Confirming DNA Replication Origins of Saccharomyces Cerevisiae A Deep Learning Approach

Abdelrahman Hosny
Graduate Student, Master's
Computer Science
University of Connecticut
Email: abdelrahman@engr.uconn.edu

Anthony Parziale
Undergraduate Student, Junior
Computer Science
University of Connecticut
Email: anthony.parziale@uconn.edu

***Abstract*—In the past, the study of medicine used to focus on observing biological processes that take place in organisms, and based on these observations, biologists would make conclusions that translate into a better understanding of how organisms systems work. Recently, the approach has changed to a computational paradigm, where scientists try to model these biological processes as mathematical equations or statistical models. In this study, we have modeled an important activity of cell replication in a type of bacteria genome using different deep learning network models. Results from this research suggest that deep learning models have the potential to learn representations of DNA sequences, hence predicting cell behavior. Source code is available under MIT license at: http://abdelrahmanhosny.github.io/DL-Cerevesiae/**

1. Introduction

Cells are the fundamental working units of every living system. They are the smallest units that can function independently. Rather than having brains, cells instead make decisions through complex networks of chemical reactions, called pathways. The reactions result in either synthesizing new material, breaking other material down, signaling to eat another protein, or signaling to die. A cell stores all the information it needs to function properly as well as the information needed to replicate. As shown in figure 1, a cell consists of a nucleus and cytoplasm.

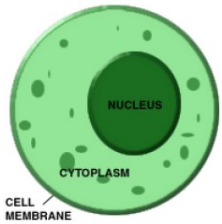

Figure 1: Cell Structure

The most important piece of information a cell stores in its nucleus is the Deoxyribonucleic Acid (DNA) sequence. This is also called the genome sequence. Every cell in an organism contains the same DNA sequence as it uniquely defines each organisms characteristics. The DNA carries most of the genetic instructions used in growth, development, functioning and reproduction of all known living organisms. The structure of the DNA, as shown in figure 2, is a double helix that is connected by base pairs of molecules, called nucleotides. A typical DNA sequence has the four well-known base pairs: *Adenine (A)*, *Cytosine (C)*, *Guanine (G)* and *Thymine (T)*. Adenine complements Thymine and Cytosine complements Guanine. Therefore, given one strand, we can infer the other complementary strand. In a computational domain, we deal with only one strand of the genome.

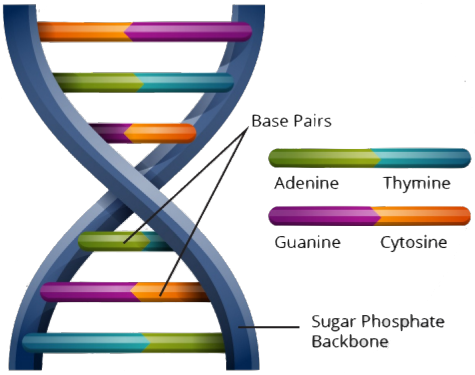

Figure 2: DNA Structure

One of the most important biological processes before a cell divides, for growth or reproduction, is *cell replication*. The first step in cell replication is replicating the DNA inside the nucleus. A replication fork sparks this replication processes by breaking the tight link between the two strands by breaking the connected base pairs. Then each strand synthesizes the complementary strand resulting in four strands, each couple form the new DNA sequence. After that, the cell divides forming two identical cells. Figure 3 shows both DNA replication and cell division processes. The importance of DNA replication process comes from the observation that during replication, bases in complementary strands might

