Suitably impressive thesis title

1044935

St Cross College

University of Oxford

Submitted in partial completion of the

MSc in Computer Science

Trinity 2020

Contents

List of Abbreviations

1-D, 2-D . . . One- or two-dimensional, referring in this thesis to spatial dimensions in an image.

Otter . . . . . One of the finest of water mammals.

Hedgehog . . . Quite a nice prickly friend.

Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...

There is no one who loves pain itself, who seeks after it and wants to have it, simply because it is pain...

— Cicero's *de Finibus Bonorum et Malorum*

1 Introduction

Contents

2
Background

Gene expression is fundamental to all life. It is the process whereby a sequence of nucleotides is used to direct the synthesis of a functional gene product (protein, functional RNA). Gene expression occurs in two steps: during transcription, the DNA is transcribed into messenger RNA (mRNA) and during translation, the mRNA is decoded into proteins.

In more detail, during transcription, an initially transcribed precursor mRNA (pre-RNA) is translated into a mature RNA by a process called splicing. Splicing is based on DNA being made up of exons (predominantly coding regions), and, typically longer, introns (non-coding regions). Only exons are contained in the mature mRNA. Introns are still contained in the initially transcribed precursor mRNA (pre-mRNA). However, they are spliced out by the spliceosome to form the mature mRNA. The spliceosome is a complex molecular machine consisting of as many as 150 proteins [2]. This is visualized in 2.1.

Exons which are always included in the mRNA are called constitutive exons. However, 95% of human genes with multiple exons are alternatively spliced, that is, they may only sometimes be included or may be included with different splice sites. The most common types of alternative splicing in higher eukaryotes are [3][4]:

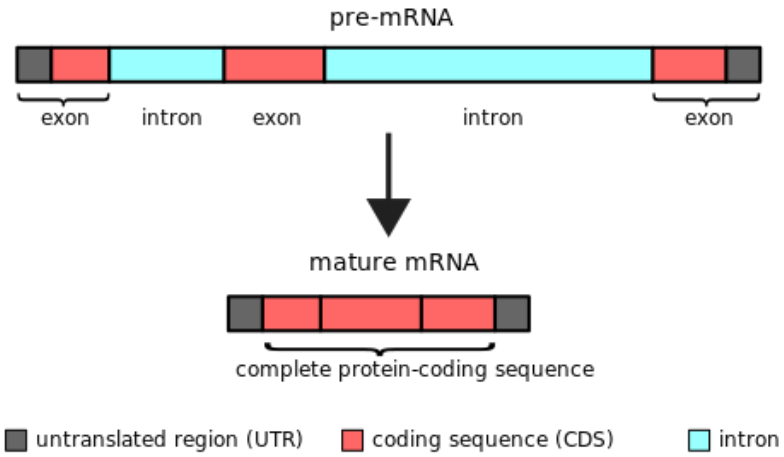

Figure 2.1: The process of splicing [1]. Introns are removed from the pre-mRNA to obtain the mature mRNA only consisting of exons. Apart from coding regions, exons may also consist of non-coding untranslated regions (UTRs). Like introns, UTRs influence gene expression.

- Cassete exons are exons who are sometimes included in the mature mRNA and sometimes skipped. This is the most common form of alternative splicing in higher eukaryotes (so also humans), accounting for roughly 40% of all AS splicing events [2].

- Exons with an alternative 3' or 5' splice-site. The 3' splice site or splice junction is the end of the exon towards the 3' end of the RNA strand (typically towards the right). The 5' splice site or splice junction is the end of the exon towards the 5' end of the RNA strand (typically towards the left). An alternative 3' or 5' splice-site may be located deeper inside the exon or outside the exon in a typically intronic region. Alternative 3' and 5' site splicing respectively constitute approximately 18% and 8% of all AS splicing events in higher eukaryotes [2].

- intron retention, that is, when an intron between exons is not spliced out. It accounts for roughly 5% of AS activity in higher eukaryotes [2].

Different forms of alternative splicing are visualized in Figure 2.2. More complex forms of alternative splicing, such as mutually exclusive exons, also exist, but they are currently believed to be more uncommon. Alternative splicing occurs in nearly

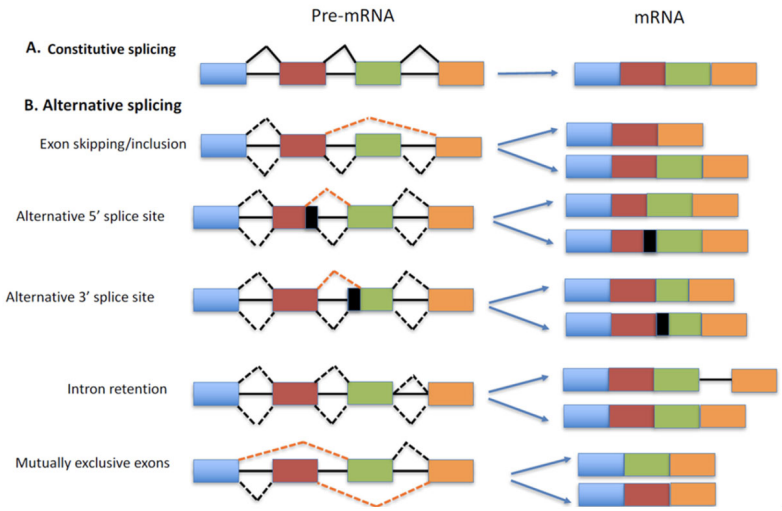

Figure 2.2: Visualization of the most common forms of alternative splicing and the resulting different possible mature mRNAs [5].

all organisms that carry out pre-mRNA splicing as such as plants or animals and its frequency varies across organisms [2].

Why does alternative splicing occur?

Alternative splicing enables a single gene to encode multiple protein variants. This massively contributes to proteomic diversity. For instance, the roughly 20,000 humans genes are estimated to encoder over 100,000 different proteins [2].
Alternative splicing may also speed up the rate of evolutionary adaption. Due to alternative splicing, a gene may evolve to fulfil a different functionality without first needing to evolve a separate copy of the same gene. [6]

How is alternative splicing regulated?

Alternative splicing was discovered 40 years ago [7], but the molecular mechanisms governing it are still poorly understood. It is known that the spliceosome recognizes exon-intron boundaries based on the 5' and 3' splice sites, the branch site located in roughly the middle of the exon, and the polypyrimidine tract located upstream of the 3' splice site. However, estimates suggest that these four factors only account for half of the information required to determine splicing behaviour. The rest is likely accounted for by intronic or exonic, cis-acting sequences of the pre-mRNA which

bind to trans-acting factors. These cis-acting sequences are usually 4-18 nucleotide long and classified as exonic splicing enhancers or silencers **??**. However, the dynamic interaction between cis-acting and trans-acting factors is highly complex, new factors are still being found and thus a lot more work needs to be done if we want to fully understand alternative splicing.

What happens when splicing is misregulated?

Since alternative splicing is such a fundamental mechanism, its correct execution is crucial. Defects in splicing are typically caused by genomic sequence variations leading to misregulation of the splicing process. An estimated 9%-30% of Mendelian disorders may act through disruption of splicing [8] Splice variants have also been shown to be biomarkers for multiple types of cancers [9]. As a result, alternative splicing has also been suggested as a biomarker and potential target for drug discovery [10].

Importance of understanding splicing

Thus, there is great interest in better understanding the mechanisms underpinning alternative splicing. Due to rapid advances in RNA-sequencing technologies, it is now possible to sequence the genome of a patient within a day. However, the genomic variants (compared to a reference genome) observed in patients are often variants of unknown significance. [6] That is, it is unknown whether these variants are pathogenic or benign. An improved understanding of alternative splicing may improve the classification of genomic variants and help with the diagnosis of patients, especially those with rare genomic diseases.

3
Related work

Splicing codes are computational models that attempt to predict splicing behaviour based on putative regulatory features (such as sequence motifs). They were first introduced in the seminal papers by [11][12]. Their introduction was motivated by the recognition that splicing is highly condition-specific and regulated by the complex interaction of many factors in such a way that it is only feasible to model this behaviour computationally.

[11] focus on cassette exons and attempt to predict the change in splicing behaviour for a given exon between different tissues. They popularized the use of the quantitative measure PSI or Ψ to describe splicing behaviour which is still commonly used today: Ψ is defined as the proportion of transcripts out of all transcripts that contain a given exon [13]. Given a random transcript, PSI denotes the probability of a particular exon being included or excluded.

Similarly, Ψ_5 is defined as the number of transcripts containing a particular alternative 3' splice site for a fixed 5' splice site. Ψ_3 is defined analogously as the number of transcripts containing a particular alternative 5' splice site for a fixed 3' splice site. Ψ_5 and Ψ_3 are particularly interesting to model the competition between different alternative splice sites. To quantify the change of splicing behaviour between conditions, these models predict the corresponding $\Delta\Psi$. They were able to find novel regulators of key genes associated with diseases and to predict how

genetic variants will affect splicing [14] [15]. Input to the model are over 1000 known and unknown motifs and higher-level features (such as exon/intron lengths and phylogenetic conversation scores) selected partially from previous studies and partially from de novo searches.

Improving upon these first models, the second 'generation' of splicing codes used several common and uncommon machine learning algorithms such as multinomial logistic regression, support vector machines (SVM) and Bayesian Neural Networks (BNN) to predict changes in alternative splicing behaviour. [16] Among these, BNNs were able to outperform the other methods when evaluated on a microarray dataset based on mouse data. In contrast to models from the first generation, BNNs based models only took in sequence information and very high-level features like tissue type which meant that the model was automatically able to learn relevant motifs from the data.

However, BNNs often rely on expensive sampling methods like Markov Chain Monte Carlo (MCMC) to be able to sample models from a posterior distribution. It can be challenging to scale these methods to larger datasets and a large number of hidden variables. As a result, the third 'generation' of splicing codes relies on deep learning models which can effectively make use of the large amount of data available with the advent of high-throughput RNA-sequencing technologies. First forays into using deep learning-based models were made by [17]. Using a Deep Neural Network (DNN) with an autoencoder, they were able to improve upon the results achieved by a BNN model. Albeit [17] initially used a different dataset and a different task formulation than [16], [18] were able to show that these improvements also lasted when directly comparing the models on the same dataset using the same task formulation. Furthermore, [18] developed a framework for integrating further experimental data, like data from CLIP-seq based measurements of in vivo splice factors bindings, into the model developed by [17]. Adding these further features improved the explained variance in splicing behaviour between tissues, as measured by the R^2 score, by roughly further 5% to an overall average value of 43.4%. Taking inspiration from advances in Natural Language Processing, [19]

developed splicing codes based on the automated feature learning approach from word2vec and doc2vec. Developing two models, one based on doc2vec and a simple MLP, and one based on word2vec and the all-convolutional Inception architecture known from Computer Vision [20], they were able to achieve an average R^2 score of 69.2% significantly improving upon the predictive power of previous models.

In contrast to these splicing codes which predict the (differential) inclusion frequency of an exon, a parallel strand of research focuses on splicing codes for distinguishing between constitutive and alternatively spliced exons. Concretely, for the first task the dataset the models are trained on only consists of alternatively spliced cassette exons and the models have to find features that are predictive of the exact inclusion rate of an exon.
For the second task, the dataset consists of alternatively spliced as well as constitutive exons and the models have to find features predictive for distinguishing between constitutive and alternatively spliced exons. While there is a large overlap between these features, there are also differences. For predicting the inclusion level of an exon, features from the cassette exon and the surrounding exons have shown been reported to be required. [14] For predicting whether an exon is constitutive or not, features around the cassette exon itself have been shown to be the most critical. [21]

[22] used 262 features extracted from an exon and it's two flanking introns to train an SVM-based splicing code for distinguishing between constitutive exons, cassette exons and exons with an alternative 5' or 3' splice site. The dataset used to train the model was based on roughly 4 million ESTs and known isoforms, as well as the alternative events, track (Alt Events) of the UCSU Genome Browser.
Their model achieved very impressive results with an AUC of roughly 0.94 when differentiating between rarely included and constitutive exons, but performance decreases to roughly 0.60 when distinguishing between frequently included and constitutive exons. [23] improved upon this work by using a deep learning model which was automatically able to learn relevant features from the raw sequence. Their

model was based on a combination of convolutional blocks for feature extraction as well as an MLP for classification based on the extracted features. Training on a similar EST-based dataset, their model is significantly more robust when distinguishing between highly included cassette exons and constitutive exons with the AUC only dropping to 0.85. When distinguishing between rarely included cassette and constitutive exons, it was still able to achieve an impressive AUC of 0.92.

