have attached onto the adapters, they are duplicated through the use of bridge amplification polymerase chain reaction (PCR).

2. **Sequence**

   Primers and modified nucleotides containing a reversible fluorescent blocker are added with the DNA fragment onto the chip. This blocker limits the addition of modified nucleotides onto the DNA fragment to one at each time.

3. **Analyse**

   An image of the chip is analysed to determine the base of the nucleotide. This is done by measuring the wavelength of the corresponding fluorescent tag. After analysis, the fluorescent blocking group is removed by a chemical deblocking process.

Because processing is conducted on fragments of the complete DNA double helix, massively parallel sequencing can be performed to sequence a significant proportion of the complete DNA on each run.

2.3.3 Nanopore Sequencing

Nanopore sequencing reads single molecules by electrophoretically driving molecules in solution through a nano-scale pore without the need for PCR amplification or labelling seen in traditional sequencing techniques, thereby eliminating the associated error rate and reducing financial and sequencing time cost [14]. Each nucleotide present in the pore changes the pore's electrical resistance, hence the corresponding base can be read as a measure of change in ionic current as the DNA strand is passed through the nanopore [26]. The process of assigning nucleotide bases to ionic current changes is known as base calling. Core nanopore sequencing patents were licensed by Oxford Nanopore Technologies (ONT). ONT has a range of open source basecallers, one of which this project would be built around.

Adapters are ligated to both ends of the DNA strand, then loaded onto a flow cell like Illumina's sequencer. These adapters facilitate strand capture (and

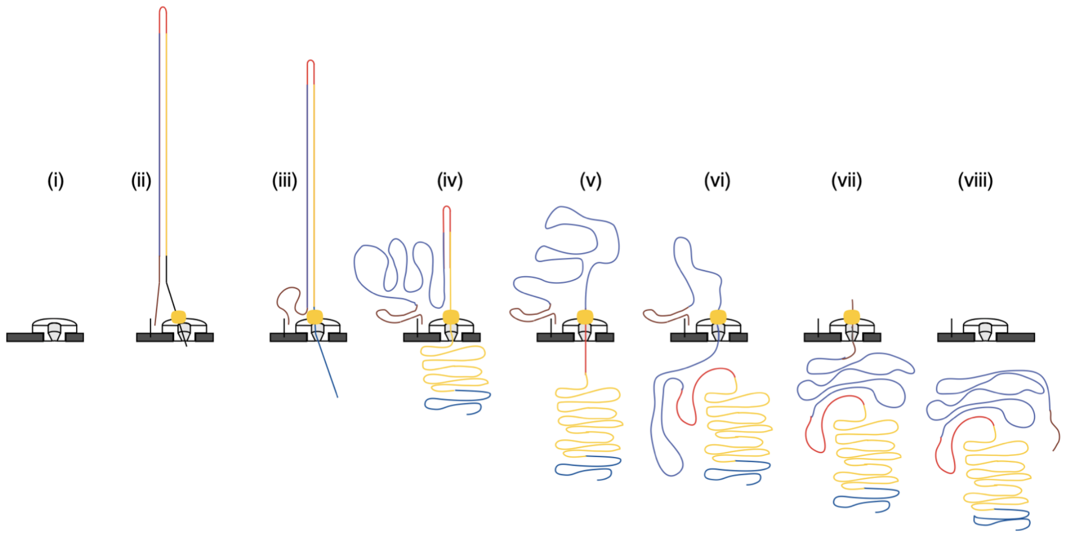


Figure 2.2: DNA translocation steps through the nanopore [16]

base calling). The flow rate of the DNA strand through the nanopore is controlled by loading an enzyme. Up to 2048 individually addressable nanopores can be controlled in groups of 512 nanopores by an application-specific integrated circuit (ASIC).

Changes in ionic current (in the region of picoamperes) are measured as the DNA strand is passed through the pore. These changes are caused by differences in the shifting nucleotides occupying the pore. The duration and amount of discrete changes in ionic current are analysed. With sufficient signal processing, the corresponding nucleotide base is identified.

2.4 Signal Processing

2.4.1 Nanopore Noise

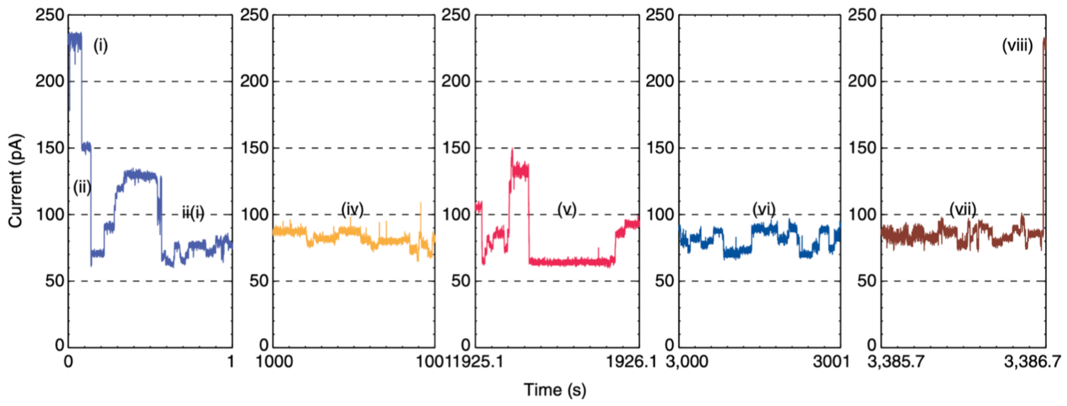


Figure 2.3: Ionic current reading of a DNA strand [16]

The figure above is an example of a typical ionic current reading from a nanopore. While clear discrete steps of varying ionic current could be observed by eye, noise present in the current output would prevent accurate decoding of the signal. The clean signal observed at the encoder after the base calling process is a stairstep plot with discrete current levels corresponding to nucleotide bases.

The amplitude noise, observed here as amplitude-wise inconsistencies could be sufficient to cause an error by misrepresenting itself as a different codeword. As the molecule is electrophoretically extruded from the nanopore, non-linear stretching of the ionic current reading is exhibited temporally due to friction. This is referred to as stretch noise and leads to incorrect 'pulse durations' which affects reading accuracy.

The decoder and codebook must be designed to ensure decoding performance in the presence of amplitude and stretch noise meets the required specification.

Noise handling built into the decoding process can be performed with traditional signal processing techniques.

2.4.2 Dynamic Time Warping (DTW)

Dynamic time warping (DTW) was invented to align words to improve the performance of speech recognition algorithms when dealing with different speaking speeds [47][48]. It is currently the most completely implemented method used by the team of research students at Monash University to correct for stretch noise.

DTW aims to find an optimal match between two given sequences. A cost matrix (sometimes referred to as distance matrix) is computed based on the Euclidean distance of each matched pair of indices. The 'path' of minimal cost denotes the optimal match [48][49].

DTW requires the first and last indexes of both sequences to match [49]. However, the application of DTW in this project will break this restriction. If DTW were applied for this project, developing a method to apply DTW on an 'open ended' or 'closed ended' target sequence would be one of the objectives.

Initialise inputs and cost matrix

- Given input sequences $x_{1:N}$ and $y_{1:M}$
- Cost matrix will be $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Where for $i = 1 : N$, $\mathbf{D}_{i,0} = \infty$ (bottommost row of matrix is infinity) and for $j = 1 : M$, $\mathbf{D}_{0,j} = \infty$ (leftmost column of matrix is infinity)

Calculate cost matrix

Iterating over the entire matrix,

$$\mathbf{D}_{i,j} = \|x_i - y_j\| + \min \begin{cases} \mathbf{D}_{i-1,j-1} & \text{(match)} \\ \mathbf{D}_{i-1,j} & \text{(insertion)} \\ \mathbf{D}_{i,j-1} & \text{(deletion)} \end{cases}$$

Find alignment

Trace lowest cost from $\mathbf{D}_{N,M}$ to $\mathbf{D}_{0,0}$

Code implementation

```
int DTWDistance(s: array [1..n], t: array [1..m]) {
    DTW := array [0..n, 0..m]

    for i := 0 to n
        for j := 0 to m
            DTW[i, j] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],
                                        DTW[i  , j-1],
                                        DTW[i-1, j-1])

    return DTW[n, m]
}
```

2.4.3 Correlation Optimised Warping (COW)

Correlation optimised warping (COW) is a newer technique originally developed to correct time shifts in chromatograms - this remains the most common use of COW today [47].