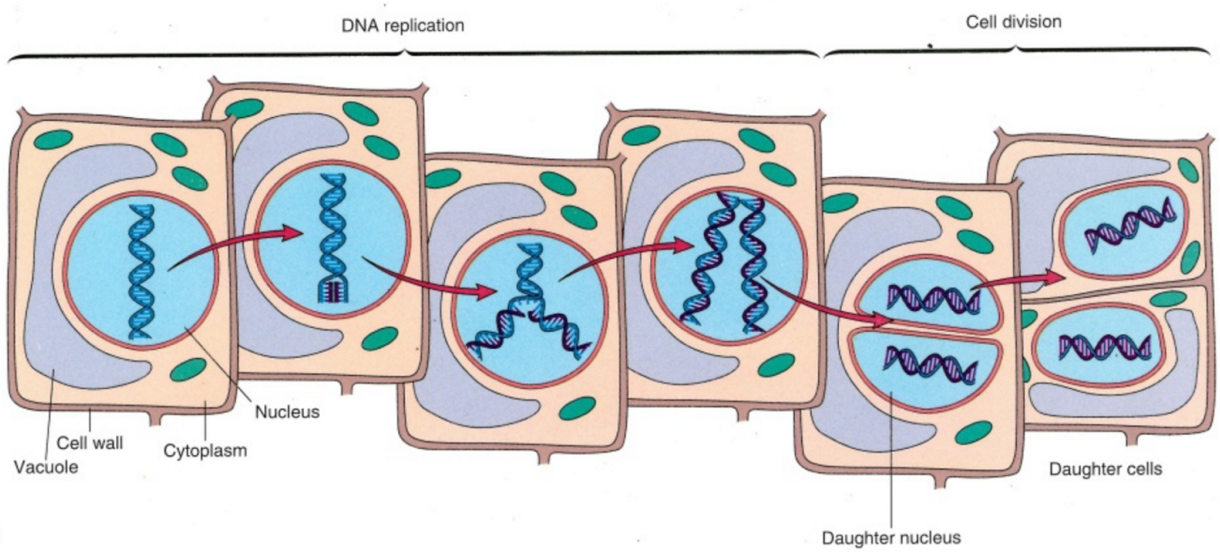

Figure 3: Cell Replication and Cell Division Processes

be mutated while being synthesized. This mutation is what causes a new cell to not function properly causing diseases.

2. Problem

The question now is: ***where in the genome does DNA replication start?*** In other words, ***where does the replication fork breaks the connected base pairs (figure 4)?*** There are two dominant approaches that biologists have followed to tackle this problem. The first one is by carrying out experiments on cells that cut a specific piece of the genome of the DNA and test if the cell can replicate itself or not. If the cell can replicate itself, that means the piece that was cut does not represent an origin of replication as the replication fork could hook to the replication origin and start the replication process. If the cell couldn't replicate itself, that means the piece cut belongs to a replication origin. Continuing on the same manner, biologists can determine the origin of replication in a specific genome. It is obvious that this approach is time-consuming and might not be accurate at the end due to large human errors. Another approach is observing the replication processes and doing some statistics on the number of base pairs being synthesized to detect skews. Although this approach proves to be successful in detecting *single* replication origins, it skew diagrams translates to no meaning in cells that has multiple origin of replications. So, the previous experimental approach should be followed. Apparently, naive mathematical models fail to compute the origin of replication in multiple-replication-origins cells. Therefore, we have proposed and tested more sophisticated mathematical paradigms inspired from deep neural network models.

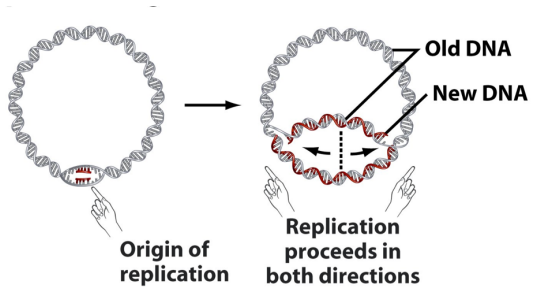

Figure 4: DNA Replication Fork

We chose to test our approach on Saccharomyces Cerevisiae genome – a species of yeast. We chose this species for its relatively simple structure and the publicly available dataset that defines different confirmed and unconfirmed replication origins. In the next subsections, we describe the data and discuss the problem and the research approach we followed.

2.1. Data Description

To the best of our knowledge, we are the first to train deep learning models on Saccharomyces Cerevisiae genome. Therefore, we had to manually pull the data from different sources and preprocess it in a way suitable for the model to be used. Cerevisiae genome consists of 16 chromosomes with a total of 12071326 base pairs. A chromosome is a subsequence of the whole genome. Connected together, the 16 chromosomes form the whole genome sequence. Cerevisiae chromosomes data can be downloaded from Yeast Genome Database [2]. Records of the replication origins of Cerevisiae can be obtained from the DNA replication origin database [3]. Replication origins have three statuses: *confirmed*, *likely* or *dubious*. Confirmed status means the origin of replication has been detected and confirmed by 2D gel analysis (one of the experimental approaches). Likely status

Chromosome	1	2	3	4	5	6	7	8
length	230218 bp	813184 bp	316620 bp	1531933 bp	576874 bp	270161 bp	1090940 bp	562643 bp
origins confirmed	14	38	21	51	22	17	30	21
average length of origins	806 bp	548 bp	772 bp	430 bp	416 bp	490 bp	345 bp	446 bp
Chromosome	9	10	11	12	13	14	15	16
length	439888 bp	745751 bp	666816 bp	1078177 bp	924431 bp	784333 bp	1091291 bp	948066 bp
origins confirmed	15	29	21	32	27	21	27	25
average length of origins	324 bp	502 bp	541 bp	346 bp	333 bp	306 bp	358 bp	458 bp