4 Methods

This chapter introduces the primary data sources, describes how these were used to obtain the training datasets, and presents the used models. It also gives implementation details regarding processing steps, model implementation, and model training. In this work, we focus on the splicing prediction of cassette exons. Previous work indicates that advances in predicting one type of alternative splicing behaviour via Machine Learning methods also translates to advances in prediction for other types. Therefore, a practical choice is to only focus on one splicing type as this reduces the number of experiments and needed training time by two to three factors. As noted, cassette exons are the most common form of alternative splicing in higher eukaryotes. Thus, for reasons of practicality we choose cassette exons as the type of alternative splicing, we will focus on.

4.1 Datasets

Three primary sources for genomic data were used in this study: the Human Exon splicing events (HEXEvent) database [24], data from the Genotype-Tissue Expression (GTEx) project [25] and data from the Human Induced Pluripotent Stem Cell Initiative (HipSci) [26]. Except for the HEXEvent database, none of these primary sources directly report the PSI values of exons or junctions. The processing

done to estimate the PSI value-based from the primary sources is described in section [estimating PSI]. All of the data sources required further processing (e.g. extraction of corresponding nucleotide sequences) to obtain the final samples which are the input for the models. These further processing steps are described in section 4.2.2.

4.1.1 HEXEvent database

The HEXEvent database contains genome-wide exon data sets of human internal exons which can be filtered for selected splicing events (e.g. constitutive or cassette exons) via an online user interface. It was compiled based on known mRNA variants as defined by the UCSC Genome Browser (newest version is hg38) as well as their associated available expressed sequence tag (EST) information. It was chosen as a data source for direct comparability with the baseline paper [23]. TODO critizie est data these are sequence reads datasets based on which I will estimate the PSI values

4.1.2 GTEx

The Genotype-Tissue Expression (GTEx) project provides the most comprehensive database for tissue-specific gene expression and regulation available to-date. It contains over 17000 samples from nearly 1000 human donors. Each sample was taken from up to 54 different tissue sites. The sequencing samples were obtained using mainly molecular assay-based techniques like Whole Genome Genotyping (WGS), Whole Exome Sequencing (WES) and RNA-Seq.

Processed data which can not be used to identify the donors is publicly available on the GTEx portal. To access raw data (e.g. raw RNA-seq reads) and meta-information about the samples one is required to undergo a data access request. It is intended that data access is requested by PIs or leader of research groups for their whole lab. Approval of a data access request usually takes upwards of 3 months. The co-collaborators Prof. Wilfred Haerty and Prof. Elizabeth Tunbridge have access to the protected part of the GTEx data. However, the scope of projects for which they were granted access does not include this thesis and changing the scope would require undergoing the data access process again. For this reason,

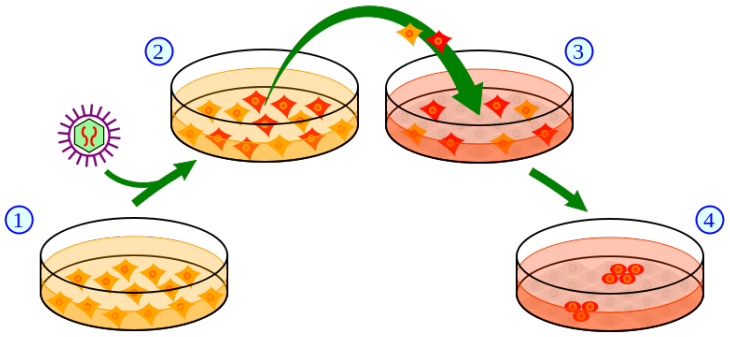

Figure 4.1: (1) Cell culture of donor cells is grown. (2) Transduce the genes associated with reprogramming into the cells via viral vectors. (3) Isolate the cells expressing the transduced genes (in red) and culture them according to embryonic stem cell culture. (4) A small subset of the transfected cells become iPSC cells. Graphic and description adapted from [**ipscprocess**].

we only use the publicly available part of the GTEx data, in particular the files containing information about exon-exon junction reads.

4.1.3 HipSci

The Human Induced Pluripotent Stem Cell Initiative (HipSci) is a large, mostly freely-available repository of human induced pluripotent stem cell (iPSC) lines. iPSC cells are mature cells which have been reprogrammed to again become pluripotent (undetermined). Figure 4.1 shows how iPSC cells are obtained. As iPSC cell lines are not directly obtained from human donors, but rather from building a cell culture based on some initial donor cells, the HipSci dataset does not suffer from the data privacy issues which prevented access to the raw RNA-seq reads as in the GTEx data.

Over 300 cell lines from over 300 donors along with their RNA-seq reads are publicly available from the HIPSCI portal. From these we selected two groups of roughly 20 biological replicates each: Neuron cell lines: One group of biological replicates were taken from undifferentiated iPSC cells from donors from which at least two cell lines are available. This allows testing how well the learned features of the model translate when applied to another library of the same tissue type from the same individual. iPSC cell lines: One group of 25 biological replicates were taken from iPSC cells differentiated to sensory neuron cells. This allows testing how

well the learned features of the model translate when applied to another library of different tissue. iPSC background probably here: some section about iPSC cells as relevant later on – might skip all of this: The Nobel Prize in Physiology or Medicine of 2012 was awarded to Shinya Yamanaka and Sir John Gurdon "for the discovery that mature cells can be reprogrammed to become pluripotent". Pluripotent cells are cells which can differentiate into other cell types. process: take cell samples, then grow cell culture, make them pluripotent and use the pluripotent cells to differentiate into other ones

4.2 Data processing

4.2.1 Estimating PSI

An often used quantitative measure is per cent spliced-in (PSI or Ψ). Ψ is defined as the proportion of transcripts out of all transcripts that contain a given exon [13]. In other words, given a random transcript, it denotes the probability of a particular exon being included or excluded. Similarly, Ψ_5 is defined as the number of transcripts containing a particular alternative 3' splice site for a fixed 5' splice site. Ψ_3 is defined analogously as the number of transcripts containing a particular alternative 5' splice site for a fixed 3' splice site. Ψ_5 and Ψ_3 are particularly interesting to model the competition between different alternative splice sites.

High-throughput sequencing methods are a potent tool to investigate RNA expression and post-transcriptional regulation. However, due to limited coverage, short read length, and experimental biases they are not able to provide a full view of post-transcriptional processing so far [27].

Naive PSI estimation

Let #IR be the number of reads giving evidence for a particular exon being included. Let #ER be the numbers of reads giving evidence for a particular exon being excluded. A PSI value of 100% indicates a constitutive exon which

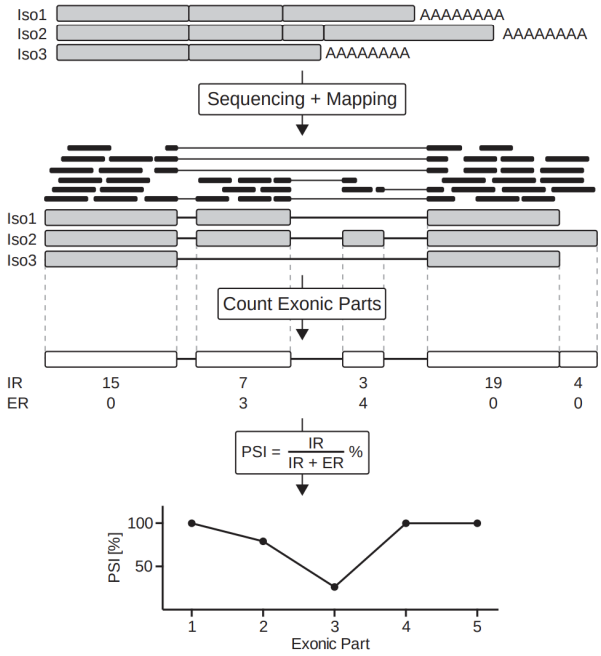

Figure 4.2: Process of estimating PSI based off RNA-seq reads [27]. The reads are mapped to the transcript. Based on annotation files, the reads overlapping exon/intron junctions are classfied as including or excluding reads for a particular exon. Based on the observed IR and ER, the PSI for a given exon is estimated.

is always included, a score below 100% includes an alternatively spliced exon. PSI can then be estimated as:

$$PSI = \frac{\#IR}{(\#IR + \#ER)}$$

Figure 4.2 illustrates the process of estimating PSI based on RNA-seq reads. The advantage of this estimate lies in its simplicity and flexibility. It is quick and easy to implement (hopefully bug-free). It is independent of library size. It can easily be adapted to estimate the PSI of a junction by redefining #IR and #ER to count the inclusion and exclusion read for that junction.

However, this estimate has various issues.

1. It doesn't account for uncertainty in the estimate. A rarely expressed gene may only experience a few reads of a particular exon leading to a very uncertain estimate. For instance, if we observe 1 IR and 1 ER for exon E1: $PSI_E1 = \frac{1}{2} = 50\%$. Compare this to E2 where we observed 15 inclusion and

15 exclusion reads: $PSI_E2 = \frac{15}{30} = 50\%$.

The estimate of PSI_E2 is likely to be more accurate, but this information is not contained in the estimate. Concretely, these two samples would be treated as equally important by the network even though perhaps they shouldn't. A further processing step to only count exons who experienced at least threshold number of reads only alleviates this problem. One would need to account for the expressiveness of each gene because 10 reads in a lowly expressed gene may be as significant as 20 reads in a highly expressed gene. There are several measures for the expressiveness of a gene: Reads Per Kilobase Million (RPKM), Fragments Per Kilobase Million (FPKM) and Transcripts Per Kilobase Million (TPM) and although TPM is the most common one, it is not obvious that it is the best choice in this scenario.

Assuming that one obtains a read count normalized by the gene expressiveness for each exon, there is not a principled way to choose the cutoff threshold. The choice between a threshold which e.g. filters 5% or 20% of the samples is a trade-off between data quality and training samples.

2. Reads which align purely to the flanking constitutive exons are ignored. While these could have occurred in either isoform, they provide latent evidence for whether the cassette exon was included or not. In isoforms where it occurred, the total length of the isoform is longer which means the reads are distributed over a larger area. This leads to a comparatively reduced proportion of reads across the flanking exons when the exon was included and vice versa. The estimate neglects to take this information into account.

3. Related to 1), typically multiple samples or biological replicates of a given experiment are available. It is desirable that reads across multiple samples can be integrated to give a more well-adjusted estimate. How to best achieve this is not obvious. Read depths between multiple libraries may vary and this should be accounted for. [possible other issues here and then make this another itemization]

4. IR and ER must be normalized for exon length to obtain meaningful results. For a long exon, the majority of reads will be IR because they can be located over a much larger area than the ER who must overlap with the 0-length feature of the splicing junction. This can be accounted for by normalizing for the possible number of start positions for each read population:

$$\#IR_norm = \frac{\#IR}{(exonlength + readlength - 1)}$$

$$\#ER_norm = \frac{\#ER}{(readlength - 1)}$$

Note that this assumes that reads are uniformly distributed across a transcript isoform. It is well-known that RNA-seq reads are biased .... [get paper from Liz/Wilfred]

Due to these issues relating to the uncertainty, to not making use of all of the available evidence and to integrating reads across multiple samples, this estimate is problematic.

