Unsupervised Representation Learning of Gene Sequences

Bachelor Thesis Project report submitted to

Indian Institute of Technology Guwahati

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Electronics and Electrical Engineering

by

Vishal Agarwal (150108043)

N. Jayanth Kumar Reddy (150102035)

under the supervision of

Dr. Ashish Anand & Dr. Tony Jacob

Department of Electronics and Electrical Engineering

Indian Institute of Technology Guwahati

Guwahati, India - 781039

2018-19

CERTIFICATE

*This is to certify that the project report entitled "**Unsupervised Representation Learning of Gene Sequences**" submitted by **Vishal Agarwal(150108043) & Jayanth Reddy(150102035)** to Indian Institute of Technology Guwahati towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Electronics and Electrical Engineering is a record of bona fide work carried out by him under my supervision and guidance during 2018-19.*

Dr. Ashish Anand
Associate Professor
Department of Computer Science & Engineering
Indian Institute of Technology Guwahati, India

Dr. Tony Jacob
Associate Professor
Deprtment of Electronics & Electrical Engineering
Indian Institute of Technology Guwahati, India

DECLARATION

The work contained in this thesis is our own work under the supervision of the guides. We have read and understood the "B.Tech. /B.Des. Ordinances and Regulations" of Indian Institute of Technology Guwahati and the "FAQ Document on Academic Malpractices and Plagiarism" of Electronics and Electrical Engineering Department of Indian Institute of Technology Guwahati. To the Best of our knowledge, this thesis is an honest representation of our work.

Vishal Agarwal	N. Jayanth Kumar Reddy
150108043	150102035

Date: _____

Place: _____

Acknowledgements

Our earnest gratitude to Dr. ***Ashish Anand*** *and* ***Dr. Tony Jacob*** *for their valuable guidance and support throughout this project. This project would not been possible without their support.*

We are thankful to Mrs. Aparajita Dutta (PhD Scholar, Dept. of Computer Science and Engineering) for her advice on genomic data processing. We would also like to thank all our parents and colleagues for their constant support and advice.

Abstract

The human genome consists of a variety of genes with more than 3 million base pairs. In addition to the presence of various genome, these data are very long and variable in length. This leads to the computational bottleneck due to its inability to handle variable length long data. This calls for the need of unsupervised techniques to unravel hidden structures in data and get some biological insights. In this project, we aim to leverage deep learning techniques to learn a meaningful latent representation of gene sequences in an unsupervised way. We use a sequence-to-sequence LSTM-based autoencoder model to learn representations without labels. We analyze this model quantitatively and qualitatively to validate that the model learns useful representations. The quantitative analysis is done on a supervised task of splice site prediction in two different settings. First, we initialize an LSTM classifier with encoder weights and compare its performance with random initialized weights network and second, we use learned latent embeddings from autoencoder as feature vectors for simple classifier models such as SVM, Feedforward network and RNN on the same supervised task. We show that these representations can be used as features or priors in closely related tasks which improves prediction accuracy and helps the classifier to converge faster, specially in the case where labeled data is very less. Further, we use a model attribution technique *Integrated Gradients* to address the issue of model interpretability. This lets us identify important regions in the sequence and hence discover motifs which induce splicing signals.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the availability of large-scale GPU compute and high volume of data, deep learning methods have been successful in various fields such as computer vision, speech, natural language, and robotics. The advancements in genomic research and data explosion such as high throughput sequencing techniques are constantly challenging traditional methods. In recent years, attention has been shifted from traditional regulatory methods to building computational models that can predict gene functions from sequences directly. In this work, we leverage deep learning techniques to learn latent gene representations in an unsupervised way. To infer that the representations capture meaningful information, we do both quantitative and qualitative analysis of the learned representation. In this chapter, we discuss the relevant biology background - describe the process of splicing, importance of predicting splice sites and the need for unsupervised techniques in genomics. We also discuss basic building block deep learning models and how the problem can be modeled with it.

1.1 Biology Background

DNA sequencing has provided researchers with the ability to read genetic blueprints and understand the process that governs all activities of an organism. The central dogma summarizes pathway from DNA to RNA to Protein. DNA consists of base

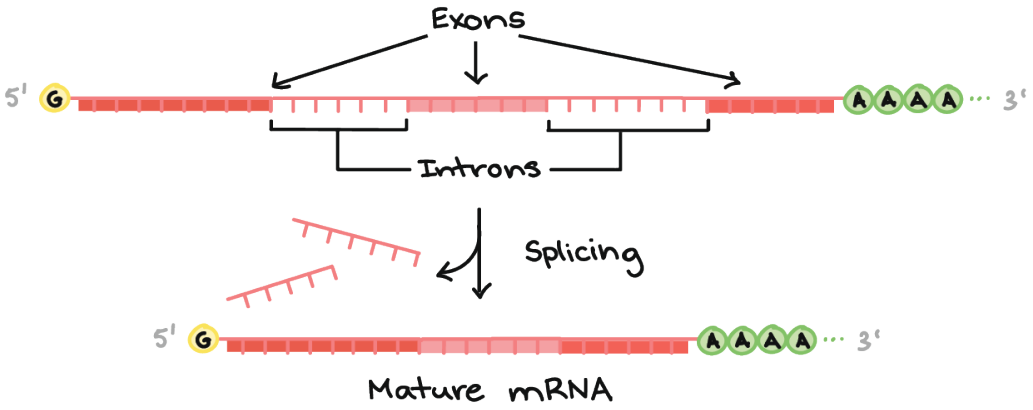


FIGURE 1.1: Illustration of Splicing phenomenon
(Source : Khan Academy)

units (A, G, T, and C), known as nucleotides. In humans, they are organized into 23 pairs of chromosomes. Chromosomes are further organized into DNA segments called genes which is responsible for encoding protein. About 2% of the entire genome encodes for protein and hence it is a key research focus area[1].

1.1.1 Splicing

Genes are composed of introns and exons. Introns are non-coding intervening sequences that do not contain any information and exons are important coding regions of genes that contain protein-encoding information. Splicing is an important process in genomics which leads to protein diversity. It happens at the junction of introns and exons where RNA departs to convert into protein and introns are removed and exons are joined together. At the time of splicing, it is possible that all the exons or flanking regions of genes are not joined properly and some of them are skipped or maybe introns are not removed. This leads to a phenomenon known as Alternative Splicing.

1.1.2 Alternative Splicing

Alternative Splicing is a regulated process whereby exons, parts or exons or normally non-coding intronic or flanking regions of genes are alternatively joined or skipped.