COW divides the sample and reference sequences into a number of segments. Each segment in the sample sequence is individually warped to best align with the corresponding segment in the reference [51].

The two parameters of COW are:

- Segment length, m which defines the number of segments from the fixed sequence length.
- Slack, t which determines the range at which the individual segments can be warped to.

Alignment with COW is achieved by optimising the combination of the m and t pair. This optimisation process is computationally intensive and pre-processing methods such as constraint randomised non-negative factor analysis (CRNNFA) is often implemented to improve computational time [52].

Segmentation and warping

Given two sequences of same length L.

The COW algorithm can be broken down into 3 steps [52][53]:

1. COW divides sequences of length L into N segments of length m, such that $m = \frac{L}{N}$.

2. Individual segments are warped (temporally compressed or expanded) up to a limit defined by the slack parameter, t.

   To ensure both ends of the sequence retain their original start and end time points, the first segment can only be warped along its right end and the last segment can only be warped along its left end. The remaining segments are free to be warped along both sides of the warping range of $[-t, +t]$.

3. A linear interpolation approach is used to synthesise the warped sequence.

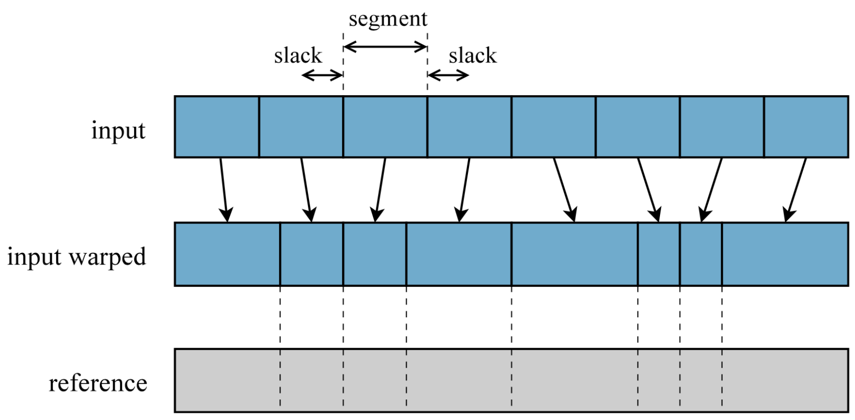


Figure 2.4: Diagram demonstrating COW [53]

Optimisation

The Pearson correlation coefficient, ρ between the segments of the reference and warped sample signals is used to optimise the m and t pair. It is evaluated by the following equation [47]:

$$\rho = \frac{\left(W_X - \text{mean}\left(W_X\right)\right)^T \left(W_{Y_{warped}} - \text{mean}\left(W_{Y_{warped}}\right)\right)}{\sigma\left(W_X\right)\sigma\left(W_{Y_{warped}}\right)}$$

where W_X and $W_{Y_{warped}}$ are segments of the reference and warped sample signals respectively and σ refers to the standard deviation.

The range of the correlation coefficient is $[-1, +1]$ where a value close to 1 indicates the achieved alignment is of good quality and a small correlation value indicates poor performance where further optimisation is required.

Dynamic programming (DP)

Dynamic programming (DP) is a mathematical optimisation technique that breaks down a complicated problem into simpler sub-problems recursively.

COW is categorised as a DP algorithm since the global problem of shifts across the sequence is broken down and optimised within individual segments. All combinations of m and t pairs are optimised individually to achieve the best alignment with the reference sequence [51].

Example

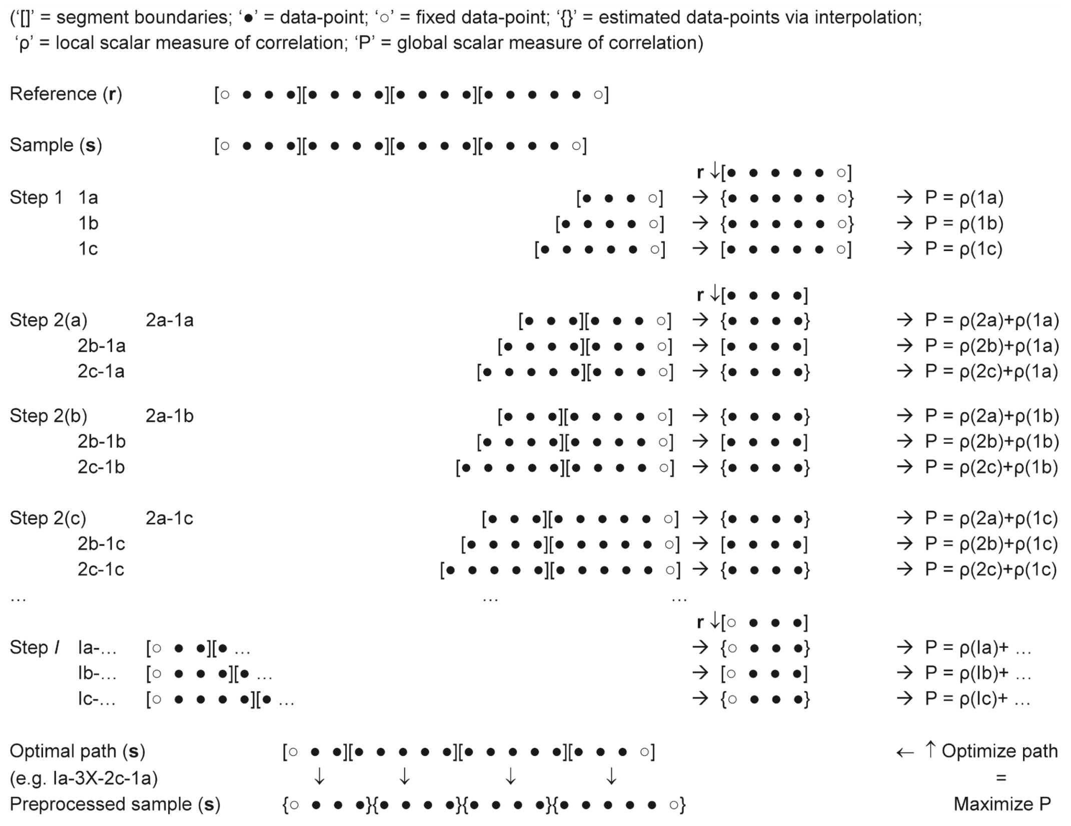


Figure 2.5: Example of COW [47]

In this example, the reference sequence ($R = 18$) and sample sequence ($S = 17$) are divided into $N = 4$ segments of length m. Because the sequence lengths are not multiples of the target segment length, the last segment is extended to cover the full length of the sequence.

Step 1 Start from rightmost segment with slack parameter $t = 1$.

There are 3 possible boundaries, 2 of which require interpolation to yield the matching number of data points with the reference segment (indicated by {}).

Step 2 There are 3 possible boundaries for the second rightmost segment, building upon the first 3 possibilities.

Step *n* Repeated until all correlation coefficients for all possible segments have been iterated. Optimum path can be traced by following the largest value of P [47].

2.5 Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) devoted to learning data to improve performance on tasks automatically through experience [30][31].

Generally, machine learning approaches can be broadly divided into three categories:

1. **Supervised learning**

   Mathematical models are built on training data containing desired outputs for the inputs, referred to as labels. Over training, the aim of the model is for it to learn to predict or determine outputs correctly with new inputs not present in the training data.
   Two primary examples of supervised learning include classification and regression. Classification algorithms have outputs belonging to a finite set of values (discrete), while regression algorithms have outputs that may be any numerical value within a defined range (continuous) [32].

2. **Unsupervised learning**

   Mathematical models are built on training data that have not been labeled, classified or categorised, and attempt to find structure and patterns within the given training data. The main application for unsupervised learning is density estimation (eg. finding the probability density function) within the field of statistics [32].

3. **Reinforcement learning**

   Algorithms interact directly within a dynamic environment, performing a set task where feedback analogous to rewards is provided for its actions. The aim of the algorithm is to take a set of actions that would maximise the cumulative reward [33]. Common applications of reinforcement learning algorithms include autonomous vehicles and gameplay against human opponents.

2.5.1 Neural Networks

The type of machine learning model used greatly depends on its application. Neural networks are the most common approach to implement for this project, however other model types, such as decision tree learning, regression analysis and support-vector machines can be implemented with unique benefits and should not be dismissed.

Neural networks loosely model the network of neurons in a biological brain. Their structure comprise of node layers - an input layer, one or more hidden layers, and an output layer. Every artificial neuron, called a node, is connected to another with an associated weight and threshold. The weight controls how much of an impact the associated neuron would have relative to other neurons.