How I implemented PSI estimation

1) Only take reads from genes across a certain TPM into account (because random spurios reads from low TPM genes could have an outsized effect on the estimation). Say that you weight reads according to the respective TPM. Show TPM distribution and say what TPM threshold you use.
2) Say how there is nothing you can do, due to only having access to the junction reads themselves.
3) Unclear how to best do this and skipped for now. However, basing estimatin based on the sample with the maximum total amount of reads.
4) Say that you estimate read length to be 150 and do this.
TODO: mention junction datasets Several approaches have been developed which try to address these issues in estimating PSI. However, many of these focus purely on the differential splicing changes between conditions (in the form of delta PSI) and don't directly report the PSI in a given condition which is what we are initially

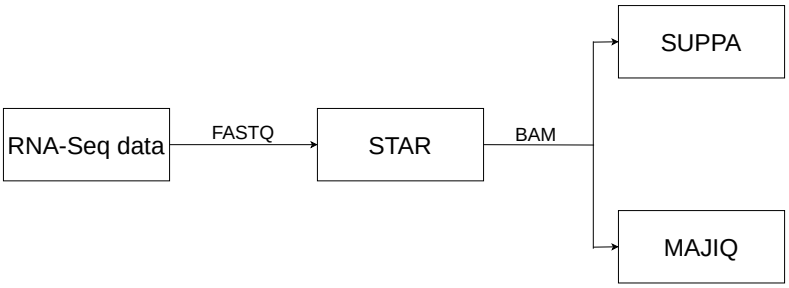

Figure 4.3: Data flow when using MAJIQ or SUPPA to create a training dataset.

interested in. Of the methods directly reporting PSI, we chose the two fast methods SUPPA and MAJIQ. SUPPA and MAJIQ represent two different approaches to PSI estimation: SUPPA is primarily based on quantification of transcripts, while MAJIQ is based on building up a splicing graph. Figure **??** shows the high-level steps taken when using MAJIQ and SUPPA to create datasets for training.

SUPPA

SUPPA2 [28] estimates the PSI value for each alternative splicing event based on transcript abundances. It operates in 2 steps for quantifying the PSI of an alternative splicing event: 1) Given an input annotation file in GTF format, it generates the transcript isoforms which count as IR or ER for a given alternative splicing event. 2) Given the information which transcript count as IR or ER from 1) and how frequently each transcript occurs, it estimates the PSI value via $PSI = TPM_{IR}/TPM_{IR} + TPM_{ER}$. SUPPA2 can integrate the TPM values from multiple samples. **Using SUPPA2:** SUPPA2 requires a GTF annotation file and a file quantifying the abundance of each transcript. A GENCODE version 34 annotation file obtained from Ensembl was used as an annotation file. Salmon was used for quantifying the relative abundance of each transcript in TPM. Salmon's [cite] quantification is based on the raw RNA-seq reads in FASTQ format, takes into account experimental attributes and correct for biases commonly observed in RNA-seq data. After extracting the TPM values provided by Salmon into the format required by SUPPA (a tsv-file with one column for the transcript id and one column for the TPM value), SUPPA2 estimates the PSI for each alternative splicing event. The latest version of SUPPA available as of July 2020 was used (SUPPA v2.3).

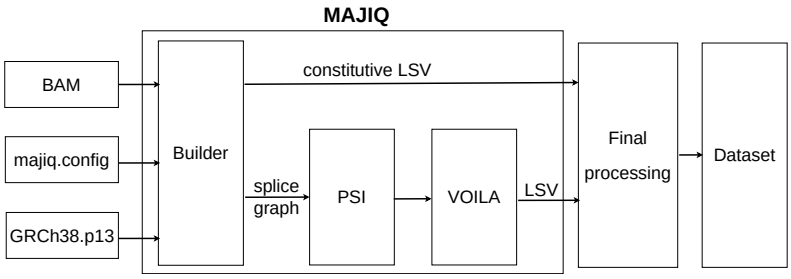

Figure 4.4: Data flow when using MAJIQ to create a training dataset.

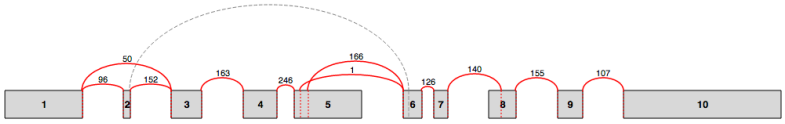

Figure 4.5: An example splice graph as displayed in VOILA [29]. Exons, represented by grey rectangles, are connected with another exon via an edge, if a junction between the exons (or a variant of them) is observed. The number above an edge displays the number of observed raw reads spanning a particular exon-exon junction.

MAJIQ

Modeling Alternative Junction Inclusion Quantification (MAJIQ) builds up a splice graph [29] which contains exon as vertices and junctions as edges. An edge is added between two vertices if a transcript isoform is found in which they share a junction (that is if the exons are neighbours and everything between them has been spliced out). Constitutive exons are vertices with two incoming edges. Vertices which have more than 3 incoming edges are denoted as local splicing variations (LSV). Due to this problem formulation, MAJIQ can model more complex splicing variations than skipped exons, alternative 3' splice sites, and alternative 5' splice sites. Figure 4.5 shows an example splice graph. Importantly, apart from estimating δ PSI between different conditions, MAJIQ is also able to directly estimate PSI for a given condition. The estimation is done using a combination of read rate modelling, Bayesian PSI modelling and bootstrapping.

Using MAJIQ: MAJIQ requires sequence files in bam format (along with the index in bai format) and an annotation DB in gff3 format. The bam format is a format for aligned RNA-seq reads. To this end, the raw reads from each

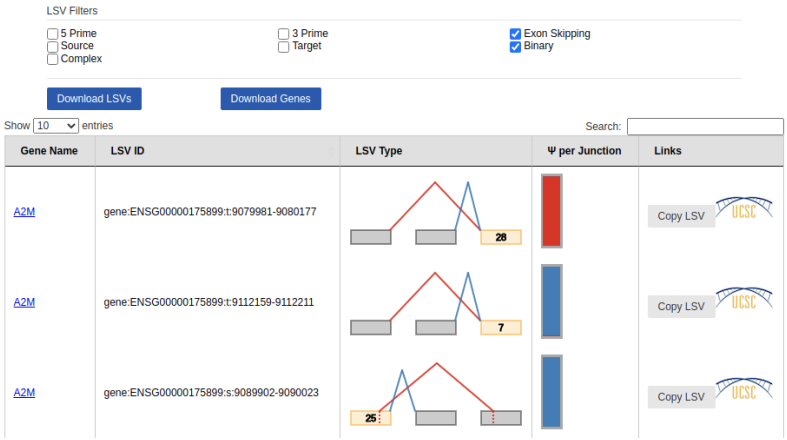

Figure 4.6: Example output of VOILA while filtering for LSV which only experience exon skipping and no other alternative splicing event.

sample were uploaded to the bioinformatics data processing platform Galaxy [30] from the European Nucleotide Archive (ENA). For each sample, the raw reads in FASTQ format were then mapped to the RNA using STAR [31]. STAR produces the required bam and bai format files as output.

The required gff3 file used is human GRCh38 release 13. MAJIQ use then proceeds in three stages. First MAJIQ Builder takes a configuration file, the gff3 annotation and the bam and bai files of all samples as input and builds up the splicing graph. At this stage, MAJIQ also identifies constitutive exons and optionally dumps them to a file. This option was used. Secondly, MAJIQ PSI estimates the PSI of the LSV candidates obtained through MAJIQ Builder. MAJIQ PSI improves the accuracy of its PSI estimate by integrating evidence across multiple samples. Finally, the obtained LSVs can be visualized using VOILA. VOILA also allows filtering of LSV according to type. See Figure 4.6 for an example output of VOILA. For this study, only LSV which denoted exon skipping were used. The filtered LSV along with the constitutive exons obtained from MAJIQ Builder were then used for processing as described in Section 4.2.2.
The latest version of MAJIQ available as of July 2020 was used (MAJIQ v2.0).

The naive way of estimating PSI was applied to the GTEx dataset as we had no access to raw RNA-seq reads and couldn't apply SUPPA nor MAJIQ. SUPPA and MAJIQ were applied to the data from the HipSci repository.

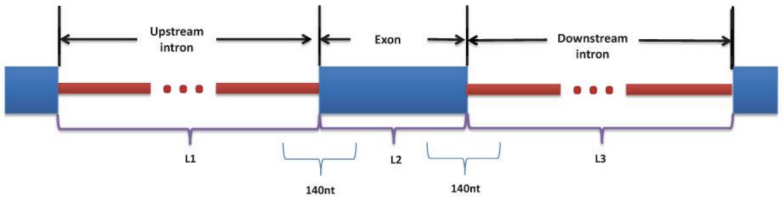

Figure 4.7: The input to our models are the 140 nucleotide region extracted around the exon start and exon end, as well as the normalized length of the exon and its flanking introns.

4.2.2 Final sample processing

Using the output of the previous preprocessing steps, now either exon-centric or intron-centric samples are created. Exon-centric samples are created when the task of the network is to classify an exon as constitutive or not. Intron-centric samples are created when the task of the network is to classify whether a junction is constitutive or not. At this point, the datasets have been preprocessed that each sample at least contains the following:

- chromosome and strand information of the to-be-classified exon or junction,
- start and end coordinates of the exon or junction within a chromosome,
- the lengths of neighbouring introns or exons.

Using the chromosome information and the start and end coordinates, a 140 nucleotide window around the start and end coordinates are extracted (see Figure 4.7). Furthermore, if an exon or junction is taken from a negative strand, the extracted start and end windows are switched, the order of the nucleotides within the start and end windows are reversed and each extracted nucleotides is converted to its reverse complement. This way the models don't have to learn different features for exons or junctions on the + and - strand. Samples from the chromosomes X, Y and M were excluded as these are known to show special splicing behaviour.
The extracted nucleotides were one-hot encoded as four dimensional vectors. Specifically, adenine 'A' was encoded as [1 0 0 0], cytosine 'C' as [0 1 0 0], guanine 'G' as [0 0 1 0] and thymine 'T' as [0 0 0 1]. Thus each 140 nucleotide long window was

converted into an 140x4 matrix containing one-hot encodings.

It is commonplace that repetitive sequences are soft-masked as lower case letters in the reference genome. As this has no bearing on alternative splicing, this information was ignored during one-hot encoded.

Introns are on average one to two orders of magnitudes larger than exons and their relative standard deviation is three times as large as that of exons. To avoid giving features to the network whose magnitude might differ by several orders of magnitude, the intron and exon lengths features were respectively normalized through the mean length and standard deviations of internal exon and introns in the human genome [32]. For brevity, these lengths are often referred to as the length features. Some of the datasets had a class imbalance issue between constitutive and alternatively spliced samples. For instance, the HexEvent dataset contains roughly times three as many constitutive as alternatively spliced samples. This leads to the issues with models getting stuck in local minima where they just predict the majority class. This was alleviated by including each alternatively spliced sample multiple times until the classes were balanced. A single sample input to the model contains two 140x4 one-hot encoded matrixes and three normalized length values. The associated ground truth with each sample is the scalar 1 if the respective exon or junction is constitutive, and 0 if not.

4.2.3 Dataset statistics

[**subsec:datasetstatistics**]

further analysis postponed until I ran experiments again and decided for which datasets I will show results

Figure 4.8 shows histograms for the exon-centric version of the four primary datasets versions. The histograms show that the distribution of PSI values is bi-model with the modes being around very rarely included ($< 5\%$) and very frequently included exons ($>95\%$). However, there are also some dissimilarities between the

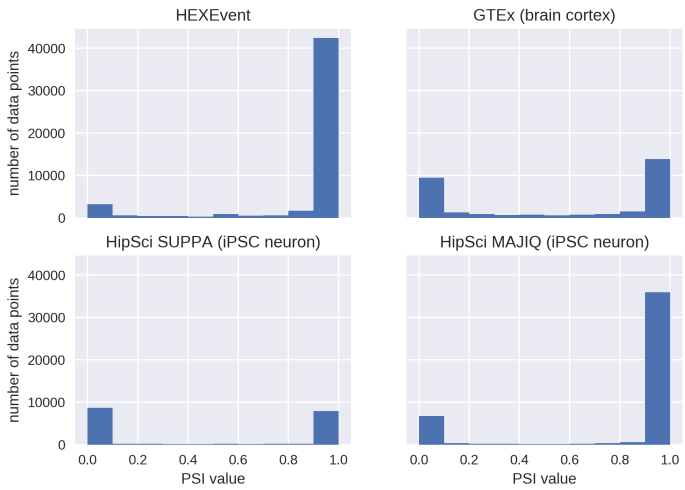

Figure 4.8: ...

datasets: the GTEx and HipSci SUPPA dataset contain the fewest samples and, in particular, they contain significantly fewer constitutive samples.