[1] `http://sitn.hms.harvard.edu/flash/2012/issue127a/`

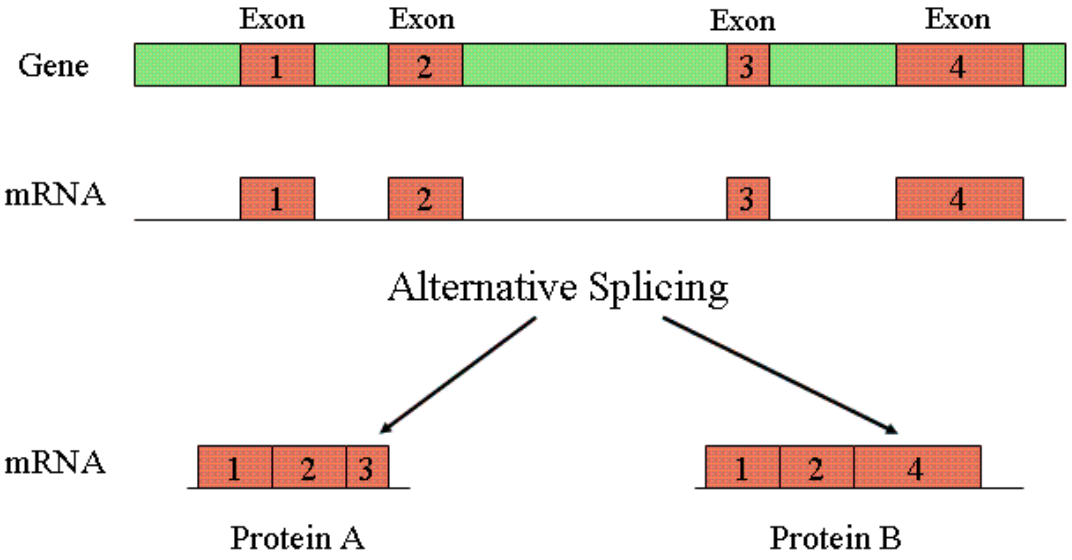


FIGURE 1.2: Illustration of Alternative Splicing
(Source : NCBI)

This enables to generate more than one mature mRNA and distinct proteins are formed known as isoforms.

Splice sites can be functionally categorized as constitutive and alternative sites. A cryptic site which is not normally recognized may exist near a well-known constitutive site[14]. Under the influence of some variant, the cryptic site may be alternatively used instead of the constitutive. This reveals that the utilization of a splice site is affected by the following two aspects. Firstly, the inherent strength of the junction determined by neighboring binding motifs and secondly, the strength of neighboring splice sites that can be alternatively used and compete to be recognized by the spliceosome.

RNA splicing is the primary phenomenon that leads to transcriptional diversity and directs a large number of activities in an organism. Recently, attention has been shifted from traditional regulatory mechanisms to computational models that can predict gene functions from sequences directly. High throughput genomics studies have revealed that more than 95% of genes experience alternative splicing. Mis-splicing may lead to genetic disorders which call for the need to accurately predict splice junctions and analyze signals that influence splicing alternatively[10].

1.2 Deep Learning Review

1.2.1 Recurrent Neural Network

For tasks which involve situations where past information can help in making decision at current time step, using vanilla feed forward neural networks or CNN's is not a very good idea since they cannot model information about past events well. Recurrent Neural Networks (RNN) perform well in such scenarios. The presence of loops in their model architecture allows for information to persist over multiple time steps. The chain like nature as shown in Figure 1.3 makes them ideal of tasks which involve sequences of data. Vanilla RNNs however suffer from a practical drawback, they can't perform well when the sequences are long and suffer from vanishing/exploding gradients problem.

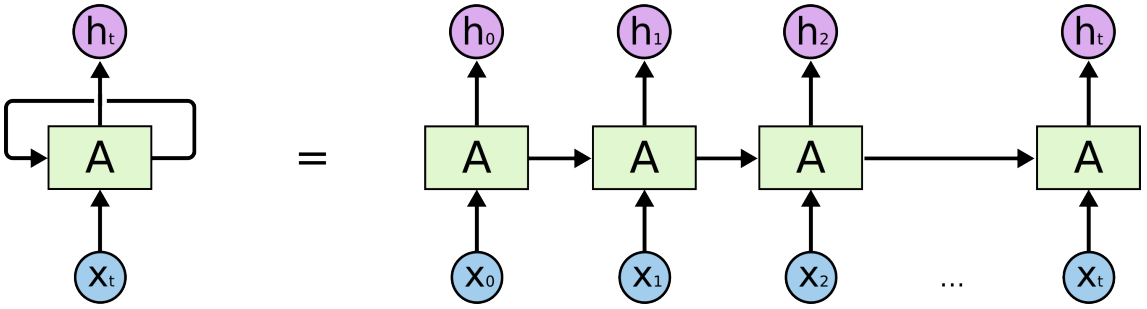


FIGURE 1.3: An unrolled recurrent neural network
(Source : Colah's Blog)

1.2.2 Long Short-Term Memory

Long Short Term Memory (LSTM) [5] is a modification to vanilla RNN, designed to be capable of learning long-term dependencies. The fundamental difference lies in the presence of a cell state and a hidden state, the values of which are updated based on the values of gates which control information flow. For each element in the input sequence, each layer computes the following function:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t c_{(t-1)} + i_t g_t \\
h_t &= o_t \tanh(c_t)
\end{aligned}
\tag{1.1}
$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, $h_{(t-1)}$ is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time 0, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function. The entire process is summarized in Figure 1.4.

These gates decide whether the information from the previous layer is to be modified or allowed to pass unchanged to the next timestep. This gets rid of the vanishing gradient problem.

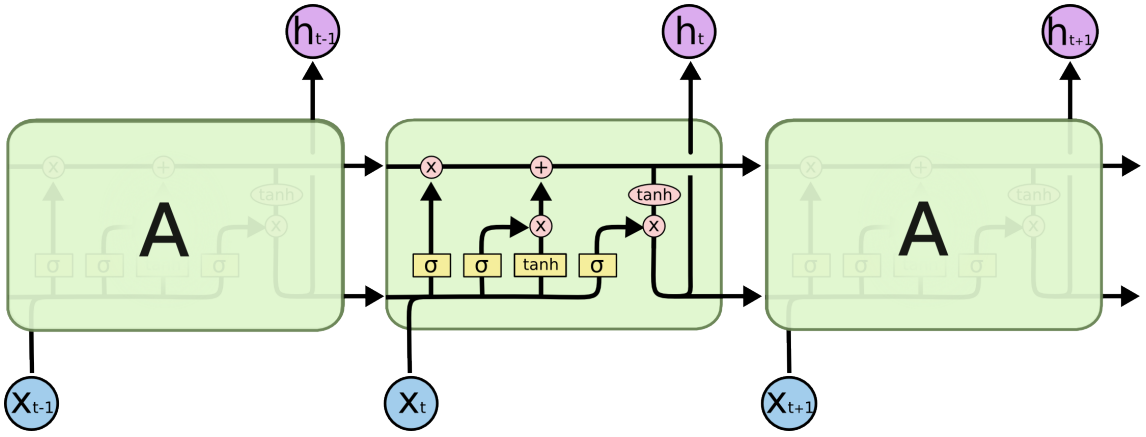


FIGURE 1.4: LSTM
(Source : Colah's Blog)

1.2.3 Bidirectional Long-Short Term Memory

There also exists a variant of LSTM known as Bidirectional Long-Short Term Memory or Bi-LSTM. This type of recurrent model process data from both the ends, i.e.,

from start to end and also from end to start. In most cases, such as natural langugae processing and gene sequencing, both past and future context matters in order to make good predictions. Since traditional LSTMs are unidirectional, they cannot incorporate context appearing in future points. Therefore, Bi-LSTMs provide a good replacement for LSTMs in such cases where the current data depends on both past and future context. Figure 1.5 shows a bidirectional recurrent model.

