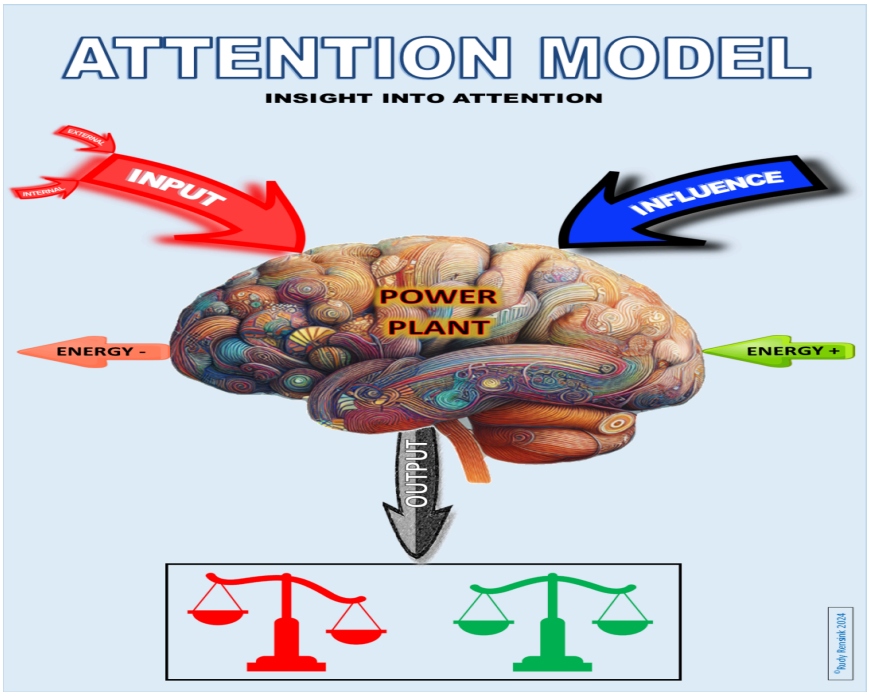

Promoter Prediction Using Attention Models

XX.XX.XX

XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX

Abstract

Accurate identification of promoter regions in DNA sequences plays a crucial role in understanding gene regulation and expression. Traditional machine learning techniques often fail to capture the complex patterns and long-range dependencies inherent in genomic data. In this project, we propose a deep learning approach that combines Convolutional Neural Networks (CNN), Bidirectional Long Short-Term Memory (BiLSTM) networks, and an attention mechanism to improve promoter prediction accuracy. The CNN component captures local motif-like patterns, while the BiLSTM learns sequential dependencies. The attention mechanism enhances model interpretability by identifying the most relevant parts of the sequence for classification. Our experiments demonstrate that incorporating attention significantly improves classification performance compared to models without it. The proposed model shows promising results and serves as a step forward in applying attention-based architectures to genomic sequence analysis.

Theoretical Background

1 What is an Attention Mechanism?

The **attention mechanism** is a computational method inspired by human cognitive processes, especially our ability to selectively focus on relevant parts of the information we perceive. In deep learning, attention allows models to **dynamically assign different weights** to different parts of the input, enabling the model to "focus" on what is most informative for a specific task.

The general structure of an attention mechanism involves three main components:

- **Query (Q)**: the vector representing the current processing point (e.g., the decoder's hidden state).
- **Keys (K)**: vectors representing the elements over which we attend (e.g., encoder outputs).
- **Values (V)**: the actual information content to be aggregated.

The attention score is computed using a similarity function (often a dot product) between the query and keys, followed by a **softmax** normalization to convert scores into probabilities. The output is a weighted sum of the values:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(QK^{t} / \sqrt{d_k} \right)$$

Different attention techniques have emerged:

- **Additive Attention (Bahdanau)**
- **Multiplicative or Scaled Dot-Product Attention (Vaswani et al.)**
- **Self-Attention (used in Transformers)**
- **Multi-Head Attention** (to learn diverse features simultaneously)

Originally designed for **natural language processing (NLP)**, attention mechanisms are now widely used in fields such as **computer vision**, **audio processing**, and increasingly, in **bioinformatics**.

2 Why Attention in Bioinformatics?

Bioinformatics is an interdisciplinary field that combines biology, computer science, and mathematics to analyze and interpret biological data. One of its key challenges is the **understanding of genomic sequences**, particularly **DNA**, which encodes all the genetic information necessary for life.

DNA is a linear sequence of nucleotides (A, T, C, G). Within these sequences, certain regions play specific roles. For example, **promoter regions** are DNA segments located upstream of genes that initiate the transcription process — they act like switches that turn genes "on" or "off." In contrast, **non-promoter sequences** do not regulate gene expression.

