

REPORT

OE3E40

**Splice Site Detection Using Chaos**

**GAME Representation and**

**Neural Networks**

**TEAM MEMBERS:**

**1.U.VISHNU -22bec126**

**2.P.HARSHA VARDHAN-22bec090**

**3.V.NASHIK -22bec129**

**4.MANIDEEP-22bec046**

**5.D.NIRUPAMA-22bec035**

**SUBMITTED TO: signature:**

**Prof . Sanjeev Narayan Sharma**

**INTRODUCTION**

**Splice sites are key genomic regions that facilitate the splicing process by marking the boundaries between introns and exons in pre-mRNA. Accurate identification of these sites is critical for understanding gene expression, annotating genomes, and detecting mutations linked to diseases.**

**Traditional computational approaches often struggle with the variability and complexity of splice sites in genetic sequences.**

**This project leverages Chaos Game Representation (CGR) and neural networks to address these challenges. CGR is a visualization technique that encodes DNA sequences into geometric patterns, providing a unique way to capture sequence information.**

**Neural networks, renowned for their pattern recognition capabilities, are employed to analyze these representations and detect splice sites. This novel approach integrates the strengths of both methods, promising improved accuracy and efficiency in splice site prediction.**

**DATA COLLECTION**

**The NN269 dataset used for this project consists of DNA sequences annotated with confirmed splice sites. Data was sourced from publicly available genomic databases, such as:**

* **Ensembl Genome Browser**
* **GENCODE**
* **UCSC Genome Browser**

**The dataset was pre-processed to ensure high-quality inputs:**

1. **Class Balance: The dataset included a balanced number of true splice sites (positive samples) and non-splice sites (negative samples) to prevent bias.**
2. **Train-Test Split: The dataset was divided into training, validation, and test sets to train and evaluate the model effectively.**

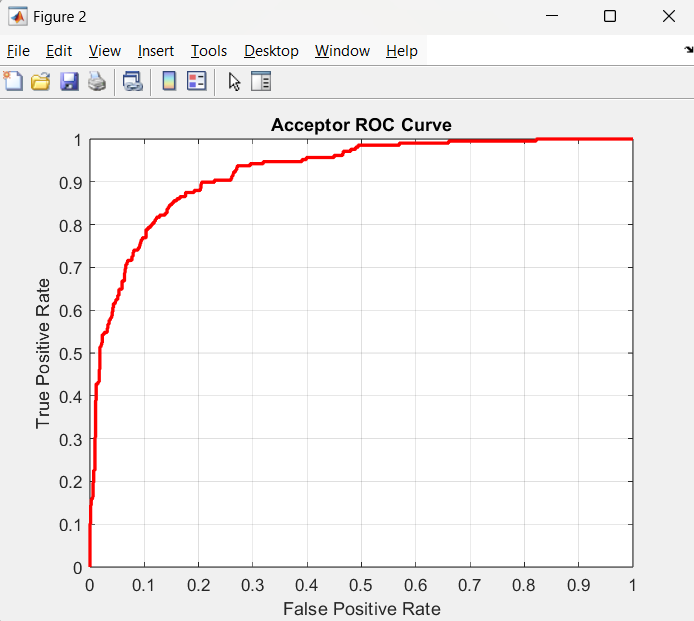
Methodology:

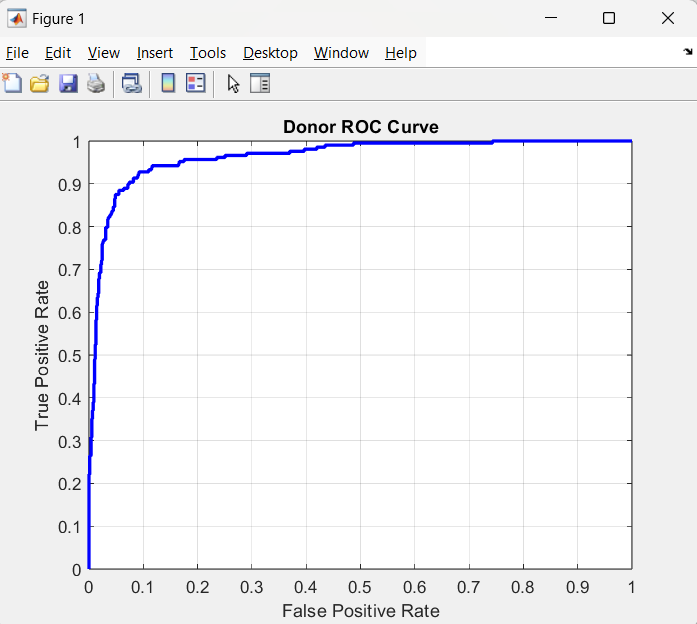
The methodology is a two-step process involving feature extraction and splice site detection:

1. **Chaos Game Representation (CGR)**:
   * DNA sequences were converted into graphical representations using CGR, transforming nucleotide sequences into fractal-like images.
   * Each sequence was mapped into a 2D plane using CGR rules, capturing sequence-specific patterns in a visual format.
2. **Neural Network Model**:
   * A feedforward neural network (ANN) was used to process the CGR images.
   * The network was trained using the cross-entropy loss function and optimized with the Adam optimizer. Hyperparameters such as learning rate, batch size, and number of epochs were fine-tuned to optimize performance.