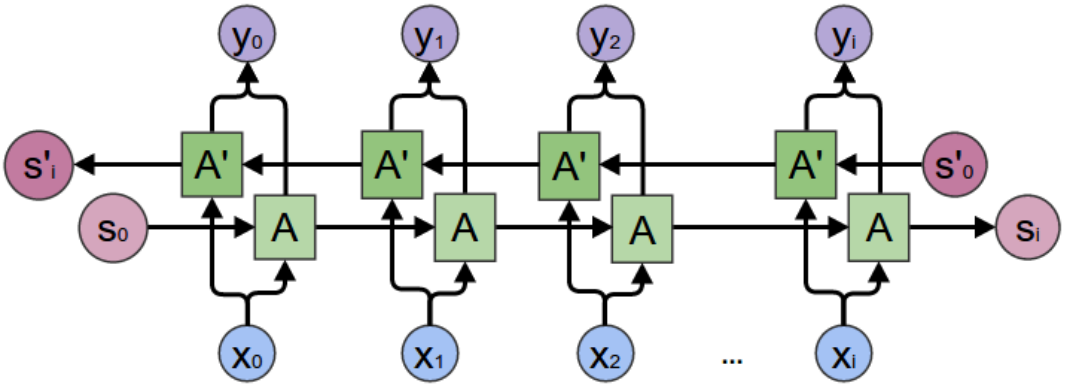


FIGURE 1.5: Bidirectional LSTM
(Source : Colah's Blog)

1.2.4 Autoencoder

Most of the data collected in the world are unlabelled while the success of deep learning has been mostly due to supervised data. One of the key challenges in artificial intelligence that still remains is to learn useful representations from data with no or little supervision. Most of the progress in this regard to representation learning have been made with autoencoder-based unsupervised models. The motivation to unravel hidden structures in unlabelled data, unsupervised learning based on neural networks have become a hot research area.

Autoencoder is an unsupervised learning model where the objective is to learn a mapping from high-dimensional inputs to a lower-dimensional latent space such that the original input can be recovered from the lower-dimensional latent space. The unsupervised nature of learning representation through the reconstruction task enforces a meta-prior which determine how useful the representations in the real world task[2]. The model consists of an encoder which encodes the data from high

to low-dimensional space, and the decoder which reconstructs the original data from lower-dimensional to high-dimensional space.

Let us denote input by $\mathbf{x}$, encoder with function $g_\phi(\mathbf{x})$ parameterized by ϕ and the decoder with function $f_\theta(\mathbf{z})$ parameterized by θ. So the latent representation can be written as $\mathbf{z} = g_\phi(\mathbf{x})$ and the reconstructed output as $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$. The parameters of the model are ϕ and θ which are learned jointly while trying to reconstruct the data same as input. The metrics that can be used to quantify the difference between the actual input and reconstructed output are cross-entropy or simply Mean Squared Loss error. The MSE loss is defined as follows :

$$L(\phi, \theta) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2 \tag{1.2}$$

The modeling choice for encoder and decoder can be a neural network, convolutional networks or recurrent networks depending upon the application. Some variations of autoencoder model are Sparse Autoencoder, Variational Autoencoders (VAE), Denoising Autoencoder, Contractive Autoencoder, etc. Figure 1.6 shows an illustration of a vanilla Autoencoder model.

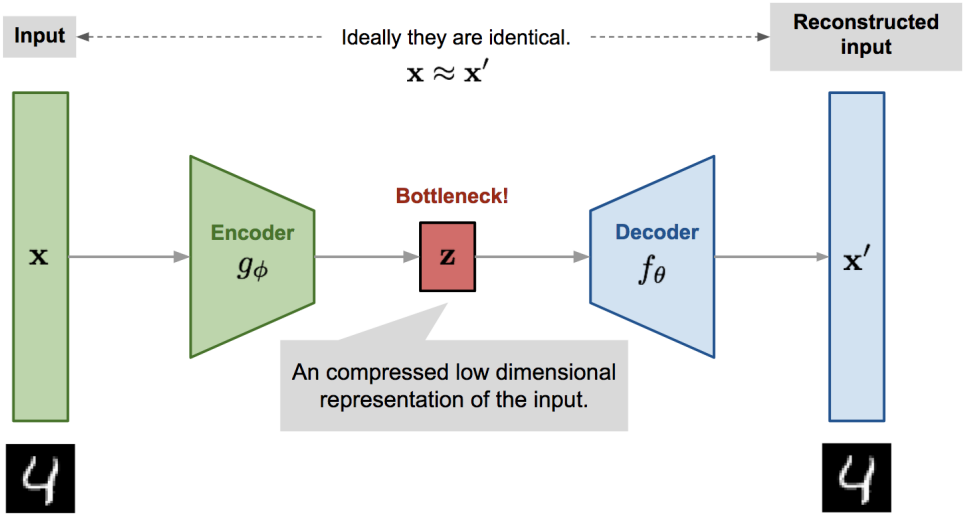


FIGURE 1.6: Autoencoder Model
(Source : Lil'Log)

1.3 Motivation

Gene sequences are very long, variable length sequences which have global as well as local temporal information. These temporal information influence splicing due to the presence of certain motifs in nearby sites. Due to computational inability to handle very long sequence, generally people use a fixed length window around acceptor and donor for any task. This may capture local neighborhood information but fail to incorporate global information and long-range dependencies. Hence, this calls for an unsupervised technique to learn some fixed length representations of gene sequences which incorporate local information as well as long-range dependencies. Further, instead of using raw variable length sequences, we can use the latent fixed-length representations for any task related to genomics since most models cannot handle variable length sequences except RNNs. It is the limitation of biological knowledge which brings in the need of machine learning algorithms to unravel hidden structures in data. Therefore, we believe there is a need to model gene sequences in an unsupervised way and discover motifs responsible for splicing.