Detecting whether a sequence is a promoter is crucial for:

- Understanding gene expression regulation
- Identifying potential disease-causing mutations
- Designing gene-editing therapies

- Synthetic biology and bioengineering

Traditional computational models often rely on fixed-length **motifs** or **pattern-matching** techniques to detect promoters (like the well-known **TATA box** motif), but these methods:

- Lack flexibility for variable patterns
- Struggle to capture long-range dependencies
- Have limited generalization across species or conditions

This is where **attention mechanisms** come in. Attention allows the model to:

- Dynamically focus on **informative subsequences** (like motifs)
- Learn **contextual relationships** across the entire DNA sequence
- Improve **interpretability**: attention weights can highlight which parts of the sequence the model used to make a decision

For example, in a 251-length DNA sequence, the model might learn to attend more heavily to positions where the TATA box commonly appears, while ignoring irrelevant noise. This makes attention not only powerful but **biologically meaningful**.

3 Why Focus on Promoter Prediction?

Identifying promoters is a foundational task in genomics. Accurate promoter prediction supports:

- **Genome annotation**
- **Transcription factor binding site analysis**
- **Gene regulation studies**
- **Drug target identification**

Yet, promoters can vary in sequence and structure, especially across organisms. Therefore, **machine learning approaches**, particularly **deep learning with attention**, offer a **data-driven, flexible alternative** to traditional rule-based models.

Our motivation in this project is to leverage the **CNN + BiLSTM + Attention** architecture to:

- Improve classification accuracy between promoter and non-promoter sequences
- Provide **interpretable outputs** via attention weights
- Demonstrate the value of attention in bio-sequence modeling

Dataset Construction and Preprocessing

Data Sources :

Our dataset is a **hybrid combination** of two distinct sources:

- **Kaggle DNA Promoter Dataset**
  🔗 Link to Kaggle Dataset
  Provided **53 promoter** and **53 non-promoter** sequences.
  This dataset, while balanced, was **too small** to train a deep learning model.

- **EPDnew (Eukaryotic Promoter Database)**
  🔗 Link to EPDnew Database
  Contained **~4000 human promoter** sequences and **~450 non-promoter** sequences (negative set generated via sampling upstream intergenic regions).
  These promoter sequences are **experimentally validated**, offering high biological relevance.

  🧩 **Data Construction Strategy**

To build a balanced dataset:

1. We used **all ~4000 promoter sequences** from EPD.
2. We **combined** the **~450 non-promoters from EPD** with the **53 non-promoters from Kaggle**, obtaining **~500 non-promoter sequences**.

3. Then, we applied **data augmentation techniques** to expand this minority class to ~**4000 synthetic non-promoters**.

👉 Final goal: approximately **4000 promoters vs. 4000 non-promoters**

Challenges and Preprocessing Steps:

During the dataset construction and preparation, we faced several issues that required targeted solutions:

🔍 **Ambiguous nucleotide characters**
Some sequences included non-standard symbols (like "N", "R") not part of the canonical bases (A, T, C, G). To ensure biological accuracy, these sequences were removed.

📏 **Inconsistent sequence lengths**
DNA sequences ranged from 57 to 60 nucleotides. We standardized them to 60 by duplicating the last 3 bases of shorter sequences, ensuring fixed input size for the model.

⚖️ **Class imbalance**
The EPD dataset provided ~4000 promoters but only ~450 non-promoters. To balance, we added 53 non-promoters from Kaggle (for a total of ~500), then applied data augmentation techniques to reach ~4000 non-promoters.

🔄 **Non-random data order**
Merged data was ordered by class (all promoters, then non-promoters), risking training bias. We randomized the sequence order to ensure better learning dynamics.

These steps were crucial to ensure a high-quality, balanced, and machine-readable dataset for effective model training.

📊 **Note:** All these challenges were uncovered through careful **exploratory data analysis (EDA)**, including sequence length histograms, class distribution charts, and visual inspection of nucleotide compositions. ***The EDA*** phase guided our cleaning, balancing, and augmentation decisions.

As a result, we decided ***to 1 : normalize all sequences*** **to 60 nt** by repeating the last three nucleotides:

```
if len(sequence) < 60:
    sequence += sequence[-3:] * ((60 - len(sequence)) // 3)
    sequence = sequence[:60]
```

2 : Data Augmentation: Techniques and Biological Justification

To balance the dataset, we applied the following **five augmentation techniques** on the ~500 non-promoter sequences to generate ≈4000 new sequences:

🧬 1. Complement

- Replace each nucleotide by its pair: A↔T, C↔G
- Biological relevance: DNA's double-helix structure has complementary strands.