TABLE 1: Statistics about Saccharomyces Cerevisiae genome (bp = base pair).
Full Statistics file can be viewed at our GitHub repository [1].

means the origin has been identified by two microarray studies, but not yet confirmed. Dubious status means the origin has been identified by one microarray study. Table 1 shows some statistics obtained during the preprocessing phase to give us a better understanding of the size of the data we have. The preprocessing phase includes two stages: First, pulling and organizing data from the two websites and doing a sanity check on their quality (by checking that the origins pulled from one website already exists in the chromosomes pulled from the other website). Second, preprocessing the data to be suitable as input to the selected deep learning network.

2.2. Computational Problem Definition

We strongly believe that there is a long-term dependency between the base pairs of the genome that makes replication start at specific given locations, and not others. In this study, we are asking the question: ***can we train a deep neural network to model the whole genome of Cerevisiae, along with the origins of replication?*** The goal is to be able to *predict* new replication origin sites as well as *confirming* the likelihood of origins marked as likely or dubious in OriDB. We can formulate the problem computationally as follows:

Input:

1) 16 chromosomes sequences.
2) Replication origin sites (start location, end location) for each chromosome.

Output:

1) Prediction of new replication origins.
2) Likelihood scores of likely and dubious replication origins.

2.3. Research Approach

With recent success in the applications of deep learning models in a plethora of fields, we were inspired to apply our recently acquired knowledge in a previously unexplored domain. Inspired by the work done by Alipanahi et.al in using deep learning models to determine sequence specifities in DNA- and RNA-binding proteins [4], the approach we initially took as we began research was a purely inquisitive one. Our initial intention was to find a research paper in the area of Bioinformatics, specifically concerning DNA replication, and provide an implementation of the model used. From our empirical study of the field, we found little to no research in the area of applying deep learning models to finding/confirming the origin sites of DNA replication. We soon realized that instead of implementing the best model for this domain area that we could find, we should shift focus on actually determining which deep learning techniques work well for this task. Once we came to this realization, we were able to formalize the objective of our research. The objective of our research is to determine which deep learning architectures are utilizable in the study of DNA origins of replication. As we found in our literature survey, the world of deep learning can be split into two sub categories, discriminate and generative models. We decided to implement various models belonging to each group in an effort to hopefully give future researchers a beginning stepping stone for further research in this direction. As expected, this requires an intuitive approach on how to preprocess the data to be suitable for these types of models. Once the data was preprocessed, we trained various deep learning models in both discriminant and generative manners.

On the discriminant side, these models include a base Recurrent Neural Network (section 3.1.1), a Long Short Term Memory (section 3.1.2), as well as a Gated Recurrent Unit (section 3.1.3). As discriminant models are those that are trained for future prediction, the focus on these models had to meet our prediction goal defined in the previous subsection.

After testing various Discriminative models, we shifted focus to the generative side. Although we realized that these models might not be able to properly learn the data in a way we intended, we wanted to test the so-called "black-magic" of deep learning and really throw a curveball its way. The goal of generative architectures is to learn a representation of the distribution of some input data. This is generally performed in an unsupervised manner and utilized in dimensionality reduction/feature learning. We preprocessed the data accordingly to get a trainable model. Hidden layers learned in these generative models could be utilized in other areas as learned features for discriminant models.

The next section presents our experimental results of training these different set of models with their variations. Then, we follow it by a discussion on how we interpret the results and plan to continue working in the future.

3. Experimental Results

Tools used – Google's Tensorflow [5], Numpy package [6], BioPython package [7]. ***Hardware configuration*** – Personal laptops.

3.1. Discriminant Models

The goal of using discriminant models is to capture long-term dependencies between the base pairs of Cerevisiae genome. Using different variations of Recurrent Neural Networks, we were able to discover that these models are able to partially capture these dependencies and make predictions. Although not highly accurate, they prove that there is a dependency between the base pairs which can be modeled via more complicated methods.

To train these models, we preprocessed the data as follows:

- For a given chromosome chr, convert all bases to lower case.
- For each origin ori in chr, capitalize bases from start location to end location.
- Introduce a space between each base.

For example, for chr-x = $ACCTGCTGCATTCG$ and one origin of replication ori-x-1 = $CTGCA$, the resulting chromosome (that will serve as input to the model) is
chr-x = a c c t g C T G C A t t c g
The above is a simple example for illustration. For actual lengths and the number of replication origins in each chromosome, refer to table 1.