The biggest contributor to this is that due to their respective processing methods the GTEx and HipSci SUPPA datasets only contain known cassette exons as constitutive exons, but not non-cassette constitutive exons. Another factor is that they only estimate PSI based on one biological sample in contrast to the other datasets. HipSci SUPPA especially has an extremely low number of non-rarely or frequently included exons which is likely an artefact of its coarse, transcription-level estimation method.

4.3 Models

The models are fundamentally all split into two components.

The first component extracts the most relevant features of the two input sequences of nucleotides. The feature extraction for the sequence around the exon start and the exon end is done independently. Depending on the exact model, the feature extraction is based either on CNNs, BiLSTMs or MLP. The extracted features of this layer are concatenated along with the length features and fed as input to the second component.

The second component uses the extracted features and length features to perform binary classification. This component is implemented as a shallow MLP for all models. It consists of one or two fully connected layers followed by dropout and activation layers. A sigmoid activation function is used after the last layer to obtain an output from [0, 1].

4.3.1 DSC: CNN-based

This model is the same as the Deep Splicing Code (DSC) model from [23] and is based on CNNs. CNNs are useful when the input data contains spatially invariant patterns. Since images are fundamentally made up of small, spatially invariant patterns which can be combined into more complex patterns, CNNs have been extremely successful in Computer Vision.

CNNs are also extremely promising for the application to nucleotide sequences:

1) We can expect to find motifs in the input sequence. Motifs are often represented as position weight matrixes (PWMs). The kernels used in CNNs can be interpreted as PWMs with learnable weights, i.e., the model can automatically learn to recognize important motifs.

2) These motifs may be found at different positions in the input sequence. Due to CNNs being spatially invariant they will be able to detect motifs at any position in the input sequence.

Keeping this motivation in mind, the first model extracts features from the two input sequences using CNN-based blocks. A CNN feature extraction block with the same architecture but independently learned weights is used for each input sequence. This is to accommodate the model learning to extract different features from the exon start and exon end sequence. The extracted features are used by a shallow MLP to obtain the final classification.

Concretely, a CNN block consists of three convolutional layers. The first convolutional layer has a window size of 7 units and 32 filters, the second has a window size of 4 units and 8 filters and the third has a window size of 3 units and 8 filters. Each convolutional layer is followed by a dropout layer with dropout

probability 0.2 to reduce overfitting [cite dropout], a ReLU activation layer and a max-pooling layer with stride and window size 2 to extract the most salient features. The hyperparameters for this model were chosen with grid search by [23]. As we are dealing with inherently 1-dimensional textual data instead of inherently 2-dimensional image data, 1D convolution layers are used.

The extracted features for both genomic sequences are concatenated along with the normalized exon and flanking introns and input to a fully connected layer with 64 neurons. A ReLU activation function, as well as a dropout layer with probability 0.5, is applied to the output of this layer. The final layer takes in the 64 outputs from the previous layer, predicts a single value and is followed by a sigmoid activation layer. a word on 1d convolution?

Applying NLP techniques to genomic data

Like text, genomic data is fundamentally a sequence of characters. This makes it possible to apply techniques known from Natural Language Processing (NLP) to genomic data. This is an especially promising area of cross-pollination because of the large strides NLP has been able to make in recent years using deep learning. Some of the most important milestones in NLP have been efficient embeddings via word2vec, applying RNNs to text as in seq2seq and extremely powerful and deep Transformers.

In this work, we will test the application of doc2vec (a derivative of word2vec) and seq2seq models to the task of alternative splicing classification. The current state-of-the-art transformer models, albeit very potent, usually require huge datasets in the order of millions of samples which aren't available for this task. While scaling them down might be an interesting avenue of research even heavily scaled-down models might still be over parametrized for the task as it has frequently been shown that very shallow networks (with less than 5 layers) perform best on genomic data. Additionally, our input sequences are very long; generally 140 tokens long. Initial tests showed that this also leads to memory problems when training on only one GPU, as the memory requirements of Transformers grows

quadratically with respect to the length of the input sequence, further complicating their application. Due to these considerations, we chose to forego the evaluation of Transformer-based models in this study.

4.3.2 D2V: MLP-based

Introduction to word embeddings

Word2Vec is a neural network-based model from Natural Language Processing (NLP) which provides a continuously distributed (vector) representation for each word within a sentence [33][34]. The main idea of Word2Vec is that the meaning of a word is characterized by its context and therefore semantically related words will be similarly represented. The classic example is that

$\vec{King} - \vec{Man} + \vec{Woman} = \vec{Queen}$ when adding the Word2Vec representations of the respective words. Doc2Vec is an extension of Word2Vec which can handle variable length texts (such as paragraphs and complete documents) [35] [36]. It returns a fixed-length representation independent of input size.

Training

To train word2vec or doc2vec a large corpus of unlabelled data is needed. As we are training on genomic data, the largest corpus is the complete genome itself. The complete human reference genome GRCh38 [37] was obtained from UCSC [38].
During training on text data, the corpus is split into documents which are ideally semantically meaningful (like paragraphs). Due to memory limitations and due to corresponding to the average gene size [39], we split the genome into sequences of 28,000 nucleotide sequences. However, tests with sequences of length 10,000 showed that this boundary length does not seem to affect model performance in a meaningful way.

Using k-mers

Naively applying word2vec or doc2vec to genomic data would mean treating each nucleotide as a token. Drawing the parallel to natural language, this would mean wanting to embed each character in a word. However, a single character or nucleotide

(which on top comes from an alphabet of only four characters) does likely not contain enough information by itself. Therefore, it is desirable to embed a k-mer. One major difference between text and genomic data is the single-token complexity: a word contains far more information than a single nucleotide. Multiple studies have explored this issue and found 3-mers to perform the best. We follow their recommendation in this work and split each sequence of nucleotides into overlapping 3-mers. Overlapping means that in the case of the sequence 'AACGAT' the resulting overlapping 3-mer sequence is 'AAC', 'ACG', 'CGA' and 'GAT'. Therefore, the 28,000 nucleotides long pre-training sequences are split into overlapping 3-mer sequences (sentences) of length 27,998.

Pre-training of word embeddings

The pre-training for word2vec uses a shallow neural network with an input, hidden and output layer. If the continuous bag-of-words technique (CBOW) is used, all words in a window except the current word are given as input and the target output is the current word. Typical window size is 5. If the skip-gram technique is used, only the current word is given as input and the task is to predict the surrounding words in the window. Doc2Vec works similarly. In the CBOW equivalent called distributed memory (DM), a document identifier (usually just an integer enumerating all documents) is given as additional input. In the skip-gram equivalent distributed bag of words (DBOW), the current word is replaced by the document identifier and the task is to predict all words in the window. All four different training methods are visualized in Figure [figure 3 from distributed]. There is no clear choice for when which training method should be used for Word2Vec or Doc2Vec as the training methods tend to perform similarly. However, DM (and also CBOW) preserve the order of words and as the order of words (3-mers) is likely significant, we chose DM as pre-training method. Pre-processing the human genome like above and using the DM training method, we trained the Doc2Vec model to output a 100-dimensional embedding for each document.

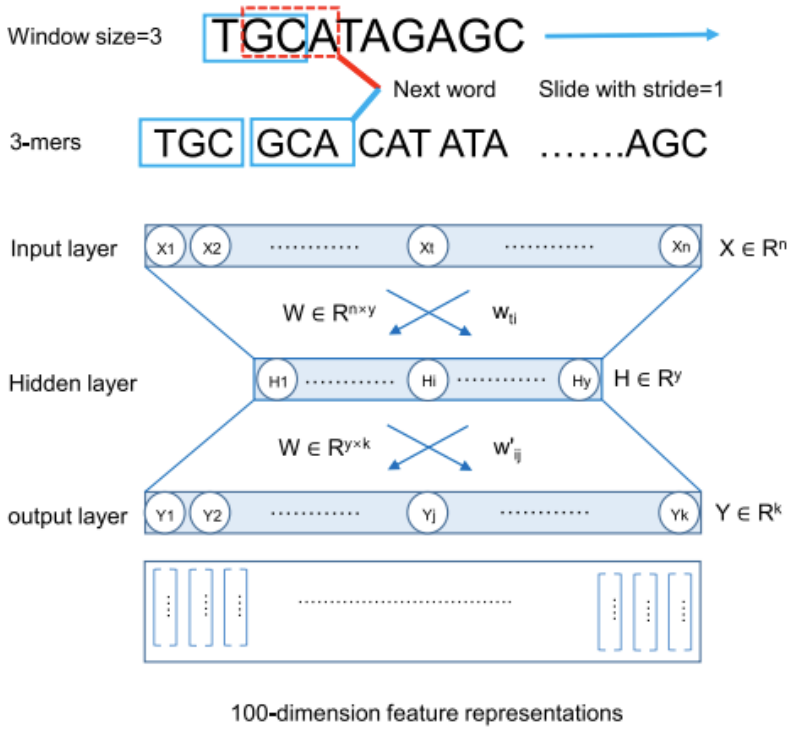

Figure 4.9

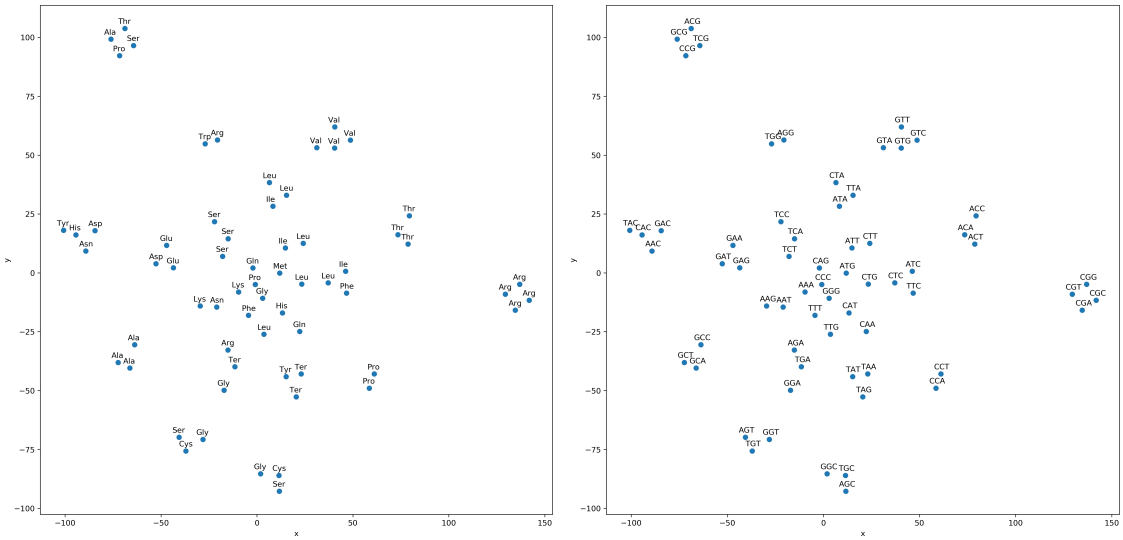

Figure 4.10

Analyzing the obtained embeddings

We visualize the embeddings for all 64 possible 3-mers using Stochastic Neighbor Embedding (t-SNE) [40] in Figure 4.10.