🔁 2. Reverse

- Reverse the entire sequence
- Mimics antisense transcription.

🔃 3. Reverse Complement

- Apply reverse, then complement
- Widely used in bioinformatics as a mirror of the sense strand.

🧪 4. Mutate

- Randomly change one or more nucleotides (point mutations)
- Simulates natural mutations and genetic noise.

🔄 5. Shift

- Cyclically shift the sequence by 1 to 3 positions
- Simulates structural variation or sequence offset

Dataset Format and Finalization:

The cleaned and augmented dataset was saved as a CSV file with the following format:

label	Sequence	Description
0	TGACTGACGTAG...	Augmented sequence

Label: 1 = promoter, 0 = non-promoter

Description: Metadata about origin or transformation applied

The final dataset was **shuffled randomly** to avoid label bias

One-Hot Encoding for Neural Network Input :

Each DNA sequence was transformed via **One-Hot Encoding**, mapping:

- A → [1, 0, 0, 0]
- C → [0, 1, 0, 0]
- G → [0, 0, 1, 0]
- T → [0, 0, 0, 1]

Result: each 60-nt sequence becomes a **60×4 matrix**, stored in a NumPy array of shape `(n_samples, 60, 4)`.

✅ This encoding is ideal for neural networks:

- Maintains **biological independence** of each base
- Prevents artificial ordinal relationships

🔢 Conversion to NumPy Arrays

The encoded sequences were converted to **NumPy arrays** for efficient manipulation and compatibility with deep learning frameworks like TensorFlow. This step enabled fast vectorized operations during training and evaluation.

📊 Dataset Splitting and Cross-Validation

To ensure a robust evaluation of our model, we:

- **Split the dataset** into training and test sets using **stratified sampling** to preserve class balance.
- Applied **K-Fold Cross-Validation** (with `k=5`) during training to minimize overfitting and provide better generalization.
- Maintained **consistency** between training folds by shuffling the data before splitting, ensuring that each fold receives a representative sample of the whole dataset.

Model Architecture and Training

I. 🧠 Model Overview

To classify DNA sequences as promoters or non-promoters, we designed a deep learning model that combines the power of convolutional and recurrent layers with an attention mechanism. The architecture was built using the Keras API from TensorFlow.

II. 📐 Model Structure: CNN + BiLSTM + Attention

Input Layer: Accepts input sequences in shape `(60, 4)` after one-hot encoding.

1D Convolution (CNN): Captures local motifs or patterns (e.g., TATA box) within the DNA sequence using 64 filters and kernel size 3.

MaxPooling: Reduces the dimensionality and retains essential features.

Bidirectional LSTM (BiLSTM): Captures long-range dependencies in both directions of the sequence.

Attention Mechanism: Highlights the most informative parts of the sequence by learning a context vector.

Global Average Pooling: Reduces the output into a single vector.

Output Layer: A single neuron with a sigmoid activation for binary classification.

This architecture allows the model to detect both **local sequence motifs** and **global contextual patterns** effectively.

III. 🔁 Training Strategy with K-Fold Cross-Validation

To ensure robustness and generalizability, we used **5-Fold Cross-Validation**,

splitting the data into 5 parts and training the model 5 times, each time using a different fold for validation. For each fold:

- A new model instance is created and compiled.
- Training is performed with **Early Stopping** (patience = 5) to prevent overfitting.
- The best model of each fold is saved using **ModelCheckpoint**.
- Accuracy is printed and saved for further analysis.

Each fold trains for up to **20 epochs**, with a **batch size of 32**.

📈 Performance Monitoring

To visualize performance, two types of accuracy plots were generated:

- **Individual Accuracy Curves** for each fold to observe variability between them.
- **Average Accuracy Curve** to show the general trend and convergence across all folds.

🔍 Why This Architecture?

- **CNN** is effective for learning position-invariant features, particularly useful for DNA where motif location may vary.
- **BiLSTM** provides the ability to model sequential dependencies from both upstream and downstream directions.
- **Attention** helps the model focus on biologically relevant regions such as core promoter elements.
- Combined, they create a powerful model well-suited for promoter prediction tasks.

Evaluation and Results

🧪 Final Evaluation on the Test Set

After training and validating the model using 5-Fold Cross-Validation, we selected the **best-performing model** (in this case, from **Fold 2**) to evaluate on the **global test set**, which had been separated prior to training.

📦 Test Dataset

The test dataset consists of unseen promoter and non-promoter sequences, properly preprocessed and encoded as in the training phase. It was used exclusively for final performance reporting to ensure an unbiased evaluation.