We trained a model for each input chromosome separately. After training, we sampled an equal length of sequence to the length of the input chromosome. Then, we measured the resulting mismatches to the original input called *Hamming Distance* and defined as follows:

def: *Hamming Distance*
input: two chromsome sequences q and p
output: an integer value representing the number of mismatches between the two chromosomes.
A mismatch is counted if two characters in the same position in the two inputs are different $q_i \neq p_i$.

The hamming distance is used to compute the percentage of mismatches, which we translate as the ability of the model to capture the dependencies between bases in their correct locations, and including the origins of replication.

For each of the following models (in the next three subsections), we have trained three different variations (for each chromosome):

- number of hidden layers = 2, size of hidden state = 128, learning rate = 0.002, decay rate = 0.97
- number of hidden layers = 3, size of hidden state = 256, learning rate = 0.002, decay rate = 0.97
- number of hidden layers = 4, size of hidden state = 512, learning rate = 0.002, decay rate = 0.97

3.1.1. Basic Recurrent Neural Network Model. RNNs make use of sequential information. Unlike traditional neural networks, where it is assumed that all inputs and outputs are independent of one another, RNNs are reliant on preceding computations and what has previously been calculated. RNNs can be conceptualized as a neural network unrolled over time. Where you would have different layers in a regular neural network, you apply the same layer to the input at each time step in an RNN, using the output, i.e. the state of the previous time step as input. Connections between entities in a RNN form a directed cycle, creating a sort of internal memory, that helps the model leverage long chains of dependencies. Therefore, we have thought that RNNs are a perfect candidate model for the DNA sequences we have. Refer to the our survey report for more information about RNNs.

After training, for each of the three models, we produce 5 samples of predictions of the same length as the chromosome length. The results came as follows:

Chromosome 1 (length = 230218 bp)
Hamming Distance in RNN Samples (from smallest RNN size to largest RNN size):
173948, 173082, 173006, 173664, 172288, 172286, 173352, 171771, 171926, 172162, 172569, 172065, 172343, 173478, 172671
Average hamming distance is 172707
Percentage of mismatches is 75.0188951342% (High)

Chromosome 2 (length = 813184 bp)
Hamming Distance in RNN Samples (from smallest RNN size to largest RNN size):
606354, 606704, 603973, 605757, 607257, 604072, 605052, 605381, 605257, 605522, 609221, 611847, 608816, 607850, 608017
Average hamming distance is 606738
Percentage of mismatches is 74.6126337951% (High)

Chromosome 3 (length = 316620 bp)
Hamming Distance in RNN Samples (from smallest RNN size to largest RNN size):
237777, 238681, 240192, 237878, 237186, 243469, 243366, 243867, 244910, 244453, 238357, 238287, 238653, 237989, 239760
Average hamming distance is 240321
Percentage of mismatches is 75.9020276672% (High)

3.1.2. Long Short Term Memory. Like RNNs, LSTMs also have a a chain-like structure (when unrolled). However, instead of a single neural network layer in the repeating module, LSTMs have four neural network layers interacting in a special harmony. This gives the model a more *memory* capabilities for long-term dependencies. Refer to our survey report for more information about LSTMs.

After training, for each of the three models, we produce 5 samples of predictions of the same length as the chromosome length. The results came as follows:

Chromosome 1 (length = 230218 bp)
Hamming Distance in LSTM Samples (from smallest RNN size to largest RNN size):
174101, 174818, 174969, 173849, 172583, 175593, 173084, 176003, 174917, 172813, 172669, 172829, 172486, 173686, 172509
Average hamming distance is 173793
Percentage of mismatches is 75.4906219323% (High)

Chromosome 2 (length = 813184 bp)
Hamming Distance in LSTM Samples (from smallest RNN size to largest RNN size):
610378, 610566, 612341, 610425, 611372, 605463, 605319, 606264, 605412, 604322, 612348, 613549, 614297, 611209, 613096
Average hamming distance is 609757
Percentage of mismatches is 74.9838904848% (High)

Chromosome 3 (length = 316620 bp)
Hamming Distance in LSTM Samples (from smallest RNN size to largest RNN size):
238424, 239983, 241530, 239029, 239367, 237177, 237764, 238371, 237359, 237589, 239475, 238510, 238740, 239757, 238172
Average hamming distance is 238749
Percentage of mismatches is 75.405533447% (High)

3.1.3. Gated Recurrent Unit. GRUs is a slightly modified version of LSTM that combines the forget and input gates into a single update gate [8]. Although it doesn't give the model more memory, it is a different architecture where it can capture different dependencies.