Doc2Vec is trained to map words (3-mers) which occur in a similar context to

similar representations. We observe that this often translates to mapping the same amino acids to similar representations. There are some instances when different amino acids are mapped together like the quartet Ala, Thr, Pro and Ser in the top-left corner. Displaying the 3-mers associated with the amino acids, that this likely occurs because these different amino acids share two out of three nucleotides at the same positions. We take this visualization as an indication that Doc2Vec was able to learn biologically useful embeddings for the overlapping 3-mers. [maybe] Making use of these pre-trained embeddings, each 140 nucleotide input sequence is split into overlapping 3-mers and mapped to a 100-dimensional vector. Like in the other models, a shallow MLP using these sequence features in conjunction with length features for classification.

Inference

During inference, the Doc2Vec model outputs 100-dimensional embeddings for each input sequence. These, along with the length features, are fed into the shallow MLP for classification. The shallow MLP consists of two layers with 16 and 8 neurons each respectively followed by a dropout layer with dropout probability of 0.2.

4.3.3 BiLSTM + Attn

Seq2Seq

Sequence-to-sequence learning (Seq2seq) [41] is a general deep learning-based framework for mapping one sequence to another. The first part of the framework is an encoder which processes the input symbol-by-symbol and produces a vector representation for each input symbol. The second part is a decoder which predicts the output sequence symbol-by-symbol based on the representation(s) computed by the decoder. In the first timestep, the decoder uses the last output of the encoder and in all other timesteps, it uses its own output from the previous time step as input. The encoder and decoder are typically RNN-based networks such as LSTMs [42] or GRUs [43]. Encoder and decoder are jointly trained to maximize the probability of correct output. This framework has been particularly successful in machine

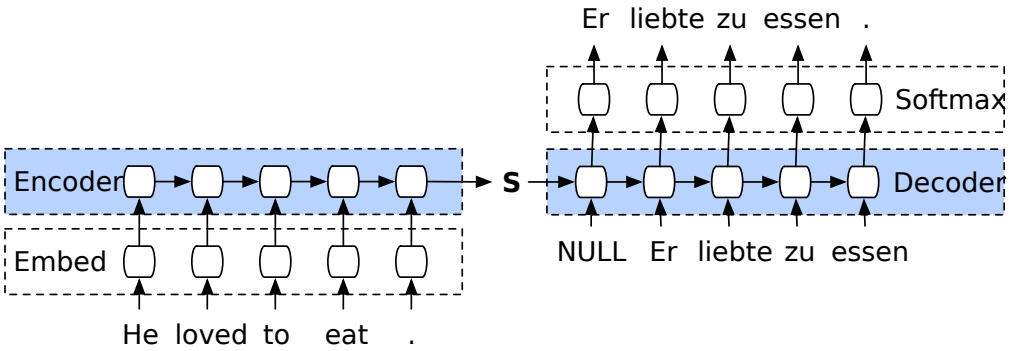

Figure 4.11: An encoder-decoder model [45] translates the English sentence 'He loved to eat' to the German sentence 'Er liebte zu essen'. The decoder takes the last hidden cell state of the encoder as initial input and generates the output token-by-token. The decoder takes its own previous output as prompt for the next output and stops generating output token when it generates a '.'-token.

translation (MT): for instance, Google announced in 2016 that it started using them for their MT [44]. Figure 4.11 visualizes the working of an encoder-decoder model.

Are you paying attention?

In the original Seq2seq framework, only the last state of the encoder is passed to the decoder. This means that the encoder needs to encode all of the information observed in the input sequence into its last state. While theoretically possible, [46] hypothesized that this informational bottleneck hampered performance in practice. They proposed removing this bottleneck via introducing attention. Attention is a mechanism whereby the decoder can look at all the representation generated by the encoder at once and can 'choose' to focus on the most important ones. Introducing attention lead to large performance gains across Seq2seq-based models and quickly became standard practice. The attention mechanism also adds interpretability to the model because it indicates which parts of the input sequence are most crucial for the model's prediction. Figure visualizes what parts of an input sequence a model is paying attention during an MT task. Attention is one of the most important innovations in deep learning research in recent years. For instance, the current state-of-the-art Transformer models [47] [48] [49] forego recurrent networks completely and solely use attention within the encoder and decoder components. However, as justified earlier, we won't focus on these models due to issues of scaling and the

sufficiency of shallow networks for genomic prediction tasks. Attention is a very general mechanism not limited to the Seq2seq framework and different forms of it (such as additive [46] or multiplicative attention [47]) are used in practice. It can theoretically be useful for any task where a model needs to make a decision based on certain parts of an input.

Integrating attention

Taking inspiration from the successful application and improved interpretability of the Seq2seq framework with attention in other domains, we also adopt it for constitutive exon classification. We now describe the modifications we made to initial Seq2Seq and attention set-up known from MT for our our context. In particular, we describe the attention mechanism in more detail as necessary for our second modification.

Modification 1: MLP instead of a recurrent decoder

As the name implies, models based on the Seq2seq framework take a sequence as input and give a sequence as output. This is achieved via using a recurrent decoder which outputs symbols until it outputs an <EOS> (end-of-sentence, e.g. a ".") token (in the case for MT, similar for other domains). However, in our case, the output will always be a single scalar: the classification of the exon as constitutive or alternatively spliced. Thus, we don't require a recurrent decoder and instead use a shallow MLP for classification of the encoder features.

Modification 2: Attention with a learned query vector

The most common form of attention (multiplicative or dot-product attention), makes uses of conceptual queries and key-value pairs. A given query is compared to all keys by computing a similarity score between the query and each key via the dot-product. The similarity scores are normalized through the use of a softmax layer. The normalized similarity scores are also commonly called the attention weights; keys similar to the query are weighted more. The output of the attention layer for a given query and input sequence is then the weighted vector sum obtained

by multiplying each value by the attention weight of its respective key. Putting the above intuition into formulas, the (dot-product self-) attention mechanism can be described using the following equations:

$$Q, K, V = IW^Q, IW^K, IW^V$$

The query, key and value matrices Q, K and V are computed by multiplying the input with query, key and value parameter matrices W^Q, W^K and W^V. where

$$W^Q, W^K, W^V \in \mathbb{R}^{in \times out}$$

, and $Q, K, V \in \mathbb{R}^{l \times out}$. *in* corresponds to the number of features of each element in the input sequence to the attention layer, *out* corresponds to the number of features in the output of the attention layer. l corresponds to the number of elements in the input sequence to the attention layer. Attention can be then computed as:

$$Z = attention(I) = softmax(QK^T)V$$

where $Z \in \mathbb{R}^{l \times out}$. The input I is a concatenation (along the first dimension) of the representations given by the two BiLSTMs. Note that this makes use of the fact that $Q \cdot K = QK^T$ where

.

denotes the dot product.

As shown, multiplicative self-attention computes a separate query for each input token. It is called self-attention because it allows a sequence to pay attention to 'itself'. (Unmasked) self-attention is mainly used in the encoder part of Transformer architectures to improve the representation for each word.

However, our objective in using attention is not to improve the representation for each symbol in a sequence (this is the task of the BiLSTM), but to select the most important features from an input sequence. The conceptual query for our task is also always the same independent of input: is this exon constitutively spliced or not?

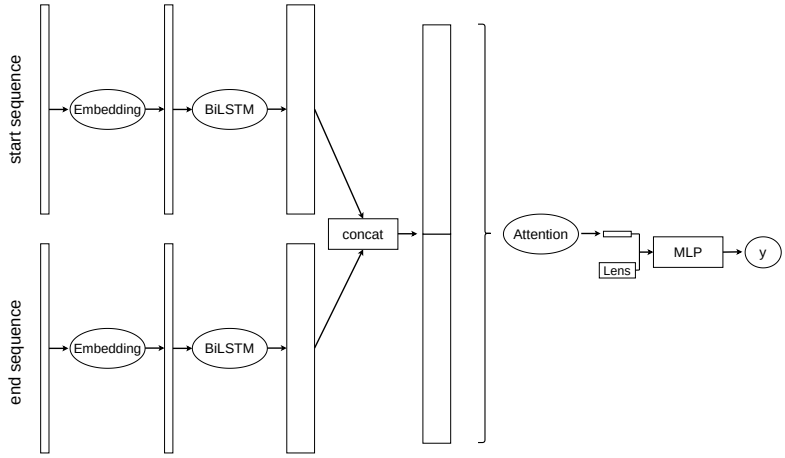

Figure 4.12: Architecture of the new attention-based model we introduce.

Thus, we adapt the attention mechanism so that we learn a single, input-independent query $Q^* \in \mathcal{R}^{1\times out}$. The output of the attention layer is now computed as:

$$Z^* = attention^*(I) = softmax(Q^* K^T)V$$

where $Z^* \in \mathbb{R}^{1\times out}$. Note that the query matrix Q^* is directly learned in the attention block and not multiplied with the input, giving rise to its input independence. The output Z is now a weighted sum of the nucleotide representation in the input sequences which the model deemed most crucial to pay attention to.

Putting it all together:

Like a Seq2Seq model, an RNN-based Encoder is used to capture a representation of the input sequence. We select the most important representations of the input using the modified attention mechanism we introduced. Instead of a recursive decoder, we use a shallow MLP for classification based on the selected features. One detail not mentioned yet is that the input to the BiLSTM blocks is a dense 4-dimensional embedding of the one-hot encoded sequences. This embedding is the same for each sequence and jointly learned during training. This extra embedding layer is included as previous works [50] have shown that otherwise, the model might suffer from limited generalization performance caused by the sparsity of the one-hot encoded representation.

4.3.4 Alternative implementations of attention

In the following, we introduce multiple extensions which were attempted which didn't lead to improved results. For most of these, initial results clearly showed that the results weren't improved by using them. As such, we didn't test these extensions more extensively than initial experiments because we felt the initial results were already instructive enough and no further investigation was warranted. Thus, the analysis of the results won't be as extensive as for the extensions which worked.

No query

As the query matrix Q^* is already the same independent of the exact input sequence, it could be possible to drop it and compute the attention weights sorely based on the key matrix K. This is possible by learning a key weight matrix $W^{K^*} \in \mathcal{R}^{in \times 1}$ and computing attention as:

$$K^* = IW^{K^*}$$

$$Z = attention(I) = softmax(K^*)V$$

where $K^* \in \mathbb{R}^{l \times 1}$ and $Z^{**} \in \mathbb{R}^{1 \times out}$ However, in initial experiments, the attention** mechanism performed significantly worse than the attention* mechanism. The extra representational capacity by having a query matrix seems to be important for performance.

Multiple attention heads

Having one set of query, key and value matrices limits the model to one representational subspace (one way of 'interpreting' the input). It might be useful for the model to have multiple representational subspaces. This is usually achieved via

$$h$$

different sets of query, key and weight matrices $W_1^Q, W_1^Q, W_1^Q, \ldots W_h^Q, W_h^Q, W_h^Q$ producing multiple outputs $Z_0, \ldots, Z_{n\_h}$. The final output matrix $Z_{final} \in \mathcal{R}^{l \times out}$ is then computed as:

$$Z_{final} = concat(Z_1, \ldots, Z_h)W^O$$

where the concatenation occurs along the second dimension and $W^O \in \mathcal{R}^{out*h \times out\_new}$. out_new represents the new output dimension of the attention layer, which is arrived at by combining the dimensions of the multiple attention heads.