1.4 Work Summary

In this work, we show a representation learning approach for problems in genomics. We use a sequence-to-sequence model to learn fixed-length latent embedding of gene sequences in an unsupervised setting without labels. The model includes an encoder and decoder LSTM. The encoder LSTM outputs a fixed-length latent representation which is then used by the decoder LSTM to reconstruct the actual input sequence. The motivation behind this is to capture important features that summarize the input sequence well-enough to be able to reconstruct it back. The learned representations are then quantitatively and qualitatively evaluated to show that it learns some meaningful representations. The quantitative analysis is carried out on a supervised task of splice site prediction in two ways. First, we use the learned encoder weights as initialization to an RNN model, instead of random weights, for splice site classification. Secondly, we use the latent embeddings as features for simple classifier models such as Support Vector Machine, Feedforward network and Vanilla RNN on the same supervised task. The evaluation on a supervised task shows that if

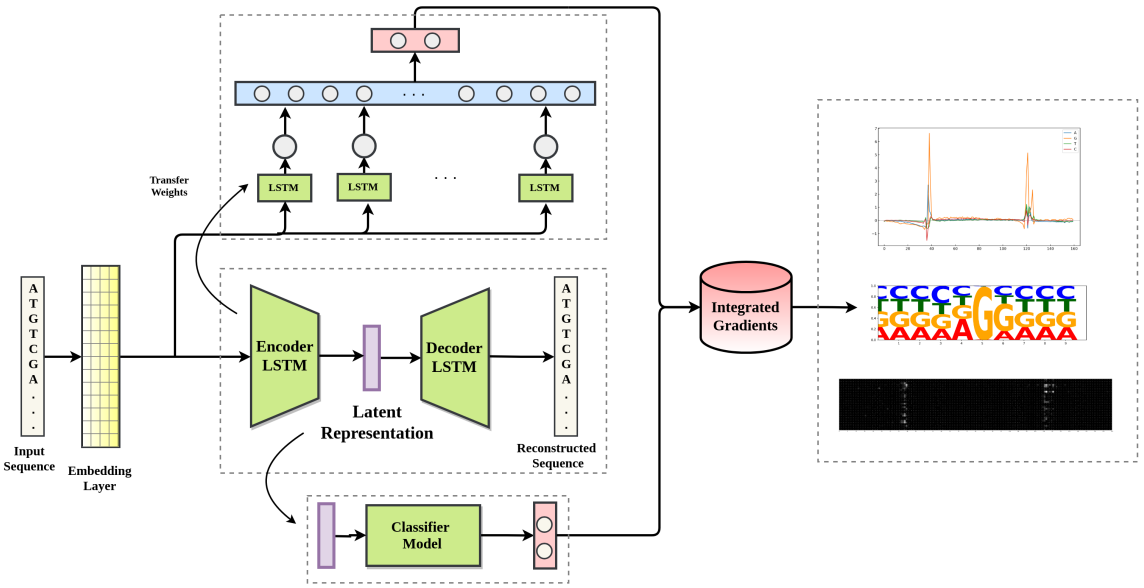


FIGURE 1.7: Graphical Illustration of the entire model

the sequence-to-sequence model learns meaningful representations then the classifier models would show good accuracy. This approach is very helpful in problems where we have very few supervised labeled data. Further, we use axiomatic attribution method known as *Integrated Gradients* to do qualitative analysis. This visualization is done in order to provide attribution to our model and discover motifs which are responsible for splicing.

Chapter 2

Previous Work

2.1 Literature Survey

Several computational based methods have been proposed which predicts splice junction. [6] and [15] used SVM for the splice site prediction task. [15] used SVM with linear kernel while [6] used the RBF kernel. These approaches produced state-of-the-art results during their time but didn't scale to many training examples and non-canonical data. In recent years, due to extensive research in the field of deep learning, many deep learning methods have been applied to this task. Lee et al.[7] proposed a Restricted Boltzmann Machine based model with a new training method called boosted Contrastive Divergence to predict non-canonical splice sites that couldn't be identified by traditional methods. [16] and [1] proposed a convolutional neural network based approach namely DeepSplice and DeepBind respectively. DeepSplice used deep convolutional neural networks to predict both donor and acceptor sites in contrast to earlier traditional methods which predict donor and acceptor sites individually. Dutta et al.[4] proposed a feature learning representation for splice junctions and used MLP for the prediction of splice junctions. Recently, Bretschneider et al.[3] introduced a model named COSSMO which predicts percent selected index (PSI) for competitive alternative splice site.

Most of the works focuses on motif discovery and splice site prediction using supervised models and fixed-length input sequences. This may capture local information

but fails to capture global splicing signals which are responsible for alternative splicing. Also the presence of various genomes, puts limitation in labelling such massive quantity of data. While most work discuss supervised techniques, almost none address the need of unsupervised algorithms. There is a need to learn hidden structures in data in an unsupervised way and get biological insights which will not be possible with traditional deep learning models and limited labelled data.

Chapter 3

Model Description

In this chapter, we provide a description of our model. First, an LSTM-based sequence autoencoder was trained in an unsupervised way to learn fixed-length latent embedding of sequences. This is done in order to learn meaningful representations which capture important motifs in a latent space. Then we evaluate our learned representations on a supervised task in two ways. One, an LSTM model was initialized with trained Encoder LSTM weights and second, the learned representations were used as features for classifiers such as SVM, 2-layer feedforward network and vanilla RNN. The classification task for both of them was splice site prediction. Further, we also do some qualitative analysis by using the axiomatic model attribution technique, Integrated Gradients, proposed by Sundararajan *et al.*[12] to attribute the predictions of our model and hence obtain important regions in sequences with focus in motifs.

3.1 Sequence Autoencoder

We use an autoencoder-like sequence-to-sequence model to learn fixed-length representation of sequences. Our approach is inspired by sequence-to-sequence learning proposed [13] which has been successfully used for machine translation, text summarizing, video analysis, conversational modeling, etc, except that we use it as an unsupervised learning model. The model has an encoder network which encodes

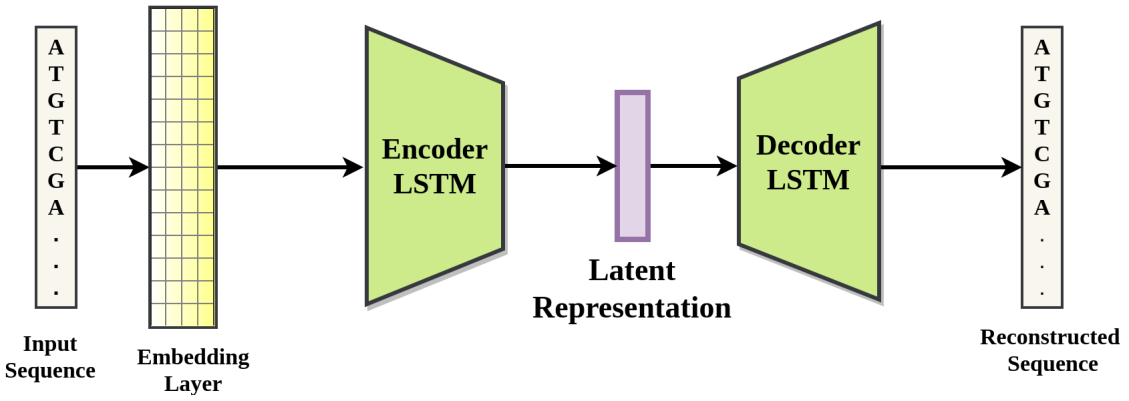


FIGURE 3.1: Sequence Autoencoder

the input sequence into a fixed length representation and a decoder networks which decodes the fixed length representation into the input sequence. The objective is to reconstruct the input sequence.