If the output of any node exceeds its threshold value, its data is passed along to the next node(s) it is connected to [34][35].

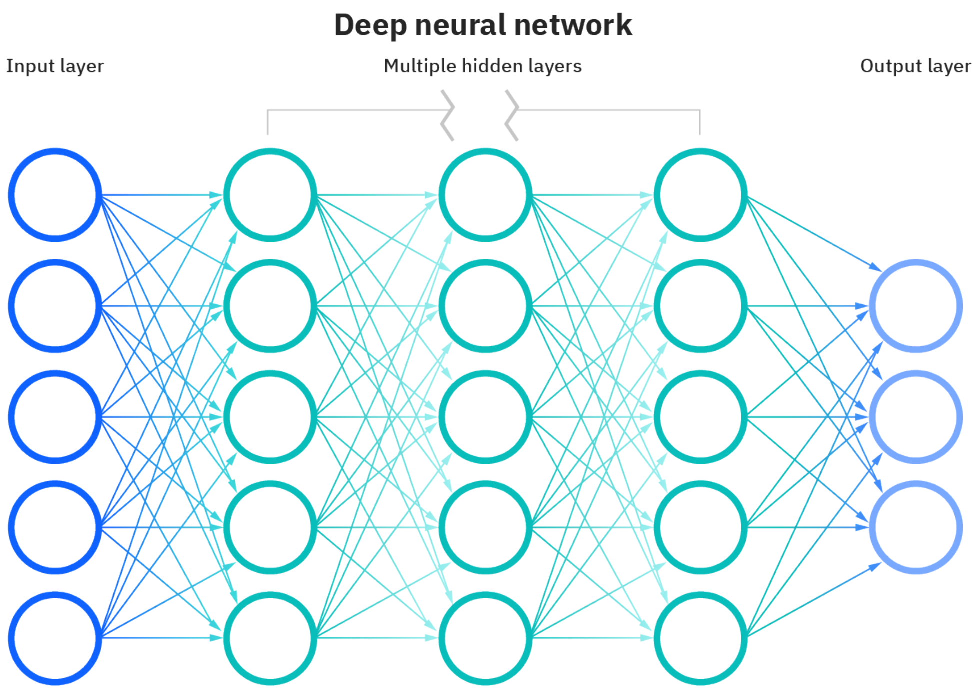


Figure 2.6: Structure of a neural network [35]

Neural networks are primarily split into three types:

1. **Artificial Neural Network (ANN)**

   An artificial neural network (ANN), also known as a feed-forward neural network, is one of the simplest types of neural network. Information is only passed in the forward direction until it reaches its final output node. ANNs are often used on tabular data for computer vision and facial recognition applications. They are generally considered to be less powerful than CNNs and RNNs.

2. **Convolutional Neural Network (CNN)**

Convolutional neural networks (CNNs) use convolutional layers to create feature maps that deconstruct an image into a grid where each grid is individually processed. CNNs are often used on image data for image classification and natural language processing (NLP) applications. They can classify images to very high accuracy but will require large amounts of training data to do so.

3. **Recurrent Neural Network (RNN)**

   Recurrent neural networks (RNNs) feed outputs of processing nodes back into the model and learns through time by holding memory of previous inputs and back-propagating to minimise loss. RNNs are often used on time-series (sequence) data for time-series prediction and text to speech applications. They are the most complex out of the three and are the most difficult to train.

2.5.2 Hyperparameters

Learning Rate

The learning rate of a model defines the size of each corrective step taken by the model to adjust for the errors from each observation. The higher the learning rate, the shorter the training time but the lower its accuracy [38]. Methods that implement adaptive learning rate can better balance training time and accuracy for maximum training efficiency.

Number of Hidden Layers

Hidden layers represent the layers of nodes between the input and output layers. The number of hidden layers directly correspond with the complexity of the network and the problem. A single hidden layer is only sufficient to represent linearly separable functions, which is not complex enough for most multi-dimensional problems [39]. However, too many hidden layers can cause overfitting which is not desirable (discussed later).

Batch Size

When one training epoch has been completed, the learning algorithm has completed one pass through the training dataset, where its samples were separated into groups in the size of the batch size [40].

2.5.3 Datasets

The dataset of a neural network can be split into three sets for different use:

- **Training data**

  Data samples used to train and fit the network.

- **Validation data**

  Data samples initially not visible to the network used to qualify performance during training to tune hyperparameters without bias. With continuous training, eventually validation data is too learned by the network [37].

- **Testing data**

  Data samples completely hidden to the network that provides an unbiased performance evaluation on the final model. Using information from the test set in any way before the model is finalised is often referred to as 'peeking' [38].

2.5.4 Limitations

Overfitting

Overfitting refers to a modelling error in statistics where the model aligns too closely to a limited set of data points and would only prove useful if used in reference to its original training dataset [41]. In machine learning, overfitting happens when the model over-learns the detail and noise present in its training data such that it negatively affects its performance when presented with new data [42]. Overfitting is often caused by an overly complex model. Ideally, a model should be tuned to the sweet spot between overfitting and underfitting.

Algorithmic Bias

Algorithmic biases often occur in machine learning especially when the model is exposed to input insufficiently or never represented by its training data. Hence, access to and management of data is crucial to develop a successful machine learning model.

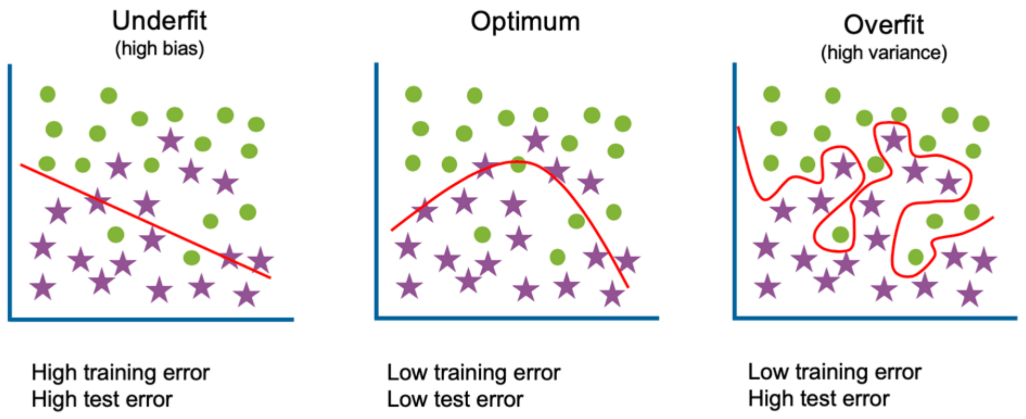


Figure 2.7: Representation of overfitting [42]

2.6 Notable Implementations

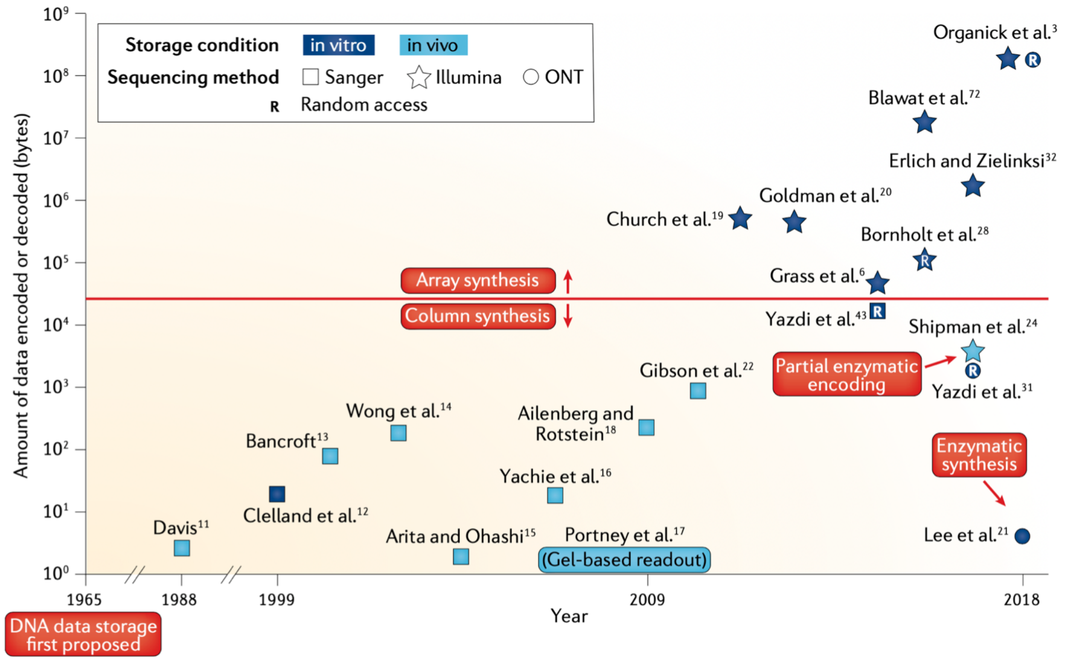


Figure 2.8: Timeline on notable published works on synthetic DNA data storage [5]

2.6.1 Nanocall

At time of development, ONT's sequencers use a cloud computing platform, Metrichor to perform base calling. Nanocall was developed as the first open-source offline basecaller to addresses this limitation and enable offline and private analysis of data measured by ONT's sequencers [28].