After training, for each of the three models, we produce 5 samples of predictions of the same length as the chromosome length. The results came as follows:

Chromosome 1 (length = 230218 bp)
Hamming Distance in GRU Samples (from smallest RNN size to largest RNN size):
173071, 173495, 173072, 172731, 171555, 172178, 176328, 173176, 173528, 173787, 173373, 172994, 171677, 0, 172386
Average hamming distance is 161556
Percentage of mismatches is 70.1752252213% (Lower)

Chromosome 2 (length = 813184 bp)
Hamming Distance in GRU Samples (from smallest RNN size to largest RNN size):
606096, 606775, 607632, 606760, 610050, 605633, 605896, 604598, 606477, 605438, 608990, 608751, 609603, 610111, 609227
Average hamming distance is 607469
Percentage of mismatches is 74.7025273493% (High)

Chromosome 3 (length = 316620 bp)
Hamming Distance in GRU Samples (from smallest RNN size to largest RNN size):
250683, 247479, 237948, 239193, 269732, 239419, 241219, 238388, 240274, 239194, 0, 236538, 238744, 237072, 237288
Average hamming distance is 226211
Percentage of mismatches is $71.4455814541\%(Lower)$

See table 2 for a summary of the results. Unfortunately, we couldn't train the remaining thirteen chromosomes as our machines were dying! The above deep learning models took days to train, and we had to restart training and make modifications when we see odd results. Find a complete log file for the training on our GitHub repository. We can notice that by the end of training, the average loss is about 0.6 over all trained models. However, the results were informative that we can make constructive conclusions based upon (see section 4).

Chromosome	**1**	**2**	**3**	...
Length	316620 bp	813184 bp	316620 bp	...
RNN	75.01889%	74.61263%	75.90202%	...
LSTM	75.49062%	74.98389%	75.40553%	...
GRU	70.17522%	74.702527%	71.44558%	...

TABLE 2: Summary of the percentages of mismatches obtained by training the generative models.

3.2. Generative Models

The goal of a Generative Model is to capture the inherent relationships that exist in a given input. In the domain of DNA sequences, we were motivated by the fact that Generative models possessed this ability to capture the distribution of the data. We postulated that these models might be able to learn characteristics of a given DNA Origin of Replication for use in future application areas. To capture these relationships, and in an effort to more efficiently train our Generative models, we had to preprocess the data a little differently. Preprocessing consisted of:

- For a given chromosome chr, convert all bases to 0
- For each origin ori in chr, convert the base character into a designated integer
- A - 1
- C - 2
- G - 3
- T - 4
- Use the beginning and ending index of each origin to create a sparse vector of 0's containing only that particular origin inserted into it

For example, for $chr\text{-}x$ = $ACCTGCTGCATTCG$ and one origin of replication $ori\text{-}x\text{-}1$ = $CTGCA$, the resulting input for the generative model is

$chr\text{-}x$ = 00000243210000

This preprocessing is done in a file included in our Github repository.

As mentioned with the discriminant models, this is for illustration purposes only. The true length of the chromosome is reflected in the table and is usually several hundred thousand characters long. With the average origin of Replication only several hundred characters long, we understood that we were providing very sparse data as an input. We hoped that the network would be able to overcome this obstacle and somehow learn a simpler representation of the data. We postulated that by preprocessing the data in this fashion, in a very sparse manner, that the model might be able to capture "local" characteristics such as

- The number of A's (1's after preprocessing) in a given origin (logic extends to other bases of course)
- The sum of the added bases in the Origin
- Other crazy "black magic" characteristics the model might find!

3.2.1. Restricted Boltzmann Machine. The first Generative model we tried was of course the Restricted Boltzmann Machine. Touted as possessing the representational power to approximate any function or distribution, we thought it proper to start here. The training is illustrated in our github repository. The model's input layer is set to the length of the Chromosome sequence being processed. For chr - 1, the input layer of the RBM would have 230219 nodes. Because our input data was so large, we decided to set the number of hidden nodes to the large number of 900. We set the learning rate $alpha$ to 0.3.
At each iteration of training, the model is partially fit with the distribution of a particular origin. Therefore, each chromosome would have a unique RBM trained on it. Each iteration of training consisted of the following. The first step was partially fitting the model with each origin. This updated the internal weights of the RBM by performing stochastic steepest ascent to maximalize log likelihood of correct Reconstruction. Each partial fit returned an error for the network. This training error is summarized in Table 4
Although these values are extremely low, this raises quite a few concerns. First off, we in no way expected Reconstruction errors to be so low. This leads us to believe one of either two things.