We use bidirectional LSTM in encoder and unidirectional LSTM in decoder network. The architecture of our model is shown in Figure 3.1. Bidirectional LSTMs work better than traditional LSTMs as it processes the input from both ends, from past to future and from future to past. Using two hidden states, we can preserve information at any time step from both past and future as opposed to unidirectional LSTMs which preserve information only from the past. Bidirectional LSTMs work better as they understand the context better. The encoder network takes in the input sequences and outputs a 128-dimensional representation, summarizing the input sequence. The fixed length representation is then fed into the decoder network which tries to reconstruct back the original sequence, but in reverse order. The reconstruction in reverse order is inspired by the stack representation of the list [11]. To make an analogy, the encoder creates a list by pushing the data in a stack and the decoder unrolls it popping it out. We explore with different models of the decoder network such as whether the decoder LSTMs are conditionally dependent on the generated output.

Another motivation for using LSTM autoencoder is its ability to handle varying length sequences. DNA sequences have varying lengths, some in the order of millions. Supervised learning has been extremely successful for classification tasks and in learning good visual representations. It not only produces good results in tasks for which it is trained for but also for closely related tasks. So it is natural to

extend the supervised learning techniques to DNA sequences but handling such high dimensional data and learning long-range structure is difficult without either collecting a large amount of data or coming up with some clever feature engineering techniques to keep the data dimensionality low. Both approaches are quite tedious and may take months to tackle a particular problem, which is highly inefficient and unsatisfying for a machine learning problem. So there is a need for an unsupervised learning technique to find structure from data and moreover, gene sequences have temporal patterns and contain lots of motifs responsible for various mechanisms which are still unknown to us.

3.2 Splice Site Prediction

After the autoencoder is trained, we perform some experiments and analysis to infer that representations learned by the model capture useful information. The quantitative analysis of learned representations was done by supervised task of splice site prediction in two different settings. The motivation behind this is to infer that the autoencoder representations learned are meaningful and capture useful information. First, an LSTM model was used to detect splice site in a gene sequence. The weights of this model were initialized with the learned weights of the encoder network. We hypothesize this since the autoencoder model learns features from the

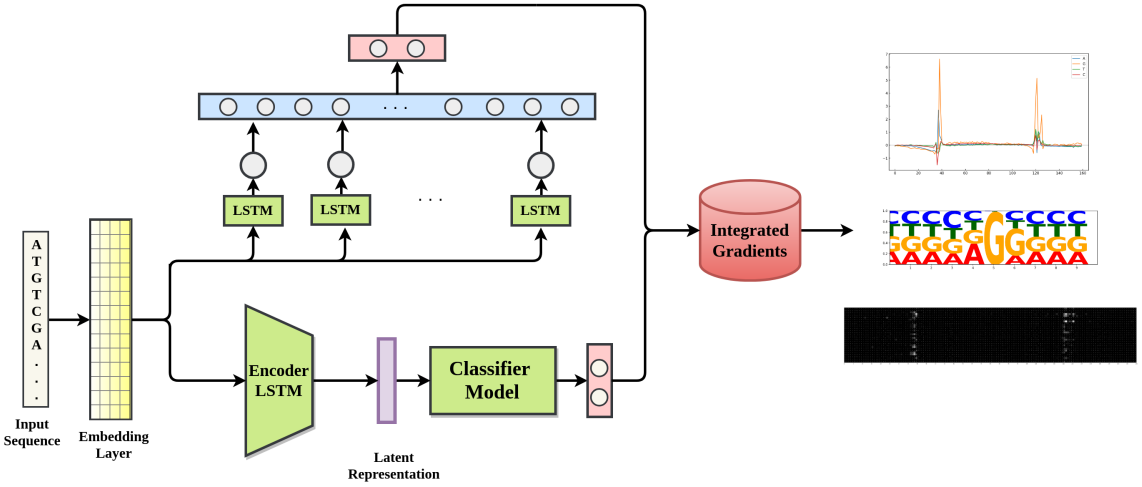


FIGURE 3.2: Graphical Illustration of Quantitative and Qualitative Analysis of learned representations

input sequence in an unsupervised manner. Initializing the supervised model with learned features help in better convergence and boosts performance as gradient takes a shortcut. This increases model capacity without actually adding more layers. In the second task, the latent embeddings are used as features on the same task of splice site identification. The difference between the previous and this model is that the former uses the entire gene sequence as input whereas the latter uses just the fixed length latent embedding as input feature vector. We use simple classifiers such as SVM, 2-layer feedforward network and a vanilla RNN model to conclude the effectiveness of latent representations. If it did capture motif information, then we expect the classifier to perform well. These approaches are helpful and provide good results particularly in cases where the amount of labeled data available is low.

3.3 Integrated Gradients

Sundararajan *et al.* proposed a model attribution technique for deep networks known as Integrated gradients[12]. They also identified two fundamental axioms which should be satisfied by all attribution methods and based on this they devised their method. The fundamental axioms include - Sensitivity and Implementation Invariance.

As deep networks are mostly considered as black boxes, the problem of attribution is important in order to understand the input-output behavior of the network. The way humans perform attribution is to relate that cause with something that has not been happened or is not the case. This serves as a baseline to compare the outcome of the prediction where we have the actual prediction and the prediction where a certain cause has been absent. In this way, we assign blame to that absented cause and provide attribution. Similarly, for deep networks we can consider a baseline input and compare the outcome of the baseline input and actual input to provide attribution to the model. In the case of images, the baseline can be a black image and in case of sequences, it is a zero embedding matrix. Formally we can define the problem of attribution for deep networks.

Suppose a deep network represents a function $F : \mathbb{R}^n \rightarrow [0, 1]$ *and the input is* $x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$. *The prediction of the model is* $F(x)$. *The attribution of the prediction of input* x *with respect to the baseline* x' *is denoted by* $A_F(x, x') = (a_1, a_2, ..., a_n) \in \mathbb{R}^n$ *where* $a_i's$ *are the attribution score or importance of input feature* x_i *to the prediction* $F(x)$.

The two fundamental axioms that the authors identified should be satisfied by every attribution method are Sensitivity and Implementation Invariance.

Sensitivity : There are two parts to this axiom.

1. If the input and the baseline differ in one feature and have different predictions, then that feature should be given a non-zero attribution.

2. If the function approximated by the model does not depend on a specific feature, then that feature should be given a zero attribution.

Implementation Invariance : Two networks or models are known as *functionally equivalent* if the outputs are same corresponding to all inputs, despite having different implementations. *Implementation Invariance* states that two functionally equivalent networks should have same attribution to their inputs. The attribution problem is defined as credit assignment for the output value to the input features and has no relation to the underlying implementation details.