Nanocall uses hidden Markov models (HMMs) and the Viterbi decoding algorithm to compute the state sequence with the highest likelihood to have stemmed from the observed event sequence. Testing conducted on *E. coli* and human DNA with and without PCR amplification averaged a decoding success rate of $\sim 68\%$ [28].

2.6.2 NanoReviser

NanoReviser is an open source basecalling reviser developed to improve the decoding accuracy of existing basecallers. Instead of completely replacing the basecaller and replicating its functionality, NanoReviser takes a novel approach by acting as a post-basecalling reviser, analysing both the raw and basecalled signal to improve basecalling accuracy [29].

NanoReviser corrects errors by feeding both original electrical signals and post-basecalling base sequences into a neural network. Its neural network structure includes a CNN, an RNN and two Bi-LSTMs [29].

Testing was conducted on a high-performance computer with 4 CPU processors (Intel Xeon E7-4809v4), 4 GPU processors (Nvidia Tesla T4) and 576 GB of DDR4 system memory. *E. coli* and human DNA datasets were used and training time over 50 epochs was 334.50 hours and 827.32 hours respectively. The following table compares the decoding accuracy between NanoReviser and other notable ONT basecallers [29].

	Success Rate (%)	
	E. coli DNA	Human DNA
Albacore	80.45	77.26
Scrappie (events)	83.16	81.11
NanoReviser	85.78	82.93

Table 2.1: Comparison of decoding accuracy between NanoReviser and notable ONT basecallers

2.6.3 DNA Fountain

In 2017, Yaniv Erlich and Dina Zielinski published DNA Fountain, which stored data that approaches the Shannon capacity limit of synthetic DNA data storage [7]. DNA Fountain builds upon Illumina's technology with the application of fountain codes for increased robustness.

Fountain codes are rateless codes with flexibility for dynamic code rates that adapt according to channel conditions, thus allowing for the design of scalable transmission of data without the need to fix data rate according to channel quality. The most common fountain codes are Luby and Raptor codes [16][18].

A screening process added onto the original fountain code design is responsible for its aforementioned 1.98 bits/nucleotide coding potential. The DNA Fountain encoder works in three steps [7]:

1. The binary file is pre-processed into non-overlapping segments of defined length.
2. Each segment is converted into Luby Transform (LT) code, a near-optimal forward error correction (FEC) code. These converted codes are called droplets.
3. The algorithm translates binary droplet (`00,01,10,11`) to DNA sequence (`A,C,G,T`). The sequence is then screened and all droplets containing errors and invalid patters are discarded. This process is iterated until sufficient oligos (DNA fragments) have been generated within the design file.

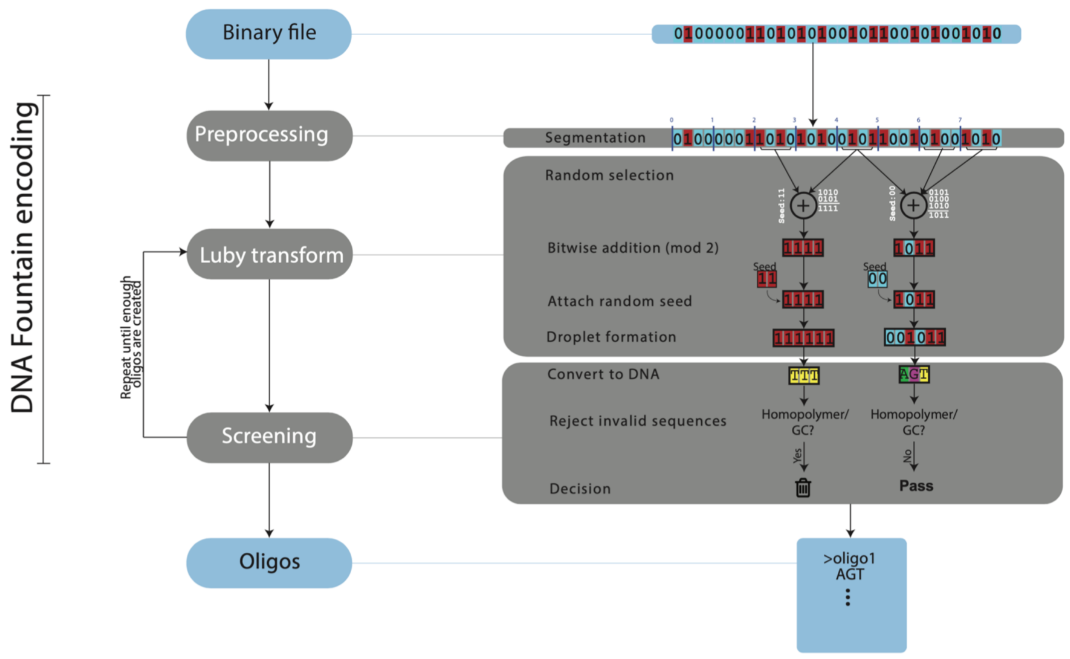


Figure 2.9: DNA Fountain encoding process [7]

Chapter 3

Methodology

After careful consideration of the possible approaches (DTW, COW and machine learning), the author decided to conduct this project with machine learning as he believes it has more promising potential and better aligns with his personal research interests.

3.1 Objectives

Currently, there have been two solutions designed by research fellows and students within Monash University. The first solution uses dynamic time warping (DTW) to temporally align signals for the decoding process and is completed in MATLAB. The second solution takes a machine learning approach and consists of a PyTorch neural network coded in Python. Both solutions utilise MATLAB to conduct the pre-processing and noise generation of basecalled sequences.

The differing platforms prevents performance comparisons from being drawn between both solutions and the potential to integrate both solutions together in the pipeline. Because the objectives are mathematically complex, MATLAB remains the preferable platform for its superior mathematical computation features.

While MATLAB includes a native deep learning toolbox, it is not an industry standard and remained unexplored by the project research team. Thus, this project aims to build a machine learning model natively within MATLAB such that the entire solution (pre and post-processing included) remains integrated within one platform. The model design looks to improve on the existing approach completed in PyTorch with an emphasis on minimising training time.

3.2 Overview

The design of the project is split into two components:

Generate Dataset

Noise is added to a sequence according to the noise variance from the basecaller that simulates the reading from a nanopore. This process is iterated until a predefined number of sequences have been generated, thus forming the dataset. This dataset is split into training, validation and testing data to be fed into the neural network.

Neural Network

A neural network is trained on the generated dataset. Its structure and parameters are designed and optimised for the application. Testing data is fed to the neural network and its performance is evaluated based on the accuracy at which it correctly decodes a noisy signal.

3.3 Squiggle Prediction with Basecaller

A *squiggle* refers to the raw electrical current signal output of the nanopore which corresponds to the sequence of DNA bases that have been passed through due to their varying electrical resistance. Basecalling refers to the process of translating the raw squiggle signal into a DNA sequence [26][27].

Basecalling is a challenging task as the electrical reading from a pore is representative of its *state*, which consists of multiple bases at a time. Additionally, the squiggle is noisy and stochastic as its measurement at any given point in time depends solely on single molecules [26].

ONT currently has multiple basecalling programs, including Albacore, Bonito, Flappie, Guppy and Scrappie. This project focuses on Scrappie for its open-source nature and familiarity within the DNA storage research team at Monash University.

Scrappie's basecalling functionality is not used in this project, as a bespoke basecaller would be implemented from scratch within MATLAB to compare with existing bespoke basecaller designs. Instead, Scrappie would be used to predict squiggles from a codebook - i.e. simulating a current read from a Nanopore if DNA encoded with sequences from the codebook were to be passed through it.