Results:

The model's performance was evaluated :

* 



**Donor AUC: 0.96395**

**Acceptor AUC: 0.92285**

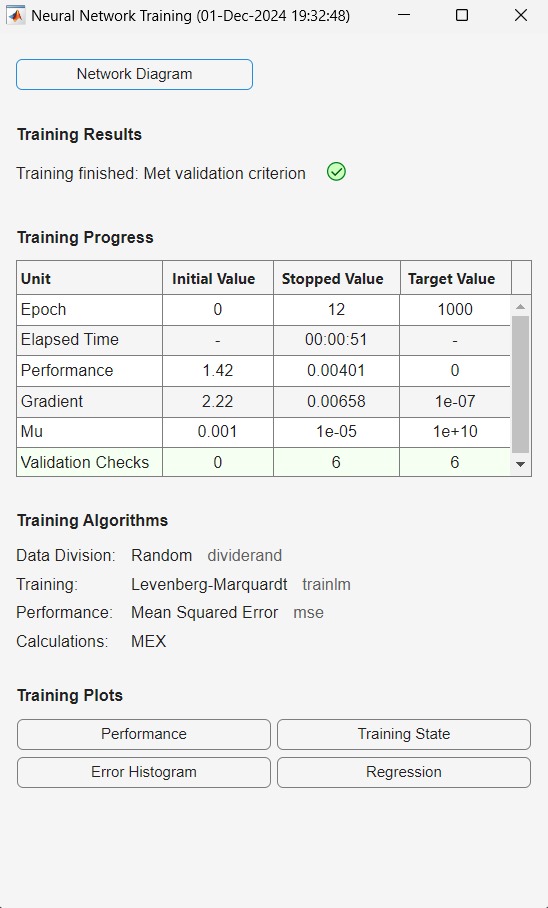
**Training Results:**

**1.The training stopped at 12 epochs as further iterations were unnecessary.**

**2.The training took about 51 seconds to complete.**

**3.The performance(mean squared error ) is very low indicating that it is a good fit.**

**\*Totally 6 validation checks are performed.**



**The results demonstrated that the CGR-based approach significantly outperformed traditional sequence-based methods, showcasing the ability of neural networks to extract meaningful patterns from CGR images.**

**5. Conclusion**

This project highlights the potential of combining Chaos Game Representation and neural networks for splice site detection. The CGR approach effectively captures the inherent patterns of DNA sequences, and the neural network's learning capabilities enable accurate classification. This method not only enhances the accuracy of splice site prediction but also opens avenues for applying similar techniques to other genomic analysis tasks. Future work will focus on scaling the model for large-scale genomic data and improving its robustness to handle diverse genetic variations.

REFERENCE:

The topic for our project on splice site detection using Chaos Game Representation and Neural Networks has been referred from the research presented in the article "Splice Site Detection Using Chaos Game Representation and Neural Networks" published in *ScienceDirect*

**Link:** [**https://www.sciencedirect.com/science/article/pii/S0888754318302118**](https://www.sciencedirect.com/science/article/pii/S0888754318302118)

Code:

% Step 1: Load the FASTA files for donor and acceptor splice sites

Donor\_Train\_Positive = fastaread('Donor\_Train\_Positive.fasta');

Donor\_Train\_Negative = fastaread('Donor\_Train\_Negative.fasta');

Donor\_Test\_Positive = fastaread('Donor\_Test\_Positive.fasta');

Donor\_Test\_Negative = fastaread('Donor\_Test\_Negative.fasta');

Acceptor\_Train\_Positive = fastaread('Acceptor\_Train\_Positive.fasta');

Acceptor\_Train\_Negative = fastaread('Acceptor\_Train\_Negative.fasta');

Acceptor\_Test\_Positive = fastaread('Acceptor\_Test\_Positive.fasta');

Acceptor\_Test\_Negative = fastaread('Acceptor\_Test\_Negative.fasta');

% Step 2: Convert DNA sequence into CGR coordinates

function cgr\_coords = sequence\_to\_cgr(sequence)

% Initialize empty coordinates matrix (two dimensions for each nucleotide)

cgr\_coords = zeros(length(sequence), 2);

% Mapping nucleotides to CGR coordinates (A, T, C, G -> 4 directions)

for i = 1:length(sequence)

switch sequence(i)

case 'A'

cgr\_coords(i, :) = [1, 0];

case 'T'

cgr\_coords(i, :) = [0, 1];

case 'C'

cgr\_coords(i, :) = [1, 1];

case 'G'

cgr\_coords(i, :) = [0, 0];

end

end

end

% Step 3: Prepare the donor training dataset

X\_donor\_train = [];

y\_donor\_train = [];

for i = 1:length(Donor\_Train\_Positive)

cgr\_coords = sequence\_to\_cgr(Donor\_Train\_Positive(i).Sequence);