Based on these two axioms, the method *Integrated Gradients* is designed for model attribution. Consider a straightline path from input x to the baseline x' in the input feature space $\mathbb{R}^n$ and compute gradient at each point along the path. Then integrated gradient is the accumulation of the gradients along the path. For the i^{th} dimension input feature, the integrated gradient can be defined as follows.

$$IntegratedGradient_i := (x_i - x_i') \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha(x - x'))}{\partial x_i} \partial\alpha \tag{3.1}$$

The integral of integrated gradients can be approximated by Riemman approximation of the integral. It basically sums over sufficiently small intervals along the straightline path from the baseline x' to the input x. The approximate formula is

as follows.

$$IntegratedGradient_i^{approx} := \frac{(x_i - x_i')}{m} \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m}(x - x'))}{\partial x_i} \partial\alpha \tag{3.2}$$

Here m is the number of steps in Riemann approximation. Generally m is kept between 20 to 300 steps.

The authors have also provided proofs of completeness and uniqueness for the Integrated Gradient method. Please refer to the paper for more details.

Chapter 4

Experiments and Results

4.1 Dataset

We use GENCODE annotations based on human genome data GRCh38 to prepare the dataset. The model is trained on a earlier release version 20 and validated on a newer release version 26 for test after removing all common junction pairs present in version 20. Genome data corresponding to 24 chromosomes namely 1 to 22, X and Y is used. Each of these chromosome consists of multiple genes and each gene is made up of multiple introns. The intron length varied from 1 to about a million in length but we choose only those introns whose length is above 30 since the shortest known intron in humans is of length 30 nt and any intron less than this is due to sequencing error. For true data or positive sequences, we choose intron sequences beginning with GT, corresponding to donor, and ending with AG, corresponding to acceptor site, which have a splice site on both of its end. Each data consists of flanking of some upstream and downstream nucleotide from junction pairs. The junction pairs are extracted from protein coding gene only. This left us with 290,502 positive samples for training data and 5,612 positive samples for testing data.

For negative data generation, we use a technique based on existing works. Negative data was generated based on [16] and [4]. In this approach, false data is randomly sampled from the genome data based on some heuristics. For each false junction pair, the consensus dimer GT and AG is searched randomly such that both donor

and acceptor is in the same chromosome and its distance is in the range of true data length range. We create a huge list of negative sequences and then sampled few sequences such that the number of negative samples is equal to the number of positive samples.

4.2 Unsupervised Learning

The model takes these gene sequences as input and uses it in an unsupervised way without labels to learn latent representations. The one-hot representation is fed to an embedding layer which then goes into the encoder bi-LSTM. The encoder outputs a 128-dimensional vector which is the latent embedding summarizing the input. Then the latent vector is fed to the decoder LSTM to predict the actual input sequence at each time step but in the reverse order. The reverse order prediction is inspired by the stack representation of the list. To make an analogy, the model maintains a stack where the encoder pushes the input data and the decoder pops the data out. At the decoder side, the output is computed as softmax over nucleotide A, T, G, and C.

The loss function criterion used was minimizing the Negative Log Likelihood. The model was trained over 300 epochs with Adam as the optimization algorithm. The entire experiment was performed on NVIDIA TitanXP GPU with 12GB memory. Few example for the input sequence and reconstructed sequence is shown in Figure 4.1.

We also experiment with the dimension of latent embedding varying. We tried various dimensions such as 64, 128, 256 and 512 and found 128 to be giving the best

(a) Input sequence

(b) Reconstructed output

FIGURE 4.1: Illustration of Autoencoder input-output

results. Also, we tried variations of the decoder network such as the input at each timestep depends on the predicted output of previous timestep or the ground-truth output of previous timestep and found the latter to perform better.

The next section describes results on both quantitative and qualitative analysis of our learned latent representation.

4.3 Supervised Splice Site Prediction

In this section, we discuss our quantitative analysis on the learned representations in two different setting for a supervised task of splice site prediction. First, an LSTM classifier was used which takes a gene sequence as input and predicts the presence of splice site. We used 40nt sequences upstream and downstream, both at donor and acceptor side, as input. The one-hot encoding sequences is fed into an embedding layer which goes into a LSTM followed by hidden layer of 1024 units and a 2 unit softmax layer at the output. To show effectiveness of the learned autoencoder, the model is initialized with the learned encoder weights. This provides a good starting point for optimization and leads to a faster convergence. This type of learning is particularly helpful in the case of semi-supervised learning. We compare this model with the similar architecture random initialized model and found the former to perform better. We also experimented with different architectures such as LSTM, bidirectional LSTM and bidirectional LSTM with Attention and compared results for all. Table 4.1 shows the comparison for different type of models.

Secondly, we use latent embeddings as feature vectors on the same supervised task. Since the input for this setting is fixed-length, here we experiment with 3 simple

Table 4.1: Classification accuracy for encoder initialized LSTM model

	Model	Accuracy
Random Weights	LSTM	95.43%
	Bi-LSTM	96.04%
	Bi-LSTM Attention	97.23%
Autoencoder Initialized	Bi-LSTM Attention	99.07%

Table 4.2: Classification accuracy for simple classifier model

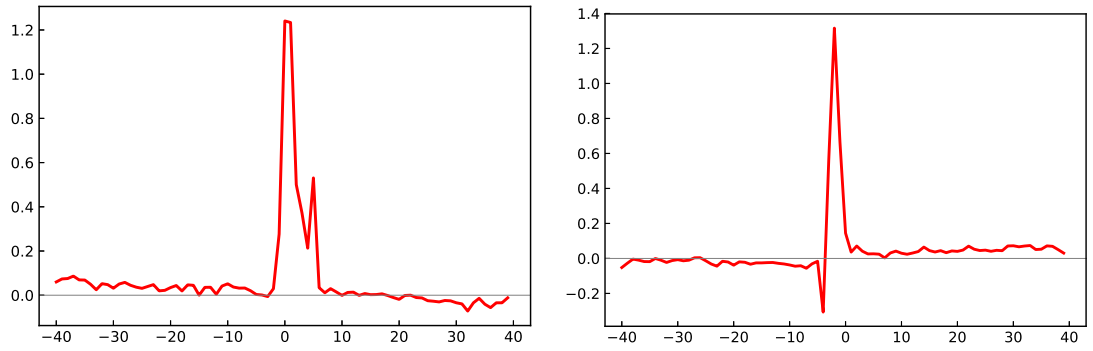
Model	Accuracy
SVM	98.63%
Feedforward network	98.88%
Vanilla RNN	98.93%

traditional classifiers to show effectiveness of the representations as features. The classifier used are SVM, 2-layer Feedforward network and a vanilla RNN. We used 128-dimensional latent embeddings as input to these classifiers. The results for this setting are shown in table 4.2.

4.4 Visualization

This section describes the qualitative analysis of our model to provide attribution of the input sequence to the predicted output of the supervised task. To achieve this, we use popular visualization technique - Integrated Gradients proposed in [12]. The visualizations provide model attribution by capturing important regions in the sequence. This helps us to identify critical motifs present in the sequence. These motifs can be interpreted as signals which influence splicing.