A set of DNA alphabets (codebook) of varying sizes, 16, 64 and 256 were graciously provided by the research team as a starting point. Each alphabet is made up of 12 nucleotides and is flanked by spacers of 6 nucleotides in size to separate the individual alphabets. Scrappie was used to predict a squiggle for each of these DNA alphabets.

3.4 Generate Dataset

3.4.1 Read Squiggle

Scrappie outputs its squiggle prediction in a table. It contains 5 columns:

1. `pos` - the index of the position along the reference
2. `base` - the base (A, T, C or G) of that state
3. `current` - the normalised current value of that state
4. `sd` - the standard deviation of the normalised current value
5. `dwell` - the number of samples

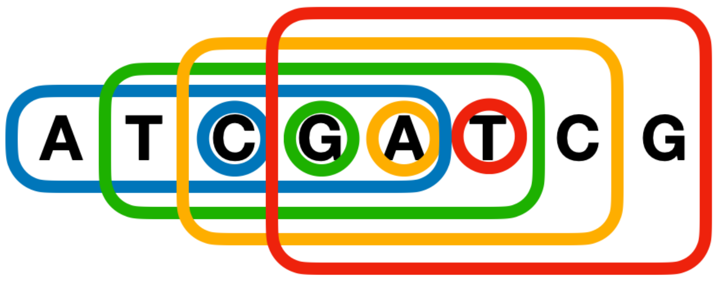
Here is an excerpt of what a typical Scrappie squiggle table would look like:

```
#Seq[1]
pos  base   current        sd      dwell
5      T      1.351073  0.182453  7.043665
6      G      0.470325  0.186138  7.385700
7      C      0.393505  0.144656  7.811590
8      G      -0.176812 0.141602  7.573686
9      T      0.632870  0.163185  6.795895
```

The table is imported into MATLAB and split into 4 arrays - base, current, standard deviation and dwell. Each sequence is indexed with its own row and its columns consist of its contents indexed by the position along reference.

3.4.2 Concatenate States

Because Scrappie's squiggle prediction consists only of mean values within a state, values from previous and future states are concatenated to return the true value. This step expands the squiggle into a measurement with true number of samples as if observed in real-time from a nanopore. The concatenated bases form an alphabet and is the label of the squiggle.



Scrappie Output

Base	C	G	A	T
Current	a	b	c	d

Concatenated Vector

Base	ATCGA	TCGAT	CGATC	GATCG
Current	aaaaa	bbbbb	ccccc	ddddd

Figure 3.1: Concatenation/expansion of states

3.4.3 Generate noise

Depending on the desired size of the dataset, noise typically observed from a nanopore reading is simulated from noise variances provided by the squiggle prediction.

Number of samples per state, N is a geometrically distributed random variable where its probability parameter P is determined by $P = \frac{1}{\text{dwell}}$ with its noise variance (standard deviation) provided by the squiggle prediction.

Thus, the parameters responsible for noise are dwell time to determine the stretch noise and standard deviation to determine the amplitude noise.

Because stretch noise acts temporally, the noisy squiggles now have inconsistent sequence lengths (number of samples). A neural network requires its inputs to be of consistent dimension, hence the noisy squiggles are resampled to a target sequence length of 150 samples. The target sequence length was chosen by finding the mean across a large batch of generated noisy signals and rounding to 10.

A clean (noise-free) version of the squiggle is also kept to visualise the added noise. If the clean squiggle were to be resampled to the same target sequence length of 150, high frequency noise artifacts would be observed due to the large transients in the stairstep signal. This was minimised by filtering and is used solely for visualisation purposes with no effect to the performance of the neural network.

3.4.4 Output

Two cells of size (number of alphabets $\times$ number of noisy squiggles generated per alphabet) by 1 are generated to be passed onto the neural network. The first cell contains the noisy squiggles and the second cell contains their corresponding labels. The noisy squiggle sequences are stored in arrays within the cell. The labels are in discrete, categorical form as there are only a finite number of labels per set of alphabets.

For example, if there are 256 alphabets in the set and 1,000 noisy squiggles were to be generated per alphabet:

- There would be 256 labels.
- The cells would be of size 256,000 × 1.
- Each entry within the noisy squiggle cell would contain an array of size 1 × 150 with data type *double*.

3.5 Neural Network

3.5.1 Previous Attempts

Initially, base-wise decoding of trying to return the noisy time-series squiggle to a noise-free, ideal stairstep signal was attempted. A windowing algorithm not based on machine learning was written to decode the noise-reduced signal into its respective bases. Performance of this network was below requirements.

A common neural network approach used in natural language processing is to pair a CNN with an RNN, before finishing with a fully connected layer. The CNN helps the RNN with feature extraction. However, extensive testing found that the performance benefit of having a convolutional block could not be justified by the computational cost in increased training time.

3.5.2 Sequence Input

To reduce the effects of algorithmic bias and allow the network to perform better unexpected stress, 15% of the dataset are generated based on artificially augmented noise variance with a coefficient between 0.8 and 1.5. This means that noise present within that 15% portion of data are the noise levels predicted by Scrappie, randomly scaled by 80% to 150%.

The scaled and unscaled dataset are combined and randomly split into training, validation and testing sets. Training and validation data make up 70% and 30% of the combined (scaled and unscaled) dataset respectively. Testing data is generated separately as a 20% proportion of the total dataset and remains unscaled.

The sequence input layer then inputs the sequence training dataset into the network. The training dataset is shuffled at every epoch to ensure the network does not learn the sequence of the training dataset.

3.5.3 Bidirectional LSTM

A bidirectional long-short-term-memory (BiLSTM) layer is a type of recurrent neural network (RNN) that remembers sequence data and makes its predictions/classifications with data patterns. BiLSTM layers differ from standard unidirectional LSTM layers such that their input flows in both directions, thereby preserving memory for past and future information.

BiLSTM layers were chosen for their ability to utilise information from past and future squiggle readings as context to decode the alphabet. This context forms part of the codeword that is being decoded. Text prediction is a common application of BiLSTM layers and shares a common concept with the task in this project. The BiLSTM layer learns context present in training text in both directions, and utilises that context to make predictions accordingly. The same approach is applied to learning and classifying squiggles.

Three BiLSTM layers, all with 150 hidden units are stacked such that the first layer takes its input from the sequence input, the second layer takes its input from the first layer and the third (last) layer takes its input from the second layer. The number of hidden units correspond to the complexity of the model have been optimised to fit the dataset of size 10,000 samples per alphabet.

3.5.4 Fully Connected

A fully connected (FC) layer connects all inputs from one layer to the every activation unit of the next layer. An FC layer is used here after the BiLSTM block to compile all extracted data from the BiLSTM layers to form the final output. The output size of the fully connected layer corresponds to the number of classes/alphabets, in this case 16.

3.5.5 Softmax Function

The softmax function is modelled after the *sigmoid* function used for logistic regresion. The softmax function is used to scale all scores to a normalised probability distribution for output and analysis. Its mathematical definition is as follows:

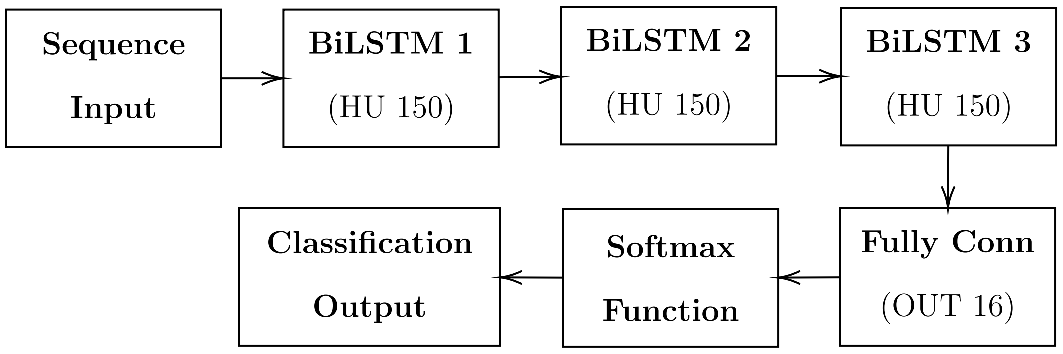
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

It is used as the penultimate layer for multi-class classification applications. Hence, a softmax layer is used here before MATLAB's classification output layer.

3.5.6 Classification Output

The classification layer is the final layer in the neural network and performs classification by computing the cross-entropy loss.

3.5.7 Structure



3.5.8 Hyperparameters

Research conducted on LSTM hyperparameters conclude that the two most crucial hyperparameters are its learning rate and network size (number of hidden layers) [42][43].