- The model is completely overfitting the small section that constitutes the Origin site. This is obviously a great concern because Overfitting is one of the dangerous products of a model that is so complex. By overfitting in this manner, the model will be unable to perform any sort of predictions on any other sites.
- Due to sparse nature of our data, the actual origin sites are being completely overlooked. From a logical standpoint, if the dataset contains over 500,000 indexes, overlooking 500 indexes that are not 0 will not leading to that great of an error. It was our hopes that the model would be able to capture complex relationships in the sparse vector, but it is very well plausible that the model may have overlooked all of the relationships!

Regardless of the lackluster performance, this model did successfully train and run. This leads us to believe that with the right set of hyper-parameters, one could potentially properly fit the Origin sites.

3.2.2. Deep Belief Network. The next logical step after implementing a Restricted Boltzmann Machine was to implement a stack of them! We then shifted focus on implementing a Deep Belief Network. Data was preprocessed in the exact same manner as in the RBM training. For this model however, we tried and trained two separate architectures. The first DBN has only 2 layers whereas the second network has 4. We will now cover our results. Results are summarized in table 3 Our DBNs can be found in our GitHub repository as well.

In an attempt to be able to compare the performance of our DBN to just one single RBM, we decided to keep the same hyper-parameters for the first RBM in the stack. Therefore, the first RBM in our Deep Belief Network had an input layer the size of the chromosome. This also implies that each chromosome would have their own unique DBN trained on it. The hidden layer of this first RBM was also set to 900 as in the single RBM implementation.

From here, we then set up the second RBM in the stack. This RBM has an input layer of 900 as it will receive the hidden layer values from the previous RBM. We set the number of nodes in this layer to be 500 for both Deep Belief Networks. We wanted to compare the training errors procured based on the depth of the Deep Belief Network so we kept the same 2 "core" RBMs in both models.

For our deeper DBN, we set the the number of hidden nodes in the third RBM as 250. Finally, in the last layer of our network, we set the number of hidden nodes to 2. In this sense, we hope to capture dependencies at each level, while progressively reducing the number of dimensions of the model.
In conclusion, we were able to successfully train the model in a greedy layer-wise manner with both 2 and 4 layers. We logged the training error at each iteration and provide a table of our results. We would like to make note of a striking pattern in table 3. Initially, as the number of nodes in each hidden layer is decreased, the training reconstruction cost follows suit. In the last layer, however, we notice a different trend. In this layer, the reconstruction errors are orders of magnitude higher. We postulate that this might be a very positive indication of the importance in depth when it comes to DBNs. As well, this higher reconstruction cost can be attributed to the fact that we set the number of hidden nodes to 2 in the final layer. This is in fact a major departure from the previously large hidden layers. We cannot confirm this, but our intuition leads us to believe that it very well may have taken 3 previous layers of non-sense reconstruction to finally reveal the very high-dimensional dependencies in the data.

We also suggest the reader note comparisons between our DBN model to just a single RBM in terms of training

Training Iteration	DBN (2 layers)		DBN (4 layers)			
	Layer 1	*Layer 2*	*Layer 1*	*Layer 2*	*Layer 3*	*Layer 4*
1	0.0381941	0.00066498	0.0381941	0.000360921	$2.03222e-9$	0.181541
2	0.034667	$8.55769e-5$	0.034667	$6.4706e-6$	$2.01938e-9$	0.133269
3	0.0161238	$4.38737e-5$	0.0161238	0.000137399	$2.00668e-9$	0.0998068
4	0.0080532	0.0014308	0.0080532	0.000435213	$1.99412e-9$	0.0764134
5	0.0817613	0.00040825	0.0817613	$1.04328e-5$	$1.98169e-9$	0.0597758
6	0.00772308	0.000352751	0.00772308	0.000414569	$1.96939e-9$	0.0476989
7	0.00799673	0.00067412	0.00799673	0.000773979	$1.95721e-9$	0.0387475
8	0.00808361	0.00100052	0.00808361	0.00180126	$1.94517e-9$	0.0319789
9	0.0356183	0.000115462	0.0356183	$8.35325e-5$	$1.93325e-9$	0.0267653
10	0.0333074	0.000257309	0.0333074	$2.58781e-5$	$1.92145e-9$	0.022681
11	0.00907397	0.00118755	0.00907397	0.000535597	$1.90977e-9$	0.0194321
12	0.0258015	0.000173275	0.0258015	$8.59969e-5$	$1.89822e-9$	0.0168118
13	0.0395059	0.000336075	0.0395059	$8.95992e-5$	$1.88678e-9$	0.014672
14	0.0292634	$5.52816e-10$	0.0292634	$8.21859e-5$	$1.87545e-9$	0.0129048