As described in section 3.3, Integrated Gradients is based out of two fundamental axioms - *Sensitivity* and *Implementation Invariance* that attribution methods are ought to follow. Integrated gradient requires a baseline against which it compares the prediction of the network and accordingly provides attribution to the feature which differs in the input and the baseline. In our case, we chose the baseline as zero embedding matrix. We cannot choose a zero input sequence as it is another input example, induces other features and does not relates to the absence of feature for comparison. Computationally, integrated gradient is calculated as Riemann approximation of the straightline path integral by summation over some steps of small intervals. The exact and approximate formula to calculate integrated gradient is given in equation 3.1 and 3.2. Basically, it is the accumulation of the gradient at each point in the straightline path from baseline to input in the input space $\mathbb{R}^n$, then multiplied by the difference in the baseline and input feature value. It signifies

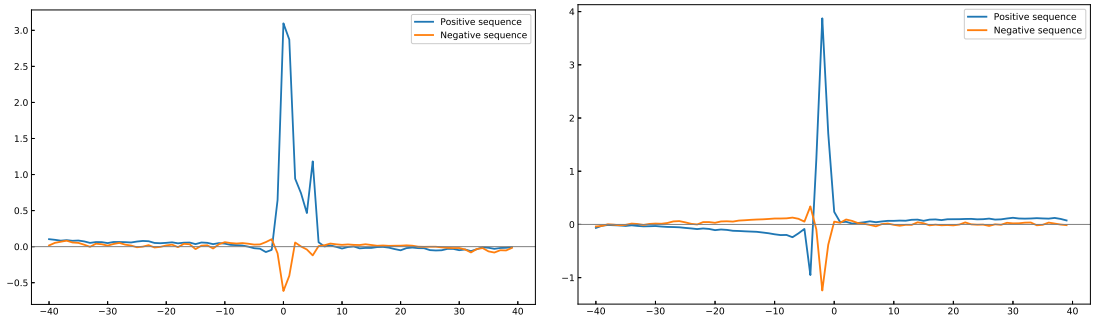


(a) Donor (b) Acceptor

FIGURE 4.2: Average attribution score per position of donor and acceptor sites

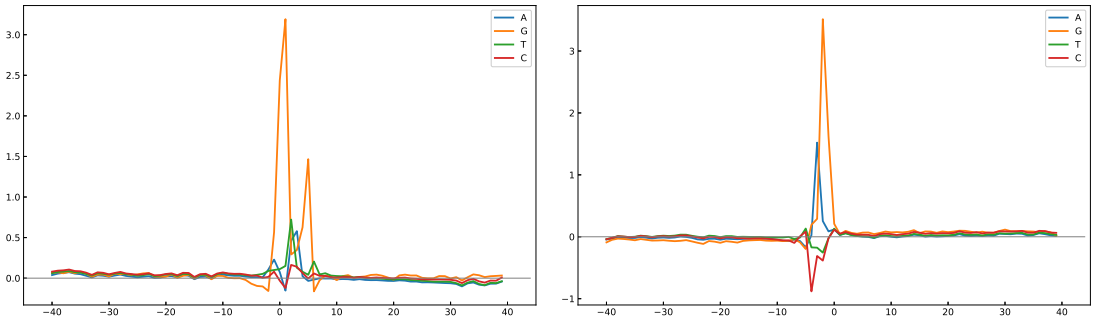
the model attribution of the input feature to the predicted score by the splice site prediction model.

In our experiment, we used 200 steps along the straightline path for gradient calculation. This can be easily implemented in a for loop or even in batches and computing the gradient is very simple and efficient in most deep learning frameworks. The integrated gradients provides attribution of the input feature so to translate it to the input sequence, we take mean of attribution score across all features at each position. As mentioned earlier, we take sequences 40nt window upstream and downstream around both at acceptor and donor. Therefore, it is expected that we get motifs around the position 40 and this is indeed the case. Figure 4.2(a) and 4.2(b) shows the attribution score averaged over all data for donor and acceptor respectively. The similar analysis for positive and negative example sequences are shown in Figure 4.3(a) and 4.3(b). We also perform analysis of attribution score for all nucleotide A, T, G and C separately and the comparison result for both donor and acceptor sites is shown in Figure 4.4(a) and 4.4(b). Finally, we generate sequence logo from the attribution score to identify important motifs or splicing signals that influence splicing. Figure 4.5(a) and 4.5(b) shows sequence logo for both donor and acceptor around the most relevant region given by the attribution score. The donor results show the presence of strong AG signal and the acceptor results shows the presence of strong GT signal with T being most influential around it. This validates the known consensus motif.



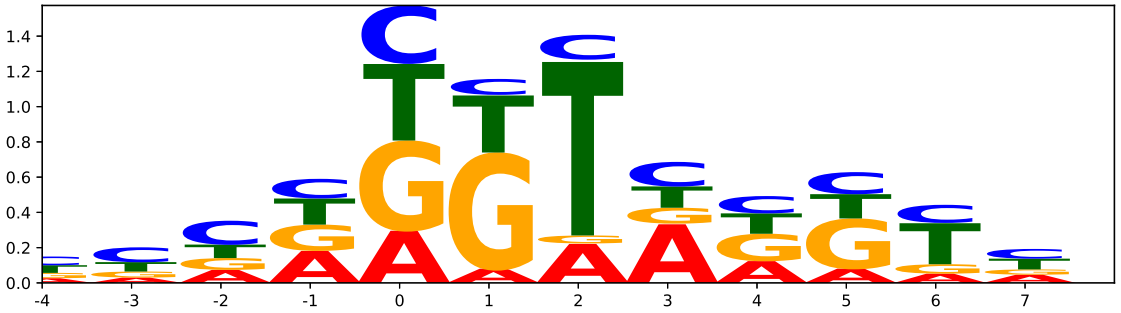
(a) Donor (b) Acceptor

FIGURE 4.3: Average attribution score per position for positive and negative sequences of donor and acceptor sites

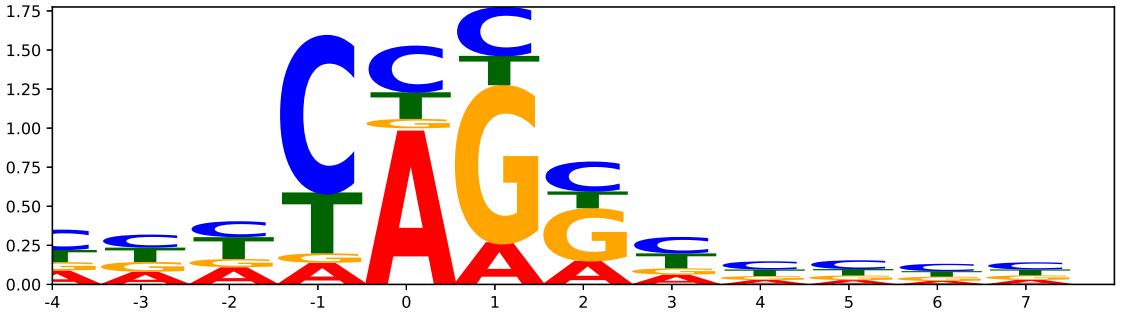


(a) Donor (b) Acceptor

FIGURE 4.4: Average attribution score per position for each nucleotide of donor and acceptor sites