To maximise training efficiency, a decaying learning rate is used to improve learning as the network converges. Initially, the learning rate is set to 1e-2 and for every epoch after, it drops by a factor of 0.5.

Every BiLSTM layer has 150 hidden layers. This depth delivered the best fit to the training data from testing. The training data is split into mini batches of size 256, which delivered the best balance between accuracy and training time from testing.

Chapter 4

Results

4.1 Testing Hardware

Development and testing was performed on a desktop personal computer with one six core CPU (AMD Ryzen 5 1600) and 16 GB of DDR4 system memory running the R2021b release of MATLAB.

The installed GPU is an AMD Radeon RX 570 which is currently not supported by MATLAB for hardware acceleration.

4.2 Hardware Resource Limitations

Hardware resource management remained a large limitation of this project. On top of implementing the network natively within MATLAB, this project aimed to improve training efficiency compared to the neural network previously implemented by the research team in Pytorch.

However, training time metrics cannot be directly compared between both implementations.

1. The previous neural network was trained with cloud-based GPU compute clusters while this neural network was CPU-trained. GPU training is more computationally efficient and decreases training time [46].

2. When observing CPU resource utilisation, the average system CPU usage is between 30 - 35% during training. This indicates that MATLAB has not been optimised to fully utilise available CPU resources, leaving additional performance on the table.

3. Previous researchers had noted that using a large dataset of around 50,000 samples per alphabet was the preferred approach, given the ability to theoretically generate an infinite dataset. The hardware used to develop this project did not contain enough memory to load a dataset of that size. Additionally, because CPU training was used instead of GPU training, the intermediates (weights and calculations during training) cannot be offloaded onto GPU memory. Thus, the architecture was optimised for smaller datasets passed through a larger number of times and might not benefit from the ability to infinitely expose the network to new training data.

The table below shows hardware resource usages during idle and training. While a dataset of 256 alphabets were used for the previous neural network, no attempt of training with 256 alphabets with at least 5000 samples per alphabet has succeeded thus far, with MATLAB crashing before completion due to memory exhaustion. A dataset below size 5000 provides insufficient information to successfully train the network. Training was conducted for dataset of size 15,000 per alphabet for 16 alphabets.

	Idle	**Training**
CPU Usage	0%	30 - 35%
Memory Usage	38% (6.0 GB)	69% (11.1 GB)

Table 4.1: Hardware resource usage during idle and training

4.3 Network Performance

Due to system memory limits, the network was trained on 16 alphabets. For a dataset size of 15,000 samples per alphabet, training was completed in 8 minutes and 19 seconds. The network achieved a success rate (testing accuracy) of 94.20% and a validation accuracy of 94.22%.

The size of the generated dataset was the primary independent variable for the accuracy of the network. The following table compares network performance across different dataset sizes, with other variables optimised for that dataset to maximise training efficiency. Note that dataset size refers to number of samples per alphabet.

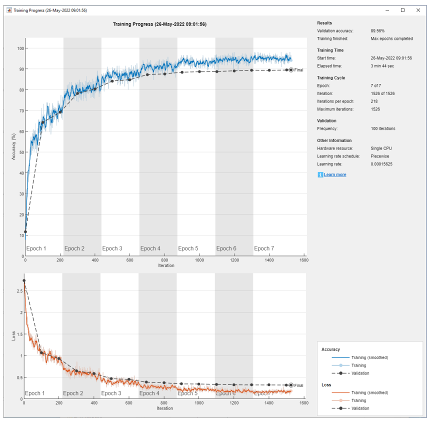
Dataset Size	**5,000**	**10,000**	**15,000**
Number of Epochs	7	6	5
Learning Rate Drop Period	1 epoch	1 epoch	1 epoch
Training Time	3 min 44 sec	5 min 31 sec	8 min 19 sec
Network Validation Accuracy	89.56%	93.09%	94.22%

Table 4.2: Effect of dataset size on network performance

The following plots visualise their respective training graphs.

For the two smaller dataset sizes of 5,000 and 10,000 samples per alphabet, the validation data (gray circles on plot) starts to detach from the training accuracy and loss as training continues. This is an indication that the network is approaching overfitting. If more epochs were to be trained, it is likely the validation accuracy would decrease as the validation loss increases.

Overfitting occurs when the network is too complex for the training dataset and it starts to pick up noise and details that negatively impact its ability to classify new data outside of its training dataset (validation and testing dataset) [42]. In this instance, dataset sizes of 5,000 and 10,000 samples represent insufficient information to make up the complexity needed by the network.



Results
Validation accuracy: 89.56%
Training finished: Max epochs completed

Training Time
Start time: 26-May-2022 09:01:56
Elapsed time: 3 min 44 sec

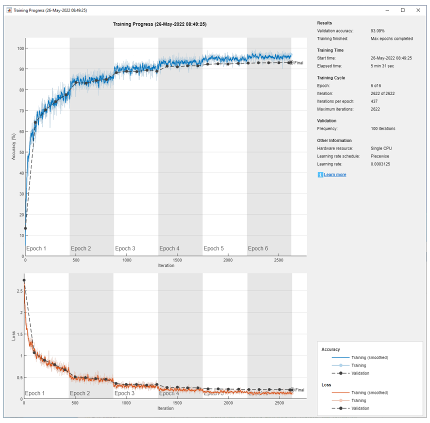
Training Cycle
Epoch: 7 of 7
Iteration: 1526 of 1526
Iterations per epoch: 218
Maximum iterations: 1526

Validation
Frequency: 100 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Piecewise
Learning rate: 0.00015625

Learn more

(a) 5,000 samples per alphabet



Results
Validation accuracy: 93.09%
Training finished: Max epochs completed

Training Time
Start time: 26-May-2022 08:49:25
Elapsed time: 5 min 31 sec

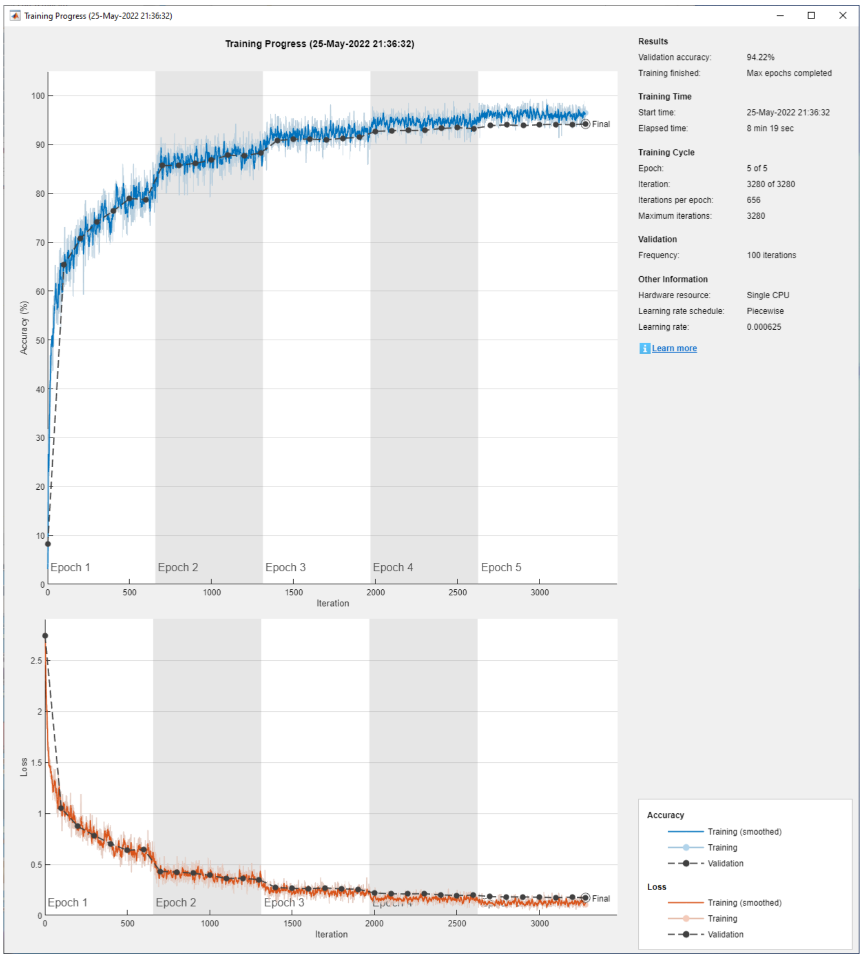
Training Cycle
Epoch: 6 of 6
Iteration: 2622 of 2622
Iterations per epoch: 437
Maximum iterations: 2622

Validation
Frequency: 100 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Piecewise
Learning rate: 0.0003125

Learn more

(b) 10,000 samples per alphabet



Results
Validation accuracy: 94.22%
Training finished: Max epochs completed

Training Time
Start time: 25-May-2022 21:36:32
Elapsed time: 8 min 19 sec

Training Cycle
Epoch: 5 of 5
Iteration: 3280 of 3280
Iterations per epoch: 656
Maximum iterations: 3280

Validation
Frequency: 100 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Piecewise
Learning rate: 0.000625

Learn more

(c) 15,000 samples per alphabet

Figure 4.1: Network training for different dataset sizes

4.4 Comparisons

Fair comparisons to other existing implementations cannot be drawn because of insufficient fixed variables.