Convolution in attention heads

As was previously discussed in the motivation for splitting the input sequence into overlapping 3-mers in 4.3.2, the information contained in a single nucleotide is very low. This motivated [51] to apply an additional 1D convolution to the query, key and value matrices Q, K and V. This extension can be understood as an alternative to splitting the input into k-mers and provided better results than splitting the input into k-mers when using a Transformer network for a genome annotation task in [51]. Keeping in mind the modification of directly learning the query matrix, this leads to:

$$K_{conv}, V_{conv} = conv(K), conv(V)$$

$$Z^* = attention^*_{conv}(I) = softmax(Q^* K^T_{conv}) V_{conv}$$

where the K and V are both fed through the same convolutional layer. The number of convolution filters and padding are chosen so that the dimensions of the K and V matrices don't change, that is, $K_{conv}, V_{conv} \in \mathbb{R}^{l \times out}$. One more detail regarding the application of this extension: to incorporate the domain knowledge that the input sequences to the neural network come from different places in the genome, we made sure that the convolution did not mix the information between the two sequences. To achieve this, the following computation actually took place :

$$K^{start}, K^{end} = I^{start} W^K, I^{end} W^K,$$

$$K^{start}_{conv}, K^{end}_{conv} = conv(K^{start}), conv(K^{end})$$

$$K_{conv} = concat(K^{start}_{conv}, K^{end}_{conv})$$

where the 'start' and 'end' superscript respectively refer to the sequence around the start and end of the exon. The concatenation is along the first dimension (so the length of the sequences). The analogue computation was done for V_{conv}.

TODO: add this separately to results section Testing this extension on the HipSci

MAJIQ dataset didn't lead to improved results. Although the convergence speed increased rapidly (from around 180 epochs to 20 epochs), we also observed heavy overfitting. To alleviate this, we added dropout and batch normalization layers. After adding two dropout layers with a high dropout probability (one after computing the attention weights and one after the convolution layers with p=0.5 each) and setting the convolutional filter size to no higher than 3, overfitting issues ameliorated. However, after adding these layers convergence was no faster than for the case without convolutional layers and the performance did not improve. Thus, for the sake of a simpler model, we opted to not use this extension for the rest of the experiments. Reasons for why this extension did not work might be related to the use of a recurrent encoder as well as task differences compared to [51]:

1. Although we process the input at single-nucleotide resolution, the use of a BiLSTM means that the encoder is aware of the nucleotides which precede and follow it. Therefore, information about the neighbouring nucleotides is likely already represented in the representation for a given nucleotide. This is in contrast to a Transformer model where the layers are non-recurrent and only work at single-symbol resolution. Thus, the convolutional layers likely don't give the model a lot more flexibility.

2. In [51], the task is a full genome transcription start site annotation. Therefore, in a sense, their task occurs at the inter-gene level while our task occurs at the intra-gene level. This likely means that the motives the model has to consider generally span more nucleotides and giving the model more capacity to integrate over multiple nucleotides is more crucial. In contrast, the single-nucleotide resolution of our model may be necessary to help it identify the more fine-grained motifs which influence splicing. Thus, these differences likely also influenced the performance differences when using the convolutional attention layer extension in the different models.

A small implementational detail regarding the attention blocks: the query, key and value parameter matrices are implemented via a linear layer with the

appropriate dimensions. These also learn bias weights which are added to the output by default. While in theory the bias weights should be turned off to implement exactly the shown formulas, in practice other attention mechanism implementations leave these turned on [52] and so we follow suit. Thus, to be precise, the query, key and value computation should be:

For this model, the hyperparameter space was a lot larger and model performance was very sensitive to the hyperparameter choice. The most crucial hyperparameters were the number of encoder dimensions and the number of dimensions of the attention layer. Hyperparameters were optimized via grid search on the HipSci MAJIQ dataset.

putting model parameters into the text instead of a table: Concretely, each LSTM block consists of a BiLSTM with one (?) layer which outputs 50 features. The shallow MLP used for classification uses a single layer with 128 (?) neurons and is followed by a dropout layer with dropout probability of 0.5.

4.4 Training and implementation details

4.4.1 Implementation

All models, except the Doc2Vec network, were implemented using the PyTorch library (version 1.5.0) [53]. The Doc2Vec model was implemented using the gensim library (version 3.8.3) [54]. The data processing was done using a mixture of Python and Bash scripts as well as the software tools MAJIQ and SUPPA described in section 4.2.1 and 4.2.1. The complexity induced by the use of a large number of different datasets and data processing methods was handled by standardizing the shape of the final training samples. Even though 12 different dataset variants and 4 different models are used, the codebase contains only 2 different data loader and trainer classes.

The structure of the repository was based on the PyTorch Deep Learning Project template available at `github.com/victoresque/pytorch-template`. This template provides access to abstract classes for data loaders, trainers and the models

themselves. These abstract classes already contain a lot of the implementation-independent code to log results and train and save models. These abstract classes can then be extended with implementation-specific code as e.g. exact data formats. Using this structure avoids the duplication of a lot of boilerplate code.
To run an experiment, the experimental settings are defined in separate JSON files. This allows for easy and unambiguous reproduction of results. All experiments for whom results in this study are shown are available as JSON files that define the exact parameters used. The code repository itself contains multiple README which go into more details towards the structuring and using of the code and also define naming conventions.

TODO: total code base contained roughly ... lines of code. the majority of these were in preprocessing section [maybe exclude this]

4.4.2 Training

The training was done using one NVIDIA GeForce GTX 1080 Ti GPU. This proved to be sufficient as initial exploratory experiments showed no advantage of using deep networks and the resulting networks used are all relatively shallow. The pre-training of the Doc2Vec model on the human genome took 3 hours. We randomly split each dataset into 10 folds; in a given run, 8 folds were used for training, 1 fold for validation and 1 fold for testing. Each model was trained with different folds 9 times. The training time for training a single model once varied between 1 minute and 25 minutes (see [table with training times]). The models were trained with early stopping on the validation fold until they stopped improving for 15 epochs. This typically occurred after 70 - 150 training epochs. Where we established in a base experiment that the variance for a given model was low between runs (lower than 1

Loss

All models were trained to minimize the binary cross-entropy loss:

$$\mathbb{L} = -(ylog(\hat{y}) + (1-y)log(1-\hat{y}))$$

where the ground truth $y \in \{0, 1\}$ and the prediction of the model $\hat{y} \in [0, 1]$. The binary cross-entropy is the standard loss for training deep learning-based (binary) classifiers.

Metrics

As in previous work [23], we evaluate all models using the Area under ROC curve (AUC). The AUC provides an aggregate measure of performance across all possible decision thresholds. However, some of our datasets were unbalanced. This may lead to a degenerating behaviour where a model only predicts the majority class. Additionally, only looking at the AUC does not give a good sense of the .... specificity.. recall? This is not captured by the AUC.
Thus, we also evaluated the F1 of our models when working with an unbalanced dataset as in the case of the HipSci-based dataset processed with MAJIQ. The output of our models is a single, continuous scalar $\hat{y} \in [0, 1]$. As the AUC itself is an aggregate measure across all decision thresholds, we don't need to set it to a certain value that translates the continuous prediction into a discrete positive or negative class prediction. To compute the F1 score, we do. Naively, this threshold may be set to 0.5; that is, all outputs above 0.5 will be interpreted as a prediction of the positive class and all other outputs as a prediction of the negative class. However, this threshold may not be optimal to maximize the F1 score, especially when working with unbalanced datasets. We choose the threshold which maximizes the F1 score on the test set.

TODO: [mention respective model sizes here] bilstm + attn 66861, dsc 20.001, d2v 6801

Exact hyperparameters used for training the models are given in table 4.1.

DSC	
Kernel filters	32, 8, 8
Kernel size	7, 5, 3
Dropout (after fully-connected layer)	0.5
Neurons fully-connected layer	64
Dropout (after convolution)	0.2, 0.2, 0.2
D2V	
Embedding dimension	100
Neurons (fully-connected layers)	32, 8
Dropout (fully-connected layers)	0.2, 0.2
BiLSTM + Attention	
Dense embedding dimensions	4
BiLSTM dimension	50
BiLSTM layers	1
Attention heads	4
Attention head dimension	50
Dropout (attention head)	0.4
Attention layer output dimension	100
Neurons fully-connected layer	128
Dropout (after fully-connected layer)	0.5

Table 4.1: Hyperparameters of the main models

5 Results

This section shows and discusses the results of our experiments based on the models and the datasets introduced in Chapter 4. It also describes the rationale behind deciding which experiments should be run.

5.1 HEXEvent dataset

To obtain a baseline, we evaluate the reimplemented and proposed model on the HEXEvent dataset as used in [23] and introduced in 4.1.1. The results of these experiments are shown in Figure 5.1.

Analysis of main findings

Generally all models perform extremely well with AUCs nearing 90%. They also perform very similarly, making it hard to differentiate them based on their ROC curves. The Attn model seems to perform slightly better than the other models.

Assessing our reimplementation, we observe a small difference between the AUC value reported in [23] (mean 0.899, standard deviation of 0.18) and the ones we observe (mean 0.873, standard deviation of 0.06). This is likely just a result of random statistical noise influenced by different random seeds between runs and differences between TensorFlow / Keras (their implementation) and PyTorch (our implementation). Notably, when reevaluating the publically available original

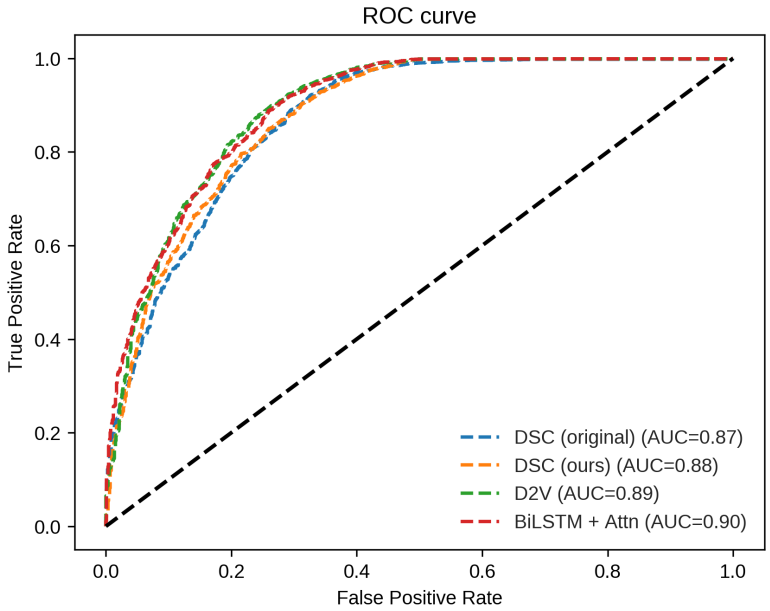

Figure 5.1: Comparison of the ROC curves of the three main models (as well as the original implementation) on the HEXEvent dataset. The values of the original implementation were obtained by rerunning the training on the publicly available implementation.

implementation, in a single run we also obtain results (0.872 AUC) closer to the results of our reimplementation. Thus, we conclude that, albeit with minor caveats, the reimplementation was successful and that we are able to replicate the results of [**DSC**].

Length features are necessary

Reimplementation was done piece-wise, that is, first the sequence features were added as input, then the networks were trained to make sure that this was done correctly and then the length features were added. This lead to an interesting observation: model performance is significantly worse when no length features are given to the models. Quantitative results for this observation are given in table 5.1. This observation is true across models and leads to an average relative performance drop of over 66%. The information about the secondary structure obtained in the lengths seems to be necessary for the models to obtain good predictive power.

Model name	Only sequences	Sequences + lengths	Relative performance difference
DSC	0.618	0.873	0.684
D2V	0.614	0.896	0.737
BiLSTM + Attn	**0.636**	**0.904**	0.663

Table 5.1: Performance of the main models on the HEXEvent dataset with and without length features given as AUC. Note that the relative performance drop was computed with reference to the baseline AUC value of 0.5.

Further investigations

To further investigate, we also test three other models: MLP100, MLP20 and MLPLinear which respectively contain 100, 20 and 20 trainable parameters. The models are simple MLPs with one hidden layer which only take the length features as inputs. MLPLinear doesn't use a non-linear activation function after its hidden layer.