TABLE 3: Summary of Training Cost of DBN

error. While this gives us a minor baseline in terms of performance between the individual RBMs in the DBN and the standalone RBM in a very convenient manner, further research is needed. Unfortunately, the reconstruction error is not always the best measure of performance of a network and therefore can only be used as guides for future research.

3.2.3. Auto-encoder. The last Generative architecture we tested was the Auto-encoder. Utilized as a dimensionality reduction technique, we postulated that an Auto-encoder might be able to produce similar effects on this huge input data. An Auto-encoder is a Neural Network with 3 layers. The input and the output layers, in a perfectly trained network, should be identical. The hidden layer, usually having a smaller number of nodes then the input layer, serves as an encoded version of the data. During training, the input is propagated to the hidden layer. This usually creates some sort of reduction in dimensionality. This is also called the *encoding phase*. Each hidden node then propagates a signal forward to the output layer. This is called the "decoding phase." The training error of the network is then determined based on how close that decoding phase was able to recreate the input layer.

From this standpoint, Auto-encoders offer something more similar to a supervised learning approach. While still falling under the domains of Generative models and Unsupervised Learning, the Auto-encoder is unique in that it had somewhat of an *Expected output* to help calculate this network error. Because the model was able to actually generate a concrete error based on how close the Output was to the Target value, we hoped that we would be able to achieve better performance with this model.

We've included our Auto-encoder model in our GitHub repository. During initial training, we tried training the model in an independent manner. We received absolutely no success in this area and were forced to make an adjustment. As you can see in the included file, our Auto-encoder is first set up with the weights obtained from a trained RBM. This provides the network with a better initial state. The downfall to this approach is that any error gathered during RBM training is inherited into the Auto-encoder. None the less, efficiencies will come as studies in this direction are continued. For the time being, we wanted to prove the trainability of the model.

Training Iteration	RBM	Auto-encoder
1	0.0381941	0.195433020592
2	0.034667	0.381623759866
3	0.0161238	0.508603200316
4	0.0080532	0.598342828453
5	0.0817613	0.884282134473
6	0.00772308	0.972163192928
7	0.00799673	1.0615876466
8	0.00808361	1.15149652958
9	0.0356183	1.34022453427
10	0.0333074	1.52272772789
11	0.00907397	1.617985107
12	0.0258015	1.77861361951
13	0.0395059	1.97737444192
14	0.0292634	2.14844004065

TABLE 4: Summary of Training Cost of RBM and Auto-encoder.

Therefore, the architecture of our Auto-Encoder is as follows. The input layer of the Auto-encoder is the length of the chromosome sequence. Similarly to the other Generative architectures, a unique Auto-encoder is trained for each chromosome. Each model is then trained on all of the respective Origins. We set the hidden layer to 900 hidden nodes in an attempt to create the most structurally similar network as the RBM we previously trained. Although the architectures are different, we reasoned that by having a very similar design, we might be able to compare results better. We provide the training results in Table 4. As you can see from the table, the Auto-encoder consistently posted a higher training cost. We believe this additional cost is due to the fact that the Auto-encoder might be able to capture the dependencies of the data! By posting a higher reconstruction error, we believe this model will offer superior performance over the RBM once properly configured. Although one generally strives for a lower reconstruction error, we believe

at this stage of research, a higher reconstruction error shows greater potential accuracy for the future.

4. Conclusions

Due to the fact that what we are attempting is original research, our aim was not at optimizing hyper-parameters or conditions of the networks to achieve optimal performance. The goal of our study as well was not to determine the efficiency of particular models. It was to test the validity of using deep learning models in identifying/confirming origins of replication in DNA sequences as well as to provide a step in the right direction for future research in this area as we believe deep learning models can provide aid in current research and has the potential to one day achieve state of the art results.

From the results shown in the previous section, we can conclude that:

- Recurrent Neural Network models could be a good candidate to use in training deep learning models on DNA sequences. Although showing a low accuracy in prediction, they could actually capture long-term dependencies in DNA sequences, especially when using the Gated Recurrent Unit variant.
- The different variations of RNNs trained proves our initial hypothesis that there is some sort of dependencies between the base pairs of DNA sequences. This opens a new door in the methods used in discovering replication origins in DNA sequences.
- Generative models probably overfit the data and were unable to capture dependencies.