X\_donor\_train = [X\_donor\_train; cgr\_coords(:)']; % Flatten to row vector

y\_donor\_train = [y\_donor\_train; 1];

end

for i = 1:length(Donor\_Train\_Negative)

cgr\_coords = sequence\_to\_cgr(Donor\_Train\_Negative(i).Sequence);

X\_donor\_train = [X\_donor\_train; cgr\_coords(:)'];

y\_donor\_train = [y\_donor\_train; 0];

end

assert(size(X\_donor\_train, 1) == length(y\_donor\_train), 'Mismatch in donor training data.');

% Step 4: Prepare the donor test dataset

X\_donor\_test = [];

y\_donor\_test = [];

for i = 1:length(Donor\_Test\_Positive)

cgr\_coords = sequence\_to\_cgr(Donor\_Test\_Positive(i).Sequence);

X\_donor\_test = [X\_donor\_test; cgr\_coords(:)'];

y\_donor\_test = [y\_donor\_test; 1];

end

for i = 1:length(Donor\_Test\_Negative)

cgr\_coords = sequence\_to\_cgr(Donor\_Test\_Negative(i).Sequence);

X\_donor\_test = [X\_donor\_test; cgr\_coords(:)'];

y\_donor\_test = [y\_donor\_test; 0];

end

% Step 5: Prepare the acceptor training dataset

X\_acceptor\_train = [];

y\_acceptor\_train = [];

for i = 1:length(Acceptor\_Train\_Positive)

cgr\_coords = sequence\_to\_cgr(Acceptor\_Train\_Positive(i).Sequence);

X\_acceptor\_train = [X\_acceptor\_train; cgr\_coords(:)'];

y\_acceptor\_train = [y\_acceptor\_train; 1];

end

for i = 1:length(Acceptor\_Train\_Negative)

cgr\_coords = sequence\_to\_cgr(Acceptor\_Train\_Negative(i).Sequence);

X\_acceptor\_train = [X\_acceptor\_train; cgr\_coords(:)'];

y\_acceptor\_train = [y\_acceptor\_train; 0];

end

assert(size(X\_acceptor\_train, 1) == length(y\_acceptor\_train), 'Mismatch in acceptor training data.');

% Step 6: Prepare the acceptor test dataset

X\_acceptor\_test = [];

y\_acceptor\_test = [];

for i = 1:length(Acceptor\_Test\_Positive)

cgr\_coords = sequence\_to\_cgr(Acceptor\_Test\_Positive(i).Sequence);

X\_acceptor\_test = [X\_acceptor\_test; cgr\_coords(:)'];

y\_acceptor\_test = [y\_acceptor\_test; 1];

end

for i = 1:length(Acceptor\_Test\_Negative)

cgr\_coords = sequence\_to\_cgr(Acceptor\_Test\_Negative(i).Sequence);

X\_acceptor\_test = [X\_acceptor\_test; cgr\_coords(:)'];

y\_acceptor\_test = [y\_acceptor\_test; 0];

end

% Step 7: Train the donor neural network

donor\_net = feedforwardnet(10);

donor\_net = train(donor\_net, X\_donor\_train', y\_donor\_train');

% Step 8: Train the acceptor neural network

acceptor\_net = feedforwardnet(20);

acceptor\_net = train(acceptor\_net, X\_acceptor\_train', y\_acceptor\_train');

% Step 9: Evaluate donor and acceptor models

y\_donor\_pred = donor\_net(X\_donor\_test')';

y\_acceptor\_pred = acceptor\_net(X\_acceptor\_test')';

% Step 10: Plot ROC for donor

[donor\_fpr, donor\_tpr, donor\_thresholds] = perfcurve(y\_donor\_test, y\_donor\_pred, 1);

figure;

plot(donor\_fpr, donor\_tpr, 'b-', 'LineWidth', 2);

xlabel('False Positive Rate');

ylabel('True Positive Rate');

title('Donor ROC Curve');

grid on;

% Step 11: Plot ROC for acceptor

[acceptor\_fpr, acceptor\_tpr, acceptor\_thresholds] = perfcurve(y\_acceptor\_test, y\_acceptor\_pred, 1);

figure;

plot(acceptor\_fpr, acceptor\_tpr, 'r-', 'LineWidth', 2);

xlabel('False Positive Rate');

ylabel('True Positive Rate');

title('Acceptor ROC Curve');

grid on;

% Step 12: Display AUC

donor\_auc = trapz(donor\_fpr, donor\_tpr);

acceptor\_auc = trapz(acceptor\_fpr, acceptor\_tpr);

disp(['Donor AUC: ', num2str(donor\_auc)]);

disp(['Acceptor AUC: ', num2str(acceptor\_auc)]);