Surprisingly, the results in Figure 5.2 show that it is possible to replicate the results of [23] with these very simple models using two to three orders of magnitude fewer parameters. Model performance is improved by adding further parameters and breaks down when no non-linearities are used in the network. This indicates that the models capture a relatively simple, but non-linear relationship between the lengths and the classification of an exon in the dataset. There are multiple possible explanations for why this is:

1. there are confounders in the dataset learned by the model. As discussed in Section 4.1.1, EST-based is inherently biased. The biases inherent in EST-based data could be captured by the exon and intron structure and the model is learning to make its prediction based on this bias. This explanation is made more likely, if the findings on the HEXEvent dataset don't replicate on the other datasets we evaluate.

2. exon splicing is extremely well predictable based on the lengths of neighbouring introns and exons. This is very unlikely given that research into splicing has been going on for over 40 years. This explanation, albeit unlikely already, can be disproven if this observation fails to replicate on other datasets.

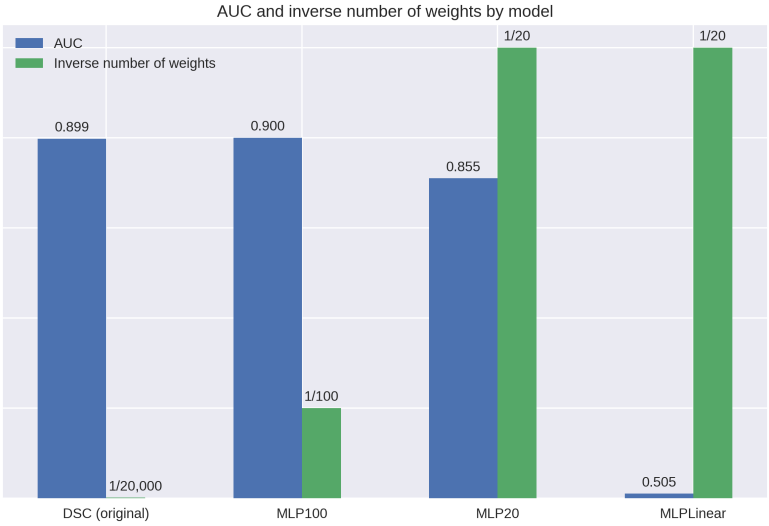

Figure 5.2: Stress testing the HEXEvent dataset used in [23]. The graph shows the performance as well as the inverse of the respective model sizes used.

3. there are bugs (e.g. mixing of testing and validation data) in our reimplementation. In the first instance, this is unlikely given that we were able to replicate the original results of [23]. To reduce complexity and further reduce the likelihood of a bug leading to these observations, we extracted the complete code for replicating the results of the simple MLP models from 5.2 into a single file. Additionally, in case of a simple bug the performance of the linear model likely wouldn't break down either. Therefore, we think that this possibility is unlikely.

Overall, we conclude that the HEXEvent-based dataset is most likely fundamentally flawed and suffers from confounders. These findings have strong implications. It calls into questions the meaningfulness of [23]'s results, showing the competitiveness of their model. These findings likely also warrant a critical investigation of any other conclusions drawn from papers based on the HEXEvent database. At the time of writing, the HEXEvent paper is cited 34 times. While most of these citations are just in passing, there are also multiple papers which use a HEXEvent-derived dataset for the training of Machine Learning models such as SVMs [22], Random Forests [55], Decision Trees [56] or AdaBoost-based algorithms [57]. Due to these data quality issue, we try to construct an alternative dataset.

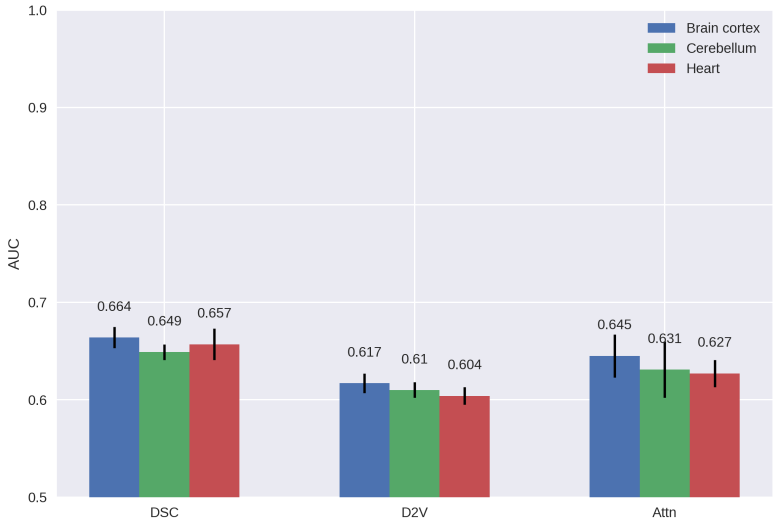

Figure 5.3: Performance on the GTEx-based exon datasets for different tissues. The error bars give the standard deviation across all cross-validation runs.

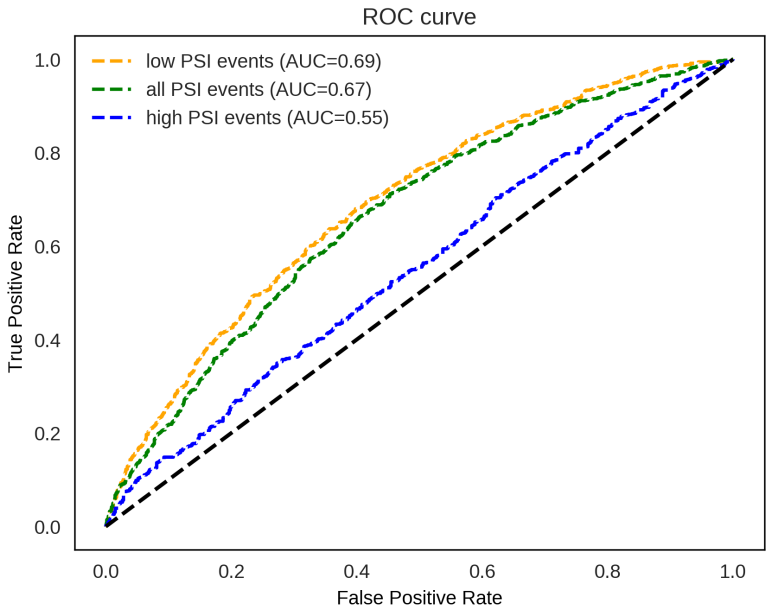

Figure 5.4: Bar chart of the performance on the GTEx-based exon datasets for different tissues. The error bars give the standard deviation across all cross-validation runs.

5.2 GTEx-based datasets

5.2.1 Cassette exon-based datasets

Results are given in Figure (....). Across all models, the performance is poor and the predictive power of the models is low. Surprisingly, performance is also very similar across tissues: the mean performance between tissues differs roughly by one

standard deviation of the mean performance between runs. This is surprising from a biological perspective as splicing across tissues is known to differ a lot. However, since our models perform so poorly they likely already struggle to learn the baseline splicing behaviour invariant across tissues. From a machine learning perspective, this is a bit surprising as the cerebellum tissue-based dataset is almost twice as large as the heart tissue-based dataset. This indicates that either 1) the models have already hit a point of diminishing returns for adding more data or 2) the models generally require magnitudes more of data. All models perform best on the dataset based on a brain cortex sample.

The relative cross-model performance also stays the same between tissues: DSC tends to perform best, followed by Attn, followed by the D2V model. The variance of the Attn model between runs is comparatively very high; the Attn model is the best performing model, as measured by the maximum rather than mean AUC value, on the brain cortex tissue-based and cerebellum tissue-based datasets. This indicates that on this dataset the Attn model needed more regularization and dataset specific fine-tuning would've lead to it performing the best.

Figure 5.4 gives more insight into model the model performance. The performance on highly included, alternatively spliced exons (80% <= PSI < 99%) is significantly worse than on more rarely included, alternatively spliced exons (PSI < 80%). From the definition of the AUC it follows that the AUC on the mixed dataset including exons with low and high PSI lies in-between these two extremes. In this case, only 28% of alternatively spliced exons belong to the exons with a high PSI and therefore the combined AUC is heavily weighted towards the AUC on the exons with low PSI. These observations align with similar observations made on the HEXEvent dataset [23].

—- removing length feature would be interesting; doing it on brain as best performance and sort of lowest combined variance

- could note that in earlier variation of the dataset where no TPM treshold was used (?) the networks didn't learn anything

– would be great finding out / remembering what exact step is the crucial one

Overall, **conclusions:**

- performance generally poor

- inter-tissue variance low

- due to issues with GTEx data, likely desirable to test with other datasets

5.2.2 Junction-based datasets

Figure 5.5: Performance on the GTEx-based junction datasets for different tissues.

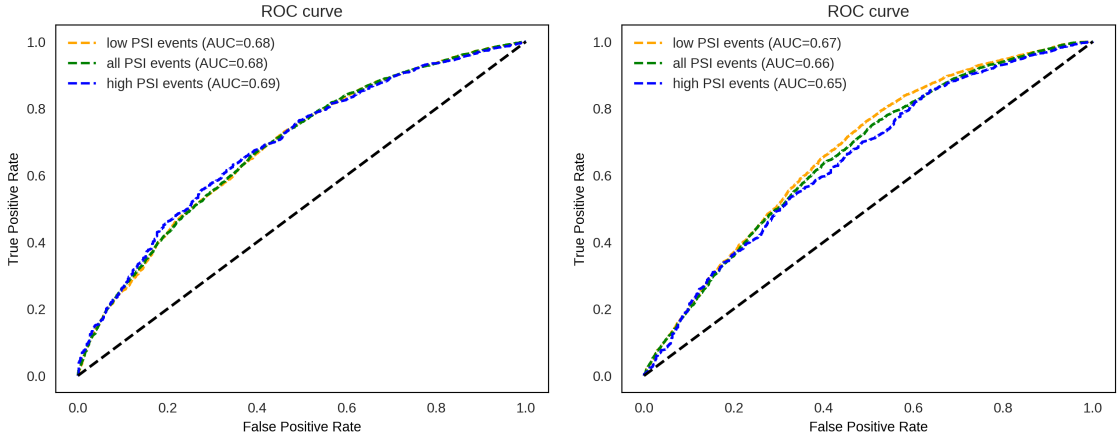

Figure 5.6: Left: ROC curve of the DSC model on heart tissue. Right: ROC curve of the Attn model on heart tissue.

TODO: rerun these with cross-validation (will take 30 hours of trainig time though)

- performance across the board a lot better; seemingly an easier task
- fairly clear correlation with more training samples -> more performance (cerebellum has most, heart has fewest)
- high / low difficulty trend doesn't really hold.

5.2.3 Reconstructing HEXEvent-dataset with GTEx data

- dataset where I only took exons which were also in GTEx data - seriously considering not showing these results as I basically already debunked HEXEvent at this point – so from a narrative perspective, why spend effort on replicating it?

In 4.2.1, we mentioned issues when trying to estimate PSI naively. Although we tried to alleviate these as far as possible in pre-processing, we don't account for the information contained in non-junction reads and that from multiple samples. As a result, data quality is likely still an issue. Alleviating these issues, we next evaluate our models based on a primary data source which gives access to raw RNA-seq reads. This enables us to use methods from the literature which leverage information from non-junction reads and multiple samples.