- **ONT Basecallers**

  This project relies on Scrappie's squiggle feature to predict squiggles based on a codebook instead of taking a true nanopore measurement. Decoding this squiggle with ONT's basecallers would not form a fair comparison as the squiggle was generated by the same models used to decode it.

- **Nanocall**

  Nanocall was tested on a different dataset thus a direct comparison would not be fair. System memory limitations prevent the (very large) dataset used to test Nanocall to be loaded onto this implementation.

- **NanoReviser**

  NanoReviser performs error correction on an already basecalled sequence. It was trained and tested on a different dataset with different hardware thus a direct comparison would not be fair. System memory limitations prevent the (very large) dataset used to train NanoReviser to be loaded onto this implementation.

- **Monash University Summer Research Program NN**

  This neural network was developed by Tin Nhut Nguyen in 2021 during Monash University's Summer Research Program (SRP) and is the closest comparison that can be drawn to this project. It was trained on the alphabets of identical formats but used a larger codebook containing 256 alphabets instead. This codebook of 256 alphabets would take longer to train and hence training time cannot be compared.

- **Monash University Research Team NN**

  This neural network was developed by the research team at Monash University under supervision of Prof. Emanuele and is the closest comparison that can be drawn to this project. It was trained on the alphabets of identical formats but used a larger codebook containing 256 alphabets instead. This codebook of 256 alphabets would take longer to train and hence training time cannot be compared.

Assuming all neural networks were optimised for the respective complexity and size of their dataset and with the understanding that these comparisons are fundamentally *flawed*, this table summarises the average success rate of the above discussed implementations.

Implementation	**Accuracy**	**Implementation**	**Accuracy**
Nanocall	68%	NanoReviser	84.36%
Albacore	78.86%	Monash SRP	70%
Scrappie (events)	82.14%	Monash Research	96.5%
This project	**94%**	**This project**	**94%**

Table 4.3: Success rate comparison of discussed implementations

Chapter 5

Discussion

Despite the hardware limitations encountered when developing and testing this project, its main objective of implementing a neural network-based nanopore basecaller natively within MATLAB was achieved.

Assuming the neural network training time scales relatively evenly with the number of alphabets present in the neural network, this network would take $\sim$ 150 minutes to train the complete 256 alphabet codebook.

$$\frac{\text{Target number of alphabets}}{\text{Current number of alphabets}} = \frac{256}{16} = 16$$

$$8.32 \text{ min} \times 16 = 133.12 \text{ min}$$

This would be lower than the $\sim$6 hours (360 minutes) taken to train the existing Monash University research team neural network. Success rates from both networks are comparable.

Most importantly, this project now forms the base to natively carry over any other neural networks developed by the research team at Monash University (and the general public) using other industry standard libraries such as PyTorch and Tensorflow, just by adapting the neural network structure within the completed MATLAB code.

Special care has been taken to develop this project in as much of a *plug and play* manner as possible by minimising the number of required inputs and automating many parameters. The code only requires the Scrappie generated squiggle and neural network structure alongside its hyperparameters as user input. Additionally, MATLAB's deep learning toolbox, while not as powerful as other ML libraries, is simple enough to allow these changes to be made easily.

While efforts were made to maximise training efficiency and network accuracy, there is still room for improvement with different network architectures and more hyperparameter optimisation. Access to more powerful hardware would allow for less limitation on the dataset size and more flexibility in choosing hyperparameters. When monitoring hardware resource use during training, it is clear that training a neural network (on CPU) within MATLAB is insufficiently optimised and may lead to inferior performance compared to other ML libraries.

Transfer learning allows a pre-trained dataset to be adapted, with training, to learn a different but similar dataset. This may be a useful approach as a generic pre-trained network that learns noise patterns exhibited by the nanopore could be further trained for the specific data application to maximise accuracy with reduced training time. However, at the point of writing, MATLAB only allows transfer learning to be performed on a list of common pretrained networks (Alexnet, Googlenet, Resnet, etc.) and not networks built and trained from scratch. More information can be found on MATLAB's transfer learning documentation at `https://www.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html`.

Chapter 6

Conclusion

In this project, a respectably performing neural network based nanopore basecaller was developed natively using MATLAB. After conducting a literature review on DNA storage technology and neural network architectures, a neural network was developed specifically to classify squiggles to their respective DNA nucleotide sequences. The pre-processing of the dataset and hyperparameters of the neural network were optimised to maximise training efficiency, that is minimise training time and maximise network accuracy.
Two objectives were set out for this project:

1. Develop a neural network based basecaller natively within MATLAB.

2. Improve upon training efficiency of existing implementations.

Both objectives were met successfully, with the network completing its training for 16 alphabets in just 8 minutes and 19 seconds while achieving a success rate of 94.20%.

6.1 Future Work

While the BiLSTMs implemented in this project have performed sufficiently well, future researchers can look to implement other modern time-series based neural network architectures. The following are two examples that have the potential to deliver promising performance improvements:

Attention

Attention is a state-of-the-art model mostly used in the field of natural language processing, which as discussed before, shares many similarities with the signal processing required in this project. Attention aims to mimic cognitive attention by devoting the focus of the neural network non-linearly to the most important parts of the data. It can be implemented as an extension of the neural network in this project [44].

Connectionist Temporal Classification (CTC)

Connectionist temporal classification (CTC) is mainly used for handwriting recognition. The CTC loss function takes the output of an existing neural network and the corresponding ground truth and attempts to align the sequence with every possible ground truth [45]. CTCs were implemented in some of ONT's latest basecallers which is indicative of its potential. It too can be implemented as an extension of the neural network in this project.

6.2 Links

Code repository:

`https://github.com/ryanwee00/FYP-DNA-Storage-Machine-Learning`

Presentation video:

`https://youtu.be/kSoiwSd_89o`

References

[1] B. Alberts, A. Johnson, and J. Lewis, "The Structure and Function of DNA," *Molecular Biology of the Cell,* 2002. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK26821/.

[2] MedlinePlus. "What is DNA." U.S. National Library of Medicine (NIH). https://medlineplus.gov/genetics/understanding/basics/dna/ (accessed 10th August 2021).

[3] "DNA Strands." AncestryDNA Learning Hub. https://www.ancestry.com/lp/double-helix/dna-strands (accessed 10th August 2021).

[4] R. F. Service. "DNA could store all of the world's data in one room." American Association for the Advancement of Science (AAAS). https://www.sciencemag.org/news/2017/03/dna-could-store-all-worlds-data-one-room (accessed 11th August 2021).

[5] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark, "Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes," *Angewandte Chemie International Edition,* vol. 54, no. 8, pp. 2552-2555, 2015, doi: https://doi.org/10.1002/anie.201411378.

[6] D. Panda, K. A. Molla, M. J. Baig, A. Swain, D. Behera, and M. Dash, "DNA as a digital information storage device: hope or hype?," (in eng), *3 Biotech,* vol. 8, no. 5, pp. 239-239, 2018, doi: 10.1007/s13205-018-1246-7.

[7] Y. Erlich and D. Zielinski, "DNA fountain enables a robust and efficient storage architecture," Department of Computer Science, Columbia University, Fu Foundation School of Engineering, 2017.

[8] M. Meyer and M. Kircher, "Illumina Sequencing Library Preparation for Highly Multiplexed Target Capture and Sequencing," *Cold Spring Harbor protocols,* vol. 2010, p. pdb.prot5448, 06/01 2010, doi: 10.1101/pdb.prot5448.

[9] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A DNA-Based Archival Storage System," presented at the Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, Atlanta, Georgia, USA, 2016. [Online]. Available: https://doi.org/10.1145/2872362.2872397.