(a) Donor



(b) Acceptor

FIGURE 4.5: Sequence logo to visualize important motifs attributed by the model

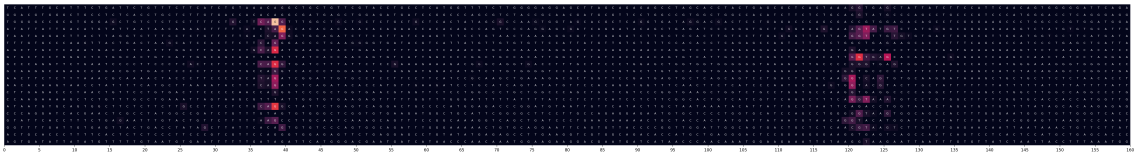


FIGURE 4.6: Visualization of integrated gradient attribution score

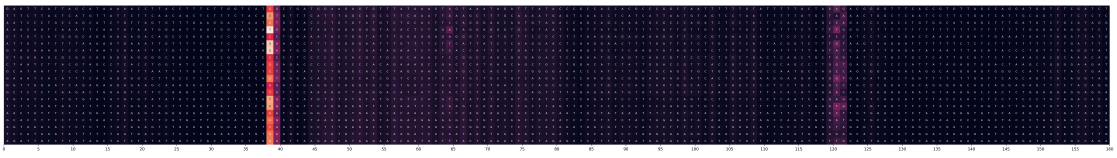


FIGURE 4.7: Visualization of attention weights

Chapter 5

Conclusion

In this work, we presented a unsupervised representation learning approach to learn representations of gene sequences in a latent space. We leveraged deep learning techniques to use an sequence-to-sequence autoencoder-like framework to learn representations in an unsupervised setting. Due to variable length sequences, we use LSTM-based models in both encoder and decoder. The encoder outputs a fixed-length latent embedding of the sequence and using this, the decoder tries to reconstruct the actual input sequence. To ensure that our model learn meaningful representations and capture important motifs in the latent space, we performed some quantitative and qualitative analysis.

In quantitative analysis, we performed splice site prediction in two different setting. First, we used learned encoder weights to initialize a LSTM-based classifier for splice site detection. We compared the encoder-initialized model with the baseline random weights initialized model. As hypothesized, the former performs better in accuracy with much faster convergence as compared to the latter. Initializing with learned encoder weights provides a good starting point for the model to converge and take gradient shortcuts. In the second task, we used the latent embeddings as input feature to the simple classifiers such as SVM, feedforward network and vanilla RNN. If the model learned some meaningful representations, then we expect the classifier to perform well and we find this is indeed the case.

Then for qualitative analysis, we did attribution for the supervised model to find important regions in the gene sequence. We used the axiomatic attribution visualization method Integrated Gradients proposed by Sundararajan *et al.*[12]. Since the data is processed as 40nt window upstream and downstream, it is expected to find splicing signals around the position 40. Integrated Gradient produced an attribution score of the input sequence with a peak around position 40 in both donor and acceptor sequences. This verifies our theoretical assumptions and also concludes that our model did learn some meaningful and useful representations.

The possible applications of our work could be to use latent embeddings for various problems in genomics instead of using directly the one-hot encoded sequences. The gene sequences are often very long and also variable in length. This eliminates the possibility to use various models since most models cannot handle variable length sequences. Besides, it also adds a huge computational bottleneck due to its very long nature. In order to tackle this issue, people mostly used fixed length sequences by windowing around donor and acceptor sites. Although this captures local information but fails to capture global and long range information. Therefore, using fixed-length latent embeddings learned from full length sequences bypasses all mentioned hurdles and provides more general representation.

Next we describe some extensions or ideas of the work that we would like to pursue in future.

1. Study "de-noising" of the sequences using explicitly added noise in the latent embedding and then reconstruct the sequence.

2. Using Variational Autoencoders, we can generate the probability distribution of latent embeddings and sample unknown adversarial examples.

3. One of the challenging problem related to splicing is to understand the signals influencing Alternative Splicing. So it is desirable to formulate a model which can predict the probability of a given acceptor site over combining with multiple donor sites and vice-versa.

Bibliography

[1] Babak Alipanahi, Andrew Delong, Matthew T. Weirauch, and Brendan J. Frey. Predicting the sequence specificities of dna- and rna-binding proteins by deep learning. *Nature Biotechnology*, 33:831 EP –, Jul 2015.

[2] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013.

[3] Hannes Bretschneider, Shreshth Gandhi, Amit G Deshwar, Khalid Zuberi, and Brendan J Frey. Cossmo: predicting competitive alternative splice site selection using deep learning. *Bioinformatics*, 34(13):i429–i437, 2018.

[4] Aparajita Dutta, Tushar Dubey, Kusum Kumari Singh, and Ashish Anand. Splicevec: Distributed feature representations for splice junction prediction. *Computational Biology and Chemistry*, 74:434 – 441, 2018.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[6] J. Huang, T. Li, K. Chen, and J. Wu. An approach of encoding for prediction of splice sites using svm. *Biochimie*, 88(7):923 – 929, 2006.

[7] Taehoon Lee and Sungroh Yoon. Boosted categorical restricted boltzmann machine for computational prediction of splice junctions. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2483–2492, Lille, France, 07–09 Jul 2015. PMLR.

[8] Chris Lonsdale, Richard R. Rosenkranz, Louisa R. Peralta, Andrew Bennie, Paul Fahey, and David R. Lubans. A systematic review and meta-analysis

of interventions designed to increase moderate-to-vigorous physical activity in school physical education lessons. *Preventive Medicine*, 56(2):152 – 161, 2013.

[9] Christoher Olah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015.

[10] Marina M. Scotti and Maurice S. Swanson. Rna mis-splicing in disease. *Nature Reviews Genetics*, 17:19 EP –, Nov 2015. Review Article.

[11] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 843–852. JMLR.org, 2015.

[12] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[14] Rita Vaz-Drago, Noélia Custódio, and Maria Carmo-Fonseca. Deep intronic mutations and human disease. *Human Genetics*, 136(9):1093–1111, Sep 2017.

[15] Masaki Yamamura and Osamu Gotoh. Detection of the splicing sites with kernel method approaches dealing with nucleotide doublets. *Genome Informatics*, 14:426–427, 01 2003.

[16] Yi Zhang, Xinan Liu, J. N. MacLeod, and Jinze Liu. Deepsplice: Deep classification of novel splice junctions revealed by rna-seq. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 330–333, Dec 2016.