5.3 HipSci-based datasets

5.3.1 SUPPA with neuron tissue induced iPSCs

Takeaways:

1. All models perform similarly, BiLSTM + Attn tends to perform best
2. All models perform very poorly perhaps a dataset issue (due to HIPSCI SUPPA being coarse-grained)

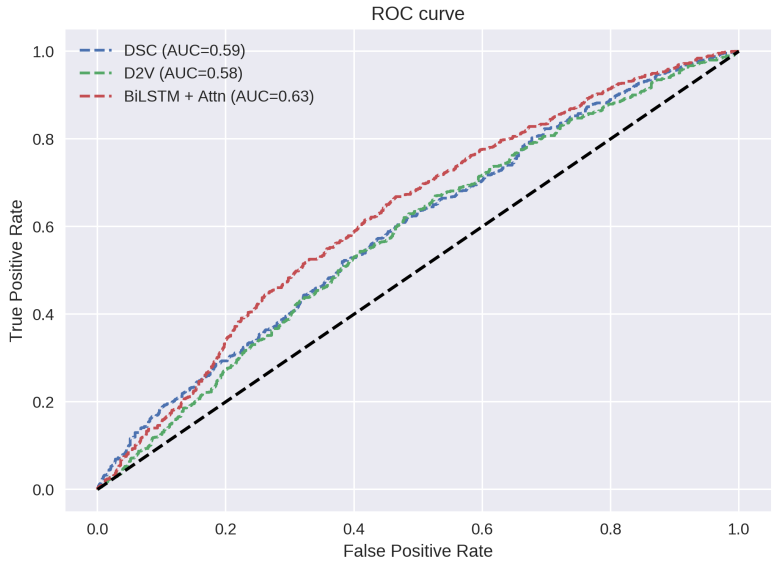

Figure 5.7: Comparison of the ROC curves of the three main models on the HipSci dataset derived by using SUPPA.

5.3.2 MAJIQ with iPSCs differentiated to neurons (exons)

5.3.3 MAJIQ with iPSCs differentiated to neurons (junctions)

haven't run experiments yet, not sure if results are interesting (but probably yes) due to training times, probably don't want to run these with cross-validation as that would take 15 hours+

5.3.4 MAJIQ with undifferentiated iPSCs

cross-tissue comparison here with very low variance across tissues;
seems like learned features are tissue-invariant

interpretation of attention mechanism etc follows here; some pretty graphics to follow

6
Discussion / Conclusion

Appendices

The first kind of intellectual and artistic personality belongs to the hedgehogs, the second to the foxes ...

— Sir Isaiah Berlin [**berlin_hedgehog_2013**]

References

[1] Mutundis. *File:Pre-mRNA to mRNA MH.svg*. [Online; accessed 23-August-2020]. 2015. URL: https://commons.wikimedia.org/wiki/File:Pre-mRNA_to_mRNA_MH.svg.

[2] E. Kim, A. Goren, and G. Ast. "Alternative splicing: current perspectives". In: *Bioessays* 30.1 (Jan. 2008), pp. 38–47.

[3] N. L. Barbosa-Morais et al. "The evolutionary landscape of alternative splicing in vertebrate species". In: *Science* 338.6114 (Dec. 2012), pp. 1587–1593.

[4] C. W. Sugnet et al. "Transcriptome and genome conservation of alternative splicing events in humans and mice". In: *Pac Symp Biocomput* (2004), pp. 66–77.

[5] L. Liu et al. "Aberrant regulation of alternative pre-mRNA splicing in hepatocellular carcinoma". In: *Crit. Rev. Eukaryot. Gene Expr.* 24.2 (2014), pp. 133–149.

[6] Hannes Bretschneider. "Alternative Splice Site Prediction with Deep Learning ". In: (June 2019).

[7] Susan M. Berget, Claire Moore, and Phillip A. Sharp. "Spliced segments at the 5' terminus of adenovirus 2 late mRNA". In: *Proceedings of the National Academy of Sciences* 74.8 (1977), pp. 3171–3175. eprint: https://www.pnas.org/content/74/8/3171.full.pdf. URL: https://www.pnas.org/content/74/8/3171.

[8] C. F. Rowlands, D. Baralle, and J. M. Ellingford. "Machine Learning Approaches for the Prioritization of Genomic Variants Impacting Pre-mRNA Splicing". In: *Cells* 8.12 (Nov. 2019).

[9] B. M. Brinkman. "Splice variants as cancer biomarkers". In: *Clin. Biochem.* 37.7 (July 2004), pp. 584–594.

[10] K. Q. Le et al. "Alternative splicing as a biomarker and potential target for drug discovery". In: *Acta Pharmacol. Sin.* 36.10 (Oct. 2015), pp. 1212–1218.

[11] Y. Barash et al. "Deciphering the splicing code". In: *Nature* 465.7294 (May 2010), pp. 53–59.

[12] Y. Barash, B. J. Blencowe, and B. J. Frey. "Model-based detection of alternative splicing signals". In: *Bioinformatics* 26.12 (June 2010), pp. i325–333.

[13] J. P. Venables et al. "Identification of alternative splicing markers for breast cancer". In: *Cancer Res.* 68.22 (Nov. 2008), pp. 9525–9531.

[14] M. R. Gazzara et al. "In silico to in vivo splicing analysis using splicing code models". In: *Methods* 67.1 (May 2014), pp. 3–12.

[15] H. Y. Xiong et al. “RNA splicing. The human splicing code reveals new insights into the genetic determinants of disease”. In: *Science* 347.6218 (Jan. 2015), p. 1254806.

[16] H. Y. Xiong, Y. Barash, and B. J. Frey. “Bayesian prediction of tissue-regulated splicing using RNA sequence and cellular context”. In: *Bioinformatics* 27.18 (Sept. 2011), pp. 2554–2562.

[17] M. K. Leung et al. “Deep learning of the tissue-regulated splicing code”. In: *Bioinformatics* 30.12 (June 2014), pp. i121–129.

[18] A. Jha, M. R. Gazzara, and Y. Barash. “Integrative deep models for alternative splicing”. In: *Bioinformatics* 33.14 (July 2017), pp. i274–i282.

[19] M. Oubounyt et al. “Deep Learning Models Based on Distributed Feature Representations for Alternative Splicing Prediction”. In: *IEEE Access* 6 (2018), pp. 58826–58834.

[20] C. Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826.

[21] P. J. Shepard et al. “Efficient internal exon recognition depends on near equal contributions from the 3’ and 5’ splice sites”. In: *Nucleic Acids Res.* 39.20 (Nov. 2011), pp. 8928–8937.

[22] A. Busch and K. J. Hertel. “Splicing predictions reliably classify different types of alternative splicing”. In: *RNA* 21.5 (May 2015), pp. 813–823.

[23] Z. Louadi et al. “Deep Splicing Code: Classifying Alternative Splicing Events Using Deep Learning”. In: *Genes (Basel)* 10.8 (Aug. 2019).

[24] A. Busch and K. J. Hertel. “HEXEvent: a database of Human EXon splicing Events”. In: *Nucleic Acids Res.* 41.Database issue (Jan. 2013), pp. D118–124.

[25] L. J. Carithers et al. “A Novel Approach to High-Quality Postmortem Tissue Procurement: The GTEx Project”. In: *Biopreserv Biobank* 13.5 (Oct. 2015), pp. 311–319.

[26] I. Streeter et al. “The human-induced pluripotent stem cell initiative-data resources for cellular genetics”. In: *Nucleic Acids Res.* 45.D1 (Jan. 2017), pp. D691–D697.

[27] S. Schafer et al. “Alternative Splicing Signatures in RNA-seq Data: Percent Spliced in (PSI)”. In: *Curr Protoc Hum Genet* 87 (Oct. 2015), pp. 1–11.

[28] J. L. Trincado et al. “SUPPA2: fast, accurate, and uncertainty-aware differential splicing analysis across multiple conditions”. In: *Genome Biol.* 19.1 (Mar. 2018), p. 40.

[29] J. Vaquero-Garcia et al. “A new view of transcriptome complexity and regulation through the lens of local splicing variations”. In: *Elife* 5 (Feb. 2016), e11752.

[30] E. Afgan et al. “The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update”. In: *Nucleic Acids Res.* 44.W1 (July 2016), W3–W10.

[31] A. Dobin et al. “STAR: ultrafast universal RNA-seq aligner”. In: *Bioinformatics* 29.1 (Jan. 2013), pp. 15–21.

[32] M. K. Sakharkar, V. T. Chow, and P. Kangueane. "Distributions of exons and introns in the human genome". In: *In Silico Biol. (Gedrukt)* 4.4 (2004), pp. 387–393.

[33] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv e-prints*, arXiv:1301.3781 (Jan. 2013), arXiv:1301.3781. arXiv: 1301.3781 [cs.CL].

[34] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *arXiv e-prints*, arXiv:1310.4546 (Oct. 2013), arXiv:1310.4546. arXiv: 1310.4546 [cs.CL].

[35] Quoc V. Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *arXiv e-prints*, arXiv:1405.4053 (May 2014), arXiv:1405.4053. arXiv: 1405.4053 [cs.CL].

[36] Ryan Kiros et al. "Skip-Thought Vectors". In: *arXiv e-prints*, arXiv:1506.06726 (June 2015), arXiv:1506.06726. arXiv: 1506.06726 [cs.CL].

[37] Eric Lander et al. "Initial sequencing and analysis of the human genome". In: *Nature* 409 (Mar. 2001), pp. 860–921.

[38] W Kent et al. "The human genome browser at UCSC". In: *Genome research* 12 (July 2002), pp. 996–1006.

[39] R. Milo et al. "BioNumbers–the database of key numbers in molecular and cell biology". In: *Nucleic Acids Res.* 38.Database issue (Jan. 2010). https://bionumbers.hms.harvard.edu/bionumber.aspx?id=105336&ver=6, pp. D750–753.

[40] L.J.P. van der Maaten and G.E. Hinton. "Visualizing High-Dimensional Data Using t-SNE". In: (2008).

[41] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *arXiv e-prints*, arXiv:1409.3215 (Sept. 2014), arXiv:1409.3215. arXiv: 1409.3215 [cs.CL].

[42] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[43] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *arXiv e-prints*, arXiv:1406.1078 (June 2014), arXiv:1406.1078. arXiv: 1406.1078 [cs.CL].

[44] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *arXiv e-prints*, arXiv:1609.08144 (Sept. 2016), arXiv:1609.08144. arXiv: 1609.08144 [cs.CL].

[45] Sebastian Ruder. *A Review of the Neural History of Natural Language Processing.* http://ruder.io/a-review-of-the-recent-history-of-nlp/. 2018.

[46] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv e-prints*, arXiv:1409.0473 (Sept. 2014), arXiv:1409.0473. arXiv: 1409.0473 [cs.CL].

[47] Ashish Vaswani et al. "Attention Is All You Need". In: *arXiv e-prints*, arXiv:1706.03762 (June 2017), arXiv:1706.03762. arXiv: 1706.03762 [cs.CL].

[48] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv e-prints*, arXiv:1810.04805 (Oct. 2018), arXiv:1810.04805. arXiv: 1810.04805 [cs.CL].

[49] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: (2020). arXiv: 2005.14165 [cs.CL].

[50] Byunghan Lee et al. “DNA-Level Splice Junction Prediction using Deep Recurrent Neural Networks”. In: *arXiv e-prints*, arXiv:1512.05135 (Dec. 2015), arXiv:1512.05135. arXiv: 1512.05135 [cs.LG].

[51] Jim Clauwaert and Willem Waegeman. “Novel transformer networks for improved sequence labeling in genomics”. In: *bioRxiv* (2020). eprint: https://www.biorxiv.org/content/early/2020/02/28/836163.full.pdf. URL: https://www.biorxiv.org/content/early/2020/02/28/836163.

[52] Guillaume Klein et al. “OpenNMT: Open-Source Toolkit for Neural Machine Translation”. In: *Proc. ACL*. 2017. URL: https://doi.org/10.18653/v1/P17-4012.

[53] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[54] Radim Řehůřek and Petr Sojka. “Software Framework for Topic Modelling with Large Corpora”. English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[55] Xiaokang Zhang et al. “Recognition of alternatively spliced cassette exons based on a hybrid model”. In: *Biochemical and Biophysical Research Communications* 471 (Feb. 2016).

[56] Juan A. Botia et al. “G2P: Using machine learning to understand and predict genes causing rare neurological disorders”. In: *bioRxiv* (2018). eprint: https://www.biorxiv.org/content/early/2018/03/27/288845.full.pdf. URL: https://www.biorxiv.org/content/early/2018/03/27/288845.

[57] L. Li et al. “A classification of alternatively spliced cassette exons using AdaBoost-based algorithm”. In: *2014 IEEE International Conference on Information and Automation (ICIA)*. 2014, pp. 370–375.