📊 Evaluation Metrics

We evaluated the model using two standard metrics:

- **Binary Crossentropy Loss** – measures how well the predicted probabilities match the true class labels.
- **Accuracy** – measures the percentage of correctly classified sequences

✅ Result Example

(Values to be updated after execution)

- **Test Loss**: ~0.27
- **Test Accuracy**: ~91.5%

🧬 Why Use Attention in DNA Classification?

The inclusion of an **attention mechanism** in our deep learning architecture was not arbitrary. In biological terms, DNA promoter regions often contain **specific motifs** or patterns, such as the **TATA box**, which play a critical role in transcription initiation.

🧠 How Attention Helps

Traditional models like CNN or LSTM treat all parts of the sequence equally or sequentially. However, with attention:

✨ **The model learns to focus** on the most biologically relevant parts of a sequence — areas that may correspond to motifs like:

- TATA box
- CAAT box
- GC-rich regions

⚡ Benefits of Attention in Promoter Detection:

- Highlights **important nucleotide regions** contributing to classification.
- Improves **interpretability** — researchers can analyze attention weights to discover potential motifs.
- Enhances **model performance** by reducing noise from irrelevant parts of the sequence.

🧬 Example Insight (hypothetical)

In several test sequences, attention maps showed that the model gave **higher weights to**

regions upstream of the transcription start site — exactly where biological promoters are expected to be located.

📦 Model Deployment Using Streamlit

🎯 Objective

The goal of this section is to make our promoter prediction model accessible to users through a simple and interactive web interface, without requiring programming skills or knowledge of deep learning.

🛠️ Tools Used

- **Streamlit**: A Python library for building interactive web apps.
- **Trained Keras Model**: The `.keras` model trained using CNN + BiLSTM + Attention.

🚀 Deployment Workflow

1. **Save the Trained Model**
2. **Create the Streamlit App (app.py) :**

   The application file contains the following core functionalities:

- Text input for DNA sequences
- One-hot encoding of the sequence
- Loading the trained model
- Predicting and displaying whether the sequence is a promoter or not

3. Run the App The app is launched using the command:

   streamlit run app.py

💡 Benefits of Using Streamlit

- User-Friendly: Anyone can use the model via the browser.
- Fast Deployment: Minimal setup required.
- Real-Time Predictions: Interactive and immediate.

📌 Limitations and Future Work

- For now, the app only accepts sequences of fixed length (251 bp).
- Future versions may include:
    - Batch processing of multiple sequences
    - Uploading FASTA files
    - Visualization of attention weights

Conclusion

This project presents a complete pipeline for promoter classification using deep learning. Starting from dataset preparation with EPD and Kaggle sequences, we performed preprocessing, encoding, and data augmentation to build a balanced and biologically relevant dataset.

We designed a hybrid CNN–BiLSTM–Attention model to learn complex DNA patterns and evaluated it using k-fold cross-validation to ensure robustness. The model was then deployed using Streamlit, offering an intuitive interface for real-time promoter prediction.

Promoter classification is crucial in genomics as it helps understand gene regulation and supports medical and research advancements. This project proves the effectiveness of attention-based models and shows how AI can make bioinformatics more accessible and impactful.

🔗 Resources and Links

- 📂 **Kaggle Dataset (Promoter/Non-promoter sequences)**
  Link:

  https://www.kaggle.com/datasets/nayanack/promoter-gene-prediction

  https://github.com/AyA3524/IsItAPromoter/blob/main/non_promoters.fasta

- 🧬 **EPDnew Database (Eukaryotic Promoter Database)**
  Link:

  https://epd.expasy.org/epd

  https://github.com/AyA3524/IsItAPromoter/blob/main/epd.fasta

- 📄 **Final Processed Dataset (CSV)**
  Custom `.csv` file containing the balanced dataset used for training and evaluation:
  👉
  https://github.com/AyA3524/IsItAPromoter/blob/main/nv_sequences_balancees.csv

- **GitHub Repository (full code + deployment)**:

  https://github.com/AyA3524/IsItAPromoter

📑 Structure of the Report

Abstract

Theoretical Background

- What is an Attention Mechanism?
- Why Attention in Bioinformatics?
- Why Focus on Promoter Prediction?

Dataset Construction and Preprocessing

- Data sources
- Challenges and Preprocessing Steps

  Normalize all sequences to 60 nt

  Data Augmentation: Techniques and Biological Justification
- Dataset Format and Finalization
- One-Hot Encoding for Neural Network Input
- Conversion to NumPy Arrays
- Dataset Splitting and Cross-Validation

Model Architecture and Training

- Model Overview
- Model Structure: CNN + BiLSTM + Attention
- Training Strategy with K-Fold Cross-Validation

Evaluation and Results

Model Deployment Using Streamlit

Conclusion

Resources and links

Structure of the Report