[10] L. Ceze, J. Nivala, and K. Strauss, "Molecular digital data storage using

DNA," *Nature Reviews Genetics,* vol. 20, no. 8, pp. 456-466, 2019/08/01 2019, doi: 10.1038/s41576-019-0125-3.

[11] D. Deamer, M. Akeson, and D. Branton, "Three decades of nanopore sequencing," *Nature Biotechnology,* vol. 34, no. 5, pp. 518-524, 2016/05/01 2016, doi: 10.1038/nbt.3423.

[12] L. Organick *et al.*, "Random access in large-scale DNA data storage," *Nature Biotechnology,* vol. 36, no. 3, pp. 242-248, 2018/03/01 2018, doi: 10.1038/nbt.4079.

[13] D. Branton *et al.*, "The potential and challenges of nanopore sequencing," (in eng), *Nature biotechnology,* vol. 26, no. 10, pp. 1146-1153, 2008, doi: 10.1038/nbt.1495.

[14] "DNA-Based Digital Storage," ed: [White Paper] Twist Bioscience, n.d.

[15] M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community," (in eng), *Genome Biol,* vol. 17, no. 1, pp. 239-239, 2016, doi: 10.1186/s13059-016-1103-0.

[16] Techopedia. "Direct Access Storage Device (DASD)." https://www.techopedia.com/definition/6728/direct-access-storage-device-dasd (accessed.

[17] S. Chanayai and A. Apavatjrut, "Fountain Codes and Their Applications: Comparison and Implementation for Wireless Applications," *Wireless Personal Communications,* vol. 121, no. 3, pp. 1979-1994, 2021/12/01 2021, doi: 10.1007/s11277-021-08749-w.

[18] A. Shokrollahi, "Fountain codes," *Iee Proceedings-communications - IEE PROC-COMMUN,* vol. 152, 01/01 2005.

[19] National Human Genome Research Institute. "DNA Sequencing Fact Sheet." https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Fact-Sheet (accessed.

[20] J. Shendure and H. Ji, "Next-generation DNA sequencing," *Nature Biotechnology,* vol. 26, no. 10, pp. 1135-1145, 2008/10/01 2008, doi: 10.1038/nbt1486.

[21] R. S. Daniels *et al.*, "A Sanger sequencing protocol for SARS-CoV-2 S-gene," *Influenza and Other Respiratory Viruses,* vol. 15, no. 6, pp. 707-710, 2021, doi: https://doi.org/10.1111/irv.12892.

[22] Merck KGaA. "Sanger Sequencing Steps and Method." Sigma Aldrich. https://www.sigmaaldrich.com/AU/en/technical-documents/protocol/genomics/sequencing/sanger-sequencing (accessed.

[23] J. Read and S. Brenner, "DNA Synthesis," in *Encyclopedia of Genetics*, S. Brenner and J. H. Miller Eds. New York: Academic Press, 2001, p. 577.

[24] J. Louhelainen, "DNA Is No Longer Just ACGT: Synthetic DNA Is Here To Stay," University of Helsinki, 2018. [Online]. Available: https://www.global-engage.com/life-science/dna-is-not-just-acgt-synthetic-dna-is-here/

[25] J. Yang *et al.*, "Solid-Phase Synthesis of Phosphorothioate Oligonucleotides Using Sulfurization Byproducts for in Situ Capping," *The Journal of Organic Chemistry,* vol. 83, no. 19, pp. 11577-11585, 2018/10/05 2018, doi: 10.1021/acs.joc.8b01553.

[26] R. R. Wick, L. M. Judd, and K. E. Holt, "Performance of neural network basecalling tools for Oxford Nanopore sequencing," *Genome Biol,* vol. 20, no. 1, p. 129, 2019/06/24 2019, doi: 10.1186/s13059-019-1727-y.

[27] Y.-z. Zhang *et al.*, "Nanopore basecalling from a perspective of instance segmentation," *BMC Bioinformatics,* vol. 21, no. 3, p. 136, 2020/04/23 2020, doi: 10.1186/s12859-020-3459-0.

[28] M. David, L. J. Dursi, D. Yao, P. C. Boutros, and J. T. Simpson, "Nanocall: an open source basecaller for Oxford Nanopore sequencing data," *Bioinformatics,* vol. 33, no. 1, pp. 49-55, 2016, doi: 10.1093/bioinformatics/btw569.

[29] L. Wang, L. Qu, L. Yang, Y. Wang, and H. Zhu, "NanoReviser: An Error-correction Tool for Nanopore Sequencing Based on a Deep Learning Algorithm," *bioRxiv,* p. 2020.07.25.220855, 2020, doi: 10.1101/2020.07.25.220855.

[30] T. Mitchell, *Machine Learning.* McGraw Hill, 1997.

[31] IBM Cloud Education. "Machine Learning." https://www.ibm.com/cloud/

learn/machine-learning (accessed.

[32] UC Berkeley School of Information, "What is Machine Learning (ML)?," 2020. [Online]. Available: https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/.

[33] M. van Otterlo and M. Wiering, "Reinforcement Learning and Markov Decision Processes," in *Reinforcement Learning: State-of-the-Art*, M. Wiering and M. van Otterlo Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3-42.

[34] S. Walczak and N. Cerpa, "Artificial Neural Networks," in *Encyclopedia of Physical Science and Technology (Third Edition)*, R. A. Meyers Ed. New York: Academic Press, 2003, pp. 631-645.

[35] IBM Cloud Education, "Neural Networks," 2020. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks.

[36] K. J. Max Kuhn, *Applied Predictive Modelling*, 1 ed. New York: Springer New York, 2013.

[37] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[38] Y. Li, Y. Fu, H. Li, and S. Zhang, "The Improved Training Algorithm of Back Propagation Neural Network with Self-adaptive Learning Rate," in *2009 International Conference on Computational Intelligence and Natural Computing*, 6-7 June 2009 2009, vol. 1, pp. 73-76, doi: 10.1109/CINC.2009.111.

[39] R. Reed and R. J. Marks, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*: The MIT Press, 1999. [Online]. Available: https://doi.org/10.7551/mitpress/4937.001.0001.

[40] J. Brownlee. "How to Control the Stability of Training Neural Networks With the Batch Size." Machine Learning Mastery. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ (accessed.

[41] A. Twin, M. James, and K. R. Schmitt. "Overfitting." Investopedia. https://www.investopedia.com/terms/o/overfitting.asp (accessed.

[42] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 28, no. 10, pp. 2222-2232, 2017, doi: 10.1109/TNNLS.2016.2582924.

[43] T. M. Breuel, "Benchmarking of LSTM Networks," *ArXiv,* vol. abs/ 1508.02774, 2015.

[44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *ArXiv,* vol. 1409, 09/01 2014.

[45] H. Scheidl, "An Intuitive Explanation of Connectionist Temporal Classification," *Towards Data Science,* 2018. [Online]. Available: https:// towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c.

[46] J. Dsouza, "What is a GPU and do you need one in Deep Learning," *Towards Data Science,* 2020. [Online]. Available: https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d.

[47] G. Tomasi, F. Berg, and C. Andersson, "Correlation Optimized Warping and Dynamic Time Warping as Preprocessing Methods for Chromatographic Data," *Journal of Chemometrics,* vol. 18, pp. 231-241, 05/01 2004, doi: 10.1002/cem.859.

[48] H. Kamper. Dynamic time warping. Available: https://www.youtube.com/ channel/UCBu4J-JIs-UORp5pQ6M48nw.

[49] T. Körting. How DTW (Dynamic Time Warping) algorithm works. Available: https://www.youtube.com/watch?v=_K1OsqCicBY.

[50] MathWorks Help Center. dtw [Online] Available: https:// www.mathworks.com/help/signal/ref/dtw.html

[51] K. Kumar, "Chemometric assisted correlation optimized warping of chromatograms: optimizing the computational time for correcting the drifts in chromatographic peak positions," *Analytical Methods,* 10.1039/C8AY00084K vol. 10, no. 9, pp. 1006-1014, 2018, doi: 10.1039/C8AY00084K.

[52] D. Zhang, X. Huang, F. E. Regnier, and M. Zhang, "Two-Dimensional Correlation Optimized Warping Algorithm for Aligning GCxGC-MS Data,"

Department of Statistics, Department of Chemistry, Bindley Bioscience Centre, Purdue University, West Lafayette, Indiana, United States, n.d.

[53] S. A. Etemad and A. Arya, "Correlation-optimized time warping for motion," *The Visual Computer,* vol. 31, no. 12, pp. 1569-1586, 2015/12/01 2015, doi: 10.1007/s00371-014-1034-2.