5. Future Work

Discovering DNA origins of replication has had little to no inter-disciplinary work with the field of deep learning at this point. From a research standpoint, we believe the direction of this paper constitutes the potential beginning of an area of application of deep learning models. Due to the lack of computational power, it took us a long time to train the different set of models we have proposed in the beginning that some models took days to train to reach acceptable loss value. To build on this work, we may extend our research to implement a hybrid approach

5.1. Hybrid Approach

An interesting approach we found was collaborative work between both the discriminant and generative models. For example, the hidden layers learned by our Deep Belief Network alone might prove useless. But in a collaborative manner with a discriminant architecture, these hidden layers might prove useful as inputs to a discriminant model. Although further research is needed on this in the area of confirming DNA replication origins to confirm our hypothesis, this is a tried and true method in deep learning. The collaborative effort can be extended to the other generative models as well. The hidden layer of an Auto-encoder is often utilized for dimensionality reduction. This reduced form of data can be input into a discriminant model in an effort to improve the efficiency of the model.

5.2. Improved Hardware

Finally, with the proper hardware configuration, along with a GPU unit, we can train discriminant predictive model that takes the whole Cerevisiae genome (16 chromosome) as one input to capture longer-term dependencies. Only with an improvement in hardware will it be possible to train all 16 chromosomes in a timely manner.

Proper hardware will also enable the testing of additional models with a shorter delay in receiving results. Our average training time was around 24 hours. These models are incredibly computationally intensive and require the machine to dedicate most of its processing power towards the training procedure. Errors are generally not realized until well into the training procedure which leads to a lot of wasted time. Allievating this downtime allows more models to be trained and researched.

In addition to an increase amount of computation capabilities, the proper hardware configuration will allow the testing of many more sets of hyper-parameters. The act of training and type of neural network is an art-form that can only be mastered with much trial and error. Hyper-parameters must be custom fit in order to improve the efficiency of the network. This just simple required to much training time to be feasible in the scope of this research. Future work has an entire area of exploration just in this limited realm of setting proper hyper-parameters. We believe they are key to capturing the dependencies of the data and would be an initial step of any further research in this direction.

Author contributions – Abdelrahman did an extensive study on the biological principles involved. This includes both understanding the biology as well as relating it to the field of deep learning. In addition, he handled the preprocessing of data to be suitable for training the discriminant architectures. Finally, he trained the discriminant models and compiled the results. Anthony focused more attention to the deep learning side of things. This includes investigating previous research efforts in this area as well as potential related work. He focused on the generative side of things, which included determining potential generative models to learn representations of the DNA sequences. He trained the generative models presented in this study and concluded the results. Both authors agreed on the research approach and collaborated on understanding the results, inferring conclusions, and planning for the future work. Our team established a clear allocation of responsibilities while still retaining strong collaborative efforts throughout the process.

References

[1] A. Hosny and A. Parziale, “Deep learning models for saccharomyces cerevisiae,” 2016.

[2] J. M. Cherry, E. L. Hong, C. Amundsen, R. Balakrishnan, G. Binkley, E. T. Chan, K. R. Christie, M. C. Costanzo, S. S. Dwight, S. R. Engel, D. G. Fisk, J. E. Hirschman, B. C. Hitz, K. Karra, C. J. Krieger, S. R. Miyasato, R. S. Nash, J. Park, M. S. Skrzypek, M. Simison, S. Weng, and E. D. Wong, "Saccharomyces genome database: the genomics resource of budding yeast," *Nucleic Acids Research*, vol. 40, no. D1, pp. D700–D705, 2012. [Online]. Available: http://nar.oxfordjournals.org/content/40/D1/D700.abstract

[3] C. C. Siow, S. R. Nieduszynska, C. A. Müller, and C. A. Nieduszynski, "Oridb, the dna replication origin database updated and extended," *Nucleic Acids Research*, vol. 40, no. D1, pp. D682–D686, 2012. [Online]. Available: http://nar.oxfordjournals.org/content/40/D1/D682.abstract

[4] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, "Predicting the sequence specificities of dna- and rna-binding proteins by deep learning," *Nat Biotech*, vol. 33, no. 8, pp. 831–838, 08 2015. [Online]. Available: http://dx.doi.org/10.1038/nbt.3300

[5] Google, "Google tensorflow."

[6] Numpy, "Numpy package."

[7] BioPython, "Biopython package."

[8] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078